IBM

ABCs of z/OS System Programming: Volume 9

- z/OS UNIX, TCP/IP installation
- zSeries file system, z/OS UNIX security
- Shell and programming tools

Paul Rogers
Richard Conway

Redbooks

ibm.com/redbooks

International Technical Support Organization

ABCs of z/OS System Programming: Volume 9

April 2011

SG24-6989-05

Note: Before using this information and the product it supports, read the information in “Notices” on page xvii.

Sixth Edition (April 2011)

This edition applies to version 1 release 12 modification 0 of IBM z/OS (product number 5694-A01) and to all subsequent releases and modifications until otherwise indicated in new editions.

© Copyright International Business Machines Corporation 2011. All rights reserved.
Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices . . . xvii
Trademarks . . . xviii

Preface . . . xix
The team who wrote this book . . . xx
Now you can become a published author, too! . . . xx
Comments welcome . . . xxi
Stay connected to IBM Redbooks . . . xxi

Chapter 1. Products and components . . . 1
1.1 z/OS UNIX System Services . . . 2
1.2 z/OS UNIX System Services and z/OS . . . 4
1.3 z/OS and z/OS UNIX . . . 6
1.4 Product and component support for z/OS UNIX . . . 8
1.5 Security Server RACF . . . 10
1.6 Data Facility System-Managed Storage (DFSMS) . . . 11
1.7 Transmission Control Protocol/Internet Protocol (TCP/IP) . . . 12
1.8 System Modification Program Extended (SMP/E) . . . 14
1.9 System Management Facility (SMF) . . . 15
1.10 Resource Measurement Facility (RMF) . . . 16
1.11 Virtual Lookaside Facility (VLF) . . . 17
1.12 Time Sharing Option/Extended (TSO/E) . . . 19
1.13 Workload Manager (WLM) . . . 20
1.14 Tivoli Storage Manager (TSM) . . . 21

Chapter 2. z/OS UNIX overview . . . 23
2.1 z/OS UNIX and UNIX applications . . . 24
2.2 z/OS UNIX terminology overview . . . 26
2.3 HFS and zFS file system PFSes . . . 28
2.4 Using z/OS UNIX . . . 29
2.5 z/OS UNIX System Services . . . 31
2.6 Physical file systems . . . 33
2.7 z/OS UNIX file sharing in a sysplex . . . 35
2.8 z/OS UNIX file systems . . . 37
2.9 File system data sets . . . 38
2.10 Root file system . . . 39
2.11 File and directory permission bits . . . 40
2.12 MVS data sets versus file system files . . . 41
2.13 zFS or HFS data sets . . . 42
2.14 z/OS UNIX components . . . 43
2.15 z/OS UNIX programs (processes) . . . 45
2.16 Create a process . . . 47
2.17 z/OS UNIX processes . . . 49
2.18 z/OS UNIX interactive interfaces . . . 51
2.19 ISPF Option 6 . . . 53
2.20 ISHELL command (ish) . . . 54
2.21 User's files and directories . . . 55
2.22 OMVS shell session . . . 56
2.23 OMVS command shell session . . . 57

© Copyright IBM Corp. 2011. All rights reserved.

2.24 ls -al command - list files in the root 59
2.25 ISPF edit mode for a z/OS UNIX file 60
2.26 ISPF edit mode for a z/OS UNIX file 61
2.27 Specifying the z/OS UNIX path name 62
2.28 ISPF ENQs on z/OS UNIX files 63
2.29 Support for editing ASCII data 64
2.30 Handling line feed characters 66
2.31 Direct login to shell 68
2.32 Telnet access to z/OS UNIX 70

Chapter 3. z/OS UNIX System Services preinstallation requirements 71
3.1 Customization of the root 72
3.2 Installing z/OS using ServerPac 74
3.3 Installing z/OS using CBPDO 76
3.4 ServerPac and CBPDO 77
3.5 z/OS UNIX System Services installation 79
3.6 z/OS UNIX security 80
3.7 RACF definitions 81
3.8 RACF OMVS segments 83
3.9 OMVS segment fields 85
3.10 UNIX security 87
3.11 z/OS UNIX superuser 89
3.12 RACF commands and user IDs 91
3.13 RACF commands to define groups 92
3.14 RACF commands to define users 93
3.15 LU and LG command examples 94
3.16 Define a terminal group name 95
3.17 TSO/E support 96
3.18 User access to TSO/E commands 97

Chapter 4. z/OS UNIX System Services installation 99
4.1 z/OS UNIX PARMLIB - PROCLIB members 100
4.2 IEASYSxx PARMLIB member 101
4.3 z/OS UNIX minimum mode 102
4.4 Minimum mode: TFS 104
4.5 z/OS UNIX full-function mode 106
4.6 z/OS HFS root 108
4.7 zFS with z/OS V1R7 110
4.8 HFS or zFS data sets 111
4.9 Set data set type 112
4.10 Choosing zFS 113
4.11 ServerPac changes if using zFS 114
4.12 UNIX utilities: TSO/E commands 115
4.13 UNIX commands to move and copy data 117
4.14 The pax and tar utilities 118
4.15 ServerPac z/OS UNIX installation 120
4.16 Non-volatile root file system 122
4.17 Installation of other products 123
4.18 z/OS UNIX System Services installation 124

Chapter 5. z/OS UNIX shell and utilities 125
5.1 The z/OS UNIX shell 126
5.2 Input, output, errors with UNIX shell 127
5.3 Accessing the z/OS UNIX shell 129

5.4 Controlling session resources . . . 130
5.5 Dynamic /dev . . . 132
5.6 Invoking the shell via TSO/E . . . 133
5.7 Invoking the shell via rlogin or telnet . . . 135
5.8 rlogin and telnet access . . . 137
5.9 Customizing z/OS UNIX initialization . . . 139
5.10 Initializing z/OS UNIX . . . 141
5.11 Environment variables . . . 142
5.12 Environment variables . . . 143
5.13 The /etc/init.options file . . . 144
5.14 The etc/rc file . . . 145
5.15 The /etc/inittab file with z/OS V1R8 . . . 148
5.16 The _BPXK_INITTAB_RESPAWN variable . . . 149
5.17 Rules for coding /etc/inittab . . . 150
5.18 Customizing the OMVS command . . . 152
5.19 Shell environment variables . . . 154
5.20 Customizing your shell environment . . . 156
5.21 Global variables in /etc/profile . . . 157
5.22 User-defined settings . . . 159
5.23 Setting the time zone . . . 161
5.24 Customizing the C89/CC compilers . . . 163
5.25 Code page tables . . . 165
5.26 Specifying a code page . . . 166
5.27 Internationalization variables (locales) . . . 167
5.28 Setting the region size . . . 169
5.29 Setting up printers for shell users . . . 170
5.30 Installing books for OHELP . . . 171
5.31 Using the man command . . . 172
5.32 Enabling various tools . . . 174
5.33 SVP for z/OS UNIX and tools . . . 176
5.34 Setup Verification Program (SVP) . . . 178

Chapter 6. Security customization . . . 181
6.1 RACF OMVS segments . . . 182
6.2 z/OS UNIX UIDs and GIDs . . . 184
6.3 z/OS UNIX users and groups . . . 185
6.4 BPXROOT user ID . . . 187
6.5 Superuser with appropriate authority . . . 189
6.6 Commands for superusers . . . 190
6.7 z/OS UNIX security and RACF profiles . . . 191
6.8 z/OS UNIX security: BPX.SUPERUSER . . . 194
6.9 z/OS UNIX superuser granularity . . . 196
6.10 Resource names: UNIXPRIV . . . 198
6.11 z/OS UNIX UNIXPRIV class profiles . . . 201
6.12 Assigning UIDs . . . 202
6.13 Shared UID prevention . . . 204
6.14 Automatic UID and GID assignment . . . 205
6.15 Automatic assignment requirements . . . 206
6.16 Automatic assignment examples . . . 208
6.17 Automatic assignment with RRSF . . . 210
6.18 z/OS UNIX security: File security packet . . . 211
6.19 Octal values for permission bits . . . 213
6.20 Data set security versus file security . . . 215

6.21 z/OS UNIX user's security environment 216
6.22 Access checking flows 218
6.23 File authorization checking flow 219
6.24 POSIX standard and UNIX ACLs 220
6.25 Limitations of current permission bits 221
6.26 FSPs and ACLs. 222
6.27 Access control list table. 223
6.28 File authorization check summary. 224
6.29 Profiles in UNIXPRIV class 225
6.30 Profiles in UNIXPRIV class (2) 226
6.31 RACF RESTRICTED attribute 228
6.32 z/OS UNIX file access checking 229
6.33 RESTRICTED user profile 231
6.34 Restricted user access checking. 232
6.35 Access checking with ACLs (1). 233
6.36 Access checking with ACLs (2). 234
6.37 Create ACLs 235
6.38 ACL types 237
6.39 OMVS shell commands for ACLs 238
6.40 Create ACLs for a specific directory 239
6.41 Create an access ACL 240
6.42 Display the access ACL 241
6.43 Create a directory default ACL 242
6.44 Create a file default ACL. 243
6.45 Creating all ACL types 244
6.46 Using the ISHELL panel 245
6.47 Create an access ACL using ISHELL 247
6.48 File attributes panel for /u/harry 248
6.49 File attributes panel showing ACLs. 249
6.50 Select option to create an access ACL 250
6.51 Create an access ACL 251
6.52 Add an access ACL. 252
6.53 Access ACL after creation. 253
6.54 ACL inheritance: New directory/new file 254
6.55 Multilevel security with z/OS V1R5 255
6.56 Multilevel security (MLS) 256
6.57 MLS support for z/OS UNIX 257
6.58 Mandatory access control (MAC) 259
6.59 Discretionary access control (DAC) 261
6.60 SECLABELs and MAC 262
6.61 Special SECLABELs and definitions. 264
6.62 SYSMULTI SECLABEL. 265
6.63 z/OS UNIX and SECLABELs 266
6.64 Understanding UMASK. 267
6.65 Displaying the UMASK 268
6.66 Default permissions and UMASK 269
6.67 Example of creating a new file 271
6.68 Can user JOE access the file 273
6.69 Can user ANN copy the file. 274
6.70 Setting file permissions 275
6.71 Setting file permissions 277
6.72 List file and directory information 278
6.73 Introducing daemons. 279

6.74 z/OS UNIX daemons . . . 281
6.75 UNIX-level security for daemons . . . 283
6.76 z/OS UNIX security: BPX.DAEMON . . . 284
6.77 RACF program control . . . 285
6.78 z/OS UNIX-level security for daemons . . . 287
6.79 Start options for daemons . . . 288
6.80 Define daemon security . . . 289
6.81 Auditing options for z/OS UNIX . . . 291
6.82 File-based auditing . . . 292
6.83 Audit z/OS UNIX events . . . 294
6.84 Chaudit command . . . 296
6.85 List audit information for files . . . 298
6.86 Auditing reports . . . 299
6.87 Maintain z/OS UNIX-level security . . . 300
6.88 Setting up z/OS UNIX (1) . . . 301
6.89 Setting up z/OS UNIX (2) . . . 302
6.90 Setting up z/OS UNIX (3) . . . 303
6.91 Setting up z/OS UNIX (4) . . . 304
6.92 Setting up z/OS UNIX (5) . . . 305
6.93 RACF definitions for zFS . . . 306
6.94 UNIXPRIV class with z/OS V1R3 and zFS . . . 307
6.95 List current user IDs with the ISHELL . . . 308
6.96 The BPXBATCH utility . . . 309
6.97 The BPXBATCH job . . . 310
6.98 BPXBATCH and shell commands . . . 312

Chapter 7. zFS file systems . . . 315
7.1 zSeries File System (zFS) . . . 316
7.2 zFS compatibility mode aggregate . . . 317
7.3 BPXPRMxx definitions for zFS . . . 318
7.4 zFS colony address space . . . 319
7.5 HFS data sets and zFS data sets . . . 320
7.6 zFS utilities and commands . . . 321
7.7 zfsadm command . . . 323
7.8 Allocate Linear VSAM data set . . . 324
7.9 Create the aggregate from ISHELL . . . 325
7.10 Format VSAM space - create aggregate . . . 326
7.11 Format the aggregate . . . 328
7.12 Ioeagfmt successful messages . . . 330
7.13 Grow an aggregate . . . 332
7.14 Dynamic aggregate extension . . . 333
7.15 The -grow option . . . 335
7.16 -grow option for formatting . . . 337
7.17 Mounting the file system . . . 338
7.18 ISHELL support for zFS . . . 339
7.19 Panel of attached zFS aggregates . . . 340
7.20 Display aggregate attributes . . . 341
7.21 Display attached aggregates . . . 342
7.22 List file systems . . . 343
7.23 zFS aggregate space commands . . . 344
7.24 Command for aggregate display . . . 346
7.25 zFS threshold monitoring space usage . . . 347
7.26 Dynamic configuration parameters . . . 348

7.27 zfsadm configquery command options 350
7.28 zfsadm config command options. 351
7.29 Defining IOEFSPRM options. 353
7.30 Logical PARMLIB support 354
7.31 Specifying PARMLIB members. 355
7.32 Add a volume to a zFS aggregate. 357
7.33 zFS migration considerations 358
7.34 HFS/zFS as generic file system type 359
7.35 Migration considerations 360
7.36 Migration tool. 361
7.37 Migration checks file system type 362
7.38 Migration tool enhancements with APAR OA18196 363
7.39 REXX exec - BPXWH2Z 365
7.40 BPXWH2Z panels 366
7.41 Space allocations - HFS versus zFS. 367
7.42 Using the migration tool 368
7.43 BPXWH2Z panels 369
7.44 Using SMS if required 370
7.45 Migrate in the foreground 371
7.46 Migration steps 372
7.47 Migration steps 373
7.48 Migration steps continued 374
7.49 Alter allocation parameters panel 375
7.50 APAR OA18196 - Exact data set match 376
7.51 Migrating a list of data sets 377
7.52 Data set list displayed 378
7.53 Health Checker USS_HFS_DETECTED. 379
7.54 Health Checker USS_HFS_DETECTED. 380
7.55 Check with RUN_ON_MOUNT=YES 382
7.56 Special characters in zFS aggregates 384
7.57 BPXMTEXT shell command 385

Chapter 8. File sharing in a sysplex 387
8.1 Shared file systems in a sysplex 388
8.2 Sysplex environment setup 389
8.3 File systems in a shared sysplex 391
8.4 Multiple systems: Different versions 392
8.5 Update BPXPRMxx for sysplex. 393
8.6 OMVS couple data set 395
8.7 OMVS couple data set commands 397
8.8 D XCF,COUPLE,TYPE=BPXMCDS 398
8.9 Change to the SETXCF command 399
8.10 New message for command failure. 400
8.11 File sharing in a sysplex 401
8.12 Mounting shared sysplex file systems. 402
8.13 Accessing shared sysplex file systems 403
8.14 Shared file system AUTOMOVE takeover 404
8.15 Moving file systems in a sysplex 406
8.16 Requests to LFS to PFS to files 407
8.17 Systems accessing file systems 408
8.18 zFS sharing mode terminology 409
8.19 Defining zFS as sysplex-unaware. 410
8.20 zFS File systems with APAR OA29712 411

8.21 Defining zFS as sysplex-aware 413
8.22 APAR OA29619 414
8.23 New zFS configuration options (OA29619) 415
8.24 Admin levels in a mixed sysplex 417
8.25 Defing zFS as syplex-aware 419
8.26 Using the sysplex=filesys parameter 421
8.27 Using the sysplex=filesys parameter 422
8.28 The sysplex_filesys_sharemode parameter 424
8.29 sysplex_filesys_sharemode considerations 425
8.30 sysplex_filesys_sharemode considerations 427
8.31 zFS mixed environment 429
8.32 zFS cache management 431
8.33 zFS sysplex-aware on a file system basis 432
8.34 Automatic movement of file systems 434

Chapter 9. Managing file systems 435
9.1 Hierarchical file system (HFS) 436
9.2 File linking 437
9.3 Hard links 438
9.4 Symbolic links 439
9.5 External links 440
9.6 File system structure 442
9.7 Temporary directory space 443
9.8 Temporary file system (TFS) 445
9.9 Colony address space 447
9.10 Mounting file systems 449
9.11 Mount and unmount 451
9.12 Managing user file systems 452
9.13 User file systems: Direct mount 453
9.14 Mounting file systems 455
9.15 Option 3: Mount 457
9.16 Automount facility 458
9.17 Automount setup 460
9.18 Generic match on lowercase names 462
9.19 Automount facility overview 463
9.20 Activating automount 464
9.21 SETOMVS RESET=xx implementation 465
9.22 Issue the SETOMVS command 466
9.23 Updating an existing automount policy 467
9.24 Example of new options 468
9.25 One auto.master for a sysplex 469
9.26 HFS to zFS automount 470
9.27 HFS to zFS automount 471
9.28 Automount migration considerations 473
9.29 How to mount zFS file systems 474
9.30 Using direct mount commands 475
9.31 Mounting zFS file systems 477
9.32 MOUNT commands 478
9.33 zFS file system clone 480
9.34 Backup file system - zFS clone 482
9.35 zFS clone mounted 483
9.36 Using the clone 484
9.37 Mounting File Systems - (HFS - zFS) 485

9.38 MOUNT command options 486
9.39 UNMOUNT option 487
9.40 UNMOUNT option support 488
9.41 UNMOUNT option support 489
9.42 Mount file system panel 490
9.43 Set AUTOMOVE options 491
9.44 AUTOMOVE system list (syslist) 492
9.45 AUTOMOVE parameters for mounts 493
9.46 AUTOMOVE wildcard support 494
9.47 AUTOMOVE wildcard examples 495
9.48 Stopping zFS 496
9.49 Restarting the PFS 498
9.50 Mounting file systems with SET OMVS 499
9.51 Messages from shutdown of a ZFS single system 500
9.52 Messages for the restart of ZFS 501
9.53 Shutdown and recovery scenarios 502
9.54 zFS commands in a sysplex 504
9.55 zfsadm command changes for sysplex 505
9.56 zfsadm command changes 506
9.57 Configuration options 507
9.58 Command forwarding support 508
9.59 Indirect volsers with zFS data sets 509
9.60 Using indirect volume serials with cloned zFS data sets 510
9.61 Define a VSAM LDS and format 512
9.62 Delete the data set and IDCAMS DEFINE 513
9.63 Centralized BRLM support 514
9.64 Distributed BRLM 515
9.65 Define BRLM option in CDS 516
9.66 BRLM problems in a sysplex 517
9.67 z/OS V1R8 BRLM recovery of locks 518
9.68 File system access 519
9.69 File access 521
9.70 List file and directory information 522
9.71 File security packet - extattr bits 523
9.72 Extended attributes 524
9.73 APF-authorized attribute 526
9.74 Activate program control 527
9.75 Shared address space attribute 528
9.76 Shared library attribute 529
9.77 File format attribute 530
9.78 Extended attribute command example 532
9.79 Sticky bit 533
9.80 Set the UID/GID bit 535

Chapter 10. Overview of TCP/IP 537
10.1 Introduction to TCP/IP 538
10.2 TCP/IP terminology 540
10.3 IPv4 addressing 542
10.4 IPv6 addresses 544
10.5 User login to the z/OS UNIX shell 545
10.6 Resolver address space 547
10.7 TCPDATA search order 549
10.8 Create configuration files used by TCP/IP 551

10.9 Customize the TCP/IP profile data set 553
10.10 Customize TCPDATA 555
10.11 z/OS IP search order. 557
10.12 z/OS IP search order (2) 559
10.13 Customize the TCP/IP procedure 560
10.14 Customizing PARMLIB members for TCP/IP 562
10.15 PARMLIB members to customize for TCP/IP 563
10.16 RACF customization for TCP/IP 564
10.17 Customizing TCP/IP 566
10.18 TCP/IP shell commands 567

Chapter 11. TCP/IP applications 569
11.1 Overview of z/OS UNIX data access 570
11.2 Sockets 572
11.3 z/OS Communications Server. 574
11.4 z/OS UNIX sockets support 576
11.5 Customizing sockets 577
11.6 Logging in to the z/OS UNIX shell. 578
11.7 Using inetd - master of daemons 579
11.8 Customize inetd 580
11.9 Customize inetd (2) 581
11.10 Login to a Unix system 583
11.11 rlogin to z/OS UNIX services. 584
11.12 Activating z/OS UNIX rlogin daemon 585
11.13 Comparing shell login methods. 587
11.14 Define TCP/IP daemons 588
11.15 The syslogd daemon. 589
11.16 The FTPD daemon 591
11.17 z/OS IP search order for FTP 592
11.18 z/OS IP search order for /etc/services 593
11.19 Start the TCP/IP daemons 594
11.20 Message integration support. 595
11.21 Message routing to z/OS. 596
11.22 syslogd command options. 597
11.23 syslogd defined instances 598
11.24 syslogd configuration file. 599
11.25 Start procedure for syslogd 601
11.26 syslogd availability considerations 602

Chapter 12. z/OS UNIX PARMLIB members 605
12.1 BPXPRMxx PARMLIB member 606
12.2 BPXPRMFS PARMLIB member 607
12.3 BPXPRMxx control keywords 608
12.4 BPXPRMxx PARMLIB member 609
12.5 Controlling the number of processes 610
12.6 Resource limits for processes 612
12.7 MAXFILEPROC statement 614
12.8 Setting file descriptors. 616
12.9 Setting file descriptor for a single user 618
12.10 Display BPXPRMxx limits 620
12.11 Memory mapped files 622
12.12 Controlling thread resources. 623
12.13 Creating a process using fork() 625

12.14 Values for forked child process 627
12.15 Starting a program with exec() 628
12.16 Values passed for exec() program 630
12.17 z/OS UNIX processes get STEPLIBs 631
12.18 Locating programs for z/OS UNIX processes 633
12.19 Shared pages for the fork() function 634
12.20 Spawn function 635
12.21 Interprocess communication functions 637
12.22 Address Space Memory Map z/OS V1R5 638
12.23 Control IPC resources 640
12.24 Kernel support for IBM 5.0 JVM 642
12.25 Interprocess communication signals 644
12.26 Pipes 646
12.27 Other BPXPRMxx keywords 647
12.28 More BPXPRMxx parameters 648
12.29 FILESYSTYPE statement 649
12.30 FILESYSTYPE and NETWORK 650
12.31 ROOT and MOUNT statements 651
12.32 Examples of MKDIR in BPXPRMxx 652
12.33 Allocating SWA above the line 653
12.34 z/OS UNIX Web site 654

Chapter 13. Maintenance 655
13.1 Example of SMP/E SMPMCS 656
13.2 Active root file system 657
13.3 Inactive root file system (clone) 658
13.4 /SERVICE directory 660
13.5 Sample SMP/E DDDEFs 661
13.6 Prepare for SMP/E 662
13.7 SMP/E APPLY process 664
13.8 Supporting multiple service levels 666
13.9 Supporting multiple service levels (2) 667
13.10 ISHELL display of root 668
13.11 The chroot command 669
13.12 Testing a root file system 670
13.13 Testing the updated root 671
13.14 Dynamic service activation 672
13.15 Dynamic service activation commands 673
13.16 Using the new service 675
13.17 Deactivate service 677
13.18 Display service 678

Chapter 14. z/OS UNIX operations 679
14.1 Commands to monitor z/OS UNIX 680
14.2 Display summary of z/OS UNIX 682
14.3 Display z/OS UNIX options 683
14.4 Display BPXPRMxx limits 684
14.5 Display address space information 685
14.6 Display process information 687
14.7 Display the kernel address space 689
14.8 z/OS V1R7 command options 691
14.9 Mount error messages displayed 692
14.10 Mount failure messages 693

14.11 Stopping BPXAS address spaces 694
14.12 LFS soft shutdown 695
14.13 z/OS V1R8 file system shutdown 696
14.14 Options with the D OMVS,F command 698
14.15 Options with the D OMVS,F command 699
14.16 New command examples 700
14.17 New command examples 701
14.18 New command examples 702
14.19 z/OS UNIX shutdown 703
14.20 Recommended shutdown procedures 705
14.21 Application registration 706
14.22 Display application registration 707
14.23 F OMVS,SHUTDOWN 708
14.24 Blocking processes completion 709
14.25 Shutdown processing completion 710
14.26 Shutdown for permanent processes 711
14.27 Shutdown processing final cleanup 712
14.28 F OMVS,RESTART 713
14.29 Alternate sysplex root 714
14.30 Defining an alternate root and mount 715
14.31 BPXPRMxx parmlib member - ALTROOT 717
14.32 Display information about processes 719
14.33 Stop a process 720
14.34 Superkill function 721
14.35 Superkill example 722
14.36 Changing OMVS parameter values 723
14.37 Manage interprocess communication 724
14.38 System problems 725
14.39 z/OS UNIX abends and messages 726
14.40 USS errors and codes 727
14.41 CTIBPX00 and CTCBPXxx 729
14.42 Tracing z/OS UNIX events 730
14.43 Debugging a z/OS UNIX problem 731
14.44 IPCS OMVSDATA reports 732

Chapter 15. z/OS UNIX shell and programming tools 733
15.1 Language Environment run-time library 734
15.2 Using pre-LE run-time libraries 736
15.3 Overview of c89/cc/c++ 738
15.4 Customization of /etc/profile for c89/cc/c++ 739
15.5 Compile, link-edit, and run 741
15.6 Customization of Java for z/OS 743
15.7 Java virtual machine 744
15.8 Management of software and the make utility 745
15.9 The dbx debugger 747
15.10 The dbx debugger 749
15.11 Introduction to shells 750
15.12 REXX, CLISTs, and shell scripts 752
15.13 Shell script syntax 754
15.14 BPXBATCH enhancements 755
15.15 BPXBATCH implementation 756
15.16 BPXBATCH summary 758
15.17 TSO/E ALLOCATE command for STDPARM 760

15.18 STDERR and STDOUT as MVS data sets 761
15.19 BPXBATCH sample job 763
15.20 Child process created for MVS data sets 764
15.21 BPXBATCH utility 765

Chapter 16. Performance, debugging, recovery, and tuning 767
16.1 z/OS UNIX performance overview 768
16.2 WLM in goal mode 769
16.3 Defining service classes 770
16.4 Workload Manager service classes 771
16.5 Subsystem type panel 772
16.6 WLM work qualifiers 773
16.7 OMVS work qualifiers 774
16.8 Defining classification rules 775
16.9 Classification rules 776
16.10 Classification rules for STC 777
16.11 Virtual lookaside facility (VLF) 778
16.12 VLF for z/OS UNIX 780
16.13 COFVLFxx updates for z/OS UNIX 781
16.14 AIM Stage 3 782
16.15 Further tuning tips 784
16.16 zFS performance tuning 786
16.17 zFS cache 787
16.18 zFS cache locations 789
16.19 Metadata backing cache 790
16.20 Performance APIs 791
16.21 Performance monitoring APIs 792
16.22 zfsadm query command 794
16.23 The IOEZADM utility from TSO for commands 795
16.24 Directory cache 796
16.25 Directory caching display tool 797
16.26 File system monitoring tool (FSMON) 798
16.27 The zfsadm query -iobyaggr command 799
16.28 SMF recording 800
16.29 SMF 92 subtype 14 enhancement 802
16.30 RMF reporting 803
16.31 RMF Monitor III support for zFS 804
16.32 zFS access to file systems 805
16.33 RMF Overview Report Selection Menu 807
16.34 zFS Summary Report 808
16.35 zFS Summary I/O details by type 809
16.36 User and vnode cache detail 811
16.37 DFSMSdss dump and restore for zFS file systems 813
16.38 UNQUIESCE command 814
16.39 zFS recovery support 815
16.40 zFS aggregate corruption 817
16.41 Debugging data sets 819
16.42 zFS hang detection 821
16.43 zFS hang detection 823
16.44 z/OS UNIX Internet information 824

Related publications 825
IBM Redbooks 825

Other publications 825
Online resources 826
How to get IBM Redbooks 826
Help from IBM 826

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

© Copyright IBM Corp. 2011. All rights reserved.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at http://www.ibm.com/legal/copytrade.shtml

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

AIX®
BookManager®
C/370™
C/MVS™
CICS®
DB2®
Domino®
GDPS®
Geographically Dispersed Parallel Sysplex™
IBM®
IMS™
Language Environment®
Lotus®
NetView®
Open Class®
OS/390®
Parallel Sysplex®
ProductPac®
RACF®
Redbooks®
Redbooks (logo) ®
S/390®
SystemPac®
Tivoli®
VTAM®
WebSphere®
z/Architecture®
z/OS®
zSeries®

The following terms are trademarks of other companies:

PostScript, and Portable Document Format (PDF) are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, other countries, or both.

Novell, the Novell logo, and the N logo are registered trademarks of Novell, Inc. in the United States and other countries.

ABAP, SAP R/3, SAP, and SAP logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries.

Java, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Preface

The ABCs of z/OS® System Programming is a 13-volume collection that provides an introduction to the z/OS operating system and the hardware architecture. Whether you are a beginner or an experienced system programmer, the ABCs collection provides the information that you need to start your research into z/OS and related subjects. If you would like to become more familiar with z/OS in your current environment, or if you are evaluating platforms to consolidate your e-business applications, the ABCs collection will serve as a powerful technical tool.

This IBM® Redbooks® publication describes z/OS UNIX System Services (z/OS UNIX). It will help you install, tailor, configure, and use the z/OS Version 1 Release 6 version of z/OS UNIX. Topics covered in this volume are the products and components used with z/OS UNIX:

- An overview of z/OS UNIX
- z/OS UNIX pre-installation requirements
- Step-by-step installation of z/OS UNIX System Services using the ServerPac
- The z/OS UNIX shell and utilities
- z/OS UNIX security customization using RACF®
- The Hierarchical File System (HFS) and zFS
- TCP/IP overview and customization; TCP/IP applications
- z/OS UNIX PARMLIB member definitions
- How to perform maintenance
- Using z/OS UNIX operator commands
- The z/OS UNIX shell and programming tools
- Using the workload manager; performance and tuning considerations
- z/OS UNIX printing options

The contents of the volumes are as follows:

Volume 1: Introduction to z/OS and storage concepts, TSO/E, ISPF, JCL, SDSF, and z/OS delivery and installation

Volume 2: z/OS implementation and daily maintenance, defining subsystems, JES2 and JES3, LPA, LNKLST, authorized libraries, SMP/E, Language Environment®

Volume 3: Introduction to DFSMS, data set basics storage management hardware and software, catalogs, and DFSMStvs

Volume 4: Communication Server, TCP/IP, and VTAM®

Volume 5: Base and Parallel Sysplex®, System Logger, Resource Recovery Services (RRS), global resource serialization (GRS), z/OS system operations, automatic restart management (ARM), Geographically Dispersed Parallel Sysplex™ (GDPS®)

Volume 6: Introduction to security, RACF, Digital certificates and PKI, Kerberos, cryptography and z990 integrated cryptography, zSeries® firewall technologies, LDAP, and Enterprise identity mapping (EIM)

© Copyright IBM Corp. 2011. All rights reserved.

Volume 7: Printing in a z/OS environment, Infoprint Server and Infoprint Central

Volume 8: An introduction to z/OS problem diagnosis

Volume 9: z/OS UNIX System Services

Volume 10: Introduction to z/Architecture®, zSeries processor design, zSeries connectivity, LPAR concepts, HCD, and HMC

Volume 11: Capacity planning, performance management, WLM, RMF, and SMF

Volume 12: WLM

Volume 13: JES3

The team who wrote this book

This book was produced by a team of specialists from around the world working at the International Technical Support Organization, Poughkeepsie Center.

Paul Rogers is a Consulting IT Specialist at the International Technical Support Organization, Poughkeepsie Center. He writes extensively and teaches IBM classes worldwide on various aspects of z/OS, z/OS UNIX, JES3, and Infoprint Server. Before joining the ITSO 20 years ago, Paul worked in the IBM Installation Support Center (ISC) in Greenford, England for seven years providing OS/390® and JES support for IBM EMEA and also in the Washington Systems Center for three years. He has worked for IBM for 40 years.

Richard Conway is a systems programmer at the International Technical Support Organization, Poughkeepsie Center. He has extensive knowledge of z/OS UNIX System Services and all of the products installed at the ITSO. He has worked in the ITSO for the last 15 years.

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author - all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:

  ibm.com/redbooks

- Send your comments in an email to:

  redbooks@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- Find us on Facebook:

  http://www.facebook.com/IBMRedbooks

- Follow us on Twitter:

  http://twitter.com/ibmredbooks

- Look for us on LinkedIn:

  http://www.linkedin.com/groups?home=&gid=2130806

- Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

  https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm

- Stay current on recent Redbooks publications with RSS Feeds:

  http://www.redbooks.ibm.com/rss.html

1

Products and components

This chapter provides a brief overview of z/OS UNIX System Services (z/OS UNIX) and related components. z/OS UNIX interacts with the following elements and features of z/OS:

- BCP (WLM and SMF components)
- XPG4-compliant shell application
- Data Facility Storage Management Subsystem (DFSMS) (HFS is a component of DFSMS)
- Security Server RACF
- Resource Measurement Facility (RMF)
- Time Sharing Option/Extended (TSO/E)
- z/OS Communications Server (TCP/IP)
- System Modification Program Extended (SMP/E)
- Virtual Lookaside Facility (VLF)
- Tivoli® Storage Manager (TSM)
- z/OS Distributed File Service zSeries File System (zFS)

© Copyright IBM Corp. 2011. All rights reserved.

1.1 z/OS UNIX System Services

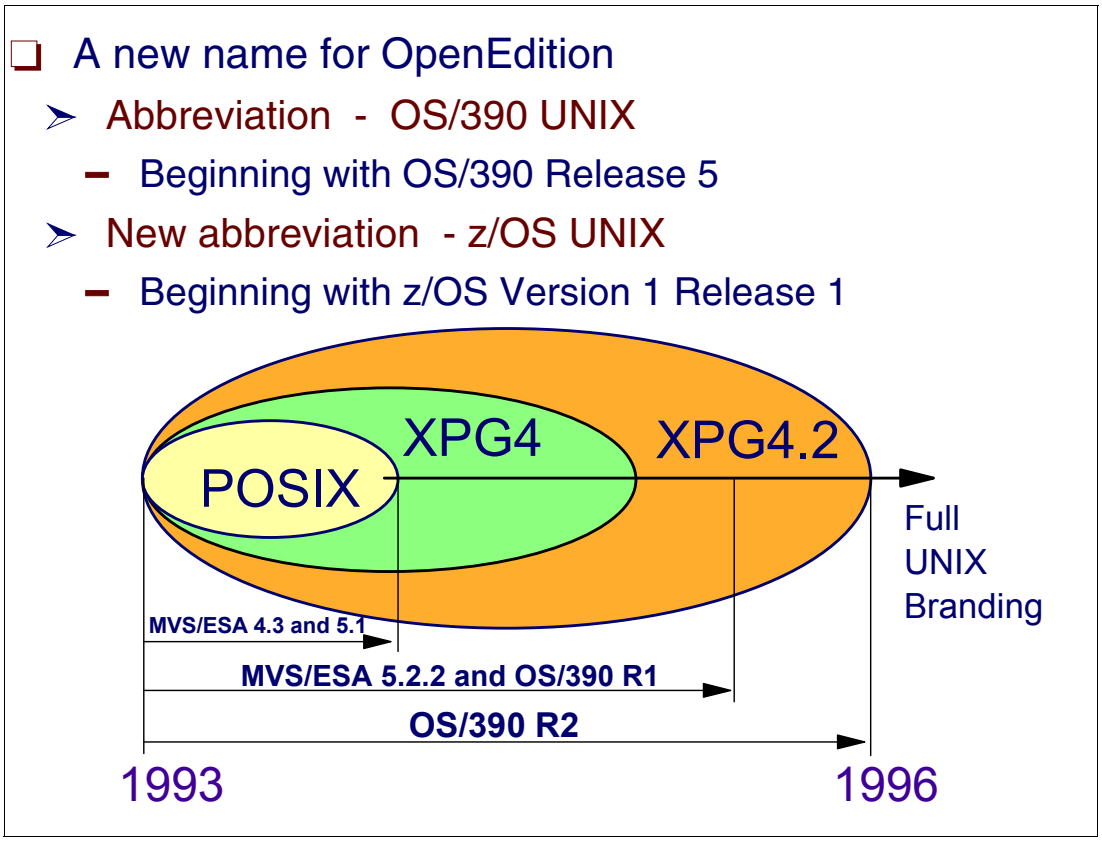


Figure 1-1 Implementation of z/OS UNIX System Services

z/OS UNIX System Services

The name OpenEdition was changed to OS/390 UNIX System Services beginning with OS/390 Release 5. UNIX Services can then be abbreviated OS/390 UNIX.

When OS/390 was renamed to z/OS, the new abbreviation for z/OS UNIX System Services became z/OS UNIX.

z/OS UNIX was originally implemented in MVS/ESA 4.3 as OpenEdition and supported the POSIX standards (1003.1, 1003.1a, 1003.1c, and 1003.2) with approximately 300 functions. In MVS/ESA 5.2.2 many additional functions were added to meet the XPG4 requirements.

In MVS/ESA 5.2.2 more than 1100 functions were included in the OpenEdition implementation. This incorporated the full X/Open Portability Guide issue 4 (XPG4) and over 90% of the single UNIX specification as defined in XPG4.2. The remaining functions were added afterwards, and OpenEdition became branded as a UNIX system.

The XPG4.2 support includes all commands and utilities, most of the additional C services defined in the standard and curses, which was included in specification 1170 but not in the XPG4.2 itself. Curses is the UNIX multi-color, multi-language screen control package which comes from the Novell SVID Edition 3 package.

Since OS/390 V2R2, the following items were added: STREAMS, X/Open Transport Interface (XTI), XPG4.2 regular expressions, XPG4.2 context switching, and XPG4.2 behavior specific to sockets.

XPG4 branding

XPG4 branding means that products use a common set of UNIX APIs. X/Open branding is the procedure by which a vendor certifies that its product complies with one or more of X/Open's vendor-independent product standards and OpenEdition in MVS 4.2.2 received base branding. In 1996, OpenEdition in MVS/ESA SP Version 5 Release 2 received a full XPG4.2 branding. Branding allows applications that are developed on one branded flavor of UNIX to run unchanged on other branded UNIX systems.

It is called branding because it allows the right to use the X/Open Trade Mark. OpenEdition in MVS was shown to comply with the specifications, and therefore IBM is entitled to use the X/Open Trade Mark in relation to OpenEdition having X/Open-compliant features. That right continues for as long as z/OS UNIX remains compliant and registered in the Directory of X/Open Branded Products.

POSIX standards

The work on Portability Operating Systems Interface (POSIX) started as an effort to standardize UNIX and was performed by a workgroup under the Institute of Electrical and Electronics Engineers (IEEE). What they defined was an application programming interface which could be applied not only to UNIX systems but to other operating systems like MVS.

POSIX is not a product. It is an evolving family of standards describing a wide spectrum of operating system components ranging from C language and shell interfaces to system administration.

The POSIX standard is sponsored by the International Organization for Standardization (ISO) and is incorporated into X/Open Portability Guides (XPG). Each element of the standard is defined by a 1003.* number.

POSIX defines the *interfaces* and not the solution or implementation. In this way POSIX can be supported by any operating system. Implementation of POSIX can be different in areas such as performance, availability, and recoverability. All POSIX-compliant systems aren't the same, although they all support basically the same interface.

The support for open systems in z/OS is based on the POSIX standard.

1.2 z/OS UNIX System Services and z/OS

- z/OS UNIX System Services is an element of z/OS
  - A UNIX operating environment implemented within the z/OS operating system
- The z/OS support enables two open systems interfaces on the z/OS operating system:
  - An application program interface (API)
  - An interactive shell interface
- z/OS UNIX System Services provides:
  - XPG4 UNIX 1995 conformance
  - Assembler callable services
  - TSO/E commands to manage the file system
  - ISPF shell environment

Figure 1-2 z/OS UNIX System Services implementation in z/OS

z/OS UNIX and z/OS

The z/OS UNIX System Services element of z/OS is a UNIX operating environment, implemented in z/OS. It is also known as z/OS UNIX. The z/OS support enables two open systems interfaces on the z/OS operating system:

- An application program interface (API)

  With the application programming interface, programs can run in any environment, including in batch jobs, in jobs submitted by TSO/E users, and in most other started tasks, or in any other MVS application task environment. The programs can request only MVS services, z/OS UNIX, or both MVS and z/OS UNIX.

  The application interface is composed of C interfaces. Some of the C interfaces are managed within the C Run-Time Library (RTL), and others access kernel interfaces to perform authorized system functions on behalf of an unauthorized caller.

- An interactive shell interface

  z/OS UNIX responds to requests from programs and the shells. z/OS UNIX has two shells, the z/OS shell and the tcsh shell. They are collectively called the z/OS UNIX shells. The interactive shell interface is an execution environment analogous to TSO/E, with a programming language of shell commands analogous to the Restructured eXtended eXecutor (REXX) Language. The shell work consists of:

  - Programs run by shell users
  - Shell commands and scripts run by shell users
  - Shell commands and scripts run as batch jobs

z/OS UNIX System Services

z/OS UNIX System Services as a component of z/OS provides:

- XPG4 UNIX 1995 conformance

  Application programmers create UNIX-compliant application programs by designing, coding, and testing the programs on their workstations using XPG4 UNIX-conforming systems.

- Assembler callable services

  As an interface between the z/OS operating system and the functions specified in the Single UNIX Specification and earlier standards, z/OS UNIX System Services (z/OS UNIX) provides access to assembler callable services (syscalls). The z/OS UNIX callable services have a standard set of syntax and linkage requirements, as well as parameter specification details necessary for successful invocation.

- TSO/E commands to manage the file system

  You can use TSO/E commands to do certain tasks with the file system. Some of these are tasks that UNIX users traditionally perform while in the UNIX shell.

- ISPF shell environment

  The ISPF shell is a panel interface that you can use instead of TSO/E commands or shell commands to perform certain tasks. For example, you can use the ISPF shell to display all mounted file systems or their attributes, such as total blocks.

1.3 z/OS and z/OS UNIX

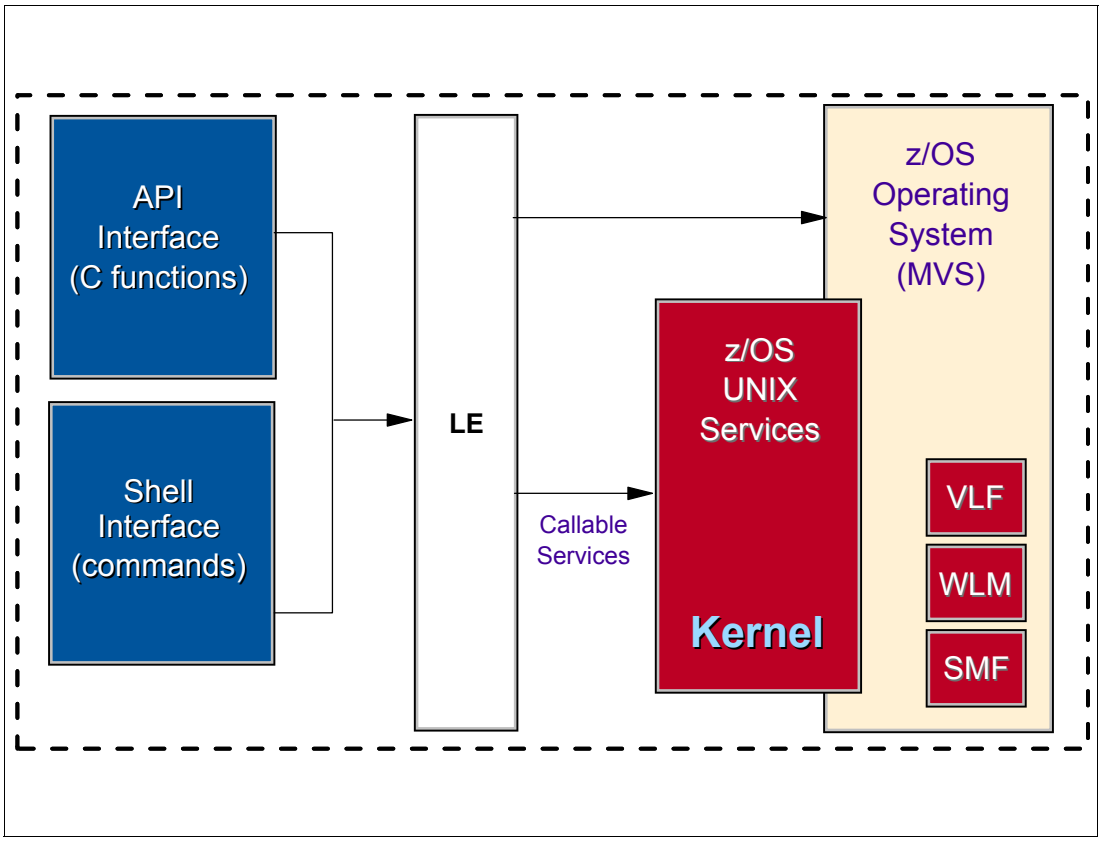


Figure 1-3 z/OS UNIX and the z/OS operating system

z/OS UNIX services

The z/OS support for z/OS UNIX System Services (z/OS UNIX) enables two open system interfaces on z/OS:

- An application program interface (API)
- An interactive z/OS shell interface

API interface

With the APIs, programs can run in any environment—in batch jobs, in jobs submitted by TSO/E users, and in most other started tasks—or in any other MVS application task environment. The programs can request:

- Only MVS services
- Only z/OS UNIX
- Both MVS and z/OS UNIX

Shell interface

The shell interface is an execution environment analogous to TSO/E, with a programming language of shell commands analogous to the Restructured eXtended eXecutor (REXX) language. The shell work consists of:

- Programs run by shell users
- Shell commands and scripts run by shell users
- Shell commands and scripts run as batch jobs

To run a shell command or utility, or any user-provided application program written in C or C++, you need the C/C++ run-time library provided with the Language Environment (LE).

With the z/OS UNIX System Services Application Services, users can:

- Request services from the system through shell commands. Shell commands are like TSO/E commands.
- Write shell scripts to run tasks. Shell scripts are analogous to REXX EXECs.
- Run programs interactively (in the foreground) or in the background.

Application programmers are likely to do the following when creating UNIX-compliant application programs:

- Design, code, and test programs on their workstations using XPG4 UNIX-conforming systems.
- Send the source modules from the workstation to z/OS.
- Copy the source modules from the MVS data sets to HFS files.
- Compile the source modules and link-edit them into executable programs.
- Test the application programs.
- Use the application programs.

A z/OS UNIX program can be run interactively from a shell in the foreground or background, run as a z/OS batch job, or called from another program. The following types of applications exist in a z/OS system with z/OS UNIX:

- Strictly conforming XPG4 UNIX-conforming applications
- Applications using only kernel services
- Applications using both kernel and z/OS services
- Applications using only z/OS services

A z/OS program submitted through the job stream or as a job from a TSO/E session can request kernel services through the following:

- C/C++ functions
- Shell commands, after invoking the shell
- Callable services

Products and components

Following are the z/OS components used by z/OS UNIX:

WLM The workload manager (WLM) transaction initiators provide address spaces when programs issue the fork(), spawn(), C function, or z/OS callable services.

VLF RACF allows caching of UID and GID information in the Virtual Lookaside Facility (VLF). Add the following VLF options to the COFVLFxx member of SYS1.PARMLIB to enable caching since it will improve the performance of z/OS UNIX. VLF is still valid if you have not converted to AIM stage 3.

```
 CLASS NAME(IRRUMAP)
    EMAJ(UMAP)
CLASS NAME(IRRGMAP)
    EMAJ(GMAP)
```

SMF System management facilities (SMF) collects data for accounting. SMF job and job-step accounting records identify processes by user, process, group, and session identifiers. Fields in these records also provide information about resources used by the process. SMF File System records describe file system events such as file open, file close, file system mount, unmount, quiesce, and unquiesce.

1.4 Product and component support for z/OS UNIX

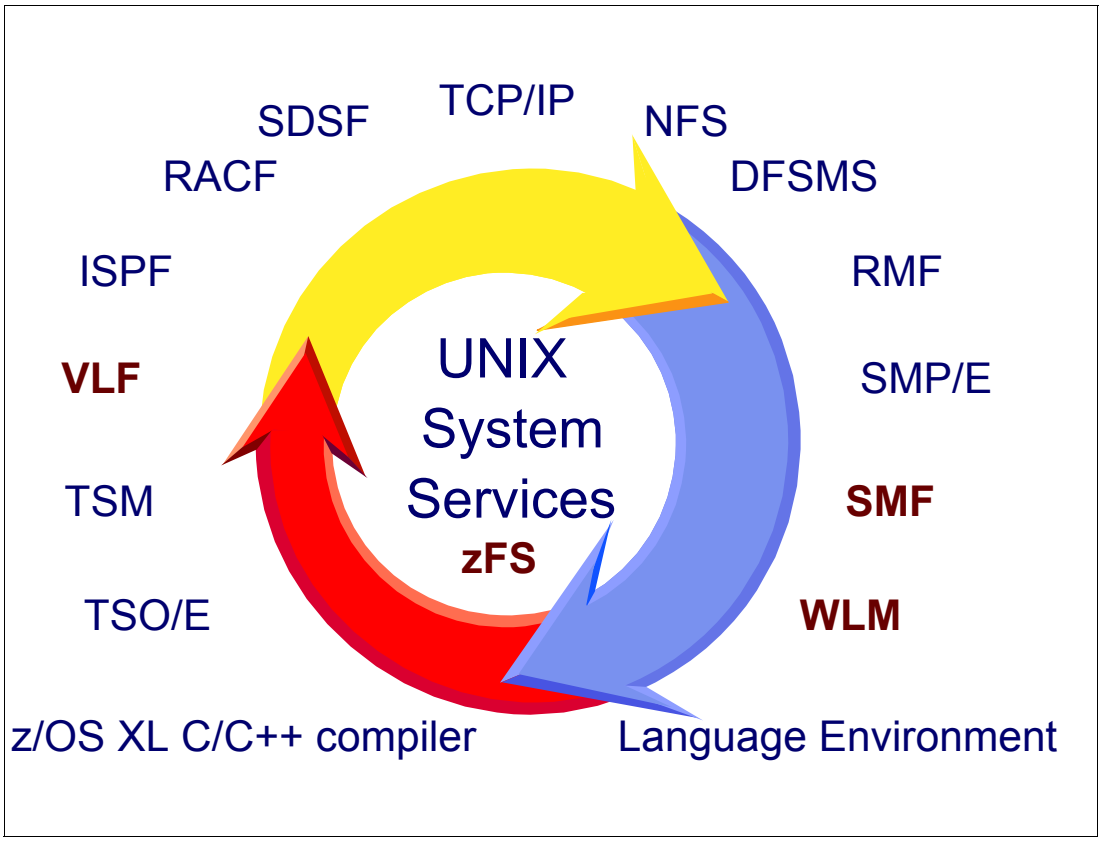


Figure 1-4 Products and components used by z/OS UNIX

z/OS UNIX and z/OS facilities

The z/OS UNIX facility allows users to write and invoke shell scripts and utilities, and to use the shell programming language. z/OS UNIX also interacts with the following elements and features of z/OS to interact directly with the operating system:

- BCP (WLM, VLF, and SMF components)
- z/OS XL C/C++ compiler, to compile programs

  In V1R6 and earlier, the compiler was known as the z/OS C/C++ compiler.

  C is a programming language designed for a wide variety of programming purposes including system-level code. To compile C code using the c89 command, or to compile C/C++ code using cxx, you need the z/OS XL C/C++ compiler that is available with z/OS.
- Language Environment, to execute the shell and utilities or any other XPG4-compliant shell application

  Language Environment establishes a common language development and execution environment for application programmers on z/OS. To run a shell command or utility, or any user-provided application program written in C or C++, you need the C/C++ run-time library provided with Language Environment.
- Data Facility Storage Management Subsystem (DFSMS) (HFS is a component of DFSMS.)
- Security Server for z/OS (RACF is a component of the Security Server.)

- Resource Measurement Facility (RMF)
- System Display and Search Facility (SDSF)

  System Display and Search Facility (SDSF) is an IBM-licensed program that provides a menu-driven full-screen interface that is used to obtain detailed information about jobs and resources in a system. Shell users can enter TSO/E sessions and use SDSF to monitor z/OS activities. For example, they can:

  - Monitor printing
  - Monitor and control a batch job
  - Monitor and control forked address spaces
  - Find out which users are logged on to TSO/E sessions
- Time Sharing Option Extensions (TSO/E)

  You can also enter the shell environment by logging on to a TSO/E session and entering the OMVS command. Other TSO/E commands logically mount and unmount file systems, create directories in a file system, and copy files to and from MVS data sets. Users can switch from the shell to their TSO/E session, enter commands or do editing, and switch back to the shell.
- Communications Server (TCP/IP)

  TCP/IP enables users to enter the shell environment by using **`rlogin`** or **`telnet`** from a workstation in the TCP/IP network. User-written socket applications can use TCP/IP services as a communication vehicle. Both client and server socket applications can use the socket interface to communicate over the Internet (AF_INET and AF_INET6) and between other socket applications by using local sockets (AF_UNIX). An assembler interface is also provided for those applications that do not use the C/C++ run-time library.
- ISPF, to use the dialogs for OEDIT, or ISPF/PDF for the ISPF shell

  Users of ISPF can use the ISPF shell environment to create, edit, browse, and perform other functions for files and directories in the z/OS UNIX file system.
- Network File System (NFS)

  Network File System (NFS) enables users to access files on other systems in a network.
- z/OS Distributed File Service zSeries File System (zFS)

  zSeries File System (zFS) is a UNIX file system that can be used, along with HFS.

1.5 Security Server RACF

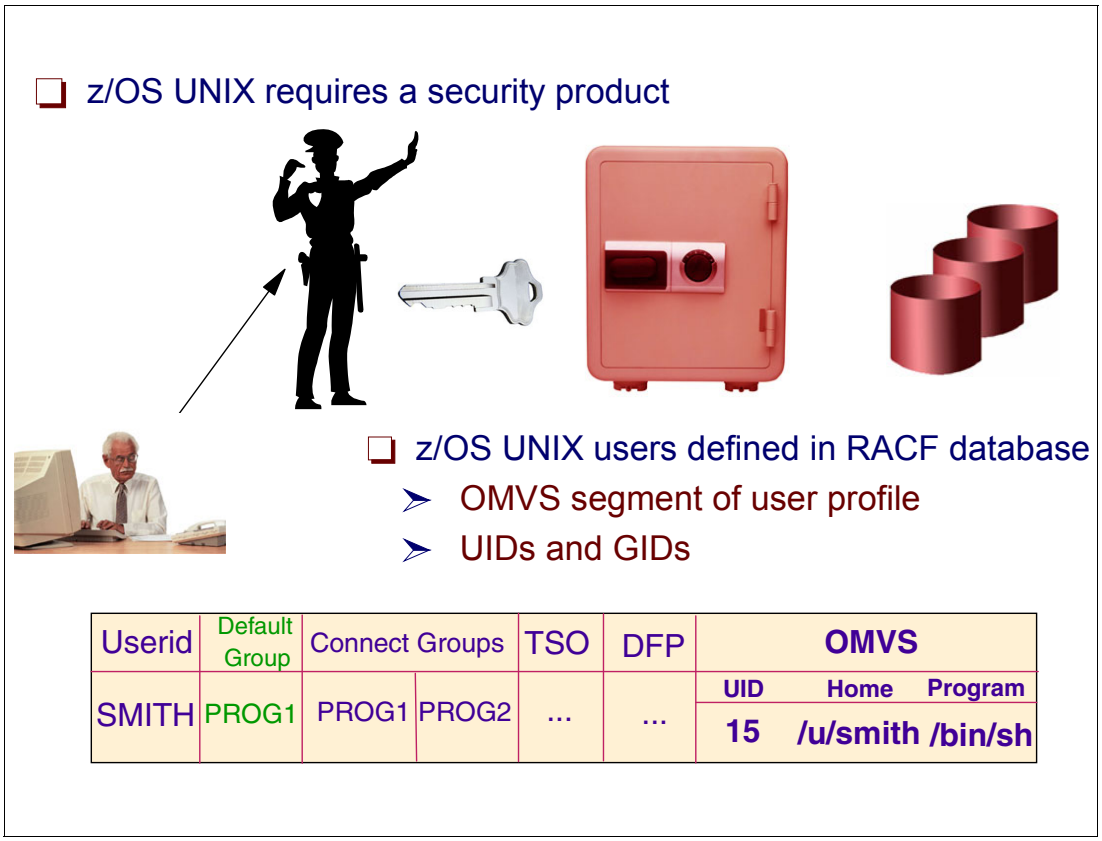


Figure 1-5 Security Server RACF

Security Server RACF

RACF or an equivalent security product like ACF2 can manage system and data set security by verifying a user and checking that the user can access a resource.

The security products also take care of who is allowed to issue special commands, define new users, and change access permissions.

The permission to access directories and files is verified by the security product. RACF is responsible for the OMVS segment which allows the user to access UNIX Services.

In other words, if someone wants to access z/OS UNIX file systems, whether a directory or a file, RACF checks whether this user is allowed to have access in the z/OS UNIX environment. The security checking to access specific directories or files is then handled by z/OS UNIX System Services itself.

RACF is not able to change permission bits or owner statements in the z/OS UNIX environment. RACF only checks whether a user or program is allowed to access z/OS UNIX services.

The RACF user profile definition was expanded with a segment called OMVS for z/OS UNIX support. All users and programs that need access to z/OS UNIX must have a RACF user profile defined with an OMVS segment which has, as a minimum, a UID specified. A user without a UID cannot access z/OS UNIX.

1.6 Data Facility System-Managed Storage (DFSMS)

- File system data sets can reside on SMS-managed volumes
  - SMS-managed is optional
  - Although the file system does not have to be SMS-managed, it is still highly suggested
  - Multivolume file systems are only supported as SMS-managed

Figure 1-6 DFSMS and z/OS UNIX

DFSMS and z/OS UNIX

During the first quarter of 2000, PTFs removed the requirement that HFS data sets reside on SMS-managed volumes. HFS data sets still need to be cataloged in the master or user catalog in order for the HFS data sets to be mounted by z/OS UNIX. However, you do not need to catalog the HFS data sets if you plan on dumping them using DFSMSdss.

Note: Although the file system does not have to be SMS-managed, this is still highly suggested. Multivolume file systems are only supported as SMS-managed (that is, you cannot have multivolume non-SMS-managed data sets). As a user adds files and extends existing files, the data set increases in size to a maximum of 123 extents if secondary extents are specified in the allocation.

Data Facility System-Managed Storage (DFSMS) manages the z/OS UNIX data sets used for processing. These hierarchical file system (HFS) data sets make up a file hierarchy.

DFSMS manages the hierarchical file system data sets that contain the file systems. To use kernel services in full-function mode, SMS must be active.

1.7 Transmission Control Protocol/Internet Protocol (TCP/IP)

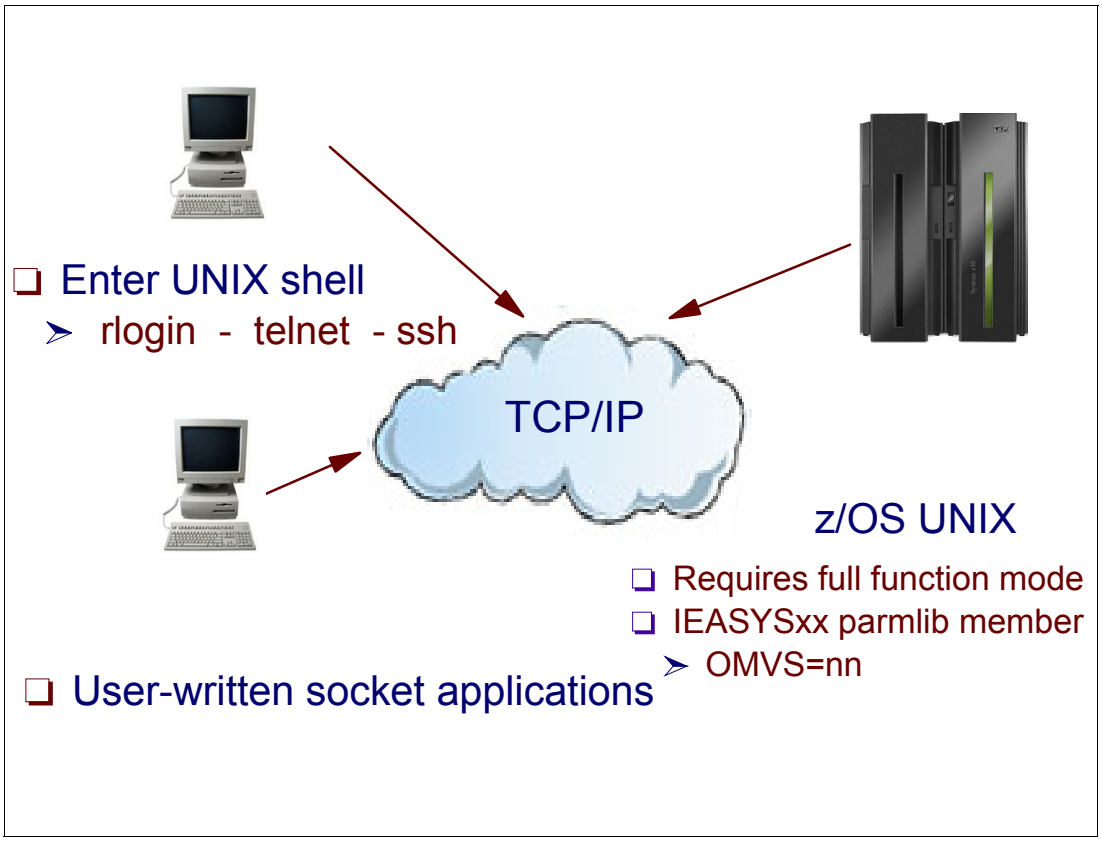


Figure 1-7 TCP/IP and z/OS UNIX

TCP/IP and z/OS UNIX

Transmission Control Protocol/Internet Protocol is a peer-to-peer network protocol. It is officially named the TCP/IP Internet Protocol Suite and is referred to as TCP/IP (after the name of its two main standard protocols). TCP/IP is a set of industry standard protocols and applications.

The TCP/IP Internet Protocol Suite is a layered set of protocols that allow cooperating computers to share resources across a network or networks. The physical networks can be of different types.

UNIX shell users

One way to enter the shell environment is by using **`rlogin`** or **`telnet`** from a workstation in the TCP/IP network. **`ssh`** is a client program for logging into a z/OS shell. It is a more secure alternative to **`rlogin`**.

User-written socket applications

User-written socket applications can use TCP/IP as a communication vehicle. Both client and server socket applications can use the socket interface to communicate over the Internet (AF_INET) and between other socket applications by using local sockets (AF_UNIX). An assembler interface is also provided for those applications that do not use the C/C++ run-time library.

Running in full function mode

The OMVS= parameter in the IEASYSxx parmlib member specifies the BPXPRMxx member or members that determine the configuration of the kernel service. Minimum mode provides the minimum requirements for kernel services; you will not be able to use the shell or TCP/IP in this mode. If you do not specify OMVS= in the IEASYSxx parmlib member, or if you specify OMVS=DEFAULT, then kernel services start up in a minimum configuration mode with all BPXPRMxx parmlib options taking their default values when the system is IPLed. This mode is for installations that do not plan to use the kernel services.

1.8 System Modification Program Extended (SMP/E)

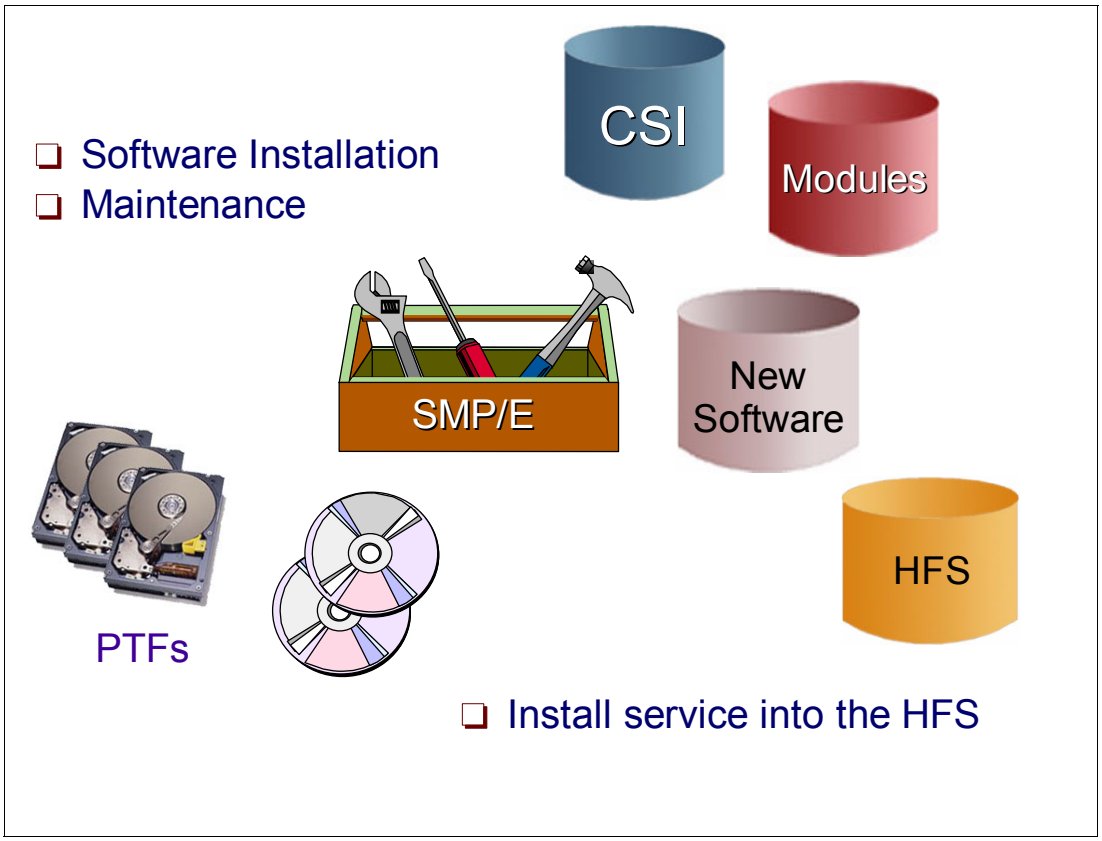


Figure 1-8 SMP/E and z/OS UNIX

SMP/E and z/OS UNIX

SMP/E is a basic tool for installing and maintaining software in MVS systems and subsystems. It controls these changes by:

- Selecting the proper levels of elements to be installed from a large number of potential changes.
- Calling system utility programs to install the changes.
- Keeping records of the installed changes (in the CSI).

SMP/E is an integral part of the installation, service, and maintenance processes for CBIPOs, CBPDOs, ProductPacs, ServicePacs, and selective follow-on service for CustomPacs. In addition, SMP/E can be used to install and service any software that is packed in SMP/E system modification (SYSMOD) format.

SMP/E can be run either using batch jobs or using dialogs under Interactive System Productivity Facility/Program Development Facility (ISPF/PDF).

Two types of dialogs are provided by SMP/E:

- CBIPO dialogs for installing and redistributing Custom-Built Installation Process Offering (CBIPO) packages.
- SMP/E dialogs help you interactively query the SMP/E database, as well as create and submit jobs to process SMP/E commands.

1.9 System Management Facility (SMF)

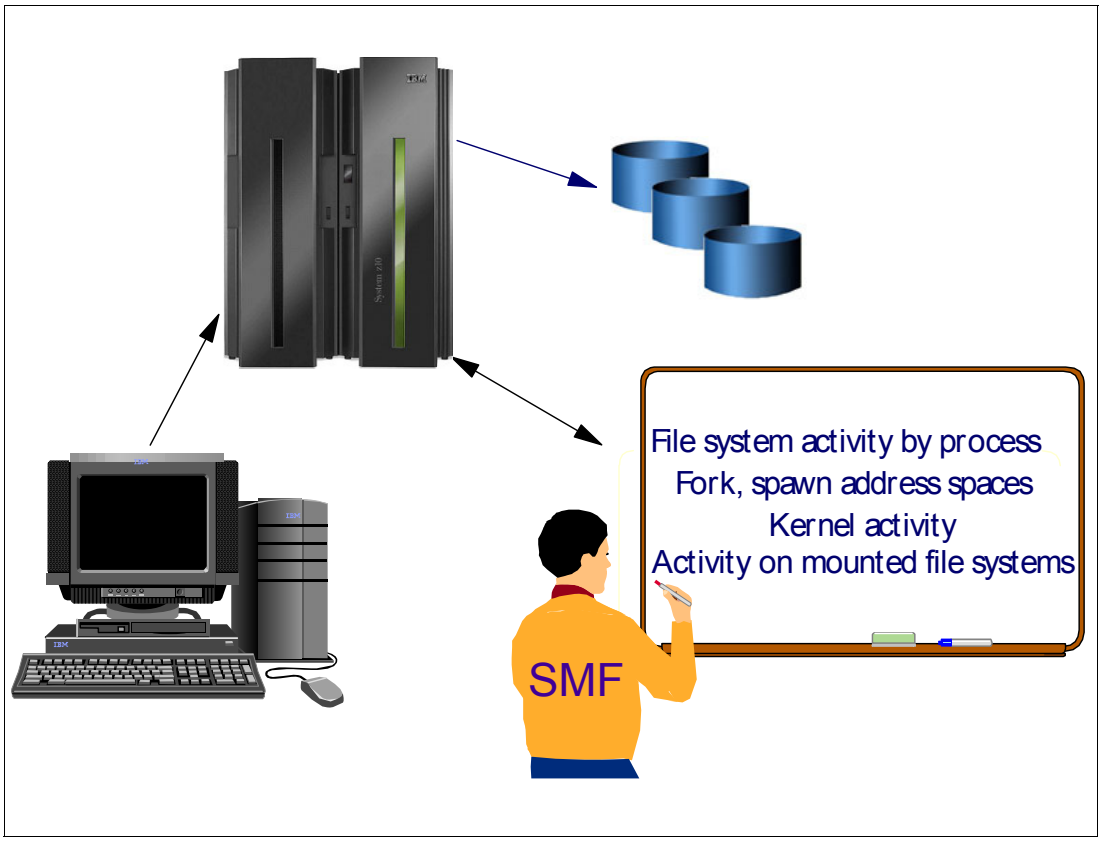


Figure 1-9 SMF and z/OS UNIX

SMF recording

System Management Facility (SMF), which is a component of the BCP element, collects data for accounting. SMF job and job-step accounting records identify processes by user, process, group, and session identifiers. Fields in these records also provide information on resources used by the process. SMF file system records describe file system events such as file open, file close, and file system mount, unmount, quiesce, and unquiesce.

You can use SMF to report on activity from a user application, to report activity on a job and jobstep basis, and to report activity of mounted file systems and files.

SMF job and job-step accounting records identify z/OS UNIX processes by:

- User (UID)
- Process (z/OS address space)
- Group (GID)
- Session identifiers

1.10 Resource Measurement Facility (RMF)

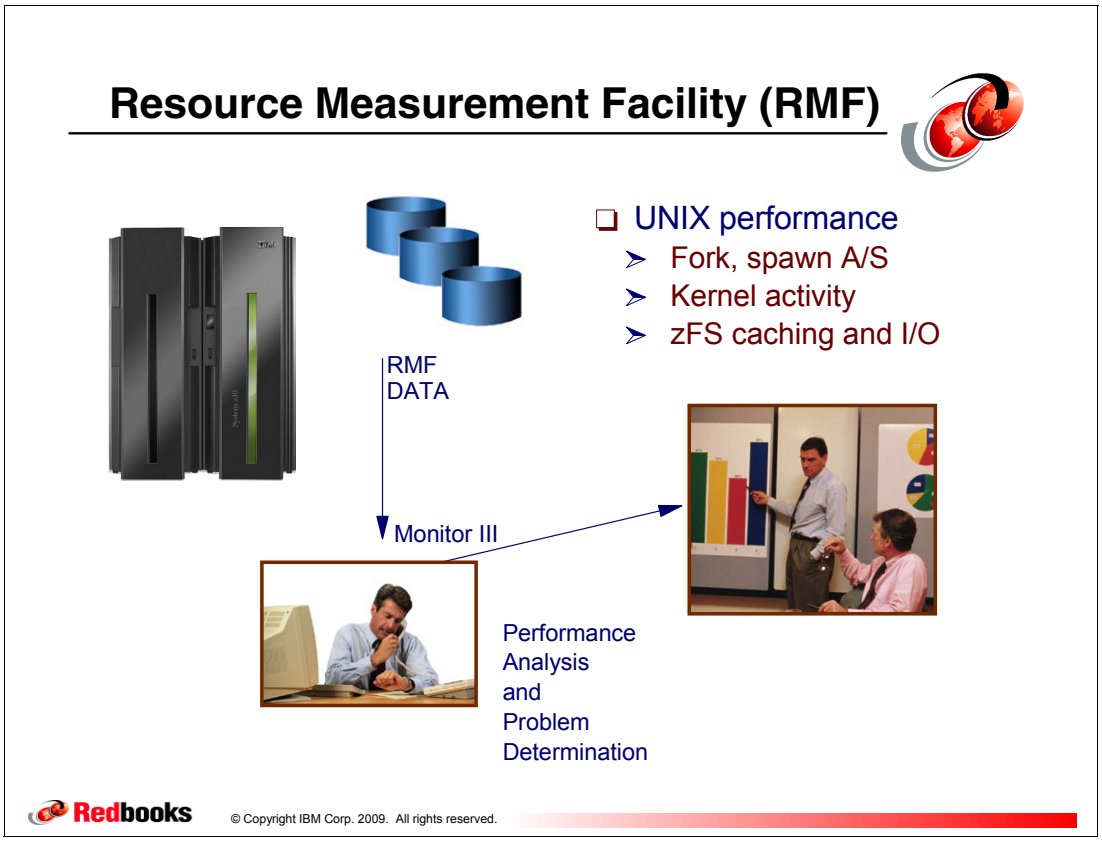


Figure 1-10 RMF and z/OS UNIX

RMF and z/OS UNIX

Resource Measurement Facility (RMF) collects data used to describe z/OS UNIX performance. RMF reports support an address space type of OMVS for address spaces created by fork or spawn callable services and support two swap reason codes.

RMF monitors the use of resources in an OMVS Kernel Activity report.

The software products supporting system programmers and operators in managing their systems heavily influence the complexity of their jobs, and their ability to keep systems at a high level of availability.

RMF collects data used to describe z/OS UNIX performance. RMF reports support an address space type of OMVS for address spaces created by fork or spawn callable services.

When an installation specifies an OMVS subsystem type in the workload manager service policy, RMF shows the activity of forked address spaces separately in the RMF Workload Activity report.

1.11 Virtual Lookaside Facility (VLF)

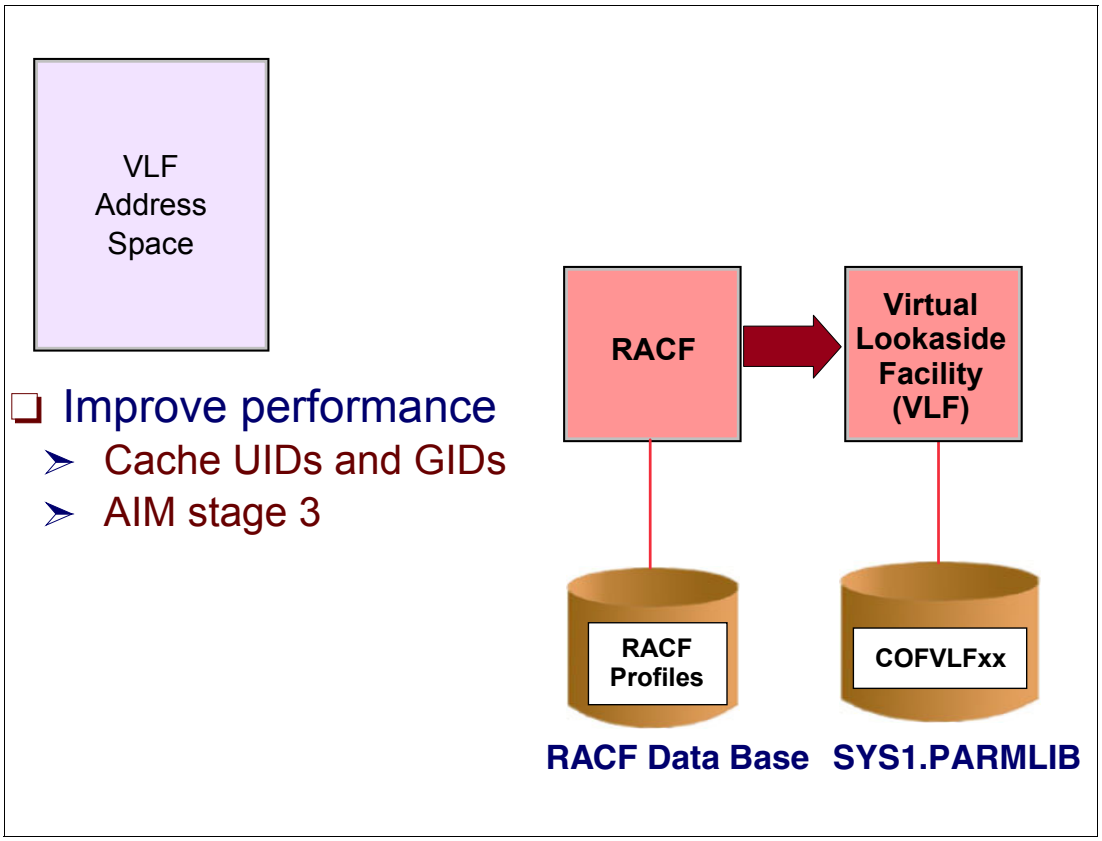


Figure 1-11 VLF and z/OS UNIX

VLF and z/OS UNIX

You can assign a z/OS UNIX user identifier (UID) to a RACF user by specifying a UID value in the OMVS segment of the RACF user profile. When assigning a UID to a user, make sure that the user is connected to at least one group that has an assigned GID. This group should be either the user's default group or one that the user specifies during logon or on the batch job. A user with a UID and a current connect group with a GID can use z/OS UNIX functions and access z/OS UNIX files based on the assigned UID and GID values. If a UID and a GID are not available as described, the user cannot use z/OS UNIX functions.

VLF

The Virtual Lookaside Facility is a set of easy-to-use high-performance services that provide an alternate fast path method for making frequently used named data objects available to many users. It will reduce I/O operations for frequently accessed programs that are provided now in central or expanded storage.

Whether you are developing a new application or modifying an existing one, the kind of application that can use VLF effectively is an application that frequently retrieves a repetitive set of data from DASD on behalf of many end users.

COFVLFxx PARMLIB member

Caching UIDs and GIDs improves performance for commands such as `ls -l`, which must convert UID numbers to user IDs and GID numbers to RACF group names. RACF allows you

to cache UID and GID information in VLF. Add the following VLF options to the COFVLFxx member of SYS1.PARMLIB to enable the caching:

```
CLASS NAME(IRRUMAP)
  EMAJ(UMAP)
CLASS NAME(IRRGMAP)
  EMAJ(GMAP)
CLASS NAME(IRRGTS)
  EMAJ(GTS)
CLASS NAME(IRRACEE)
  EMAJ(ACEE)
CLASS NAME(IRRSMAP)
  EMAJ(SMAP)
```

For details about the VLF classes, see *z/OS Security Server RACF System Programmer's Guide,* SA22-7681.

AIM stage 3

In stage 3, RACF locates application identities, such as UIDs and GIDs, for users and groups by using an alias index that is automatically maintained by RACF. This allows RACF to more efficiently handle authentication and authorization requests from applications such as z/OS UNIX than was possible using other methods, such as the UNIXMAP class and VLF. Once your installation reaches stage 3 of application identity mapping (AIM), you will no longer have UNIXMAP class profiles on your system, and you can deactivate the UNIXMAP class and remove VLF classes IRRUMAP and IRRGMAP.

Important: Associating RACF user IDs and groups to UIDs and GIDs has important performance considerations. If your installation shares the RACF database with systems running releases prior to OS/390 Version 2 Release 10, it is important to use the VLF classes IRRUMAP and IRRGMAP and the UNIXMAP class to improve performance by avoiding sequential searches of the RACF database for UID and GID associations.

If your installation shares the RACF database with only systems running z/OS, or OS/390 Version 2 Release 10 or above, you may be able to achieve improved performance without using UNIXMAP and VLF. However, before you can avoid using UNIXMAP and VLF, you need to implement stage 3 of application identity mapping by running the IRRIRA00 conversion utility.

1.12 Time Sharing Option/Extended (TSO/E)

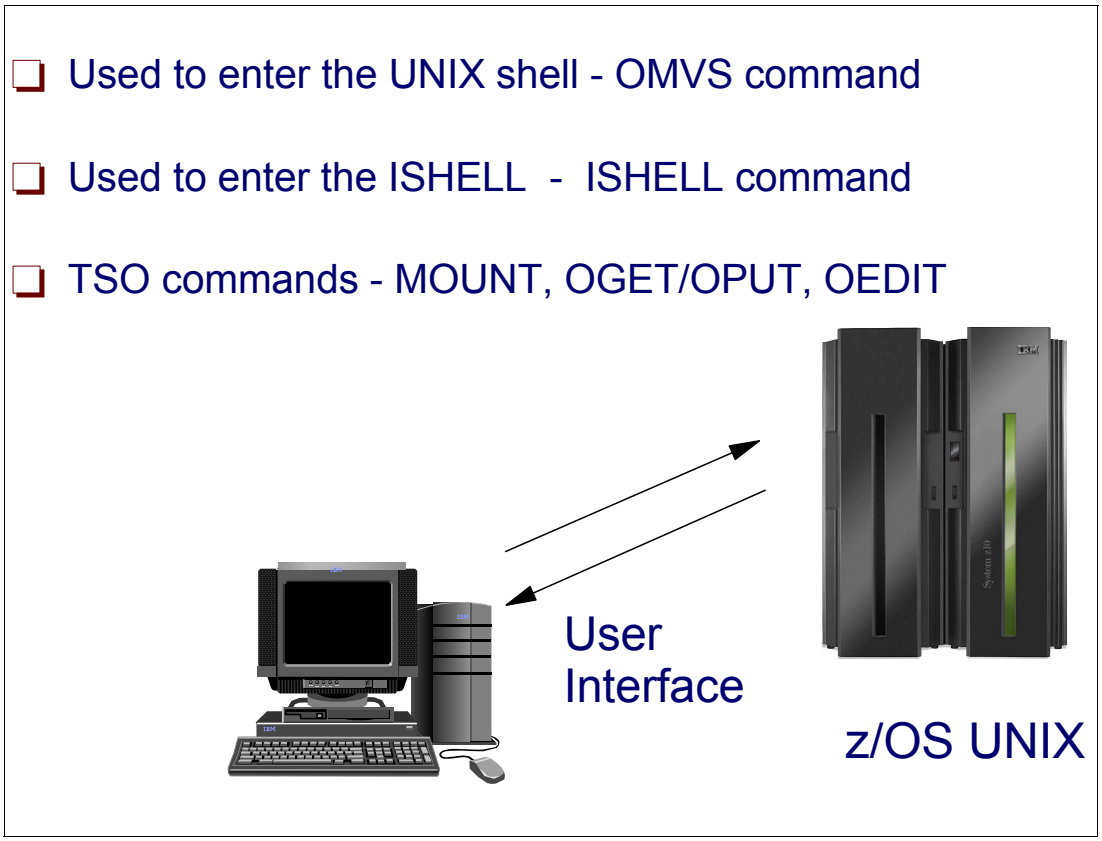


Figure 1-12 TSO/E and z/OS UNIX

TSO/E and z/OS UNIX

The TSO Extensions (TSO/E) licensed program is based on the Time Sharing Option (TSO), which allows users to interactively share computer time and resources.

TSO/E is composed of modules that communicate with the user and perform the work requested by the user. When a user logs on to TSO/E, the user must either specify the name of a LOGON procedure by the LOGON command or accept that user's default procedure name from the user attribute data set.

One way to enter the UNIX shell environment is by using TSO/E. A user logs on to a TSO/E session and enters the TSO/E OMVS command.

The z/OS environment has other TSO/E commands, for example, to logically mount and unmount file systems, create directories in a file system, and copy files to and from MVS data sets. Users can switch from the shell to their TSO/E session, enter commands, or do editing, and switch back to the shell. For information about how to perform these tasks using TSO/E commands, see *z/OS UNIX System Services User's Guide,* SA22- 7801.

1.13 Workload Manager (WLM)

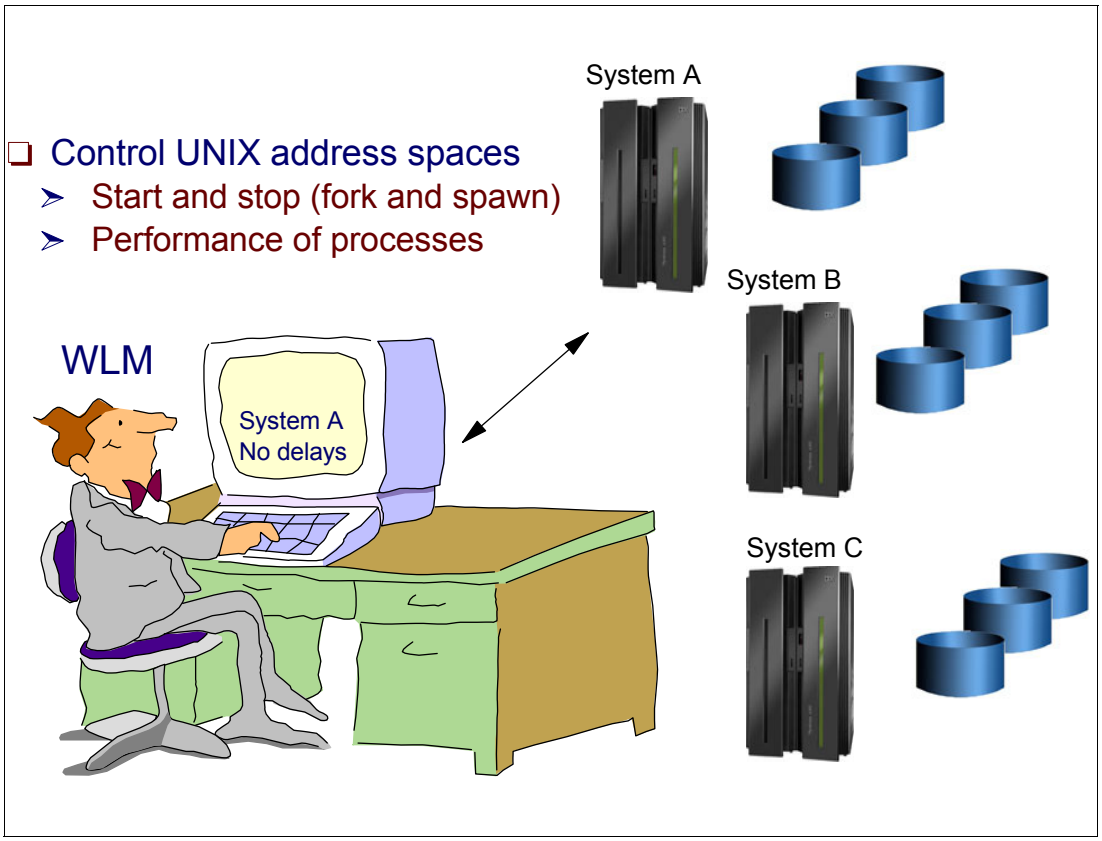


Figure 1-13 WLM and z/OS UNIX

WLM and z/OS UNIX

The purpose of workload management is to balance the available system resources to meet the demands of subsystem work managers such as CICS®, BATCH, TSO, UNIX Services, and WebServer, in response to incoming work requests.

The workload manager is a component of the BCP element. When using WLM, you do not need to do any tuning or issue any commands. The kernel uses WLM to create child processes when running in goal mode or compatibility mode, or both.

The workload manager is a component of z/OS that provides the ability to run multiple workloads at the same time in one z/OS image or across multiple images. When using WLM, you do not need to do any tuning or issue any commands. The kernel uses WLM to create child processes while running in goal mode.

When programs issue fork() or spawn(), the BPXAS PROC found in SYS1.PROCLIB is used to provide a new address space. For a fork(), the system copies one process, called the parent process, into a new process, called the child process. The forked address space is provided by WLM.

1.14 Tivoli Storage Manager (TSM)

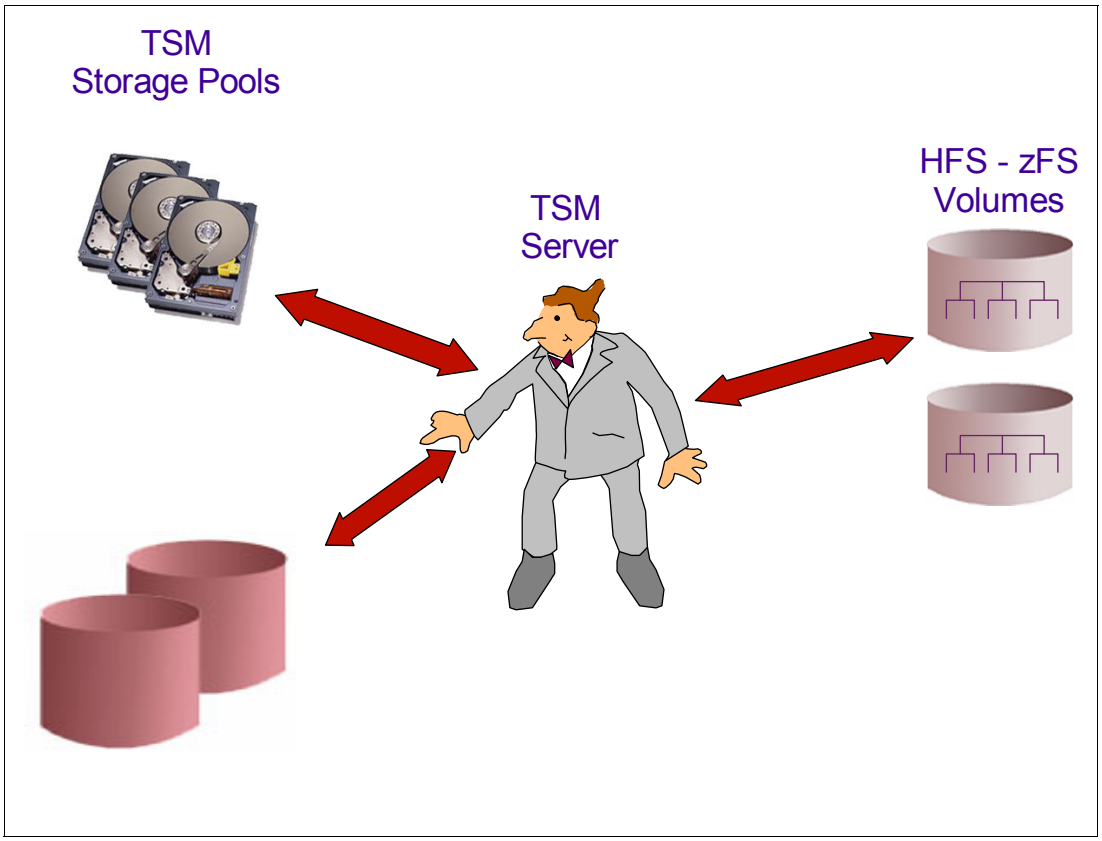


Figure 1-14 Tivoli and z/OS UNIX

TSM and z/OS UNIX

Tivoli Storage Manager (TSM) is a client/server storage management product that provides administrator-controlled, highly automated, centrally scheduled, network-based backup and archive functions for workstations and LAN file servers. A TSM server backs up and/or archives data from a TSM client and stores the data in the TSM server storage pool for z/OS UNIX clients.

There are two types of backup: incremental, in which all new or changed files are backed up; and selective, in which the user backs up specific files.

Backup can be performed automatically or when the user requests it. The user can initiate a specific type of backup or start the scheduler, which will run whatever action the TSM administrator has scheduled for the user's machine.

As a TSM authorized user, you also have the authority to back up and archive all eligible files in all locally mounted file systems on your workstation, restore and retrieve all backup and archive files for your workstation from TSM storage, within the limits imposed by the UNIX file access permissions. A TSM authorized user can also grant users access to specific files in TSM storage.

Information about using the z/OS UNIX client is documented in:

- *TSM Using the Backup-Archive Clients*, SH26-4105
- *TSM Installing the Clients*, SH26-4102

Data Facility System-Managed Storage Hierarchical Storage Manager (DFSMShsm) provides automatic backup facilities for HFS data sets. The system programmer uses DFSMShsm facilities to back up mountable file systems by backing up the HFS data sets that contain them on a regular basis; the data sets can be restored when necessary. DFSMShsm is also used for migrating (archiving) and restoring unmounted file systems.

2

z/OS UNIX overview

This chapter provides a brief overview of the most important components of z/OS UNIX System Services (z/OS UNIX). The following topics are discussed:

- z/OS UNIX terminology
- z/OS UNIX physical file systems
- Hierarchical file system structure
- File system data sets
- The z/OS UNIX components
- z/OS UNIX processes
- z/OS UNIX interfaces
- Methods for direct login to the z/OS UNIX shell

© Copyright IBM Corp. 2011. All rights reserved.

2.1 z/OS UNIX and UNIX applications

- IBM implemented UNIX based on SPEC1170 specifications
- Reasons for z/OS UNIX
  - Run all types of applications on z/OS and zSeries
  - Portability of programmer skills
  - Portability of source code and applications
- Accessing databases
  - DB2 can be called directly from a UNIX application
- Importance on UNIX to MVS programmers

Figure 2-1 UNIX applications using z/OS UNIX

z/OS UNIX and SPEC1170

IBM added UNIX to the MVS environment to compete for new work on S/390®. In addition to the characteristics of UNIX described earlier, portability is important due to the openness of UNIX. The compliance of systems to SPEC1170, which is the Open Group's single UNIX standard. IBM looked at the SPEC1170 specifications that say “these are what the UNIX interfaces are” and implemented those specifications directly into its operating system as OpenEdition services, which is now called z/OS UNIX. The OpenEdition MVS Shell and Utilities, with many of the Korn shell features, were first to take advantage of the new z/OS UNIX System Services now in z/OS.

Reasons for z/OS UNIX

For installations, portability of skills from one UNIX to another is important. If you work on a UNIX system, you can port most of your skills onto another UNIX platform.

For applications, portability of source code from one UNIX platform to another is important. If applications consisted of only interfaces that meet the X/Open SPEC1170 specifications (also known as X/Open XPG4.2, or UNIX95, or Single UNIX Specification), a port would require no changes to the applications. But, in the real world, where applications contain some non-standard UNIX code, the applications do require some changes.

Accessing UNIX from database applications

As an example of database applications using z/OS UNIX, when using the DB2® ODBC product for data set access, if you build a DB2 ODBC application in z/OS UNIX, you can use

the c89 **`compile`** command to compile your application. Although you compile your application under z/OS UNIX, you can directly reference the non-HFS DB2 ODBC data sets in the c89 command. There is no need to copy the DB2 ODBC product files to HFS.

Importance of UNIX to MVS programmers

MVS-experienced system programmers now installing, maintaining, and debugging z/OS UNIX are faced with new responsibilities that require learning new vocabulary and new concepts. Perhaps they must now interact with UNIX or TCP/IP computer personnel from whom they have been isolated before now. Differences between the familiar MVS world and the new UNIX world sometimes seem cultural and philosophical, as well as technical.

2.2 z/OS UNIX terminology overview

- An address space is a process in UNIX
- A task (TCB) is called a thread
- A started task (STC) is a daemon
- z/OS has TSO/E and ISPF and UNIX has a shell
- Applications are written in different languages and jobs are submitted for processing in different ways
- HFS - Hierarchical file system
  - Structure of the file system - directories and files
- PFS - Physical file system
- z/OS UNIX kernel
- Shared file system in a sysplex

Figure 2-2 z/OS UNIX terminology

z/OS UNIX process

With z/OS, processes can be created by fork or spawn functions. Existing MVS address space types such as TSO, STC, Batch, and APPC can request z/OS UNIX services. When one of those address spaces makes its first request to the z/OS kernel, the kernel dubs the task; that is, it identifies the task as a z/OS UNIX process.

z/OS UNIX thread

A process can have one or more threads; a thread is a single flow of control within a process. Application programmers create multiple threads to structure an application in independent sections that can run in parallel for more efficient use of system resources.

z/OS UNIX daemon

Daemon processes perform continuous or periodic system wide functions, such as a Web server. Daemons are programs that are typically started when the operating system is initialized and remain active to perform standard services.

z/OS UNIX shell

The UNIX shell interface is an execution environment analogous to TSO/E, with a programming language of shell commands analogous to the Restructured eXtended eXecutor (REXX) Language. The shell work consists of:

- Programs run by shell users
- Shell commands and scripts run by shell users

- Shell commands and scripts run as batch jobs

z/OS UNIX has two shells, the z/OS shell and the tcsh shell. They are collectively called the z/OS UNIX shells.

z/OS UNIX applications

The following types of applications exist in z/OS UNIX:

- Strictly conforming XPG4-conforming applications
- Applications using only kernel services
- Applications using both kernel and MVS services
- Applications using only MVS services

z/OS UNIX file systems

z/OS UNIX files are organized in a hierarchical file system, as in other UNIX systems. Files are members of a directory, and each directory is in turn a member of another directory at a higher level. The highest level of the hierarchy is the root directory. Each instance of the system contains only one root directory.

z/OS UNIX physical file systems

A physical file system (PFS) is the part of the operating system that handles the actual storage and manipulation of data on a storage medium. In z/OS UNIX, for creating and accessing HFS files, there are two PFSes, zFS and HFS.

z/OS UNIX kernel

The kernel address space contains the MVS support for z/OS UNIX services. This address space can also be called the *kernel* and is the part of z/OS UNIX that contains programs for such tasks as I/O, management, control of hardware, and the scheduling of user tasks.

Shared file system in a sysplex

By establishing the shared file system environment, sysplex users can access data throughout the file hierarchy from any system in the sysplex. With shared file system support, all file systems that are mounted by a system participating in a shared file system are available to all participating systems. In other words, once a file system is mounted by a participating system, that file system is accessible by any other participating system.

Although it is suggested that you exploit shared file system support when running in a sysplex environment, you are not required to do so. If you choose not to, you will continue to share file systems as you have before.

2.3 HFS and zFS file system PFSes

- HFS is a UNIX File System PFS for z/OS
  - Introduced with OpenEdition
- zFS is a UNIX File System PFS for z/OS
  - Component of the Distributed File Service since 1995
  - HFS now stabilized - will be removed in future release
- Using zFS, you can
  - Run applications just like HFS
  - Use zFS in addition to HFS or replace HFS
- Advantages of zFS over HFS:
  - Better performance
  - Enhanced administrative functions
  - Less loss of data on system failures

Figure 2-3 HFS and zFS file system PFSes

HFS and zFS file systems

The zSeries File System (zFS) is a UNIX file system that can be used in addition to the HFS. This file system actually has been available since 1995 as part of the DCE component of MVS/ESA V5R2.2, OS/390, and z/OS V1R1. It is a high-performance, log-based file system. The DCE DFS Local File System is part of the OSF DFS product, and as such is included in the Distributed File Service base element of z/OS V1R2. zFS is a separate component of the DFS base element, so it can be serviced separately from the other components of DFS.

HFS stabilization

In early 2004, IBM announced that Hierarchical File System (HFS) function is stabilized. The zSeries File System (zFS) is the strategic z/OS UNIX System Services file system for z/OS. IBM has enhanced zFS function in z/OS V1R7 so that you can use zFS file systems at all levels within the file hierarchy. HFS may no longer be supported in future releases and you will have to migrate the remaining HFS file systems to zFS.

Using zFS

The zSeries File System (zFS) is a UNIX file system that can be used in addition to the HFS. zFS is the strategic file system for z/OS. Therefore, in z/OS V1R7, zFS file system functions are extended beyond those provided by HFS by improving usability, performance, and making it easier to migrate your data from HFS file systems to zFS file systems.

2.4 Using z/OS UNIX

- UNIX applications and users
  - Create files in the hierarchical file system
    - z/OS UNIX kernel passes access request to a PFS
    - Beginning with OpenEdition, this PFS was called HFS
    - Beginning with z/OS V1R2, zFS became a PFS
  - Physical file systems (PFSs)
    - File system requests can be passed to either HFS or zFS physical file systems
    - Defined in the BPXPRMxx parmlib member

Figure 2-4 Using z/OS UNIX for applications and users

Creating UNIX files

When you are logged into the shell you have a choice of editors to create and change files, depending on which terminal interface you are using, OMVS or the asynchronous terminal interface. Accessing UNIX files can be done from application programs. Also, ISPF Edit provides a full-screen editor to create and edit HFS files. You can access ISPF Edit in several ways:

- Using the **`oedit`** shell command
- Using the TSO/E OEDIT command at the TSO/E READY prompt or from the shell command line
- From the ISPF menu (if a menu option is installed)
- From the ISPF shell (accessed using the TSO/E ISHELL command)

Requests from kernel to PFS

When users access file data, the kernel passes the user request to the correct PFS to do the I/O to the file system.

Physical file systems

A physical file system (PFS) controls access to data. PFSs receive and act upon requests to read and write files that they control. The format of these requests is defined by the PFS interface.

A physical file system (PFS) is packaged as one or more MVS load modules. These load modules must be installed in an APF-authorized MVS load library. The hierarchical file system is not available when a PFS is loaded, so it cannot be installed in the file system.

A PFS is defined to z/OS UNIX through the BPXPRMxx PARMLIB member you specify when you start the kernel address space (OMVS=xx). The FILESYSTYPE statement defines a single instance of a PFS.

HFS and zFS

When IBM created OpenEdition as part of MVS, the PFS that did the I/O to the file system was called HFS.

Beginning with z/OS V1R2, a second PFS, called ZFS, was added. In current z/OS operating systems, both the HFS and ZFS PFSs can be used to access file system data.

User-written programs use the POSIX API to issue file requests. These requests are routed by the logical file system (LFS) to the appropriate PFS through the PFS interface.

2.5 z/OS UNIX System Services

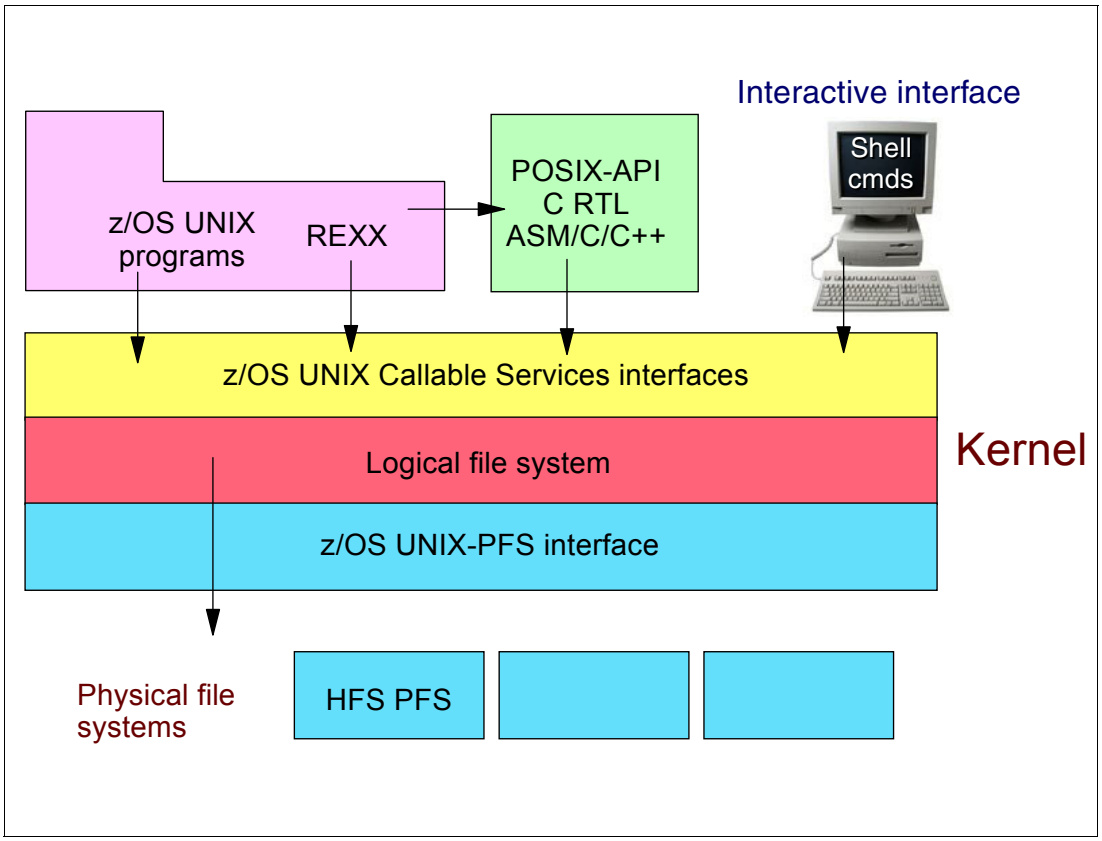


Figure 2-5 z/OS UNIX System Services and the Kernel

POSIX-API

The Application Programming Interface (API) consists of C programming calls that can be used by C/370™ programs to access z/OS UNIX. These C calls are defined in the POSIX 1003.1 standard. Many of the C calls will use callable services to interact with the z/OS system to perform the services requested. Some C calls will interface directly with the z/OS UNIX kernel.

The callable services can be used directly by Assembler programs to access z/OS UNIX, for example to access files in the hierarchical file system. This possibility allows other high-level languages (excluding C) and Assembler to use z/OS UNIX.

The API interface provides the ability to run XPG4.2 programs on z/OS. A program that conforms to the XPG4.2 standard can be developed on one system and then ported to another system, compiled and link-edited, and then executed on that system. Such a program is referred to as *portable*.

A programmer can develop a program that uses a mix of standard z/OS services and z/OS UNIX. Such a program is often referred to as a *mixed program*. A mixed program can, for example, be a z/OS program that uses some of the Assembler callable services to access files in the hierarchical file system, or a pipe for temporary data storage.

Shell interactive interface

The interactive interface is called the z/OS UNIX shell. The shell is a command interpreter that accepts commands defined in the POSIX 1003.2 standard. Shell commands can be put together in a sequence, stored in a text file as a shell script, and then executed. The request is then passed to the callable services interface. The shell script is similar to z/OS CLISTs and REXX execs.

REXX

TSO REXX has been extended to provide access to the z/OS UNIX callable services. A REXX exec using z/OS UNIX System Services can be run from TSO/E, in z/OS batch, or in the shell.

You can use a set of z/OS UNIX extensions to TSO/E REXX—host commands and functions—to access kernel callable services. The z/OS UNIX extensions, called syscall commands, have names that correspond to the names of the callable services that they invoke, for example, **`access`**, **`chmod`**, and **`chown`**.

You can run a REXX program with z/OS UNIX extensions from MVS, TSO/E, the shell, or a C program. The REXX exec is not portable to an operating system that does not have z/OS UNIX installed.

Physical file system (PFS)

The PFS interface is a set of protocols and calling interfaces between the logical file system (LFS) and the PFSs that are installed on z/OS UNIX. In a z/OS UNIX System Services environment, UNIX programs and UNIX users access their files through these interfaces. PFSs mount and unmount file systems and perform other file operations.

Supported file types

z/OS UNIX supports the following types of files:

- Regular files
- Directories
- Symbolic links
- Character special files (for example, terminals)
- Pipes (both FIFOs and unnamed)
- Sockets

2.6 Physical file systems

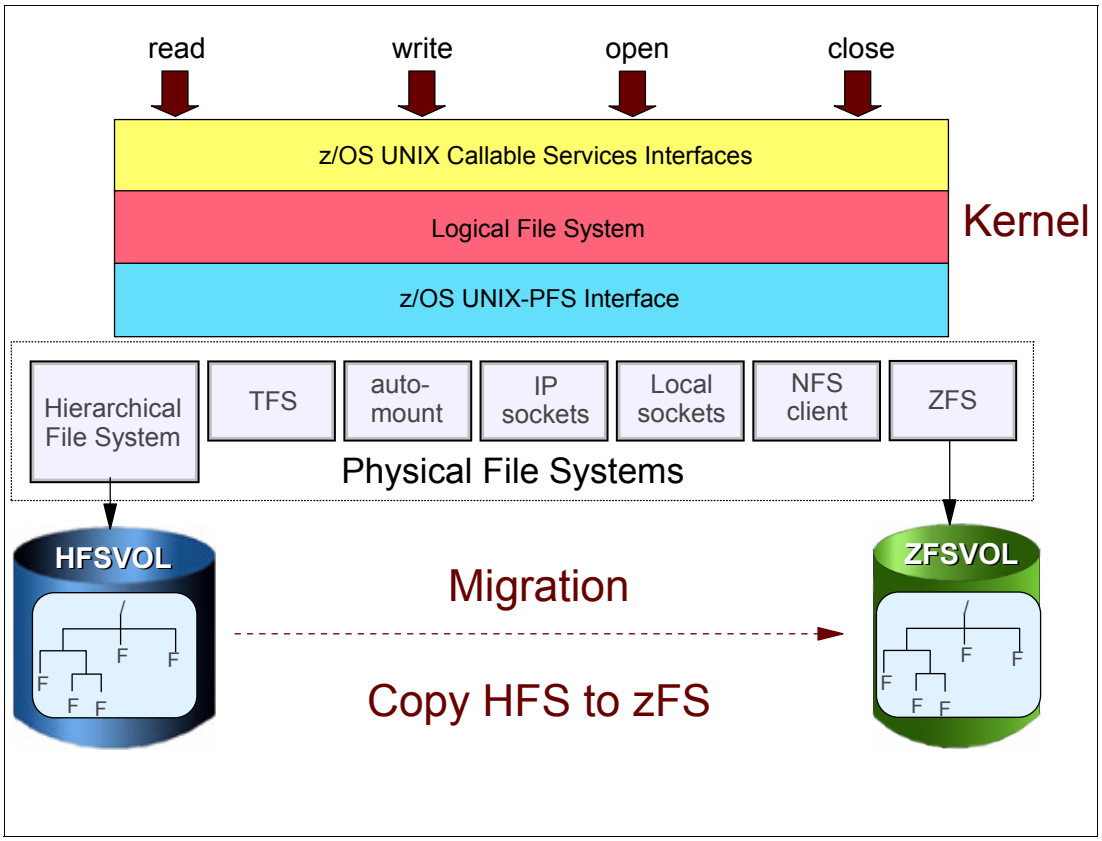


Figure 2-6 z/OS UNIX and physical file systems

Physical file systems

A physical file system (PFS) is packaged as one or more MVS load modules. These load modules must be installed in an APF-authorized MVS load library. The hierarchical file system is not available when a PFS is loaded, so it cannot be installed in the file system.

A PFS controls access to data. PFSs receive and act upon requests to read and write files that they control. The format of these requests is defined by the PFS interface. In a z/OS UNIX, the physical file systems are defined in the BPXPRMxx PARMLIB member. zFS, as a physical file system, is also defined in the PARMLIB member. Figure 2-6 shows all the physical file systems that can be defined in a z/OS UNIX System Services environment. The logical file system (LFS) is called by POSIX programs, non-POSIX z/OS UNIX programs, and VFS servers.

PFS interface‘

The PFS interface is a set of protocols and calling interfaces between the LFS and the PFSs that are installed on z/OS UNIX. PFSs mount and unmount file systems and perform other file operations.

There are two types of PFSs, those that manage files and those that manage sockets:

- File management PFSs, such as HFS and zFS, deal with objects that have path names and that generally follow the semantics of POSIX files.

- Socket PFSs deal with objects that are created by the socket() and accept() functions and that follow socket semantics.

HFS to zFS migration

Because IBM has announced the stabilization of the HFS PFS, a migration of HFS file systems (both mounted and unmounted) to zFS file systems must occur over time.

Advantages of zFS

Like HFS, zFS is a UNIX file system. It contains files and directories that can be accessed with the APIs available for HFS. In general, the application view of zFS is the same as the application view of HFS. Once a zFS file system is mounted, it is almost indistinguishable from any other mounted HFS. The benefits of using zFS are:

- Improved performance

  zFS provides significant performance gains in many customer environments accessing files approaching 8K in size that are frequently accessed and updated. The access performance of smaller files is equivalent to HFS.

- Underlying architecture supports additional functions

  Only zFS supports security labels. Therefore, in a multilevel-secure environment, you must use zFS file systems instead of HFS file systems.

  As an optional function, zFS allows the administrator to make a read-only clone of a file system in the same data set. This clone file system can be made available to users to provide a read-only point-in-time copy of a file system.

  zFS runs as a z/OS UNIX colony address space. Therefore, zFS can be stopped using the **`p zfs`** operator command. zFS file systems should be unmounted or moved to another sysplex member before stopping zFS.

- Improved crash recovery

  zFS provides a reduction in exposure to loss of updates. zFS writes data blocks asynchronously and does not wait for a sync interval. zFS is a logging file system. It logs metadata updates. If a system failure occurs, zFS replays the log when it comes back up to ensure that the file system is consistent.

2.7 z/OS UNIX file sharing in a sysplex

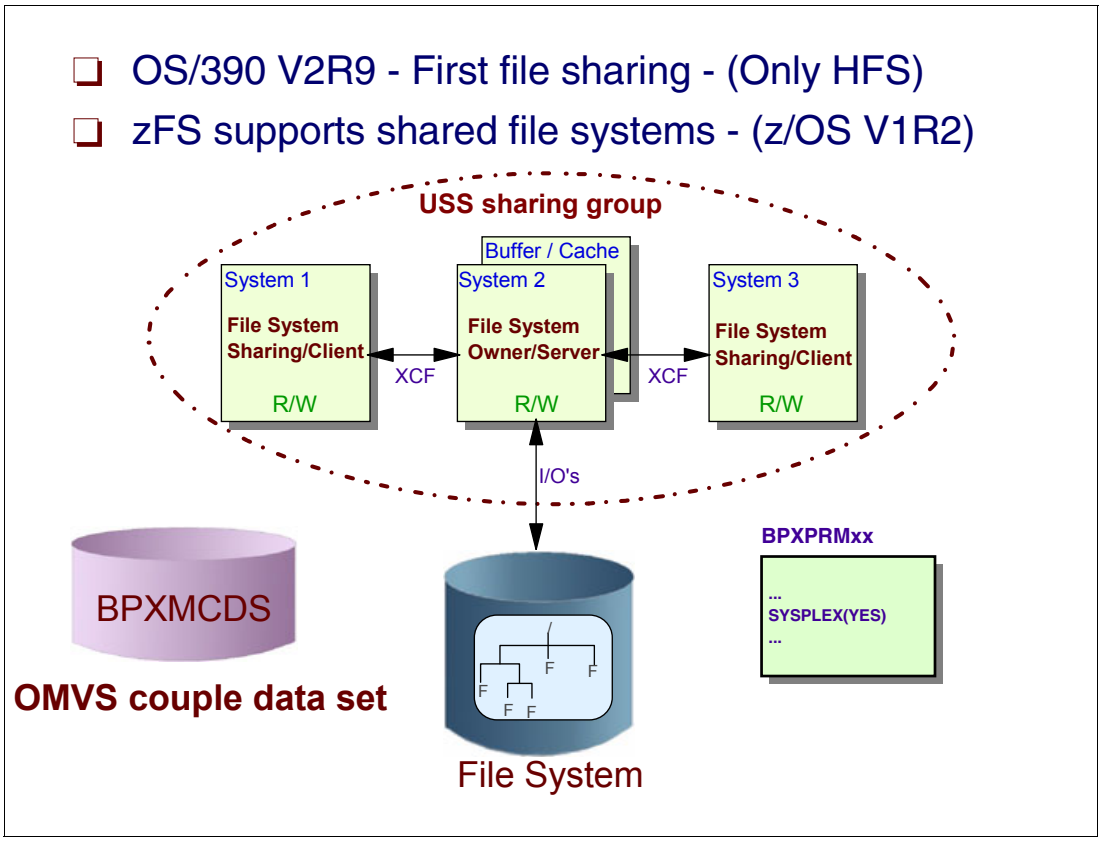


Figure 2-7 z/OS UNIX file sharing

z/OS UNIX file sharing

Although it is suggested that you exploit shared file system support when running in a sysplex environment, you are not required to do so. If you choose not to, you will continue to share file systems as you have before. With shared file system support, all file systems that are mounted by a system participating in a shared file system are available to all participating systems. The system that mounts the file system becomes the owner. All other systems have to go to the owning system to access the file system. In other words, once a file system is mounted by a participating system, that file system is accessible by any other participating system. It is not possible to mount a file system so that it is restricted to just one of those systems. This file sharing mode is supported by both zFS and HFS.

OMVS couple data set

The couple data set (CDS) contains the sysplex-wide mount table and information about all participating systems, and all mounted file systems in the sysplex. To allocate and format a CDS, customize and invoke the BPXISCDS sample job in SYS1.SAMPLIB. The job will create two CDSs: one is the primary and the other is a backup that is referred to as the alternate. In BPXISCDS, you also specify the number of mount records that are supported by the CDS.

SYSPLEX(YES)

In the BPXPRMxx parmlib member, SYSPLEX(YES) specifies a shared file system for those who are in the SYSBPX XCF group, the group that is participating in a shared file system. The systems that specify SYSPLEX(YES) make up the participating group for the sysplex. If

SYSPLEX(YES) is specified in the BPXPRMxx member, but the system is initialized in XCF-local mode, either by specifying COUPLE SYSPLEX(LOCAL) in the COUPLExx member or by specifying PLEXCFG=XCFLOCAL in the IEASYSxx parmlib member, then the kernel will ignore the SYSPLEX(YES) value and initialize with SYSPLEX(NO). This system will not participate in a shared file system support after the initialization completes.

File system owner

The system where the file system is mounted is called the file system owner. The owner of a file system is the first system that processes the mount. This system always accesses the file system locally; that is, the system does not access the file system through a remote system. Other non-owning systems in the sysplex access the file system either locally or through the remote owning system, depending on the PFS and the mount mode.

2.8 z/OS UNIX file systems

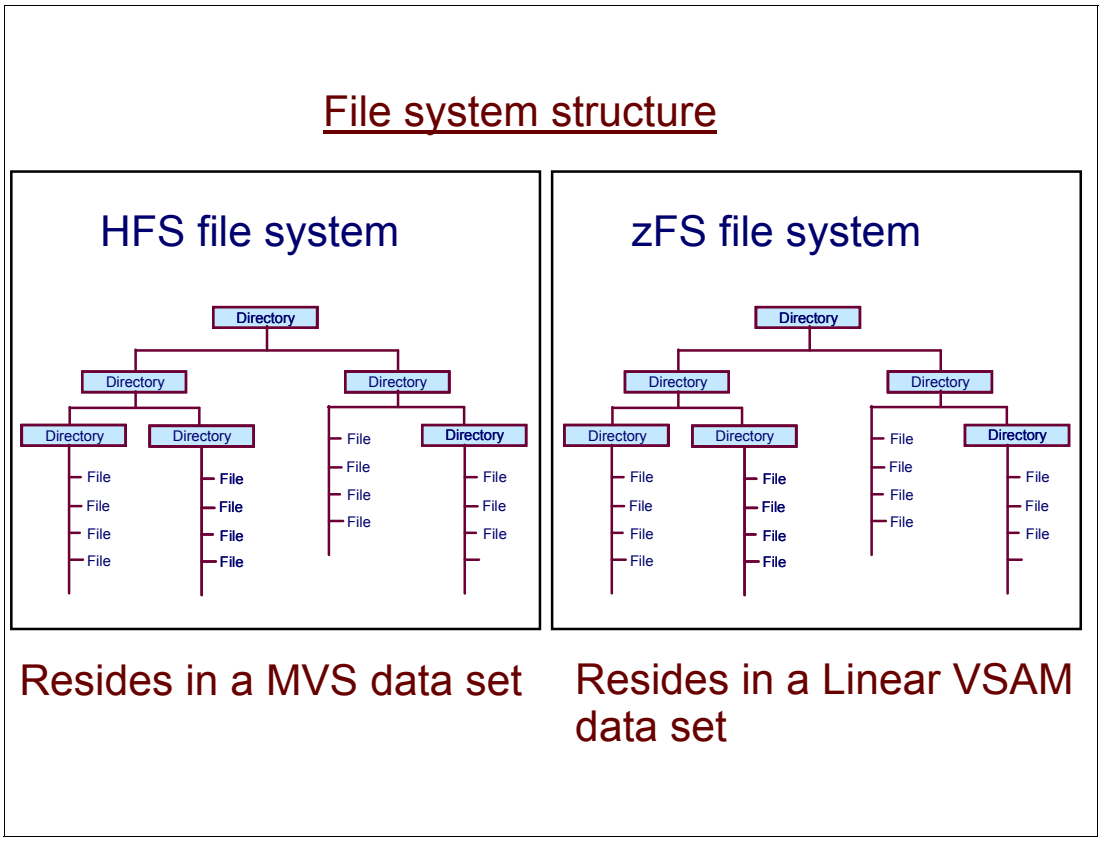


Figure 2-8 Hierarchical file system structure for HFS and zFS file systems

HFS and zFS file systems

The hierarchical file system is used to store data and organize it in a hierarchical way by employing file system entries such as directories and files. These file system entries have certain attributes, such as ownership, permission bits, and access time stamps. The data and the attributes of a file are stored with the file in the file system.

Path name

The path name is constructed of individual directory names and a file name separated by the forward-slash character, for example:

/dir1/dir2/dir3/myfile

Like UNIX, z/OS UNIX is case-sensitive for file and directory names. For example, in the same directory, the file MYFILE is a different file than myfile.

z/OS UNIX data sets

HFS data sets, zFS data sets, and z/OS data sets can reside on the same DASD volume.

The integration of the HFS file system with existing MVS file system management services provides automated file system management capabilities that may not be available on other POSIX platforms. This allows file owners to spend less time on tasks such as backup and restore of entire file systems.

2.9 File system data sets

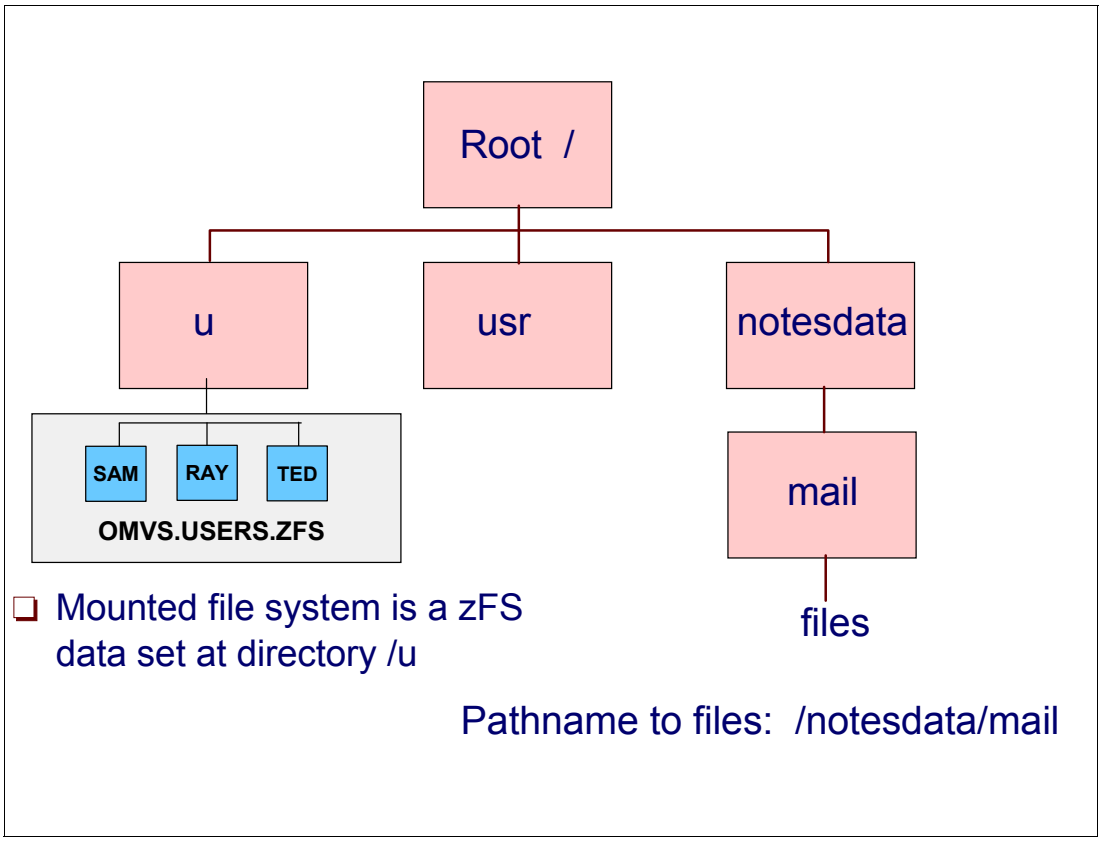


Figure 2-9 File system data sets

z/OS UNIX files

z/OS UNIX files are organized in a hierarchy, as in a UNIX system. All files are members of a directory, and each directory is in turn a member of another directory at a higher level in the hierarchy. The highest level of the hierarchy is the root directory.

MVS views an entire file hierarchy as a collection of hierarchical file system data sets (HFS data sets). Each HFS data set is a mountable file system. DFSMS facilities can be used to manage an HFS data set, and DFSMS Hierarchical Storage Manager (DFSMShsm*) is used to back up and restore an HFS data set.

A file in the hierarchical file system is called an HFS file. HFS files are byte-oriented, rather than record-oriented, as are MVS data sets.

Root file system

The root file system is the first file system mounted. Subsequent file systems can be mounted on any directory within the root file system or on a directory within any mounted file system. The root system is the starting point for the overall HFS file structure. It contains the root directory and any related HFS or zFS files or subdirectories. The root file system is created as part of the installation process, either by the ServerPac method or CPBDO, when you install z/OS.

Path name

The path name is a file name specifying all directories leading to a file.

2.10 Root file system

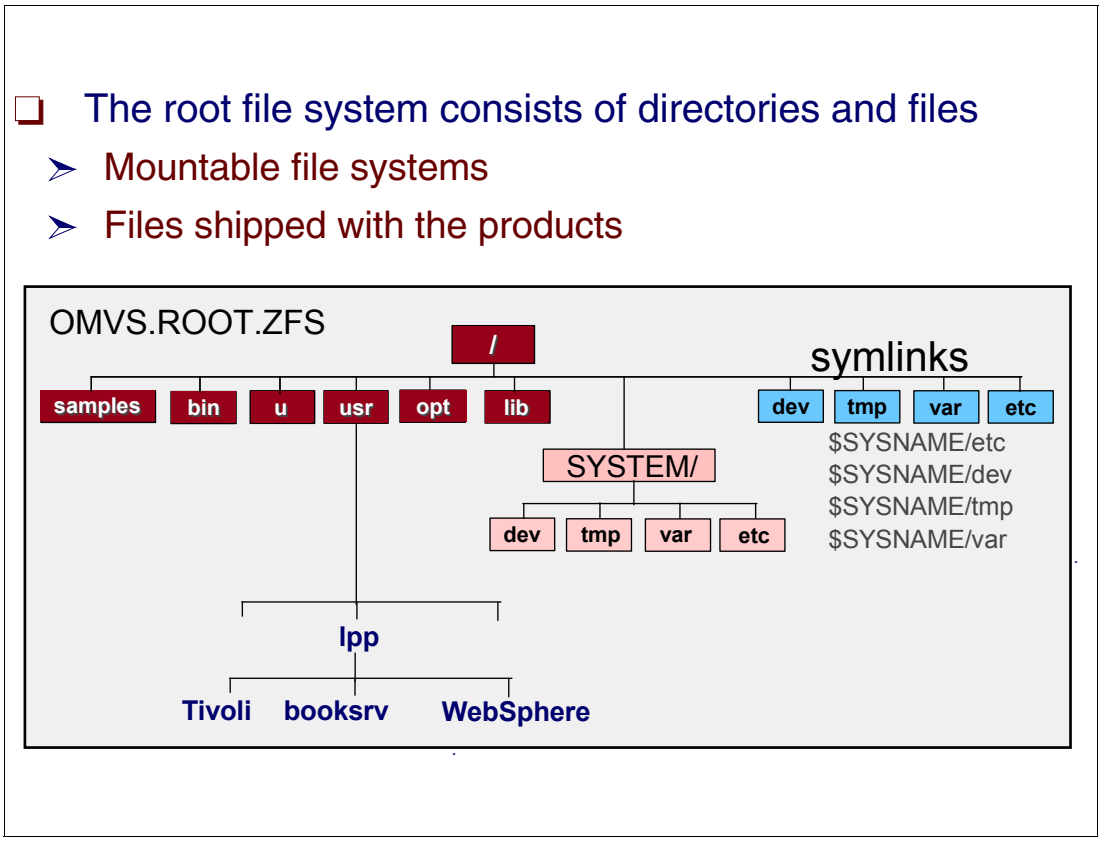


Figure 2-10 The root file system

Root file system

The root file system is the first file system mounted. Subsequent file systems can be mounted on any directory within the root file system or on a directory within any mounted file system.

Note: The root file system is called the version file system or version root.

The file system is the entire set of directories and files, consisting of all HFS files shipped with the products that are ordered and all those created by the system programmer and users. The system programmer (superuser) defines the root file system; subsequently, a superuser can mount other mountable file systems on directories within the file hierarchy. Altogether, the root file system and mountable file systems comprise the file hierarchy used by shell users and applications.

The root consists of /dev, /tmp, /var, and /etc symbolic links that point to the /dev, /tmp, /var, and /etc directories. On each of these directories, there will be a mounted file system containing the directories and files associated with those functions.

2.11 File and directory permission bits

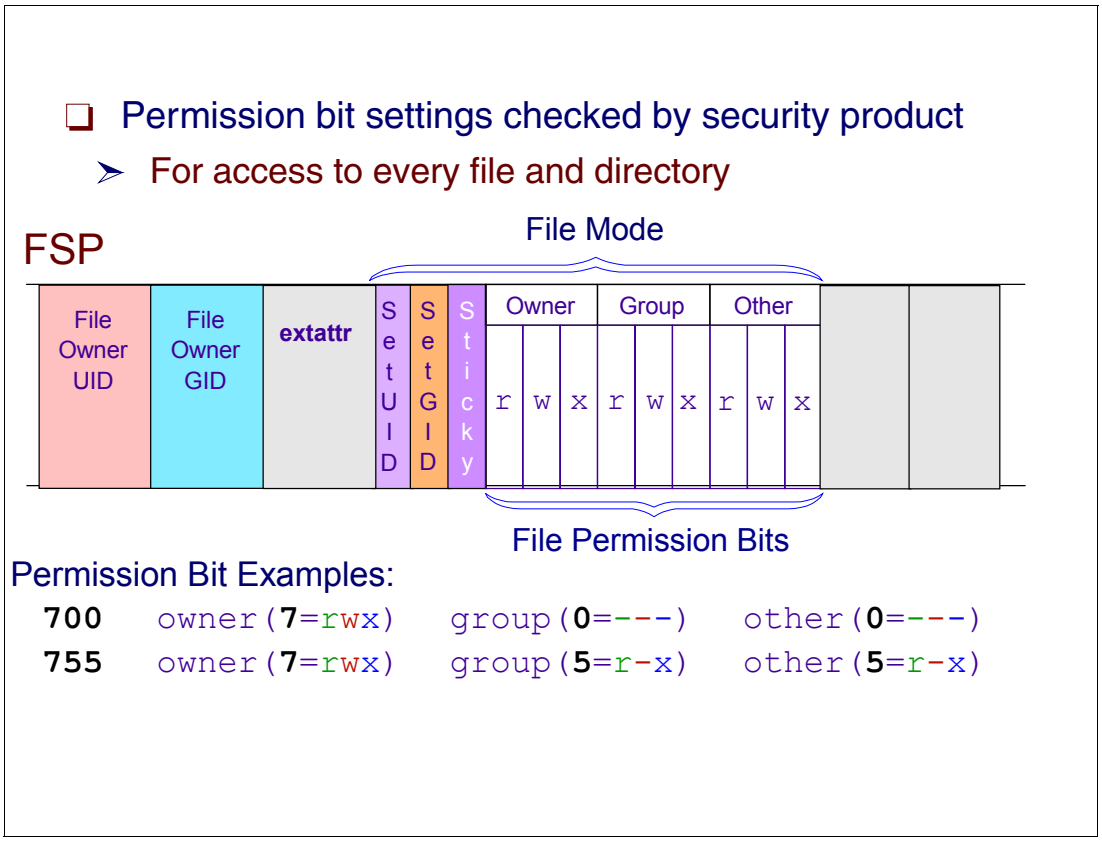


Figure 2-11 Permission bits in the FSP

Permission bits

Permission bit information is stored in the file security packet (FSP) within each file and directory. (ACLs may also be stored with the file.) Permission bits allow you to specify read authority, write authority, or search authority for a directory. They also allow specification of read, write, or execute authority for a file. Because there are three sets of bits, separate authorities can be specified for the owner of the file or directory, the owning group, and everyone else, which is the other category.

File security packet (FSP)

Security information, such as the owner's UID-GID and the permission bits for a file, is kept in a 64-byte area called the file security packet (FSP), which is mapped by IRRPIFSP. The FSP is the security-related section of a file's attributes.

The FSP is created by a SAF call from the PFS when a file is created. Some of the information is taken from the current security environment, and some of it is passed as parameters.

The PFS stores the FSP with the attributes of the file.

When an access check is to be done, the PFS calls SAF with the type of check that is being requested, the audit_structure from the current call, and the file's FSP. SAF passes these to the security product, which extracts user information from the current security environment and compares it against the access control that is stored within the FSP.

2.12 MVS data sets versus file system files

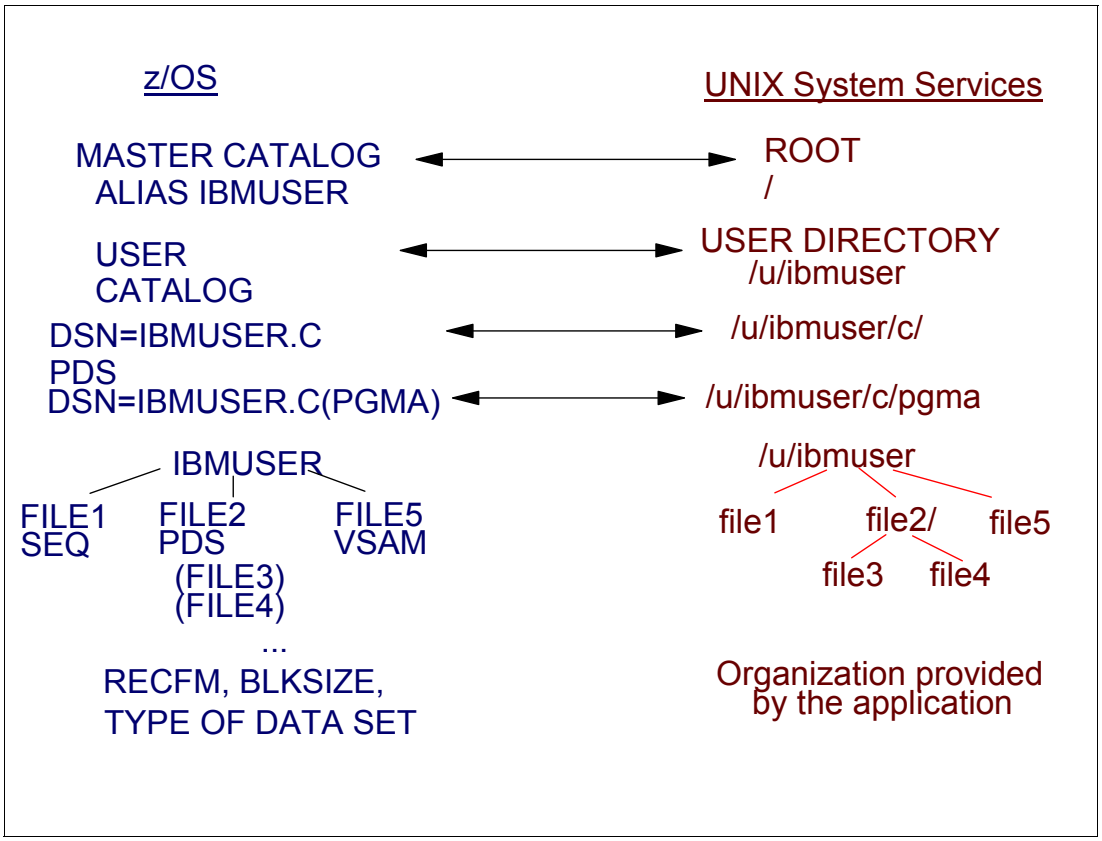


Figure 2-12 Comparison of MVS data set and file system files

Comparing MVS and UNIX files

The z/OS master catalog is analogous to the root directory in a hierarchical file system.

The user prefix assigned to MVS data sets points to a user catalog. The organization of the user catalog is analogous to a user directory (/u/ibmuser) in the file system. Typically, one user owns all the data sets whose names begin with his user prefix. For example, the data sets belonging to the TSO/E user ID IBMUSER all begin with the prefix IBMUSER. There could be data sets named IBMUSER.C, and IBMUSER.C(PGMA).

In the file system, ibmuser would have a user directory named /u/ibmuser. Under that directory there could be subdirectories named /u/ibmuser and /u/ibmuser/c/pgma.

Of the various types of MVS data sets, a partitioned data set (PDS) is most like a user directory in the file system. In a partitioned data set such as IBMUSER.C, you could have members PGMA, PGMB, and so on. For example, you could have IBMUSER.C(PGMA) and IBMUSER.C(PGMB). A subdirectory such as /u/ibmuser/c can hold many files, such as pgma, pgmb, and so on.

All data written to the hierarchical file system can be read by all programs as soon as it is written. Data is written to a disk when a program issues an fsync().

2.13 zFS or HFS data sets

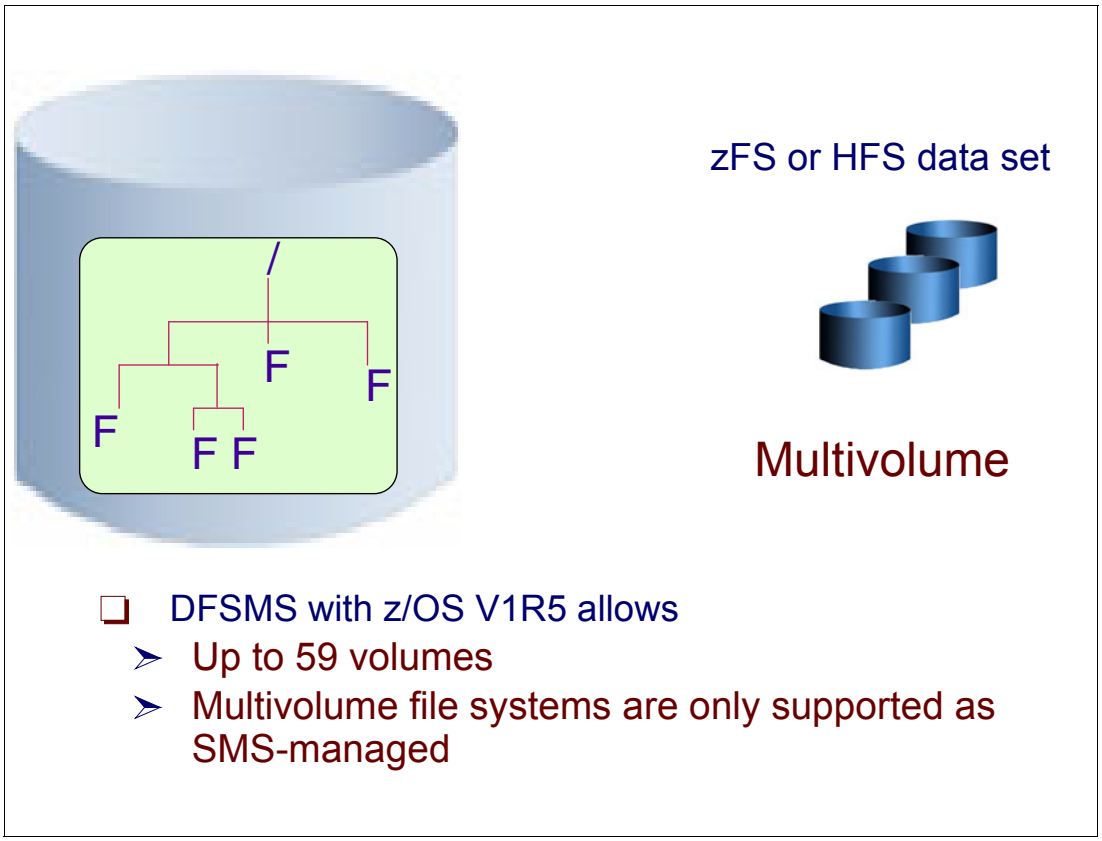


Figure 2-13 zFS or HFS data sets

File system data sets

A z/OS UNIX hierarchical file system is contained in a data set type called zFS or HFS. A zFS or HFS data set can reside on an SMS-managed volume, and it is a single volume data set if it is non-SMS-managed. zFS or HFS data sets can reside with other MVS data sets on SMS-managed volumes or non-SMS-managed volumes. Multiple systems can share a zFS or HFS data set if it is mounted in read-only mode. Beginning with OS/390 V2R9, sharing can be done in read-write mode.

Note: APAR OW35441 now gives you the ability to allocate PDSE and HFS data sets on unmanaged (non-SMS) volumes, if running DFSMS 1.4 or DFSMS 1.5.

An HFS data set is allocated by specifying HFS in the DSNTYPE parameter. You can also define a data class for zFS or HFS data sets. OS/390 V2R7 began to support multivolume access up to 59 physical volumes, but then the data set must be SMS-managed.

2.14 z/OS UNIX components

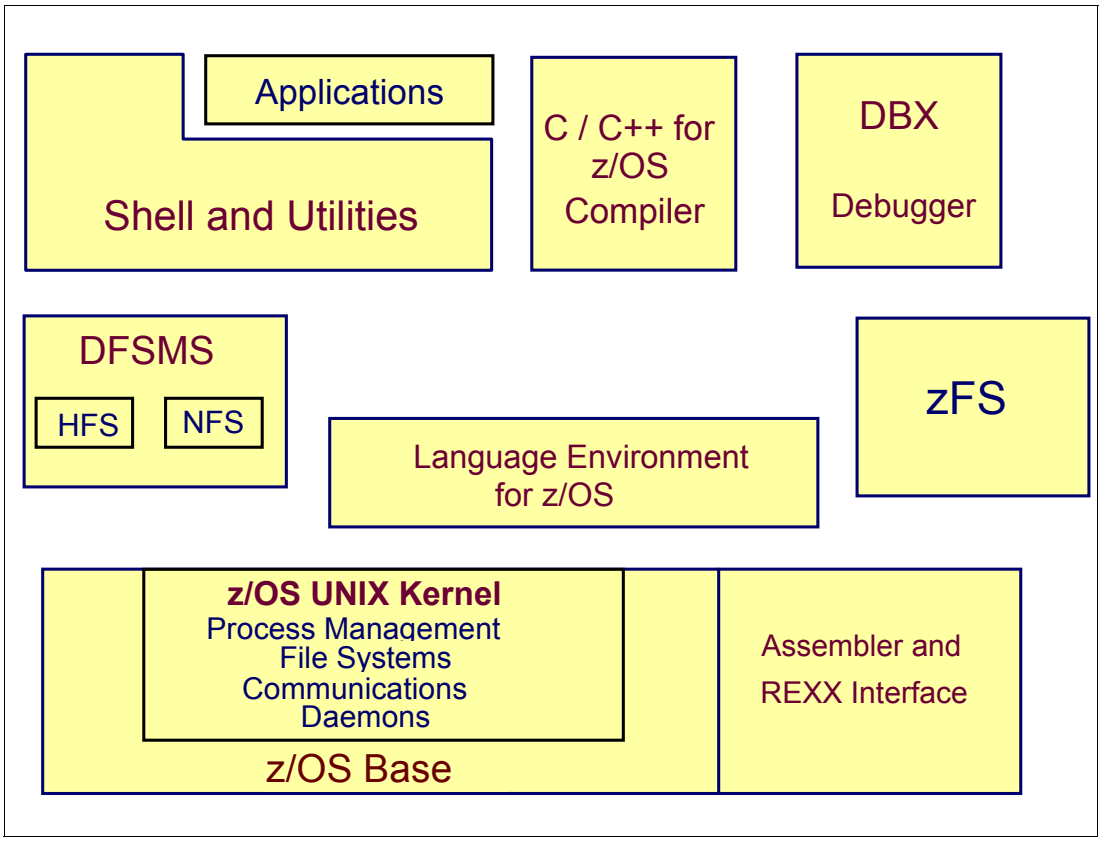


Figure 2-14 The components of z/OS UNIX

z/OS UNIX components

z/OS UNIX offers open interfaces for applications and interactive users on a z/OS system. The z/OS UNIX components and their functions are now described.

The z/OS UNIX kernel

At system IPL time, kernel services are started automatically. The kernel provides z/OS UNIX System Services in answer to requests from programs and the shell. The kernel manages the file system, the communications, and the program processes.

The hierarchical file system is shipped as part of DFSMS. zFS is shipped with the Distributed File Service.

The POSIX standard introduces a completely new terminology in the MVS environment. A typical UNIX operating system consists of a kernel that interfaces directly with the hardware. Built on the kernel is the shell and utilities layer that defines a command interface. Then there are application programs built on the shell and utilities.

In a z/OS UNIX environment the file system is considered part of the kernel because it is allocated to the kernel. The support for the file system is provided by the DFSMS product. Figure 2-14, showing the file system as part of the kernel, shows a logical view of the solution.

The z/OS UNIX API conforms to the POSIX and XPG4 standard. OS/390 was branded as a UNIX system by the Open Group in 1996. To support the APIs, the z/OS system must provide some system services that are included in the kernel, such as the file system and communication services.

Daemons are programs that are typically started when the operating system is initialized and remain active to perform standard services. Some programs are considered daemons that initialize processes for users even though these daemons are not long-running processes. z/OS UNIX supplies daemons that start applications and start a user shell session when requested.

The z/OS UNIX shell and utilities

This is an interactive interface to z/OS UNIX services that interprets commands from interactive users and programs. The shell and utilities component can be compared to the TSO function in z/OS.

The z/OS UNIX debugger (dbx)

The z/OS UNIX debugger is a tool that application programmers can use to interactively debug a C program. The dbx debugger is not part of the POSIX standard and is well known in many UNIX environments.

The C/C++ compiler and C run-time libraries

The C/C++ compiler and C run-time libraries are needed to make executables that the kernel understands and can manage.

DFSMS

DFSMS manages the hierarchical file system (HFS) data sets that contain the file systems. To use kernel services in full-function mode, SMS must be active.

Network File System (NFS) enables users to mount file systems from other systems so that the files appear to be locally mounted. You end up with a mixture of file systems that come from systems where the UIDs and GIDs may be independently managed.

Language Environment (LE)

The C compiler and Language Environment feature library are changed and extended to include support for the POSIX and XPG4 C function calls. The LE product provides a common run-time environment and language-specific run-time services for compiled programs.

To run a shell command or utility, or any user-provided application program written in C or C++, you need the C/C++ run-time library provided with the Language Environment.

zFS

The zSeries File System (zFS) is a UNIX file system that can be used in addition to the hierarchical file system physical file system. This file system has been available since 1995 as part of the DCE component of MVS/ESA V5R2.2, OS/390, and z/OS V1R1. The file system has been known as the DCE DFS Local File System (LFS), sometimes referred to as Episode. It is a high-performance, log-based file system. The DCE DFS Local File System is part of the OSF DFS product, and as such is included in the Distributed File Service base element of z/OS V1R2. zFS is a separate component of the DFS base element, so it can be serviced separately from the other components of DFS.

2.15 z/OS UNIX programs (processes)

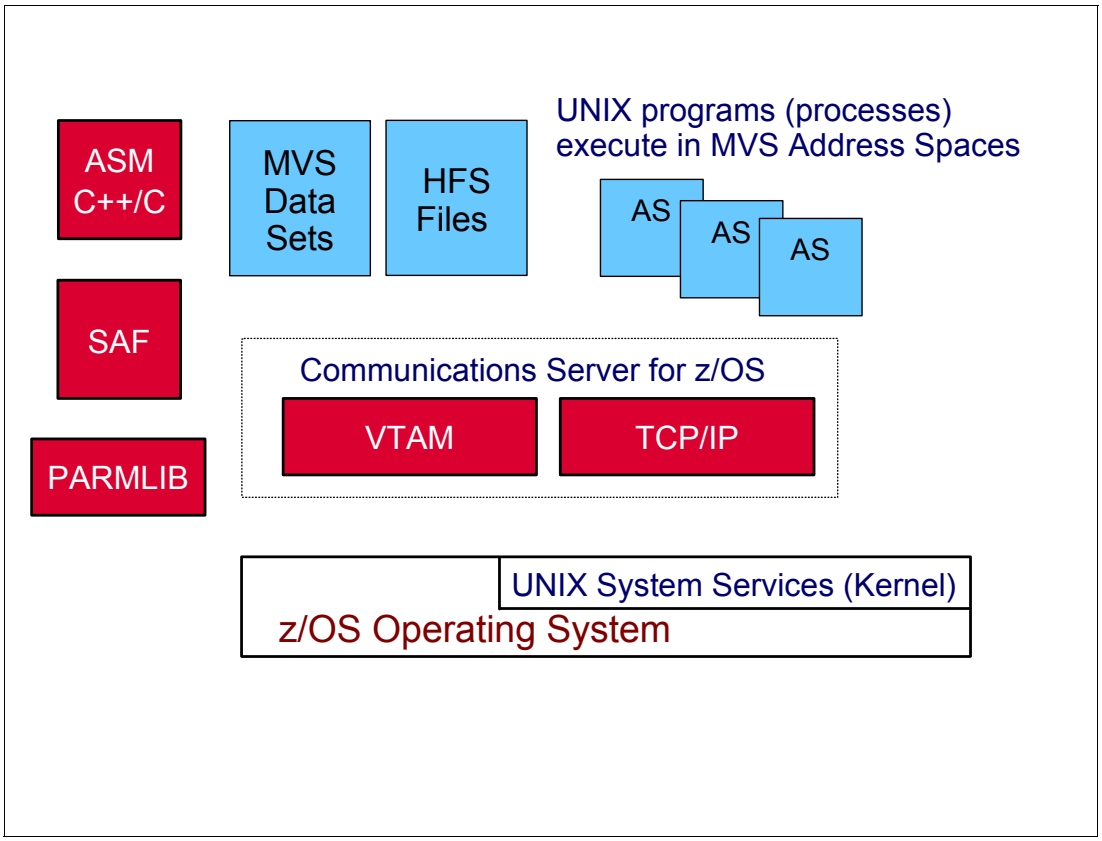


Figure 2-15 z/OS UNIX programs (processes) and components

z/OS UNIX processes

A process is a program using kernel services. The program can be created by a fork() function, fork callable service, or spawn() function; or the program can be dubbed because it requested kernel services. The three types of processes are:

- User processes, which are associated with a program or a shell user
- Daemon processes, which perform continuous or periodic system-wide functions, such as printer spooling
- Kernel processes, which perform system-wide functions for the kernel such as cleaning up zombie processes (init process)

UNIX commands

When you enter a shell command, you start a process that runs in an MVS address space. When you enter that command, the z/OS shell runs it in its own process group. As such, it is considered a separate job and the shell assigns it a job identifier—a small number known only to the shell. A shell job identifier identifies a shell job, not an MVS job. When the process completes, the system displays the shell prompt.

UNIX programs and components

UNIX programs running in MVS address spaces use or access the following:

ASM/C++/C These programming languages can be used to create the programs.

SAF	A security product is required when running z/OS UNIX System Services. SAF is the interface to RACF, Top Secret, or ACF2.
BPXPRMxx	This PARMLIB member determines the number of processes that may be started in the z/OS system.
MVS data sets	UNIX programs may access MVS data sets.
HFS files	UNIX programs may access HFS files.
TCP/IP	Workstation users can enter the shell environment by using **`rlogin`** or **`telnet`** in a TCP/IP network. User-written applications can use TCP/IP as a communication vehicle.
VTAM	Workstation users can access TSO/E through VTAM. z/OS UNIX is then accessed from TSO/E.

2.16 Create a process

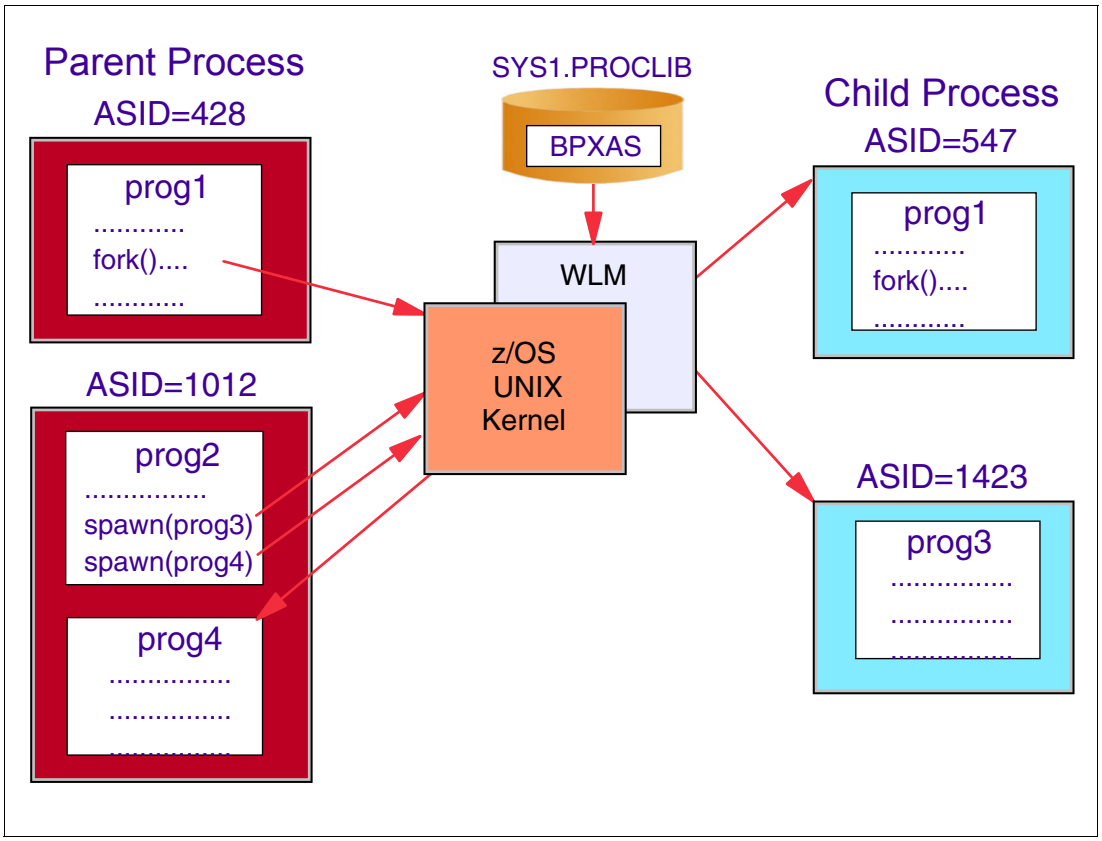


Figure 2-16 Creating a process

Creating a process

A process is created by using any of the following methods:

- C fork() function

  Creates a child process that is identical to the parent process in a new address space scheduled by the Workload Manager (WLM).

- C spawn() function

  Creates a child process that executes a different program than the parent, either in a new address space scheduled by WLM, or in the same address space as the parent process (local spawn).

- z/OS UNIX callable services

  When a program uses a z/OS UNIX assembler callable service, the z/OS address space is *dubbed* a z/OS UNIX process. The address space gets a PID. The dub does not result in a new address space.

Using the fork() function

Fork() is a POSIX/XPG4 function that creates a duplicate process referred to as a child process. The process that issues the fork() is referred to as the parent process.

With z/OS UNIX, a program that issues a fork() function creates a new address space that is a copy of the address space where the program is running. The fork() function does a

program call to the kernel, which uses WLM/MVS facilities to create the child process. The contents of the parent address space are then copied to the child address space.

After the fork() function, the program in the child address space will start at the same instruction as the program in the parent address space. Control is returned to both programs at the same point. The only difference that the program sees is the return code from the fork() function. A return code of zero is returned to the child after a successful fork(). All other return codes are valid only for the parent.

Child address space

Once the child address space has been created, the child gets the required storage from a STORAGE request. The kernel then copies the contents of the parent address space to the child address space using the MVCL instruction. Once the copy has been completed, a short conversation between the kernel and the child process takes place. At this point, both the parent and child process are activated. The program in the child address space gets control at the same point as the program in the parent address space. The only difference is the return code from the fork() function.

The child address space is almost an identical copy of the parent address space. User data, for example private subpools, and system data, like Recovery Termination Management (RTM) control blocks, are identical.

WLM and the kernel

The kernel interfaces to WLM to create the new address space for a fork or spawn. WLM uses an IBM-supplied procedure, BPXAS, to start up a new address space. This new address space then initializes the UNIX child process to run in the address space. After the child process completes, this address space can be reused for another fork or spawn. If none is waiting, BPXAS times out after being idle for 30 minutes.

2.17 z/OS UNIX processes

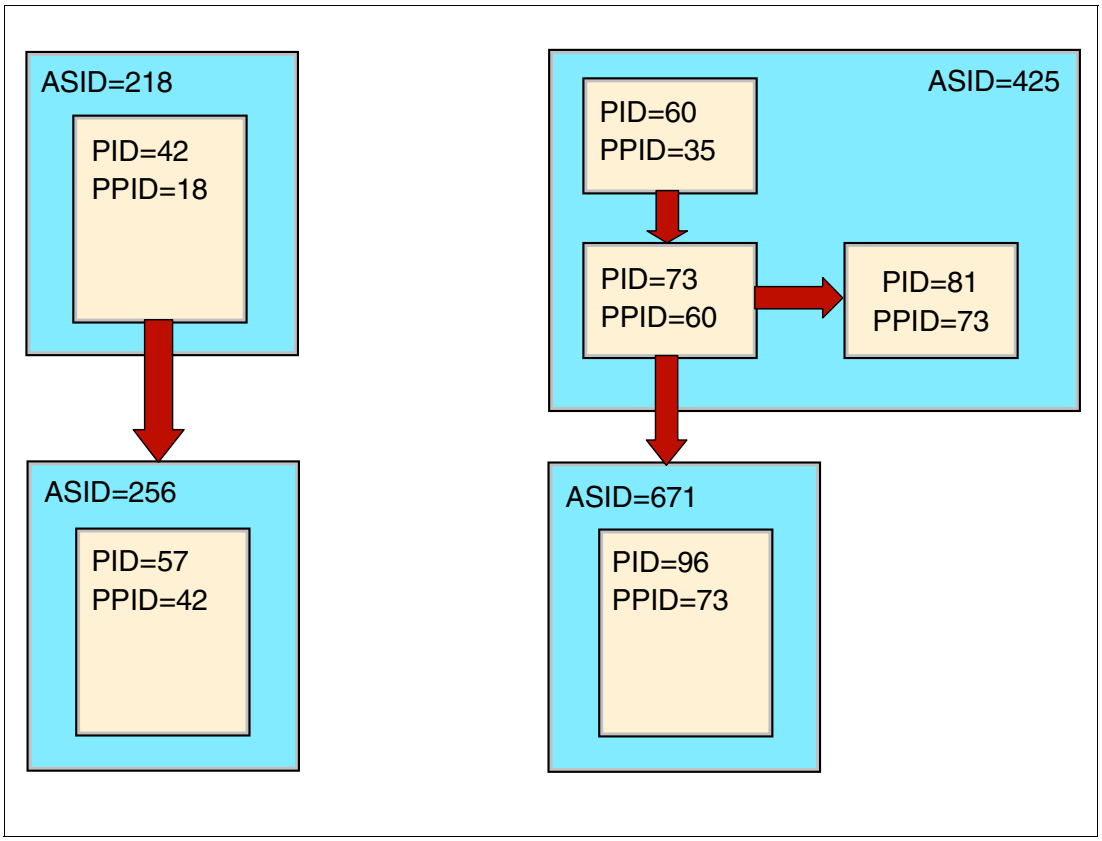


Figure 2-17 z/OS UNIX processes

z/OS UNIX processes

z/OS UNIX uses processes to run programs, and to associate resources to the programs. A z/OS address space can contain one or multiple processes. A process is created by another process, or by a request for a z/OS UNIX service. The process that creates a new process is called a parent process and the new process is called a child process. There will be a hierarchy of processes in the system.

Process ID (PID)

Every process is identified by a process ID (PID) and is associated with its parent process by a parent process ID (PPID).

z/OS UNIX supports processes that run in unique address spaces. These would be created by fork() and exec() services. It also supports local processes. These will share an address space and are created by the spawn() service.

The system also assigns a process group identifier (PGID) and a process identifier (PID). When only one command is entered, the PGID is the same as the PID. The PGID can be thought of as a system-wide identifier. If you enter more than one command at a time using a pipe, several processes, each with its own PID, will be started. However, these processes all have the same PGID and shell job identifier. The PGID is the same as the PID for the first process in the pipe.

Process identifiers

Process identifiers associated with a process are as follows:

PID A process ID. A unique identifier assigned to a process while it runs. When the process ends, its PID is returned to the system. Each time you run a process, it has a different PID (it takes a long time for a PID to be reused by the system). You can use the PID to track the status of a process with the **ps** command or the **jobs** command, or to end a process with the **kill** command.

PGID Each process in a process group shares a process group ID (PGID), which is the same as the PID of the first process in the process group. This ID is used for signaling related processes. If a command starts just one process, its PID and PGID are the same.

PPID A process that creates a new process is called a parent process; the new process is called a child process. The parent process ID (PPID) becomes associated with the new child process when it is created. The PPID is not used for job control.

A process can create one or more child processes, and the child processes can be parent processes of new child processes, thus creating a hierarchy of related processes. The PPID maintains the relationship between processes. Usually, a process creates a child process to perform a separate task, for example a shell command. The child process ends when the task is completed while the parent process continues to execute. If for some reason a parent process should terminate before a child process, the child process will become an orphan process. Orphan processes are inherited by the first process created in the system, called the *init* process.

2.18 z/OS UNIX interactive interfaces

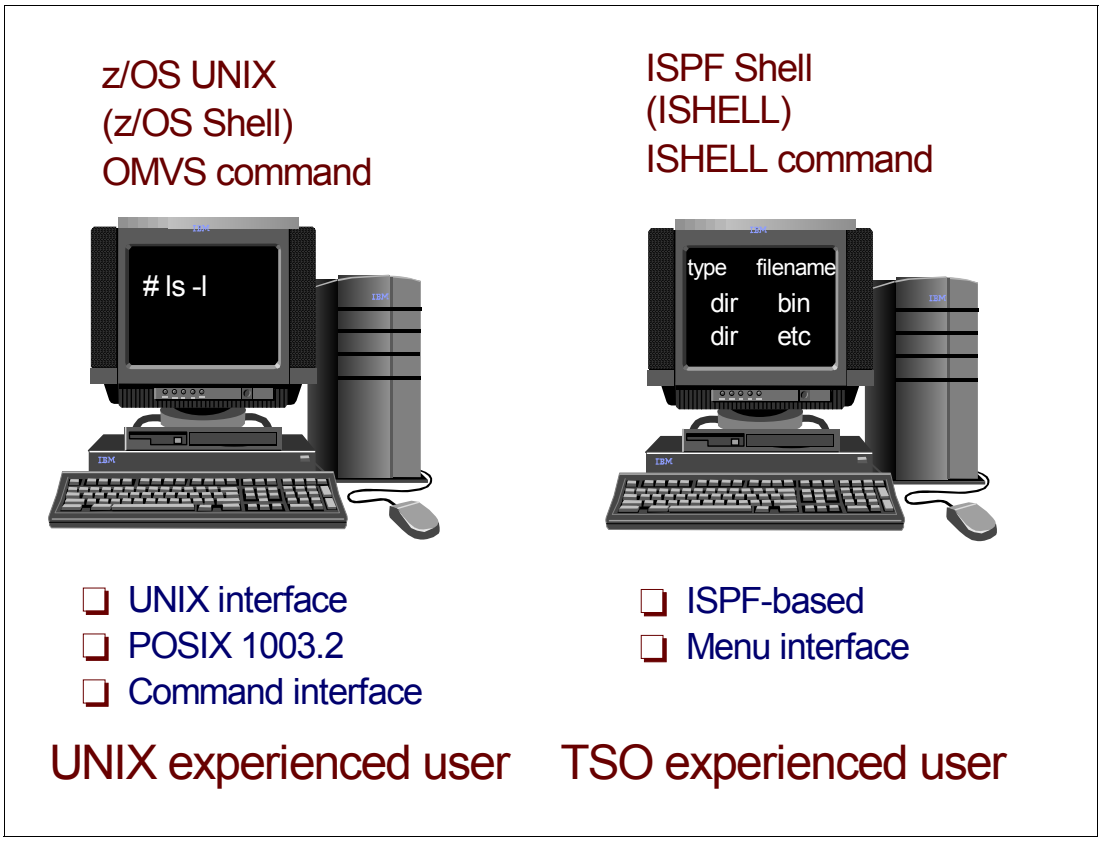


Figure 2-18 z/OS UNIX interactive interfaces

z/OS UNIX interactive interfaces

Figure 2-18 is an overview of the two interactive interfaces, z/OS UNIX shell and the ISHELL. In addition, there are some TSO/E commands to support z/OS UNIX, but they are limited to certain functions such as copying files and creating directories.

The z/OS UNIX shell is based on the UNIX System V shell and has some of the features from the UNIX Korn shell. The POSIX standard distinguishes between a *command*, which is a directive to the shell to perform a specific task, and a *utility*, which is the name of a program callable by name from the shell. To the user, there is no difference between a command and a utility.

ISHELL or OMVS shell

Interactive users of z/OS UNIX have a choice between using a UNIX-like interface (the shell), a TSO interface (TSO commands), and an ISPF interface (ISPF CUA dialog). With these choices, users can choose the interface which they are most familiar with and get a quicker start on z/OS UNIX.

The z/OS UNIX shell provides the environment that has the most functions and capabilities. Shell commands can easily be combined in pipes or shell scripts and thereby become powerful new functions. A sequence of shell commands can be stored in a text file which can be executed. This is called a shell script. The shell supports many of the features of a regular programming language.

TSO commands for interactive use

There are some TSO commands that provide support for z/OS UNIX System Services, as follows:

ISHELL The ISHELL command will invoke the ISPF shell. The ISHELL is a good starting point for users familiar with TSO and ISPF who want or need to use z/OS UNIX. The ISHELL provides CUA panels where users can work with the hierarchical file system. There are also panels for mounting/unmounting file systems and for doing some z/OS UNIX administration.

OMVS The OMVS command used to invoke the z/OS UNIX shell.

The ISHELL is an ISPF dialog for users and system administrators which can be used instead of shell commands to perform many tasks related to file systems, files, and z/OS UNIX user administration.

REXX support

The REXX support for z/OS UNIX is not really an interactive interface, but we chose to introduce it here since it is most often used in TSO or in the shell. The SYSCALL environment is not built into TSO/E, but an external function call called SYSCALLS initializes the environment.

Note: The shell is the initial host environment, which means that the SYSCALL environment is automatically initialized. A difference between the REXX support and shell scripts is that a REXX EXEC can be invoked from a C program, while a shell script can only be interpreted from the shell. A REXX EXEC can be called from a shell script.

An interactive user who uses the OMVS command to access the shell can switch back and forth between the shell and TSO/E, the interactive interface to MVS.

Programmers whose primary interactive computing environment is a UNIX or AIX® workstation find the z/OS shell programming environment familiar.

Programmers whose primary interactive computing environment is TSO/E and ISPF can do much of their work in that environment.

Interactive users of z/OS UNIX have a choice between using a UNIX-like interface (the shell), a TSO interface (TSO commands), or an ISPF interface (ISPF CUA dialog). With these choices, users can choose the interface which they are most familiar with and get a quicker start on z/OS UNIX.

2.19 ISPF Option 6

```
  Menu  List  Mode  Functions  Utilities  Help
_____________________________________________________________________________
                              ISPF Command Shell
Enter TSO or Workstation commands below:

===> ISHELL_____________________________________________________________________
_____________________________________________________________________________
_____________________________________________________________________________

Place cursor on choice and press enter to Retrieve command

=> ishell
=> omvs
=> netstat
=>
=>
=>
=>
=>
=>
=>
```

Figure 2-19 ISPF Option 6 panel

Interactive use via ISPF Option 6

With the ISPF shell (ISHELL), a user or system programmer can use ISPF dialogs instead of shell commands to perform many tasks, especially those related to file systems and files. After a logon to TSO/E, enter Option 6 under ISPF to use the OMVS and ISHELL commands. The ISPF shell is a panel interface that you can use instead of TSO/E commands or shell commands to perform certain tasks. For example, you can use the ISPF shell to display all mounted file systems or its attributes, such as total blocks. You can also use the ISPF shell to perform the following tasks, which require superuser authority or the RACF SPECIAL attribute or both.

A user can use the ISPF shell to work with:

- Directories
- Regular files
- FIFO special files
- Symbolic links, including external links

You can also run shell commands, REXX programs, and C programs from the ISPF shell. The ISPF shell can only direct stdout and stderr to a file in your file system, not to your terminal. If it has any contents, the file is displayed when the command or program completes. If you are a user with an MVS background, you may prefer to use the ISPF shell panel interface instead of shell commands or TSO/E commands to work with the file system. The ISPF shell also provides the administrator with a panel interface for setting up users for z/OS UNIX access, for setting up the root file system, and for mounting and unmounting a file system.

2.20 ISHELL command (ish)

```
  File  Directory  Special_file  Tools  File_systems  Options  Setup  Help
_____________________________________________________________________________

                     UNIX System Services ISPF Shell
Command ===> ____________________________________________________________

Enter a pathname and do one of these:

    - Press Enter.
    - Select an action bar choice.
    - Specify an action code or command on the command line.

Return to this panel to work with a different pathname.
                                                                More:     +
   /u/rogers
   ______________________________________________________________
   ______________________________________________________________
   ______________________________________________________________

EUID=0
```

Figure 2-20 Panel displayed after issuing the ish command

ISHELL command panel

Figure 2-20 shows the ISHELL or ISPF shell panel displayed as a result of the **ishell** or **ish** command entered from ISPF Option 6. Fill in a path name or select a pull-down choice. If you do not enter a path name, your working directory is used. Some pull-down choices do not require a path name.

To search a user's files and directories, type the following and press Enter:

```
/u/userid
```

Pull-down menu

At the top of the panel is the action bar or pull-down menu, with eight choices, as follows:

File - Directory - Special file - Tools - File systems - Options - Setup - Help

When you select one of these choices, a pull-down panel with a list of actions is displayed.

EUID

The effective user ID (EUID) of the current user or process, is initialized at shell startup.

2.21 User's files and directories

```
      File  Directory  Special_file  Commands  Help
_____________________________________________________________________________

                                Directory List
Command ===> ________________________________________________________________

Select one or more files with / or action codes.  If / is used also select an
action from the action bar otherwise your default action will be used.  Select
with S to use your default action.  Cursor select can also be used for quick
navigation.  See help for details.
EUID=0   /u/rogers/
  Type  Perm  Changed-EST5EDT,  Owner      ------Size  Filename   Row 1 of 27
_ Dir    755  2009-03-31 16:31  STC              8192   alice
_ Dir    700  2009-11-10 05:49  STC              8192  .
_ Dir    555  2009-11-03 10:14  STC                 0  ..
_ File   400  2009-08-18 17:27  STC             17408  .ishell-reflist-ROGERS
_ File   600  2001-03-14 15:44  STC               793  .profile
_ File   600  2009-11-10 05:51  STC              2874  .sh_history
_ File   755  2005-08-17 14:01  STC                 0  apf.file
_ Dir    755  2006-03-02 11:43  STC              8192  cachex
_ Dir    755  2006-03-02 15:58  STC              8192  cachey
_ Dir    755  2006-02-28 10:31  STC              8192  cache1
_ Dir    755  2006-02-28 11:51  STC              8192  cache2
_ File   700  2007-08-02 11:32  STC               729  cbprnt
```

Figure 2-21 Display of a user's files and directories

Directory List

This function displays all files in a directory. You can customize the columns by using the Options pull-down and then selecting the Directory List choice. Select one or more files by action code or / and press Enter. To scroll the list of files, use the Forward and Backward function keys. Processing does not begin until you press Enter.

Displaying files and directories

An action code is a single character you can enter to select a specific action for a file. To display the valid action codes for a file type, put the cursor under the file type and press the Help function key.

Shown in the figure are the files and directories of user rogers. The user can then use action codes to do the following:

b Browse a file or directory

e Edit a file or directory

d Delete a file or directory

r Rename a file or directory

a Show the attributes of a file or directory

c Copy a file or directory

2.22 OMVS shell session

```
IBM
Licensed Material - Property of IBM
5694-A01 Copyright IBM Corp. 1993, 2009
(C) Copyright Mortice Kern Systems, Inc., 1985, 1996.
(C) Copyright Software Development Group, University of Waterloo, 1989.

All Rights Reserved.

U.S. Government Users Restricted Rights -
Use,duplication or disclosure restricted by
GSA ADP Schedule Contract with IBM Corp.

IBM is a registered trademark of the IBM Corp.

PATH is set to /usr/lpp/java/J1.4/bin:/usr/lpp/Printsrv/bin:/bin:.:/usr/lpp/java
18/J1.1/bin
JAVA_HOME is set to /usr/lpp/java18/J1.1
CLASSPATH is set to /usr/lpp/Printsrv/classes/ipp.jar:/usr/lpp/Printsrv/classes/
ippserver.jar:/usr/lpp/Printsrv/classes/ippclient.jar:/usr/lpp/Printsrv/classes/
ippreal.jar:/usr/lpp/java18/J1.1/lib/classes.zip
LIBPATH is set to /usr/lpp/Printsrv/lib:/lib:/usr/lib:.:/usr/lpp/java18/J1.1/lib
/mvs/native_threads

 ===> ls -al                                                          INPUT
```

Figure 2-22 OMVS shell session display after issuing the OMVS command in ISPF

OMVS shell session

After you invoke the shell with the OMVS command, you can switch among the shell, a TSO/E session, and subcommand mode. You can switch to TSO/E to perform some tasks without interrupting a process that is running and without shutting down the shell. You can switch to subcommand mode to end the process if your process is in a loop and you are unable to interrupt the process or enter a command.

Exit shell session

To exit the shell when a foreground process has completed, type `exit` and press Enter, or press the <ESC>-D keys in sequence (where <ESC> is a defined escape character displayed at the bottom of your panel). Any processes that are spawned using the **nohup** command continue to run after you exit. Any processes spawned in the background without using the **nohup** command die when you exit.

Enter only one subcommand at a time on the command line. You can enter subcommands with leading or trailing blanks, but an empty or blank command line is ignored.

2.23 OMVS command shell session

```
U.S. Government Users Restricted Rights -
Use,duplication or disclosure restricted by
GSA ADP Schedule Contract with IBM Corp.

IBM is a registered trademark of the IBM Corp.

ROGERS @ SC74:/u/rogers>ls -al
total 202
drwxr-x---  13 SYSPROG  SYS1         960 Nov  4 04:52 .
dr-xr-xr-x  11 SYSPROG  TTY            0 Nov  4 04:46 ..
-rw-------   1 SYSPROG  SYS1        2183 Nov  4 04:58 .sh_history
drwxr-xr-x   2 SYSPROG  SYS1         256 Sep 13  2006 00000100
drwxr-xr-x   5 SYSPROG  OMVSGRP     1120 Jul 26  2006 ITSO-link
drwxrwxrwx   4 SYSPROG  SYS1         352 Aug  2 16:33
drwxr-xr-x   4 SYSPROG  SYS1         320 Aug  2 16:33
drwxr-xr-x   2 SYSPROG  SYS1         256 Sep 13  2006 db2
drwxr-xr-x   2 SYSPROG  SYS1         256 Sep 13  2006 echo
drwxr-xr-x   2 SYSPROG  SYS1         256 Sep 13  2006 ims
-rwx------   1 SYSPROG  SYS1           0 Aug 28  2006 inetd-stderr
-rwx------   1 SYSPROG  SYS1           6 Apr 17  2008 inetd-stdout
-rwxr-xr-x   1 SYSPROG  SYS1       73728 Apr 21  2006 maxcore
-rw-r--r--   1 SYSPROG  SYS1           0 Aug  3 12:42 more
ROGERS @ SC74:/u/rogers>
```

Figure 2-23 OMVS shell session after entering the ls -al command

Entering the UNIX shell using the OMVS command

Use the OMVS command to invoke the z/OS UNIX shell. Once you are working in a shell session, you can switch to subcommand mode, return temporarily to TSO/E command mode, or end the session by exiting the shell.

Shell commands often have options (also known as flags) that you can specify, and they usually take an argument, such as the name of a file or directory. The format for specifying the command begins with the command name, then the option or options, and finally the argument, if any. In Figure 2-23, the following command is shown to list the files and directories:

```
ls -al
```

where **ls** is the command name, **-al** are the options.

This figure shows the panel when the OMVS command is issued from ISPF Option 6. The **ls -al** command lists the files and directories of the user. If the path name is a file, **ls** displays information about the file according to the requested options. If it is a directory, **ls** displays information about the files and subdirectories therein. You can get information about a directory itself using the **-d** option.

If you do not specify any options, **ls** displays only the file names. When **ls** sends output to a pipe or a file, it writes one name per line; when it sends output to the terminal, it uses the **-C** (multicolumn) format.

z/OS UNIX shell (OMVS)

Using the OMVS command to invoke the z/OS shell, you can select options on the OMVS command to customize aspects of the shell interface, such as the function keys.

Once you are working in a shell session, you can switch to subcommand mode, return temporarily to TSO/E command mode, or end the session by exiting the shell.

The shell is a command processor that you use to:

- Invoke shell commands or utilities that request services from the system.
- Write shell scripts using the shell programming language.
- Run shell scripts and C-language programs interactively (in the foreground), in the background, or in batch.

OMVS shell

In the OMVS shell environment, however, things work a little differently. A transaction starts when a command is issued from the terminal. After the command is issued, polling is done to wait for output to return from the command. Every half second, there is a test for output and a test (TGET NOWAIT) for terminal input. This goes on for 20 seconds before the session goes into INPUT mode and does a TGET WAIT for terminal input only. TGET NOWAIT does not end the current transaction unless terminal input is found. If there is no more terminal input for over 20 seconds, the transaction does not end until the TGET WAIT is issued and the session goes into INPUT mode.

2.24 ls -al command - list files in the root

```
U.S. Government Users Restricted Rights -
Use,duplication or disclosure restricted by
GSA ADP Schedule Contract with IBM Corp.

IBM is a registered trademark of the IBM Corp.

ROGERS @ SC74:/u/rogers>cd ..
ROGERS @ SC74:/u>cd ..
ROGERS @ SC74:/>ls -al
total 48
lrwxrwxrwx   1 SYSPROG  SYS1           9 May 17  2009 $SYSNAME -> $SYSNAME/
lrwxrwxrwx   1 SYSPROG  SYS1           9 May 17  2009 $VERSION -> $VERSION/
drwxr-xr-x  18 SYSPROG  SYS1        1504 Oct 19 12:00 .
drwxr-xr-x  18 SYSPROG  SYS1        1504 Oct 19 12:00 ..
drwxr-xr-x   2 SYSPROG  SYS1         256 May 30  2006 ...
-rw-------   1 SYSPROG  SYS1         224 Sep 21  2007 .sh_history
drwx------   2 SYSPROG  SYS1         352 Sep  1 00:05 .ssh
drwxrwxrwx   4 SYSPROG  SYS1         352 Oct 26 11:12 PLEX75
drwxr-xr-x   7 SYSPROG  SYS1         640 Oct  4 12:48 SC74
drwxr-xr-x   6 SYSPROG  SYS1         544 May 30  2006 SC75
 ===>
                                                                    MORE...
```

Figure 2-24 A UNIX command that lists the files in the root

Displaying files and directories

Figure 2-24 shows the result of the **`ls -al`** command to display files and directories from the shell in the root file system. It displays the files and directories and shows the file access allowed for the owner user, the user's group, and other users, which are called permission bits, for example:

```
rwxrwxrwx
```

Permission bits

The default mode (read-write-execute permissions) for a directory created with the **`mkdir`** command is:

- owner=rwx
- group=rwx
- other=rwx

2.25 ISPF edit mode for a z/OS UNIX file

```
  Menu  RefList  RefMode  Utilities  Workstation  Help
 ------------------------------------------------------------------------------
                                Edit Entry Panel
 Command ===>

 ISPF Library:
    Project . . . ROGERS
    Group . . . . ZFS      . . .          . . .          . . .
    Type  . . . . MIGRTOOL
    Member  . . .                  (Blank or pattern for member selection list)

 Other Partitioned, Sequential or VSAM Data Set, or z/OS UNIX file:
    Name . . . . . /u/rogers
+
    Volume Serial . .           (If not cataloged)

 Workstation File:
    File Name  . .
                                          Options
 Initial Macro  . . . .                      Confirm Cancel/Move/Replace
 Profile Name . . . . .                      Mixed Mode
 Format Name  . . . . .                      Edit on Workstation
 Data Set Password  . .                      Preserve VB record length
 Record Length  . . . .                      Edit ASCII data
```

Figure 2-25 Edit a z/OS UNIX file from ISPF Option 2

Editing z/OS UNIX files with ISPF

In z/OS V1R9, the ISPF EDIT, VIEW and BROWSE functions are enhanced to support z/OS UNIX files. This includes support for z/OS UNIX files via Options 1 and 2 on the main ISPF panel, as well as the ISPF EDIT, VIEW, and BROWSE service interfaces. In addition, the Edit Entry, Browse/View Entry, Edit Copy, Edit Move, and Edit Create panels are modified so that the “Other” data set name field is a scrollable field that supports z/OS UNIX files and path names up to 1023 characters in length.

The Edit Option (2), shown in Figure 2-25, allows you to create, display, and change data stored in ISPF libraries, other partitioned or single-volume or multivolume sequential data sets, or now with z/OS V1R9: z/OS UNIX files by specifying a path name.

Type a z/OS UNIX file path name, such as:

```
Name . . . . . /u/jsmith/test/tst1.sh
```

Note: In z/OS V1R8, ISPF was enhanced to process z/OS UNIX files. The z/OS V1R8 support includes the ability to edit, browse, create, delete, rename, copy, and replace z/OS UNIX files. This support is implemented as a directory list utility. The utility is available as Option 17, z/OS UNIX Directory List Utility, under the ISPF Utilities menu (Option 3 from the main ISPF panel).

2.26 ISPF edit mode for a z/OS UNIX file

```
  Menu  Utilities  View  Options  Help
 ------------------------------------------------------------------------------
                          z/OS UNIX Directory List          Row 1 to 20 of 27
Command ===> _____________________________________________ Scroll ===> PAGE
Time zone EST5EDT is used to calculate the displayed date and time values.
Pathname . : /u/rogers

Command  Filename        Message          Type Permission Audit  Ext  Fmat
-------------------------------------------------------------------------------
________ .                                Dir  rwx------  fff---
________ ..                               Dir  r-xr-xr-x  ------
________ .ishell-reflist                  File r--------  fff--- --s- ----
________ .profile                         File rw-------  fff--- --s- ----
________ .sh_history                      File rw-------  fff--- --s- ----
________ apf.file                         File rwxr-xr-x  fff--- a-s- ----
________ cachex                           Dir  rwxr-xr-x  fff---
________ cachey                           Dir  rwxr-xr-x  fff---
________ cache1                           Dir  rwxr-xr-x  fff---
________ cache2                           Dir  rwxr-xr-x  fff---
e_______ cbprnt                           File rwx------  fff--- --s- nl
________ fill                             Dir  rwxr-xr-x  fff---
________ fill3                            Dir  rwxr-xr-x  fff---
________ fill4                            Dir  rwxr-xr-x  fff---
________ fill5                            Dir  rwxr-xr-x  fff---
________ gener                            File rwx------  fff--- --s- nl
________ largezfs                         Dir  rwxr-xr-x  fff---
________ rich.txt                         File rwxr-xr-x  fff--- --s- nl
________ testa                            Dir  rwxr-xr-x  fff---
________ tmp                              Dir  rwxr-xr-x  fff---
```

Figure 2-26 Files and directories with /u/rogers

Select a file to edit

When you press the Enter key on the panel shown in Figure 2-25 on page 60, and you enter the path name without specifying a file, the panel shown in Figure 2-26 is displayed.

With this panel, select a file to edit by placing an **e** next to the file you wish to edit, as shown in Figure 2-26. You then receive an ISPF panel displaying the file in ISPF edit mode, as shown in Figure 2-27.

```
  File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
 ------------------------------------------------------------------------------
EDIT       /u/rogers/gener                                Columns 00001 00072
Command ===> _____________________________________________ Scroll ===> HALF
****** ***************************** Top of Data ******************************
000001 //ROGERSC JOB    (POK,999),MSGCLASS=T,NOTIFY=ROGERS
000002 //PRINT    EXEC PGM=IEBGENER
000003 //SYSPRINT  DD  SYSOUT=T
000004 //SYSUT2    DD  SYSOUT=J
000005 //SYSUT1   DD  DISP=OLD,DSN=ROGERS.JCL.VERS5(GENER)
000006 //TAPE  DD DSN=(ROGERS.A),DISP=(NEW,KEEP),UNIT=3490
000007 //SYSIN     DD  DUMMY
000008 /*
****** ***************************** Bottom of Data ***************************
```

Figure 2-27 File selected to be edited

The file could be selected directly from Option 2 by using the following path name:

```
/u/rogers/cbprnt
```

2.27 Specifying the z/OS UNIX path name

- / (forward slash) Identifies an absolute path name
- ~ (tilde) Represents the path name for the user's home directory
  - ~/test/tst1.sh specifies /u/smith/test/tst1.sh
- . (dot) Represents the path name for the current working directory
  - ./pgma.c specifies /u/proj1/dev/pgma.c when the current working directory is /u/proj1/dev/
- .. (dot dot) Represents the path name for the parent directory of the current working directory
  - ../test/pgma.c specifies /u/proj1/test/pgma.c when the current working directory is /u/proj1/dev/

Figure 2-28 Different ways to specify the z/OS UNIX path name

Specifying the path name

ISPF assumes that a z/OS UNIX path name is entered in this field when the first character is one of the following:

/ (forward slash)	Identifies an absolute path name. /u/rogers as shown in Figure 2-25 on page 60
~ (tilde)	Represents the path name for the user's home directory. Assuming a user's home directory is /u/smith, specifying ~/test/tst1.sh is the equivalent of specifying the absolute path name /u/smith/test/tst1.sh.
. (dot)	Represents the path name for the current working directory. Specifying ./pgma.c is the equivalent of specifying the absolute path name /u/proj1/dev/pgma.c when the current working directory is /u/proj1/dev/.
.. (dot dot)	Represents the path name for the parent directory of the current working directory. Specifying ../test/pgma.c is the equivalent of specifying the absolute path name /u/proj1/test/pgma.c when the current working directory is /u/proj1/dev/.

2.28 ISPF ENQs on z/OS UNIX files

- To avoid possible data corruption, ISPF issues an exclusive ENQ request to prevent two or more users from editing the same z/OS UNIX file at the same time
  - The ENQ major name is SPFEDIT
  - The ENQ minor name is a 12 bytes string compromising these 3 binary integers:
    - The file's i-node number (4 bytes)
    - The file's device number (4 bytes)
    - A sysplex indicator (4 bytes), set to 1 when z/OS UNIX is running with SYSPLEX(YES)

Figure 2-29 IPSF ENQs for protection of files

ISPF ENQ for z/OS UNIX files

To restrict concurrent use of a z/OS UNIX file, ISPF edit issues the following ENQ macro for the file:

```
ENQ SPFEDIT,rname,E,12,SYSTEMS  (z/OS UNIX in sysplex; OextSysplexActv = '
or
ENQ SPFEDIT,rname,E,12,SYSTEM   (z/OS UNIX not in sysplex)
```

Where:

`rname` is 12 bytes in length, comprising the following 4-byte binary integers:

- The inode number of the file
- The device number of the file
- Sysplex indicator:
  - 1 z/OS UNIX is in a sysplex (OextSysplexActv is set ON).
  - 0 z/OS UNIX is not in a sysplex (OextSysplexActv is set OFF).

This enqueue is compatible with the enqueue issued by the z/OS UNIX OEDIT command.

2.29 Support for editing ASCII data

- To activate the ASCII editing facility for a file:
  - Start editing the file as you would for a normal EBCDIC file
- In z/OS V1R9, an ASCII editing facility is provided through the ISPF editor
  - The ASCII editing facility translates ASCII data in a file to EBCDIC before displaying it at the terminal and translates EBCDIC data to ASCII when receiving input from the terminal to write to the file
  - A new SOURCE primary command for the ISPF editor is provided in z/OS V1R9 to control the ASCII editing facility
  - Enter the following command:
    - SOURCE ASCII

Figure 2-30 Editing ASCII data with ISPF

Editing ASCII data

With the increasing use of Java™ and WebSphere® products on z/OS, more and more data is stored in ASCII files on z/OS. One example is XML documents for WebSphere Application Server. Prior to z/OS V1R9, there were few facilities available in z/OS to display and change ASCII data, especially under ISPF. In general, it is necessary for ISPF users to download their ASCII files to a workstation that supports the ASCII character set, edit the files on the workstation, and upload them back to the z/OS system.

In z/OS V1R9, an ASCII editing facility is provided through the ISPF editor. The ASCII editing facility translates ASCII data in a file to EBCDIC before displaying it at the terminal and translates EBCDIC data to ASCII when receiving input from the terminal to write to the file. A new SOURCE primary command for the ISPF editor is provided in z/OS V1R9 to control the ASCII editing facility.

To activate the ASCII editing facility for a file, perform these steps:

1. Start editing the file as you would for a normal EBCDIC file. Then,
2. Enter the command SOURCE ASCII.

Once SOURCE ASCII is issued, the ISPF editor treats the source data as ASCII data and converts it from ASCII to the Coded Character Set ID (CCSID) of the terminal for display purposes. The data in the file remains unchanged. When you input or modify data at the terminal, the ISPF editor translates the data entered from the CCSID of the terminal to ASCII before storing the data in the file.

To change back to the normal editing mode, where the data is not translated from and to ASCII when displaying and receiving input from the terminal, issue the primary EDIT command: RESET SOURCE.

Note: The ASCII editing facility uses the z/OS Unicode Conversion Services to translate the data between ASCII (CCSID 850) and the CCSID supported by the terminal. It is required that z/OS Unicode Conversion Services be installed and the required translations specified to it, in order for the ASCII editing facility to be operable.

2.30 Handling line feed characters

- Many times ASCII files contain line feed characters
  - When such an ASCII file is uploaded from the workstation to a fixed length data set, the data may not be structured correctly according to the line feed characters
- The LF primary command is a new ISPF editor primary command that restructures the data in the file based on the line feed characters
- After starting the editor, issue commands in order to activate the ASCII editing facility to restructure the ASCII file according to the line feed characters it contains:
  - Issue the SOURCE ASCII command
  - Issue the LF command

Figure 2-31 How to handle line feed characters in ASCII data

Handling line feed characters

Many times ASCII files contain line feed characters. When such an ASCII file is uploaded from the workstation to a fixed length data set, the data may not be structured correctly according to the line feed characters. The **LF** primary command is a new ISPF editor primary command that restructures the data in the file based on the line feed characters.

The LF command

ASCII data can contain line feed characters (X'0A'). If the data has been uploaded from another computing platform, the data may not be correctly structured based on the line feed characters. To restructure the data based on the line feed characters, issue the LF command.

Important: There is no reverse process for restructuring the data based on the line feed characters. Consequently, once you have saved the data after an LF command, the change is permanent. Do not enter the LF command more than once against the same file because blanks following line feed characters are interpreted as the leading data of the next record.

SOURCE ASCII and LF example

Using ISPF Option 2 to edit an ASCII file, it is displayed in Figure 2-32 on page 67. Enter the SOURCE ASCII command on the command line to convert the file to EBCDIC.

```
 File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
EDIT       ROGERS.JCL.VERS5(ASCII10) - 01.00              Columns 00001 00072
Command ===> source ascii                                  Scroll ===> HALF
****** ***************************** Top of Data ******************************
000001   äñèë|   ¢|â  ñèë|   äñäë  (ëå<áîá<       (ëåä< ëë ç         êáåñ|+  (
000002 +|èñãß ñèë|                            íëáê ñèë|   & ëëï|êà ñèë|
000003           ¢|â& ê( ë ëä   <                                    &ê|äë
000004 êàáê ñèë|   äñäë ¢ä<
000005                   êí+ñè    áìáä àãçëè êè
000006 &<ñà   äñèë|                                          ëè êè ñ+ñèñ < (ßëñ
000007                              ñ+àáì   ñèë|   äñäë
000008       èá (  ñèë|
****** ***************************** Bottom of Data ***************************
```

Figure 2-32 ASCII file displayed with ISPF edit mode

When you Enter the **`source ascii`** command on the command line, the ASCII text is converted to EBCDIC, as shown in Figure 2-33.

```
 File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
EDIT       ROGERS.JCL.VERS5(ASCII10) - 01.00              Columns 00001 00072
Command ===> LF                                            Scroll ===> HALF
****** ***************************** Top of Data ******************************
000001 //CITS001 JOB 'ITS001-CICS',MSGLEVEL=(1,1),MSGCLASS=H,  //    REGION=0M,
000002 NOTIFY=ITS001,                  //    USER=ITS001,PASSWORD=ITS001
000003         /*JOBPARM S=SC47,L=9999                                //PROCS
000004 RDER=ITS001.CICS.JCL                    //*
000005                 //RUNIT    EXEC DFHSTART,
000006 PLID='(CITS001)',                                //    START=INITIAL,MYSI
000007 ,                       //    INDEX1='ITS001.CICS',
000008 //    TEAM='ITS001'
****** ***************************** Bottom of Data ***************************
```

Figure 2-33 ASCII text converted to EBCDIC

When you use the LF command to restructure the data based on the line feed characters, the text now is aligned as shown in Figure 2-34.

```
 File  Edit  Edit_Settings  Menu  Utilities  Compilers  Test  Help
EDIT       ROGERS.JCL.VERS5(ASCII10) - 01.00              Columns 00001 00072
Command ===>                                               Scroll ===> HALF
****** ***************************** Top of Data ******************************
000001 //CITS001 JOB 'ITS001-CICS',MSGLEVEL=(1,1),MSGCLASS=H,
000002 //    REGION=0M,CLASS=A,NOTIFY=ITS001,
000003 //    USER=ITS001,PASSWORD=ITS001
000004 /*JOBPARM S=SC47,L=9999
000005 //PROCS JCLLIB ORDER=ITS001.CICS.JCL
000006 //*
000007 //RUNIT    EXEC DFHSTART,
000008 //    APPLID='(CITS001)',
000009 //    START=INITIAL,MYSIT=LABSIT,
000010 //    INDEX1='ITS001.CICS',
000011 //    TEAM='ITS001'
****** ***************************** Bottom of Data ***************************
```

Figure 2-34 EBCDIC text following the line feed (LF) command

2.31 Direct login to shell

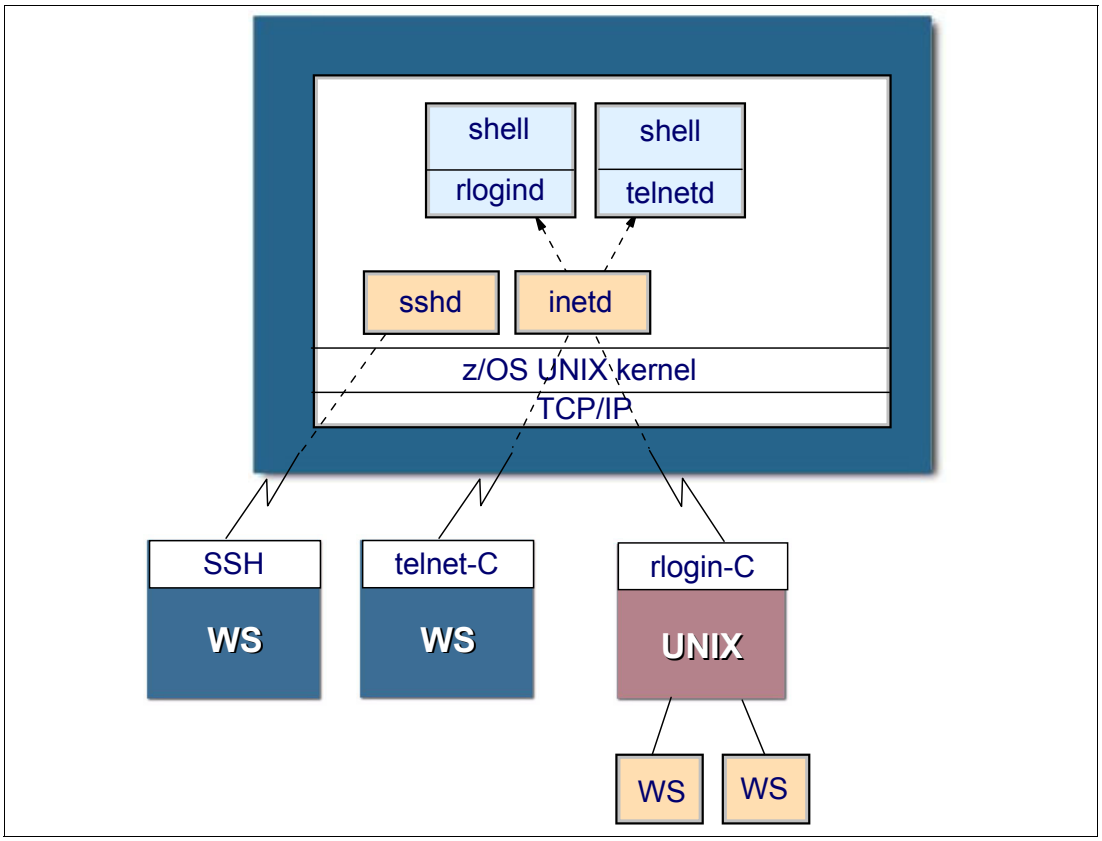


Figure 2-35 Diagram of a login to the shell from a terminal workstation

Entering a shell session from a workstation

Figure 2-35 shows two ways that you can access the z/OS UNIX shells:

- The **`rlogin`** command, which provides an ASCII interface
- The **`telnet`** command, which provides an ASCII interface

When you first log in to one of the z/OS UNIX shells, you are in line mode. Depending on how you access the shell, you might be able to use utilities that require raw mode (such as vi) or run an X-Windows application.

A z/OS UNIX user can log in directly to the z/OS UNIX shell using one of the following solutions:

rlogin When the inetd daemon is set up and active, you can **`rlogin`** to the shell from a workstation that has **`rlogin`** client support and is connected via TCP/IP or Communications Server to the MVS system. To log in, use the **`rlogin`** (remote log in) command syntax supported at your site.

telnet The **`telnet`** support comes with the TCP/IP z/OS UNIX feature. It also uses the inetd daemon, which must be active and set up to recognize and receive the incoming **`telnet`** requests.

ssh OpenSSH is intended to allow a non-z/OS server to communicate securely with a z/OS server using the **`ssh`** protocol in a security-rich environment. Data is encrypted as it passes through the network unaltered, and OpenSSL technology enables both

secure login and secure file transfer. Following is a partial list of utilities that are included in OpenSSH:

- **ssh** - A client program for logging into a z/OS shell. It is an alternative to **`rlogin`**.
- **scp** - For copying files between networks. It is an alternative to **`rcp`**.
- **sftp** - For file transfers over an encrypted **`ssh`** transport. It is an interactive file transfer program similar to **`ftp`**.
- **sshd** - A daemon program for **`ssh`** that listens for connections from clients. The IBM Ported Tools for z/OS implementation of **`sshd`** simultaneously supports both SSH protocol versions 1 and 2.

Terminal support considerations

There are some differences between the asynchronous terminal support (direct shell login) and the 3270-terminal support (OMVS command):

- You cannot switch to TSO/E. However, you can use the TSO shell command to run a TSO/E command from your shell session.
- You cannot use the ISPF editor (this includes the oedit and TSO/E OEDIT commands, which invoke ISPF edit).

When you first log in to one of the z/OS UNIX shells, you are in line mode. Depending on how you access the shell, you might be able to use utilities that require raw mode (such as vi) or run an X-Windows application. The modes are:

Line mode Input is processed after you press <Enter>. Line mode is also called canonical mode.

Raw mode Each character is processed as it is typed. Raw mode is also called non-canonical mode.

Graphical A graphical user interface for X-Windows applications.

The TCP/IP chapters have more details about this. It is mentioned here to give you an idea of the methods available to invoke the shell. Later chapters also have more information about differences between the 3270 and the direct login methods.

2.32 Telnet access to z/OS UNIX

```
Telnet - wtsc47oe
Connect  Edit  Terminal  Help
(C) Copyright Software Development Group, University of Waterloo, 1989.

All Rights Reserved.

U.S. Government users - RESTRICTED RIGHTS - Use, Duplication, or
Disclosure restricted by GSA-ADP schedule contract with IBM Corp.

IBM is a registered trademark of the IBM Corp.

 - - - - - - - - - - - - - - - - - - - - - - - - - - -
 - Improve performance by preventing the propagation -
 - of TSO/E or ISPF STEPLIBs                         -
 - - - - - - - - - - - - - - - - - - - - - - - - - - -
-----------------------------------------------------------------
Set up environment variables for Java and Servlets for OS/390 -
-----------------------------------------------------------------
PATH reset to /usr/lpp/Java/J1.1/bin:/usr/lpp/Printsrv/bin:/bin:.
-----------------------------------------------------------------
JAVA-HOME reset to /usr/lpp/Java/J1.1/
-----------------------------------------------------------------
CLASSPATH reset to .:/usr/lpp/Java/J1.1/lib/classes.zip:/usr/lpp/internet/server
_root/cgi-bin/icsclass.zip:/usr/lpp/internet/server_root/servlets/public:/u/suf/
classes
-----------------------------------------------------------------
ROGERS @ SC47:/>
```

Figure 2-36 Telnet login to the shell screen

Shell session using Telnet

This figure shows the z/OS shell returned after a Telnet login.

From this session, you cannot switch to TSO/E or use the ISPF editor.

Telnet is a terminal emulation protocol that allows end users to log on to remote host applications as though they were directly attached to that host. Telnet protocol requires that the end user have a Telnet client emulating a type of terminal the host application will understand. The client connects to a Telnet server, which communicates with the host application. The Telnet server acts as an interface between the client and host application. A PC can support several clients simultaneously, each with its own connection to any Telnet server.

3

z/OS UNIX System Services preinstallation requirements

This chapter describes the necessary preinstallation requirements to get z/OS UNIX System Services up and running.

It explains in detail the following topics:

- Customization of the root file system
- The difference between ServerPac and CBPDO
- The RACF definitions needed before initializing z/OS UNIX System Services
- Allocation of the root file
- The TSO/E debugging necessary when using the z/OS UNIX ISHELL

© Copyright IBM Corp. 2011. All rights reserved.

3.1 Customization of the root

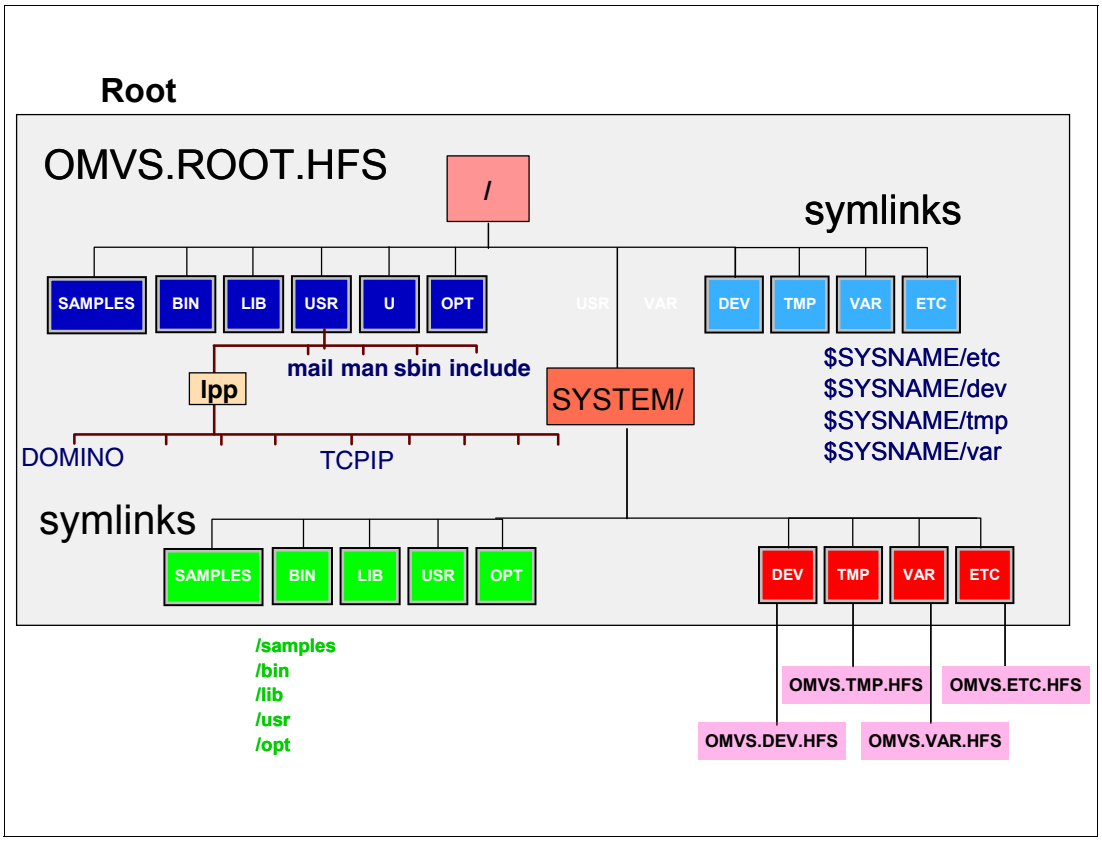


Figure 3-1 The root file system

Root file system

The root file system is the starting point for the overall file system structure. It consists of the root (/) directory, system directories, and files. A system programmer defines the root file system. The system programmer must have an OMVS UID of 0 to allocate, mount, and customize the root directories. It usually contains system-related files and files that belong to a program product.

Directories in the root

The z/OS UNIX root file system directories and their contents are as follows:

bin Contains executable modules, mostly shell commands.

var Contains dynamic data that is used internally by products and by elements and features of z/OS. Any files or directories that are needed are created during execution of code. An example of this is caching data.

dev Contains character-special files that are used when logging into the OMVS shell environment and also during c89 processing. Prior to OS/390 V2R7, these character-special files were created by running the BPXISMKD REXX exec or would be part of your ServerPac order. Beginning with OS/390 V2R7, /dev is shipped empty. The necessary files are created when the system is IPLed, and on a per-demand basis.

etc Contains customization data. Keeping the /etc file system in an HFS data set separate from other file systems allows you to separate your customization data

from IBM's service updates. It also makes migrating to another release easier. After you complete instructions for a ServerPac or CBPDO installation, you will have an /etc file system in its own HFS data set.

tmp Contains temporary data that is used by products and applications. The directory /tmp is created empty, and temporary files are created dynamically by different elements and products. You have the option of mounting a temporary file system (TFS) on /tmp.

lib This directory contains symbolic links to the TCP/IP directory for the X11 Window interface. In a z/OS UNIX system, it is used as a library containing C run-time libraries.

samples Contains examples of files that can be customized and used in the /etc directory.

u Contains user home directories

usr/lpp Contains subdirectories for other applications like Domino®, WebSphere, TCP/IP, DCE, daemons, and so on.

SYSTEM The root file system contains an additional directory, /SYSTEM; existing directories, /etc, /dev, /tmp and /var are converted into symbolic links. These changes, however, are transparent to the user who brings up a single system environment.

Symbolic links (Symlinks)

The presence of symbolic links is transparent to the user. In the illustrations used throughout this redbook, symbolic links are indicated with an arrow.

> **Note:** If the content of the symbolic link begins with $SYSNAME and SYSPLEX is specified as `NO` in the BPXPRMxx member, then $SYSNAME is replaced with /SYSTEM when the symbolic link is resolved.

Customization of the root file system

In Figure 3-1, the root file system contains an additional directory, /SYSTEM; existing directories, /etc, /dev, /tmp, and /var are converted into symbolic links. These changes, however, are transparent to the user who brings up a single system environment. During installation, IBM defines directories in the /etc directory but does not install files there. Because the configuration and customization data in your existing /etc directory might not be correct for the new system, you might need to make changes to the files in your new /etc directory. IBM recommends that you make these changes before the first IPL of the new system. The presence of symbolic links is transparent to the user.

Starting in OS/390 V2R9, place the contents of the /etc, /dev, /tmp, and /var directories for each system into separate HFS data sets if they have not already been set up that way. This task is required for both non-sysplex users and sysplex users. Some utilities that are provided by z/OS UNIX require the use of certain configuration files. Customers are responsible for providing these files if they expect to use these utilities at their installation. IBM provides default configuration files as samples in the /samples directory. Before the first use of any of these utilities, you must copy these IBM-provided samples to the /etc directory (in most cases). You can add further customization of these files to include installation-dependent information.

> **Note:** If the content of the symbolic link begins with $SYSNAME and SYSPLEX is specified as `NO`, then $SYSNAME is replaced with /SYSTEM when the symbolic link is resolved.

3.2 Installing z/OS using ServerPac

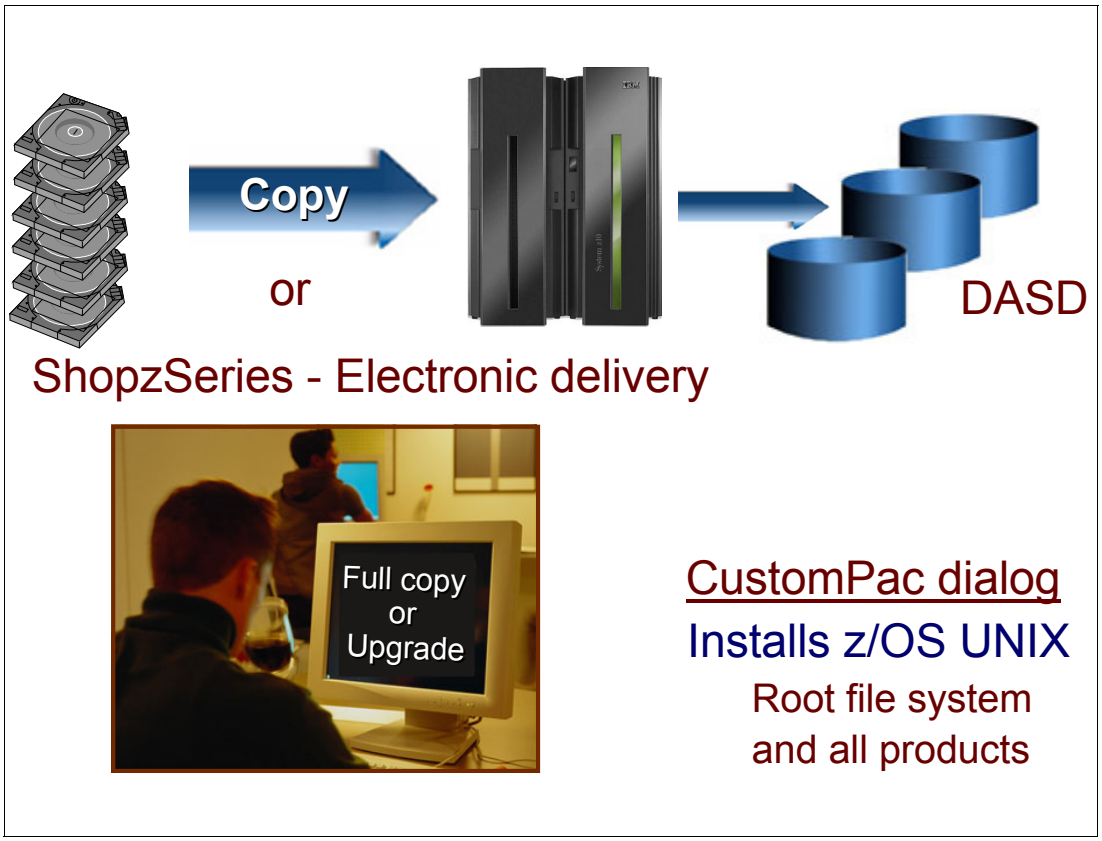


Figure 3-2 Installing z/OS using the ServerPac

ServerPac installation

ServerPac is a software delivery package consisting of products and services for which IBM has performed the SMP/E installation steps and some of the post-SMP/E installation steps. To install the package on your system and complete the installation of the software it includes, you use the CustomPac Installation Dialog, the same dialog that is used for all the CustomPac offerings, including SystemPac® and ProductPac®.

Two types of ServerPac installation are available:

- A full system replacement
- A software upgrade

Note: A full system replacement will provide system software related data sets like Dlib and Target, as well as SMP/E CSI data sets and sample libraries. A software upgrade can only be done if those data sets are available.

ServerPac and the root HFS

For z/OS ServerPac customers, IBM delivers a single-root HFS. This HFS is unloaded when you perform “Establishing UNIX Services” in the ServerPac installation process. Not only does the single-root HFS make cloning of file systems easier, but it also dramatically reduces the number of jobs run by system programmers to establish z/OS UNIX.

Electronic delivery

ShopzSeries is an Internet application you can use to order z/OS software products and service. Using ShopzSeries, you can order corrective and preventive service over the Internet, with delivery over the Internet or by tape. Service with ShopzSeries reduces your research time and effort by using your uploaded SMP/E consolidated software inventory (CSI) to ensure that all applicable service, including reachahead service, for the installed FMIDs in the target zones is selected. The ShopzSeries Web address is:

```
http://www.ibm.com/software/shopzseries
```

3.3 Installing z/OS using CBPDO

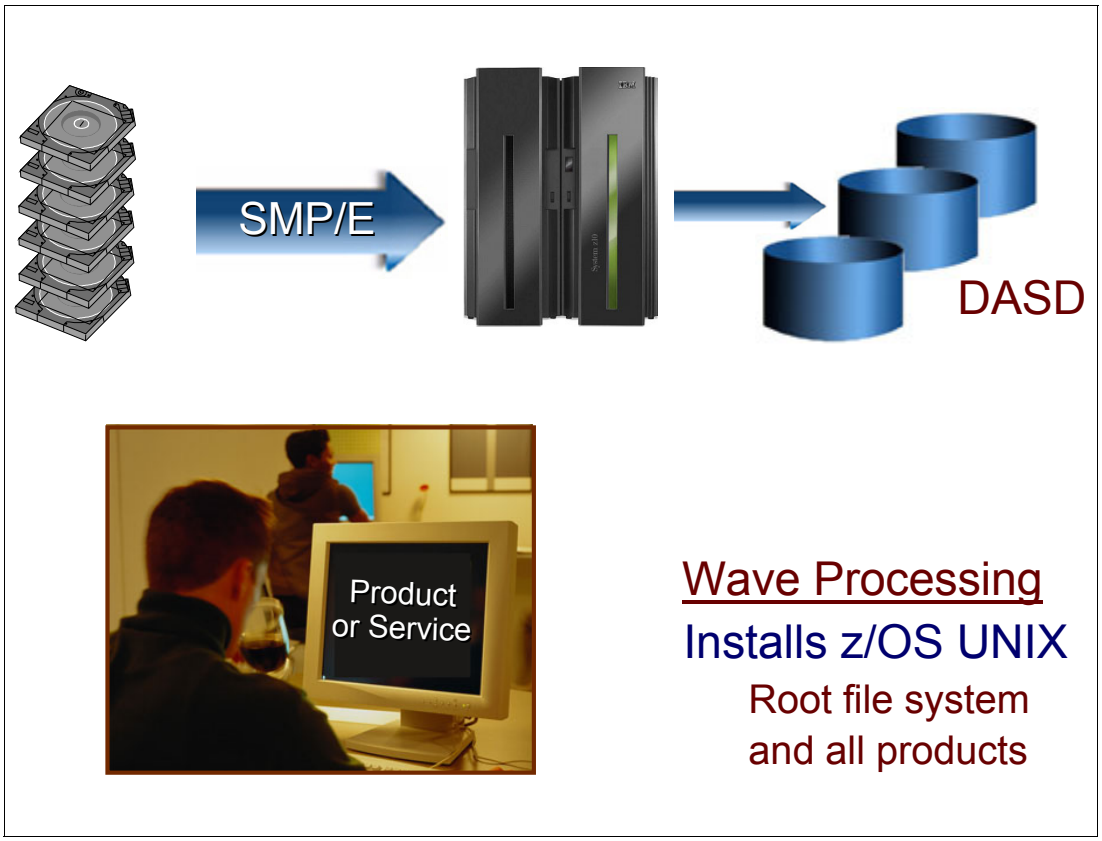


Figure 3-3 Installing z/OS using CBPDO

CBPDO installation

Custom-Built Product Delivery Option (CBPDO) is a software delivery package consisting of uninstalled products and unintegrated service. You must use SMP/E to install the individual z/OS elements and features, and their service, before you can IPL.

The CBPDO installation process is split into separate stages, called waves. These waves group related activities together so that at the completion of each wave, the wave components can be activated (like performing an IPL).

These waves are:

- Wave 0 installs prerequisite FMIDs onto the driving system, such as HLASM and SMP/E.
- Wave 1 installs FMIDs that do not install into HFS.
- Wave 2 installs FMIDs that do install into HFS.
- Wave 3 installs FMIDs for JES2 or JES3.

BPXISHFS member of samplib

For customers who use the CBPDO software delivery package, a sample job called BPXISHFS (found in SYS1.SAMPLIB) is provided. It allocates the root and /etc HFS data sets and then mounts them at a given mount point. This job allocates a file system that is mounted on the /etc directory so that z/OS-delivered code is part of a single HFS, while customized data can be kept separate. This allows for easier cloning of file systems.

3.4 ServerPac and CBPDO

- Provide two HFS data sets:
  - Root file system
    - Files and executables
  - ETC file system
    - Contains only empty directories and symlinks
- Recommendation for separate file system data sets
  - /tmp - /var - /dev - /etc
    - Separate customized data from shipped code

Figure 3-4 ServerPac and the root file system

Installation data sets

For z/OS ServerPac customers, IBM delivers a single-root HFS. This HFS is unloaded when you perform the “Establishing UNIX Services” step in the ServerPac installation process. Not only does the single-root HFS make cloning of file systems easier, but it also dramatically reduces the number of jobs run by system programmers to establish z/OS UNIX.

IBM also delivers a separate HFS data set for /etc.

Performing ServerPac installation requires that you be a superuser with UID(0) or have access to the BPX.SUPERUSER resource in the FACILITY class.

For customers who use the Custom-Built Product Delivery Option (CBPDO) software delivery package, a sample job called BPXISHFS (found in SYS1.SAMPLIB) is provided. It allocates the root and /etc HFS data sets and then mounts them at a given mount point. This job allocates a file system that is mounted on the /etc directory so that z/OS-delivered code is part of a single HFS, while customized data can be kept separate. This allows for easier cloning of file systems.

/etc file system

The /etc file system contains customization data, such as the definition files for the automount facility. The install process creates an empty /etc directory into which customers must copy their existing /etc file system. This directory is similar to SYS1.PARMLIB, but differs in some aspects. For example, the /etc file system cannot be shared between systems, nor can the

/etc file system be concatenated with other directories like SYS1.PARMLIB can. To keep your customization data separated from IBM's service updates and to make migration to another release easier, keep the /etc file system in an HFS data set separate from other file systems. When you complete all instructions for installing z/OS, whether through a ServerPac or CBPDO, you have an /etc file system in its own HFS data set.

To ensure that your customization files are not modified, IBM does not create any files in the /etc file system. IBM does, however, create directories when they are needed. Furthermore, IBM does not ship maintenance into /etc via SMP/E.

Starting in OS/390 V2R9, place the contents of the /etc, /dev, /tmp, and /var directories for each system into separate HFS data sets if they have not already been set up that way. This task is required for both sysplex and non-sysplex users.

3.5 z/OS UNIX System Services installation

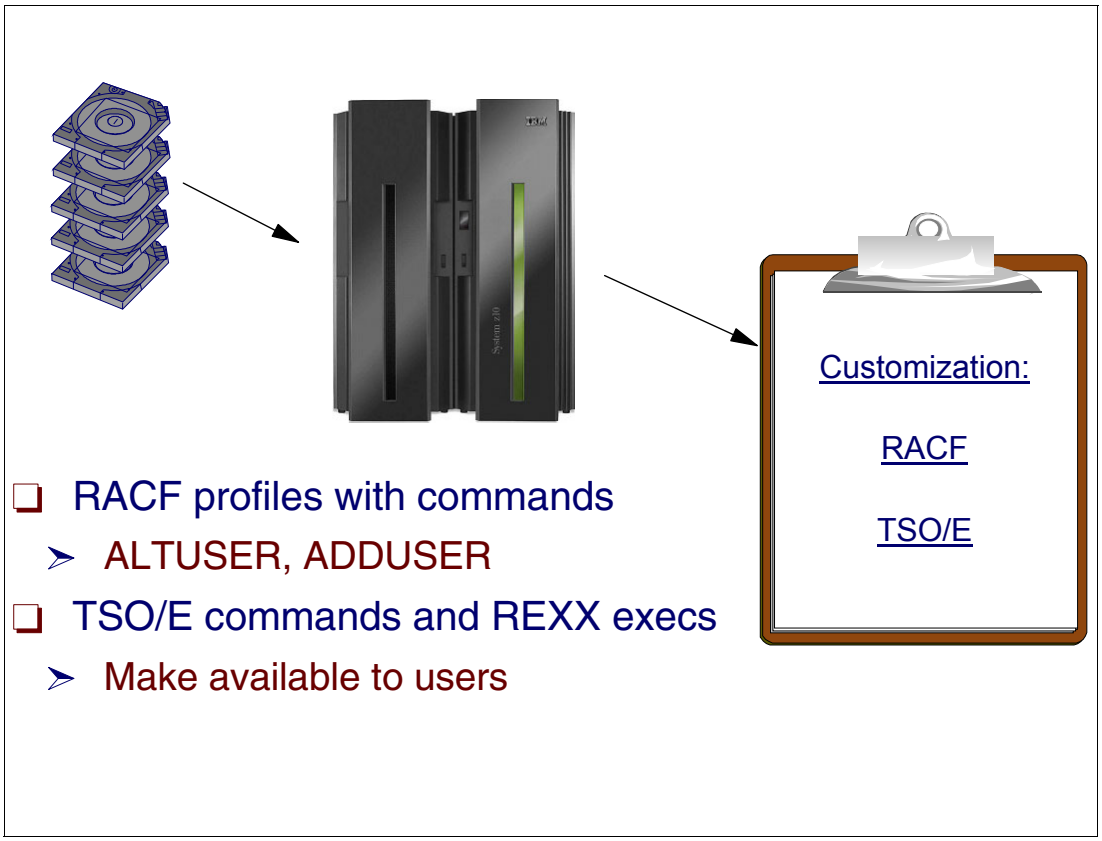


Figure 3-5 Components needed for z/OS UNIX installation

Customizing the z/OS UNIX environment

The ServerPac provides you with full system replacement and software upgrade options. Online panels contain the jobs and present the information you need to proceed through the ServerPac installation. Before you can install and set up z/OS UNIX System Services, the following must be set up in your environment:

- RACF

  The security administrator defines a user by creating a RACF user profile with an ADDUSER command or alters the user profile with an ALTUSER command to make the user a z/OS UNIX user.

- SMS

  System Managed Storage (SMS), which is part of the DFSMSdfp element of z/OS, can be configured, whether you define the kernel in minimum mode or full function mode.

- TSO/E

  To make certain TSO/E commands (such as OEDIT, OBROWSE, and ISHELL) and some shipped REXX execs available to users, concatenate the following target libraries to the appropriate ISPF data definition names (ddnames). The following data sets are for the English panels, messages, and tables:

```
SYS1.SBPXPENU concatenated to ISPPLIB
SYS1.SBPXMENU concatenated to ISPMLIB
SYS1.SBPXTENU concatenated to ISPTLIB
SYS1.SBPXEXEC concatenated to SYSEXEC or SYSPROC
```

3.6 z/OS UNIX security

- Security product required:
  - RACF
  - CA - ACF2
  - CA - Top Secret
  - SAF Exits - security product simulation

Figure 3-6 z/OS UNIX security products

Security products

To provide data and system security, the security administrator and security auditor need to set up and maintain security with the tasks used by z/OS UNIX.

z/OS UNIX provides security mechanisms that work with the security offered by the z/OS system. A security product is required, either RACF or an equivalent security product. This chapter assumes that you are using RACF. If you are using an equivalent security product, you should refer to that product's documentation. If you do not have a security product, you must write SAF exits to simulate all of the functions.

RACF stores z/OS UNIX data in OMVS segments, but Solution Developer products such as CA-ACF2 and CA-Top Secret have also implemented solutions to seamlessly support z/OS UNIX.

3.7 RACF definitions

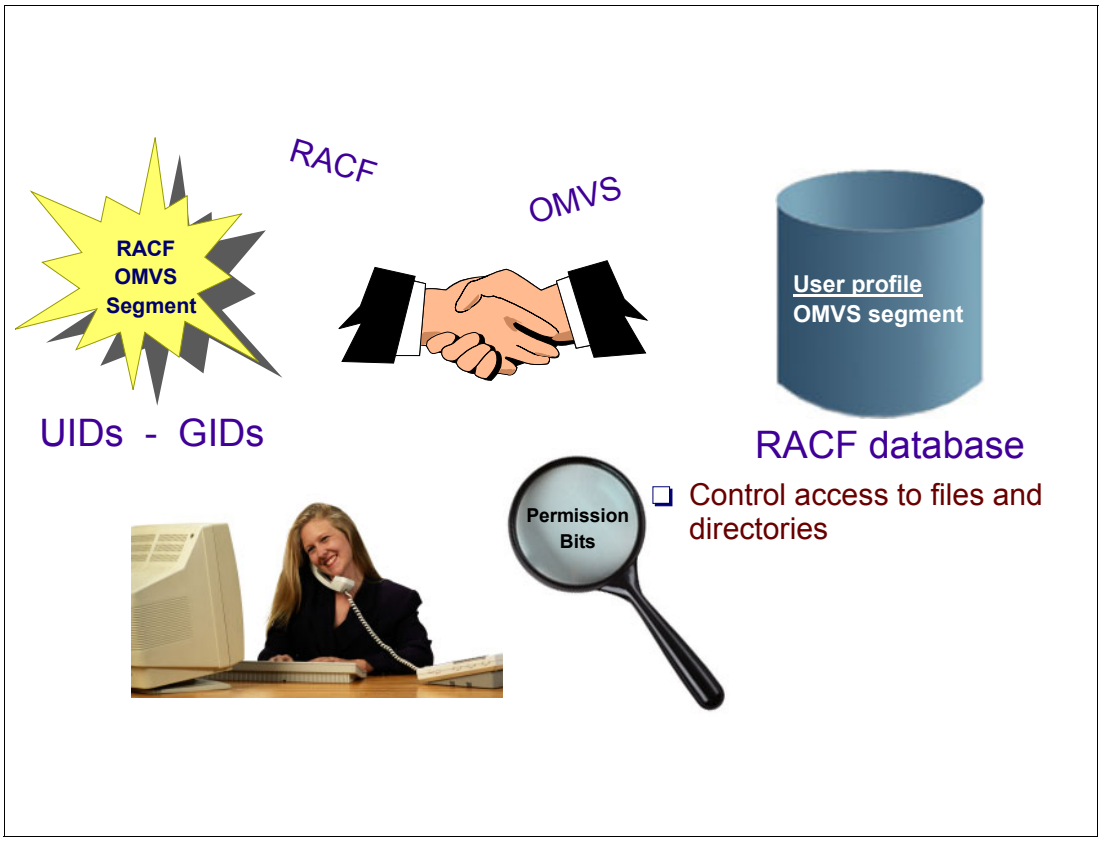


Figure 3-7 RACF definitions for z/OS UNIX

Providing security with RACF

The RACF component of the Security Server authenticates users and verifies whether they are allowed to access certain resources. An equivalent security product can be used to do those tasks.

To provide data and system security, the security administrator and security auditor need to set up and maintain security. z/OS UNIX provides security mechanisms that work with the security offered by the z/OS system. A security product is required, either RACF or an equivalent security product. It is assumed that you are using RACF. If you are using an equivalent security product, you should refer to that product's documentation. If you do not have a security product, you must write SAF exits to simulate all of the functions.

z/OS UNIX provides security mechanisms that work with the security offered by the z/OS system. Before you can install and debug z/OS UNIX System Services, you need to have access to z/OS UNIX System Services data sets and members that are named in directories and files. RACF and other security-related products allow access to the z/OS UNIX environment. This access can be allowed without being a TSO/E user.

Permission bits

RACF uses permission bit settings for files and directories to determine if a request for access to the file or directory can be granted.

Create an OMVS segment

If your user ID should have access to z/OS UNIX, your security administrator has to specify the home directory, the shell program, and a UID in the OMVS segment. In addition, the administrator has to provide a group ID (GID) for any RACF group the user is connected to. If this is done you should be able to access the UNIX shell or work with the ISPF-driven ISHELL. The decision whether you will be able to access directories or files will be made by UNIX security.

OMVS segments can be created using the RACF commands ADDUSER and ALTUSER.

Create user IDs for z/OS UNIX

In addition, your RACF administrator has to provide a user ID that is assigned to the OMVS and BPXOINIT address spaces (Started Procedure). This user ID needs only to have an OMVS segment and should be connected to a RACF group with a GID.

3.8 RACF OMVS segments

User profile

Userid	Default Group	Connect Groups		TSO	DFP	OMVS		
						UID	Home	Program
SMITH	PROG1	PROG1	PROG2	...	...	15	/u/smith	/bin/sh

Group profile

Groupid	Superior Group	Connected Users				OMVS GID
PROG1	PROGR	SMITH	BROWN	...	...	25

Group profile (no OMVS segment)

Groupid	Superior Group	Connected Users			
PROG2	PROGR	SMITH	WHITE	...	...

Figure 3-8 z/OS UNIX RACF OMVS segments

User profiles with OMVS segments

All users and programs that need access to z/OS UNIX System Services must have a RACF user profile defined, with an OMVS segment which has at least a UID specified. A user without a UID cannot access z/OS UNIX.

Note: It is possible for multiple users to have the same UID number specified. However, this is not recommended.

RACF user profile

The RACF user profile has a segment called OMVS for z/OS UNIX support. A user ID must have an OMVS segment defined in order to use z/OS UNIX System Services, for example, access the ISHELL or the shell. This segment has three fields, as follows:

UID A number from 0 to 2147483647 that identifies a z/OS UNIX user. A z/OS UNIX user must have a UID defined.

Home The name of a directory in the file system. This directory is called the home directory. This field is optional.

Program The name of a program. This is the program that will be started for the user when the user begins a z/OS UNIX session. Usually this is the program name for the z/OS UNIX shell. This field is optional.

RACF group profile

The RACF group also has a segment called OMVS to define z/OS UNIX groups. It contains only one field, as follows:

GID A number from 0 to 2147483647 which identifies a z/OS UNIX group.

The home directory is the current directory when a user invokes z/OS UNIX. During z/OS UNIX processing this can be changed temporarily by using the **cd** (change directory) shell command. The command will not change the value in the RACF profile. The directory specified as home directory in the RACF profile must exist (be pre-allocated) before a user can invoke z/OS UNIX. If a home directory is not specified in RACF, the root (/) directory will be used as default.

The example in Figure 3-8 shows a user profile for TSO/E user ID SMITH which is connected to two groups, PROG1 and PROG2. SMITH is defined as a z/OS UNIX user because he has a UID specified. His home directory is /u/smith and he will get into the shell when he issues the OMVS command because the name of the shell, /bin/sh is specified as program name.

The program name in the OMVS segment specifies the name of the first program to start when z/OS UNIX is invoked. Usually this is the name of the z/OS UNIX shell.

3.9 OMVS segment fields

OMVS Segment
UID= 0000000000
HOME= /
PROGRAM= /bin/sh
- -
CPUTIMEMAX= NONE
ASSIZEMAX= NONE
FILEPROCMAX= NONE
PROCUSERMAX= NONE
THREADSMAX= NONE
MMAPAREAMAX= NONE
MEMLIMIT | NOMEMLIMIT
SHMEMAX | NOSHMEMAX

Figure 3-9 Fields in the OMVS segment

Controlling resources for users

You can control the amount of resources consumed by certain z/OS UNIX users by setting individual limits for these users. The resource limits for the majority of z/OS UNIX users are specified in the BPXPRMxx PARMLIB member. These limits apply to all users except those with UID(0), which indicates superuser authority. Instead of assigning superuser authority to application servers and other users so they can exceed BPXPRMxx limits, you can individually set higher limits for these users. Setting user limits allows you to minimize the number of assignments of superuser authority at your installation and reduces your security risk.

Setting limits for z/OS UNIX users

Specify limits for z/OS UNIX users by choosing options on the ADDUSER or ALTUSER commands. The limits are stored in the OMVS segment of the user profile. You can set the following limits in the OMVS user segment:

ASSIZEMAX MAXASSIZE is the maximum region size (in bytes) for an address space. You can set a system-wide limit in BPXPRMxx and then set higher limits for individual users. Use the RACF ADDUSER or ALTUSER command to specify the ASSIZEMAX limit on a per-user basis as follows:

```
ALTUSER userid OMVS(ASSIZEMAX(nnnn)
```

CPUTIMEMAX MAXCPUTIME is the time limit (in seconds) for processes that were created by rlogind and other daemons. You can set a system-wide limit in BPXPRMxx and then set higher limits for individual users. Use the RACF ADDUSER or ALTUSER command to specify the CPUTIMEMAX limit on a per user basis as follows:

```
ALTUSER userid OMVS(CPUTIMEMAX(nnnn))
```

FILEPROCMAX Use MAXFILEPROC to determine the number of character-special files, /dev/fdxx, that a single process can have open concurrently. You can also limit the amount of system resources available to a single user process. You can set a system-wide limit in BPXPRMxx and then set higher limits for individual users. Use the RACF ADDUSER or ALTUSER command to specify the FILEPROCMAX limit on a per user basis as follows:

```
ALTUSER userid OMVS(FILEPROCMAX(nnnn))
```

MMAPAREAMAX For MAXMMAPAREA, you can set a system-wide limit in BPXPRMxx and then set higher limits for individual users. Use the RACF ADDUSER or ALTUSER command to specify the MMAPAREAMAX limit on a per user basis as follows:

```
ALTUSER userid OMVS(MMAPAREAMAX(nnnn))
```

PROCUSERMAX You can set a system-wide limit in BPXPRMxx and then set higher limits for individual users. Use the RACF ADDUSER or ALTUSER command to specify the PROCUSERMAX limit on a per-user basis as follows:

```
ALTUSER userid OMVS(PROCUSERMAX(nnnn))
```

THREADSMAX MAXTHREADS is the maximum number of threads that a single process can have active concurrently. If an application needs to create more than the recommended maximum in SAMPLIB, it must minimize storage allocated below the 16M line by specifying C run-time options. You can set a system-wide limit in BPXPRMxx and then set higher limits for individual users by using the RACF ADDUSER or ALTUSER command to specify the THREADSMAX limit on a per user basis as follows:

```
ALTUSER userid OMVS(THREADSMAX(nnnn))
```

SHMEMAX SHMEMMAX(shared-memory-size) specifies the maximum number of bytes of shared memory that can be allocated by the user. The shared-memory-size value must be numeric 1 - 16777215, followed by the letter M, G, T, or P. The M, G, T or P letter indicates the multiplier to be used. The maximum value is 16383P.

MEMLIMIT MEMLIMIT(nonshared-memory-size) specifies the maximum number of bytes of nonshared memory that can be allocated by the user. The nonshared-memory-size value must be numeric 0 - 16777215, followed by the letter M, G, T, or P. The M, G, T or P letter indicates the multiplier to be used. The maximum value is 16383P.

3.10 UNIX security

- UID = user identifier
  - Number in range 0 - 2,147,483,647
    - But.... 0 - 16,777,215 due to `pax` protocol
  - 0 = Superuser (Root)
- GID = group identifier
  - Number in range 0 - 2,147,483,647
    - But.... 0 - 16,777,215 due to `pax` protocol

/etc/passwd - (Not used by z/OS UNIX)
/etc/group - (Not used by z/OS UNIX)

Figure 3-10 Security on UNIX platforms

Security on z/OS UNIX

UNIX systems incorporate a concept of users and groups similar to that of RACF. A user UNIX identifier (or UID) is a number between 0 and some large number that varies between brands of UNIX. User numbers do not have to be unique and it is possible (though not recommended) for several users to share the same UID, and even be logged on at the same time. UNIX sees these users as being the same entities and they receive the same levels of authorization. A user with UID=0 is what's called a superuser (ROOT on some brands of UNIX). The superuser has unlimited authority within a UNIX environment. For obvious reasons, UID=0 needs to be strictly controlled.

Users are all related to a group. Groups allow authority to be controlled in a more economical way, in that giving access to a group is a lot easier than giving access to a couple of hundred users. A group identifier (or GID) is a number between 0 and some large number that varies between brands of UNIX. Any number of users can share the same GID.

UID and GID numbers

`pax` and **`tar`** are UNIX utilities that perform functions such as PKZIP and DFSMSdss. The z/OS implementation of UNIX allows UID and GID numbers up to 2,147,483,647. The **`pax`** protocol supports UIDs and GIDs up to 8 octal digits. The largest value supported is 77777777 octal or 16 M decimal. To allow the use of **`pax`** and **`tar`**, keep the UIDs and GIDs below this value. It is recommended that the maximum number used should be no more than 77,777,777.

Security on other UNIX platforms

The /etc/passwd file contains basic user attributes. This is an ASCII file that contains an entry for each user. Each entry defines the basic attributes applied to a user. When using the **mkuser** command to add a user to your system, the command updates the /etc/passwd file.

An entry in the /etc/passwd file has the following form:

```
Name:Password: UserID:PrincipleGroup:Gecos: HomeDirectory:Shell
```

Attributes in an entry are separated by a : (colon). For this reason, you should not use a : (colon) in any attribute. The attributes are defined as follows:

Name Specifies the user's login name. The user name must be a unique string of 8 bytes or less. There are a number of restrictions on naming users. See the **mkuser** command for more information.

Password Contains an * (asterisk) indicating an invalid password or an ! (exclamation mark) indicating that the password is in the /etc/security/passwd file. Under normal conditions, the field contains an !. If the field has an * and a password is required for user authentication, the user cannot log in.

UserID Specifies the user's unique numeric ID. This ID is used for discretionary access control. The value is a unique decimal integer.

PrincipleGroup Specifies the user's principal group ID. This must be the numeric ID of a group in the user database or a group defined by a network information service. The value is a unique decimal integer.

Gecos Specifies general information about the user that is not needed by the system, such as an office or phone number. The value is a character string. The Gecos field cannot contain a colon.

HomeDirectory Specifies the full path name of the user's home directory. If the user does not have a defined home directory, the home directory of the guest user is used. The value is a character string.

Shell Specifies the initial program or shell that is executed after a user invokes the **login** or **su** command. If a user does not have a defined shell, /usr/bin/sh, the system shell, is used. The value is a character string that may contain arguments to pass to the initial program.

The /etc/group file contains basic group attributes. This is an ASCII file that contains records for system groups. Each record appears on a single line and has the following format:

```
Name:Password:ID:User1,User2,...,Usern
```

Each attribute must be separated by a colon. Records are separated by new-line characters. The attributes in a record have the following values:

Name Specifies a group name that is unique on the system. The name is a string of 8 bytes or less. See the **mkgroup** command for information on the restrictions for naming groups.

Password Not used. Group administrators are provided instead of group passwords. See the /etc/security/group file for more information.

ID Specifies the group ID. The value is a unique decimal integer string.

User1,User2,...,Usern Identifies a list of one or more users. Separate group member names with commas. Each user must already be defined in the local database configuration files.

3.11 z/OS UNIX superuser

- User having a **UID(0)**
  - Specified in **RACF user profile**
- Can access any file or directory
- Required for many of the customization and administration tasks
- Required to perform some functions like MOUNT
- Superusers can be defined using the following:
  - BPX.SUPERUSER RACF profiles
  - UNIXPRIV class (RACF)

Figure 3-11 z/OS UNIX and superusers

Defining superusers

z/OS UNIX has no predefined numbering scheme for UIDs and GIDs, with the exception of UID=0. UID=0 is a superuser (also known as ROOT on other UNIX platforms). While some functions require a UID of 0, in most cases you can choose among the three ways. When choosing among them, try to minimize the number of “human” user IDs (as opposed to started procedures) set up with UID(0) superuser authority. To summarize the choices, UID(0) gives you access to all UNIX functions and resources, as is true for all UNIX systems.

However, in z/OS, RACF allows certain users to perform specific privileged functions without being defined as UID(0). BPX.SUPERUSER allows you to request that you be given such access, but you do not have the access unless you make the request. And, the UNIXPRIV class allows you to do other privileged functions, such as mounting a file system. Both these definitions are similar to having UID(0) in that, before RACF grants access to a system resource or use of it, the system checks these definitions.

Defined in the user profile

A superuser is a user with a UID of 0 and this UID is in the RACF profile of the user. A superuser can be a started procedure with a trusted or privileged attribute defined in the RACF STARTED Class or added to the RACF Started Procedures Table. The procedure must have a UID. However, when a started procedure is trusted or privileged, RACF does not base authority checking on the UID value; the UID can have any value.

Superuser authority

A superuser can do the following:

- Pass all security checks, so that the superuser can access any file in the file system.
- Perform the mount function.
- Change identity from one UID to another.
- Manage processes.
- Have an unlimited number of processes running concurrently. For a started procedure, this is true only if it has a UID of 0. It is not true for a trusted or privileged process with a different UID.
- Change identity from one UID to another.
- Use **`setrlimit`** to increase any of the system limits for a process.

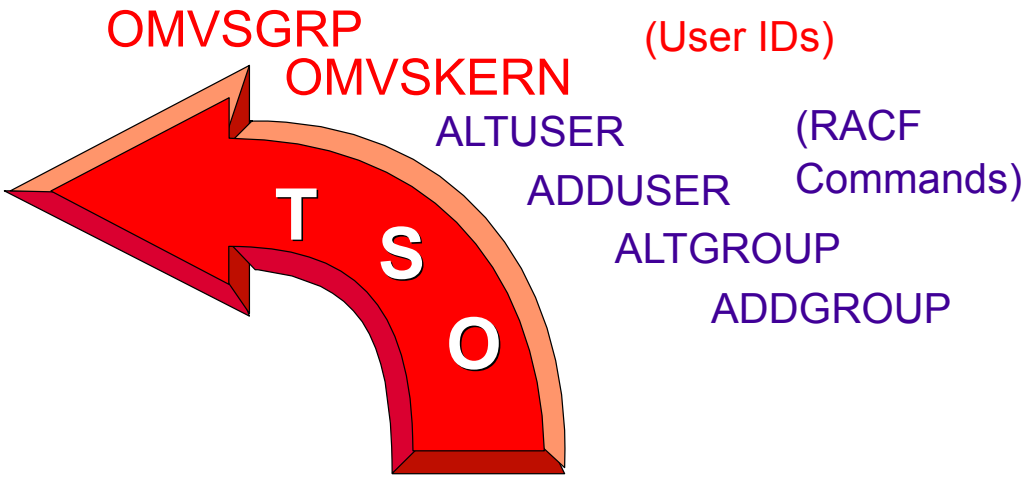
BPX.SUPERUSER profiles

Using the BPX.SUPERUSER resource in the FACILITY class is another way for users to get the authority to do most of the tasks that require superuser authority.

UNIXPRIV class

You can define profiles in the UNIXPRIV class to grant RACF authorization for certain z/OS UNIX privileges. By defining profiles in the UNIXPRIV class, you can specifically grant certain superuser privileges with a high degree of granularity to users who do not have superuser authority. This allows you to minimize the number of assignments of superuser authority at your installation and reduces your security risk.

3.12 RACF commands and user IDs




```
ALTGROUP SYS1 OMVS(GID(0))
ALTUSER KORN OMVS(UID(0) HOME('/') PROGRAM('/bin/sh'))
ADDGROUP OMVSGRP OMVS(GID(1))
ADDUSER OMVSKERN DFLTGRP(OMVSGRP) OMVS(UID(0) HOME('/') PROGRAM('/bin/sh'))
PASSWORD(ONE6PACK).
RDEFINE STARTED OMVS.* STDATA(USER(OMVSKERN) GROUP(OMVSGRP) TRUSTED(YES))
RDEFINE STARTED BPXOINIT.* STDATA(USER(OMVSKERN) GROUP(OMVSGRP) TRUSTED(NO))
```

Figure 3-12 The RACF commands used to define users and groups

Defining user IDs

For our UNIX System Service installation we should provide the following setup in RACF:

- Provide a GID for our default RACF group.
- Define the OMVS RACF segment.
- Define the OMVSKERN user ID and group.
- Define the OMVS and BPXOINIT cataloged procedures.

RACF commands

The following RACF commands can be used to fulfill the UNIX RACF prerequisites:

```
ALTGROUP SYS1 OMVS(GID(0))
ALTUSER KORN OMVS(UID(0) HOME('/') PROGRAM('/bin/sh'))
ADDGROUP OMVSGRP OMVS(GID(1))
ADDUSER OMVSKERN DFLTGRP(OMVSGRP) OMVS(UID(0) HOME('/') PROGRAM('/bin/sh'))
PASSWORD(ONE6PACK).
RDEFINE STARTED OMVS.* STDATA(USER(OMVSKERN) GROUP(OMVSGRP) TRUSTED(YES))
RDEFINE STARTED BPXOINIT.* STDATA(USER(OMVSKERN) GROUP(OMVSGRP) TRUSTED(NO))
```

3.13 RACF commands to define groups

Add OMVS segment to existing group SYSADM:

```
ALTGROUP SYSADM OMVS(GID(0))
```

Define a new group PROG1:

```
ADDGROUP PROG1 OMVS(GID(25))
```

List the OMVS segment of group TTY:

```
LG TTY OMVS
```

Figure 3-13 RACF commands used to define z/OS UNIX groups

RACF commands for defining groups

The ALTGROUP, ADDGROUP, and LISTGRP commands have keywords for administering the OMVS segment.

- Use ALTGROUP (ALG) to modify an existing RACF group, with or without an OMVS segment. The OMVS segment can be added, modified, or deleted.
- Use ADDGROUP (AG) to define a new RACF group, with or without an OMVS segment. The OMVS keyword can be used to define this group as a z/OS UNIX group.
- The LISTGRP (LG) command will display the OMVS segment if the OMVS keyword is specified.

A z/OS UNIX group is a RACF group with an OMVS segment and a GID defined. Figure 3-13 shows examples of how to use the RACF commands to add, change or delete the OMVS segment for a group.

Using `NOGID` removes a GID definition, and `NOOMVS` removes the OMVS segment.

A user can belong to (or be connected to) multiple groups. A z/OS UNIX user must belong to at least one z/OS UNIX group. It is not necessary that all the groups a z/OS UNIX user belongs to are defined as z/OS UNIX groups. Only the groups that are defined as z/OS UNIX groups will be used for authorization checking in z/OS UNIX.

3.14 RACF commands to define users

Add OMVS segment to existing user NEILOC:

```
ALTUSER NEILOC +
  OMVS(UID(0) HOME('/') PROGRAM('/bin/sh'))
```

Define a new user SMITH:

```
ADDUSER SMITH +
  OMVS(UID(15) HOME('/u/smith')PROGRAM('/bin/sh'))
```

List the OMVS segment of user SMITH:

```
LU SMITH OMVS
```

Note: UID=0 is a Superuser

Figure 3-14 RACF commands to define z/OS UNIX users

RACF commands for defining users

The RACF commands ALTUSER, ADDUSER, and LISTUSER have keywords for administering the OMVS segment, as follows:

- Use the ALTUSER (ALU) command to change the definitions for an existing TSO/E user ID. An OMVS segment can be added, modified, or deleted.
- Use the ADDUSER (AU) command to define a new TSO/E user ID with or without an OMVS segment.
- The LISTUSER (LU) command is used for listing the definitions for a TSO/E user ID. When the OMVS keyword is specified, the values specified in the OMVS segment will also be listed.

Note: The values shown are case-sensitive in the definitions for HOME and PROGRAM.

ADDUSER sets up a new user, whereas ALTUSER modifies an existing user. To remove a specification in the OMVS segment, use the keywords NOUID (to remove UID), NOHOME (to remove home directory definition), or NOPROGRAM (to remove program definition). The NOOMVS keyword will remove all the definitions in the OMVS segment. The OMVS keyword must be used on the LISTUSER command if you want the OMVS segment definitions to be displayed.

The z/OS UNIX ISPF shell (ISHELL) provides some menus for administering user IDs. However, before a user can use the ISHELL, he must have a user ID with a UID defined (and a group with a GID defined). This must be done using the RACF commands.

3.15 LU and LG command examples

```
lu smith omvs noracf
  USER=SMITH
  OMVS INFORMATION
  ----------------
  UID= 0000000015
  HOME= /u/smith
  PROGRAM= /bin/sh
  CPUTIMEMAX= NONE
  ASSIZEMAX= NONE
  FILEPROCMAX= NONE
  PROCUSERMAX= NONE
  THREADSMAX= NONE
  MMAPAREAMAX= NONE
lg prog1 omvs noracf
  INFORMATION FOR GROUP PROG1
  OMVS INFORMATION
  ----------------
  GID= 0000000025
```

Figure 3-15 RACF commands to display z/OS UNIX OMVS segments

List user and group commands

Specifying the NORACF option stops the LU and LG commands from displaying general information. You only see the OMVS segment information.

Note that users are only allowed to display their OMVS segments if RACF field-level access has been enabled.

3.16 Define a terminal group name

- Define a RACF group for pseudoterminals
  - Group name - TTY (default)
  - Group id (GID)
    - ADDGROUP TTY OMVS(GID(2))
- TTY group is used by shell programs
- Shell users are assigned group TTY

Figure 3-16 Defining a terminal group name

Terminal group name

Certain shell commands, such as **`mesg`**, **`talk`**, and **`write`** require pseudoterminals to have a group name of TTY. When a user logs in, or issues the OMVS command from TSO/E, the group name associated with these terminals is changed to TTY. As part of your installation, you have to define the group TTY.

Pseudoterminal files

Pseudoterminals (pseudo-TTYs) are used by users and applications to gain access to the shell. A pseudo_TTY is a pair of character special files, a *master file* and a corresponding *slave file*. The master file is used by a networking application such as OMVS or **`rlogin`**. The corresponding slave file is used by the shell or the user's process to read and write terminal data.

When a user enters the TSO/E OMVS command or logs in using **`rlogin`** or **`telnet`** to initialize a shell, the system selects an available pair of these files. The pair represents the connection. The maximum number of pairs is 10000.

3.17 TSO/E support

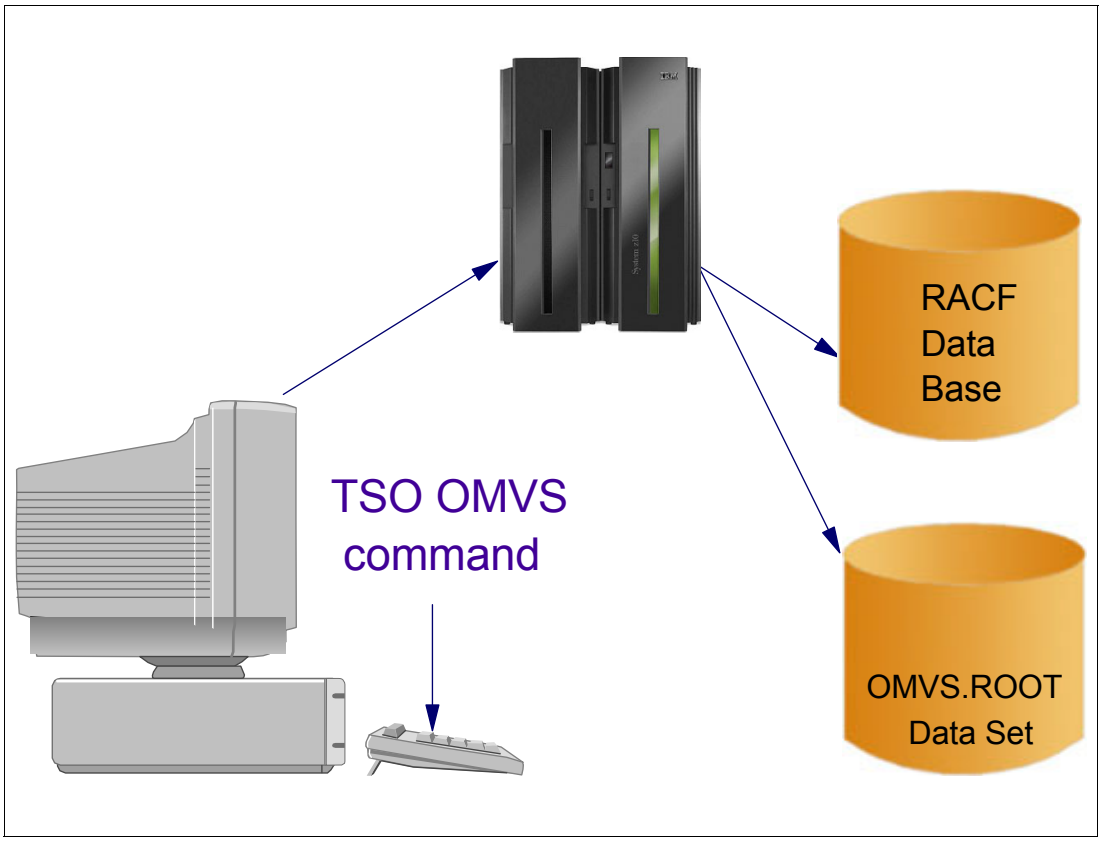


Figure 3-17 TSO/E component used with z/OS UNIX

Using TSO/E for z/OS UNIX

One benefit to users of z/OS UNIX System Services is the z/OS UNIX System Services shell, which is a command processor that you use to:

- Invoke shell commands or utilities that request services from the system (similar to TSO/E).
- Write shell scripts using the shell programming language (similar to REXX).
- Run shell scripts and C-language programs interactively in the foreground, in the background, or in batch (again, similar to REXX).

Once z/OS UNIX is installed, the shell can be invoked through the TSO/E command OMVS. Like the OBROWSE, OEDIT, and ISHELL commands, the OMVS command can also be added to an ISPF selection panel, or entered as a TSO/E command.

The z/OS UNIX ISPF Shell or ISHELL is an ISPF panel interface that you can use instead of TSO/E commands or shell commands to perform tasks against z/OS UNIX user IDs and HFSs.

If you are more comfortable using the ISPF editor and ISPF pull-down menus, the ISHELL is the tool for you.

3.18 User access to TSO/E commands

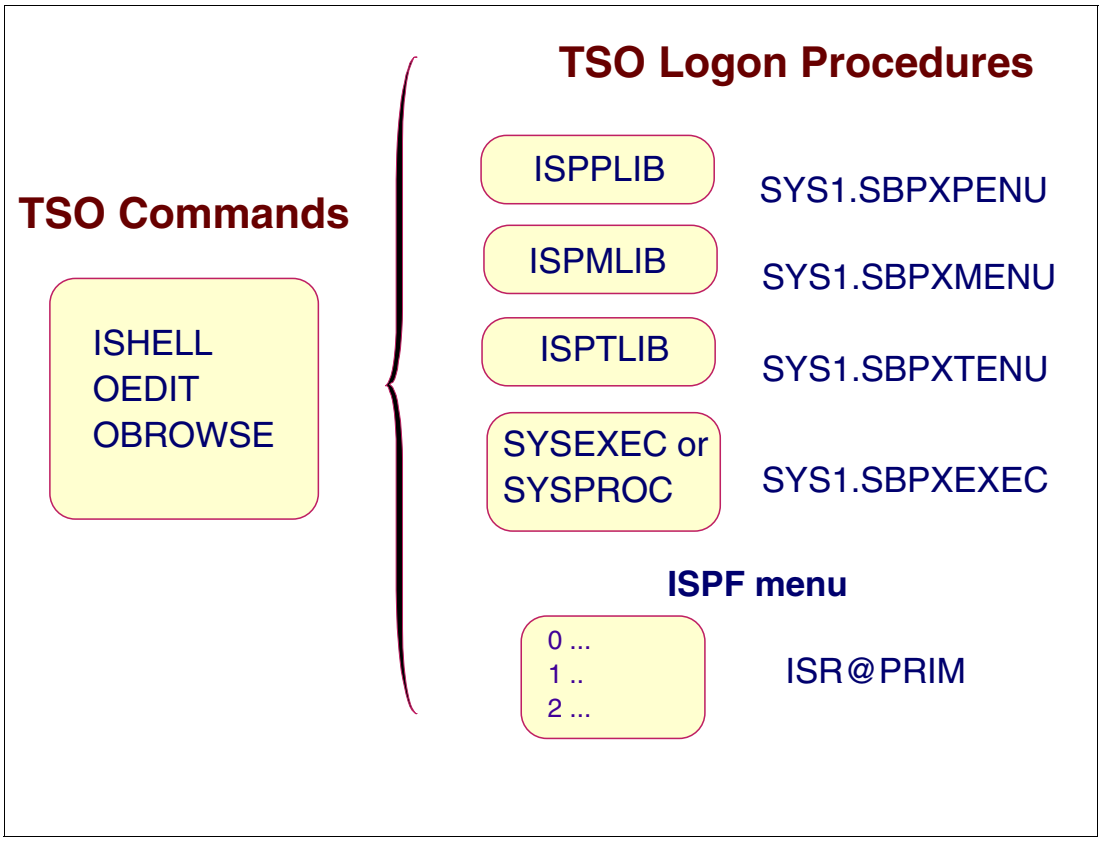


Figure 3-18 Using TSO/E commands with z/OS UNIX

Customizing the TSO/E environment for z/OS UNIX

In order to make the TSO/E commands OEDIT, OBROWSE, and ISHELL (ISPF Shell) available to users, the following data sets should be concatenated to the appropriate ISPF DD names (in the TSO/E logon procedure, or CLIST/REXX EXEC if used):

- SYS1.SBPXPENU (concatenated to ISPPLIB)
- SYS1.SBPXMENU (concatenated to ISPMLIB)
- SYS1.SBPXTENU (concatenated to ISPTLIB)
- SYS1.SBPXEXEC (concatenated to SYSEXEC or SYSPROC)

These TSO/E commands can be invoked from a TSO/E command line, but it is more convenient for users if they are added as an option to an ISPF menu. They can be added to the ISPF/PDF primary option menu (ISR@PRIM) or any other menu the installation prefers to use.

TSO/E users need to be defined with an OMVS segment in RACF before they can use the ISHELL, OEDIT, or OBROWSE commands.

4

z/OS UNIX System Services installation

This chapter provides information about installing z/OS UNIX System Services, and how to avoid or fix installation problems. It explains how to:

- Install z/OS UNIX System Services
- Allocate HFS data sets
- Mount HFS data sets
- Restore file systems

© Copyright IBM Corp. 2011. All rights reserved.

4.1 z/OS UNIX PARMLIB - PROCLIB members

- SYS1.PARMLIB
  - One or more BPXPRMxx members
  - IEASYSxx member to define BPXPRMxx members
  - PROGxx member to define language environment
  - COFVLFxx member for VLF definitions
  - COMMNDxx member for start of VLF
- SYS1.PROCLIB
  - Procedures required for z/OS UNIX address spaces
    - OMVS - Main z/OS UNIX address space
    - BPXOINIT - z/OS UNIX initilaization address space
    - BPXAS - Address spaces for all UNIX processes

Figure 4-1 PARMLIB and PROCLIB members for z/OS UNIX

Customization of SYS1.PARMLIB

SYS1.PARMLIB has several members in which some customization needs to be done. z/OS UNIX requires definitions in one or more BPXPRMxx members that are defined in the IEASYSxx member.

Use the PROGxx member to define language environment run-time libraries.

The COFVLFxx member is used for VLF definitions to cache UIDs and GIDs for performance when RACF does authorization checks for access to files and directories. The COMMNDxx member can be used to specify the start command for VLF.

Customization of SYS1.PROCLIB

The following procedures are required in SYS1.PROCLIB for the z/OS UNIX address spaces:

- OMVS - The main z/OS UNIX address space
- BPXOINIT - The z/OS UNIX initialization address space
- BPXAS - For address spaces for all UNIX processes

4.2 IEASYSxx PARMLIB member

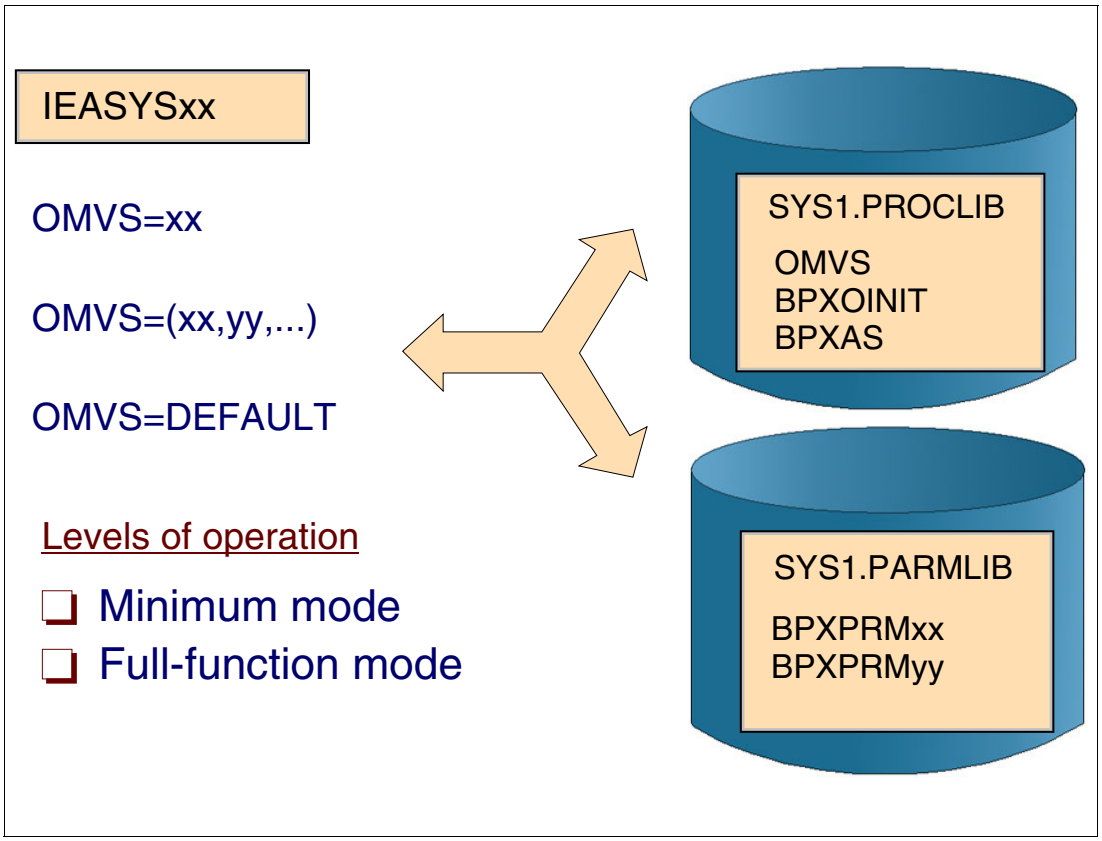


Figure 4-2 PARMLIB settings needed to start z/OS UNIX

PARMLIB definitions to start z/OS UNIX

The initial PARMLIB settings for the z/OS UNIX kernel are pointed to by the OMVS parameter in the IEASYSxx PARMLIB member. The OMVS parameter in the IEASYSxx PARMLIB member lets you specify one or more BPXPRMxx PARMLIB members to be used to specify the initial PARMLIB settings for the kernel. If you do not specify the OMVS parameter, or if you specify OMVS=DEFAULT, the kernel is started in a minimum configuration mode with all BPXPRMxx PARMLIB statements taking their default values.

OMVS may also be left out or coded as DEFAULT. This allows the z/OS UNIX kernel to start in a minimum configuration. All BPXPRMxx values will take their default values and a temporary root file system will be set up in memory.

Note: The **`start`** and **`stop`** commands for the z/OS UNIX kernel are no longer supported.

Activation of kernel services is available in two modes:

- Minimum mode
- Full-function mode

You can set up kernel services in either minimum mode or full function mode. If you want to use any z/OS UNIX service, TCP/IP, or other functions that require the kernel services, you will need to use full function mode; otherwise, you can use minimum mode. In order to apply service to the HFS, you need at least one system that can run in full function mode.

4.3 z/OS UNIX minimum mode

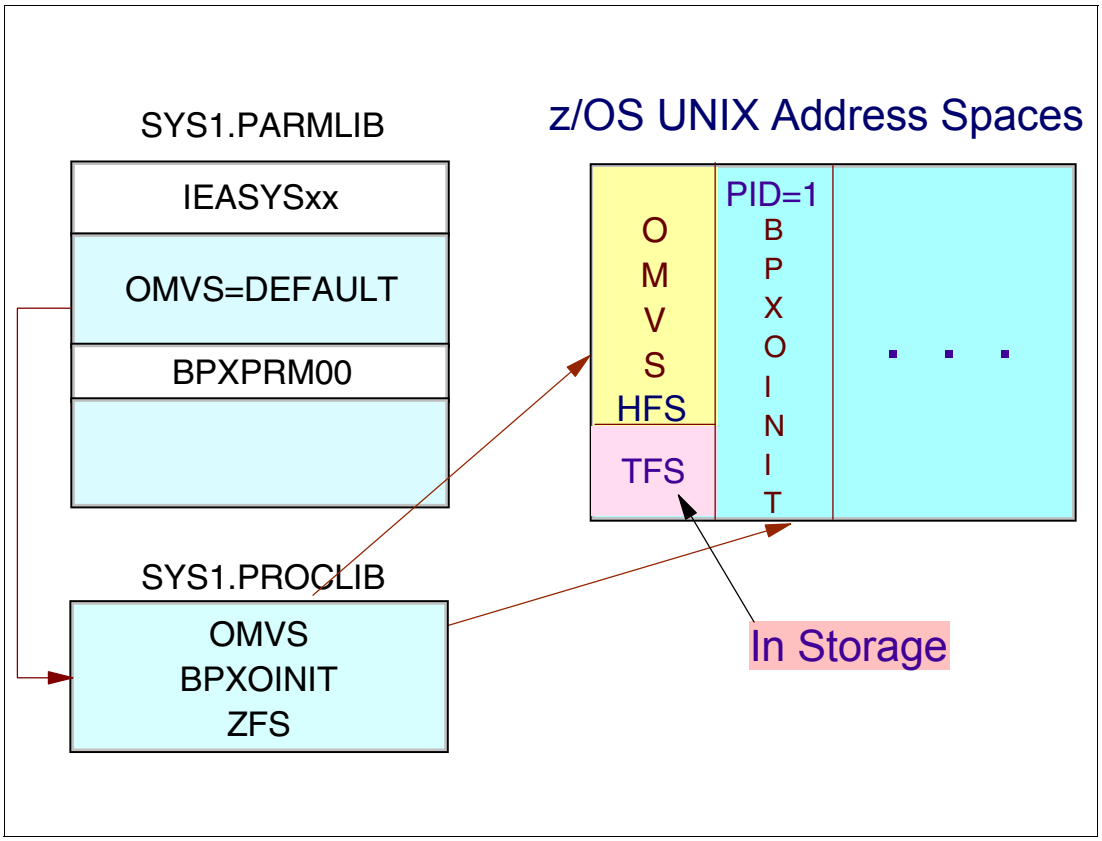


Figure 4-3 z/OS UNIX running in minimum mode

Using minimum mode

Minimum mode is intended for installations that do not intend to use z/OS UNIX System Services, but which allow the IPL process to complete. In this mode many services are available to programs. Some that require further customization, such as a fork(), will fail.

OMVS=DEFAULT

If you do not specify OMVS= in the IEASYSxx PARMLIB member or if you specify OMVS=DEFAULT, then kernel services start up in minimum mode when the system is IPLed. This mode is intended for installations that do not plan to use the kernel services.

In minimum mode:

- Many services are available, but some functions such as TCP/IP sockets that require other system customization may not work.
- TCP/IP sockets (AF_INET) are not available.
- A temporary file system (TFS) is used. A TFS is an in-storage file system, hence no physical DASD data set needs to exist or be mounted.
- In minimum mode, the kernel cannot support some functions, such as the z/OS shell and TCP/IP.

Temporary file system (TFS)

In minimum mode, a temporary file system named SYSROOT is used as the root file system. It is initialized and primed with a minimum set of files and directories. Any data written to this file system is not written to DASD.

The temporary file system does not have any executables; that is, the shell will not be available. Do not install z/OS UNIX System Services Application Services in the TFS, because data will not be written to DASD.

A temporary file system (kept in memory) is created for the root and the required directories (`/bin`, `/etc`, `/tmp`, `/dev`, and `/dev/null`) are built. There are no executables in this file system.

4.4 Minimum mode: TFS

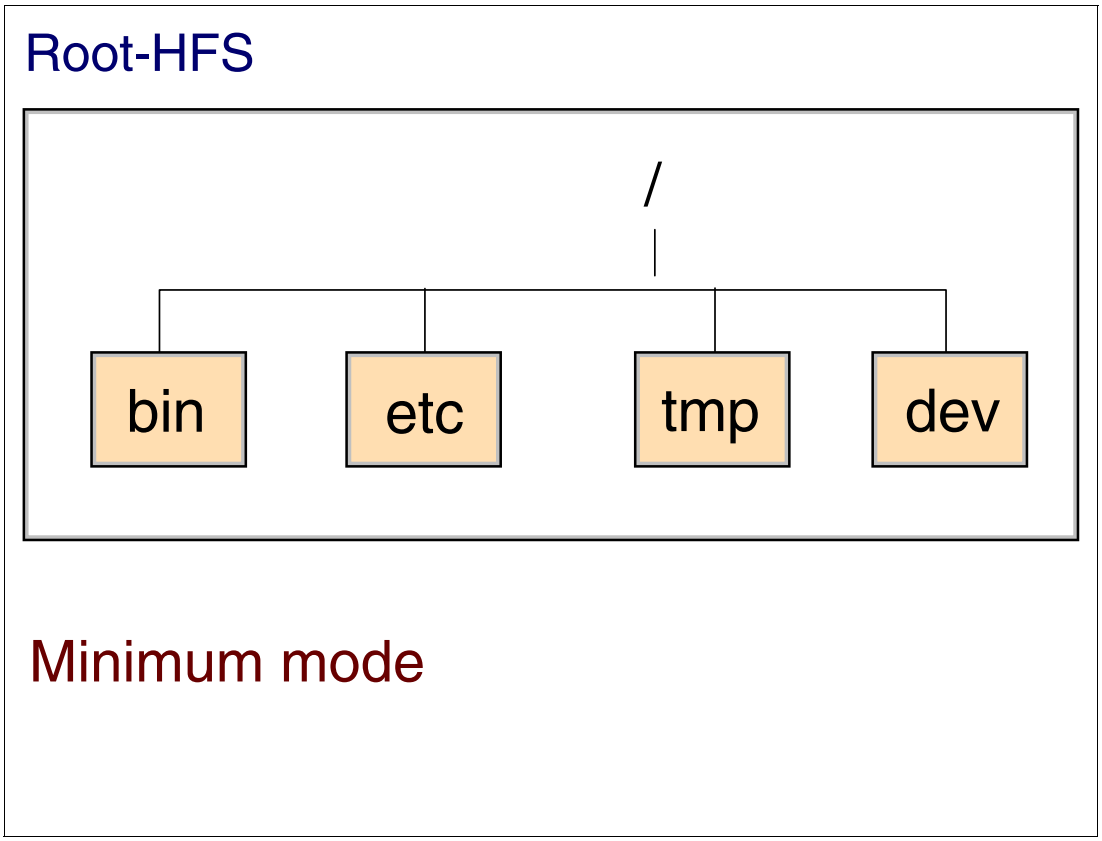


Figure 4-4 The TFS in minimum mode

TFS in minimum mode

A temporary file system is an in-memory file system which delivers high-speed I/O. If the kernel is started in minimum setup mode, the TFS is automatically mounted. The system is in minimum mode when:

- OMVS=DEFAULT.
- There is no OMVS statement in the IEASYSxx parmlib member and no BPXPRMxx parmlib member.

Files and directories

The temporary file system is used as the root file system. It is initialized and primed with a minimum set of files and directories, specifically as follows:

```
/ (root directory)
/bin directory
/etc directory
/tmp directory
/dev directory
/dev/null file
```

TFS usage

There are no executables in the temporary file system (that is, you will not find the Shell and Utilities). Do not attempt to install z/OS UNIX System Services Application Services in the TFS, since no data will be written to DASD. Because the TFS is a temporary file system,

unmounting it causes all data stored in the file system to be discarded. If, after an unmount, you mount another TFS, that file system has only a dot (.) and a dot-dot (..) and nothing else. Any data written to this file system is not written to DASD.

The TFS is automatically mounted if the kernel is started in minimum mode. In this environment, the TFS is the in-storage file system and defaults to the root file system. Because it is an in-storage file system, the TFS delivers high-speed I/O.

If the TFS is to be used in other situations, it is made available by mounting. You can mount it with a MOUNT or ROOT statement in the BPXPRMxx member or by means of one of the supported mount commands.

If you are using kernel services in full function mode, you may want to mount a TFS over `/tmp`. If you do, it can be used as a high-speed file system for temporary files. However, you cannot recover vi files if the system goes down because vi writes temporary files to TMPDIR (`/tmp` by default). To recover these files, you can use the **`exrecover`** command, which automatically runs from `/etc/rc`.

4.5 z/OS UNIX full-function mode

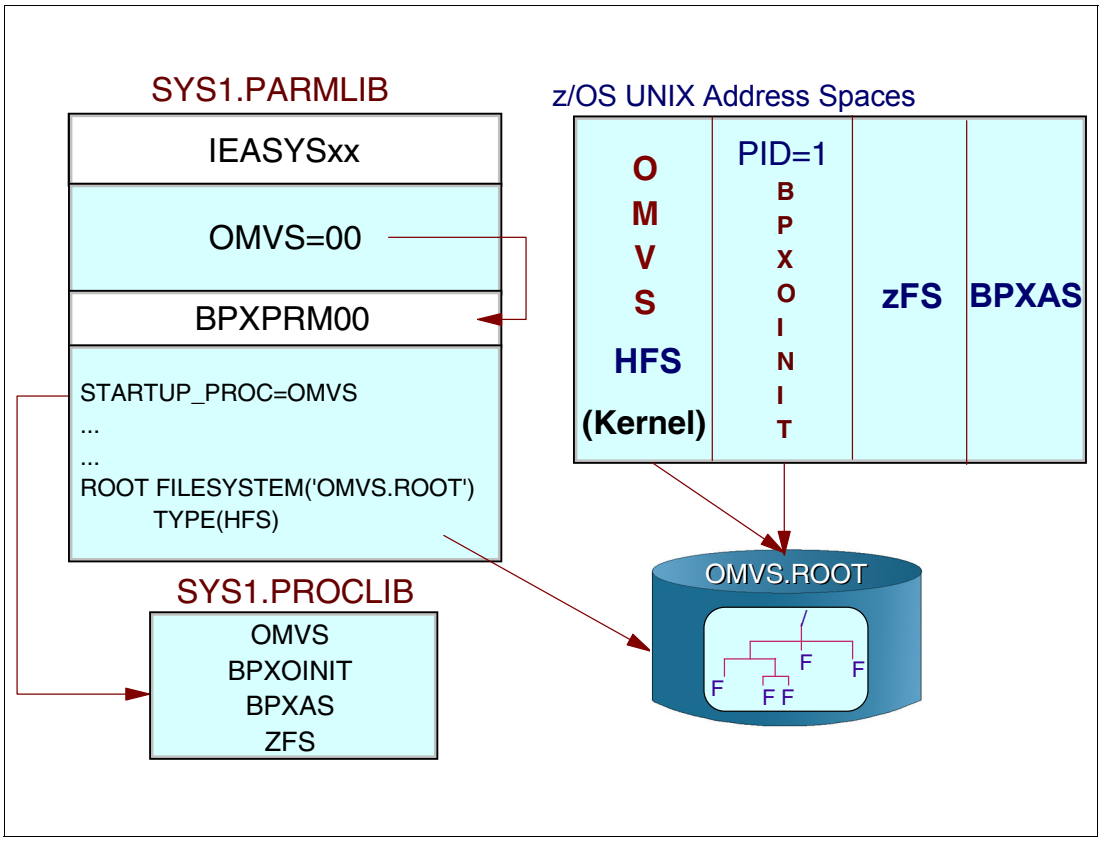


Figure 4-5 z/OS UNIX full function mode

Using full-function mode

Full-function mode is started at IPL time when the OMVS parameter in the IEASYSxx PARMLIB member points to one or more BPXPRMxx parmlib members.

There must be a root HFS built and defined in BPXPRMxx for OMVS to initialize correctly, and SMS, WLM, and RACF customization should be completed.

BPXPRMnn

STARTUP_PROC is a 1 to 8 character name of a started procedure JCL that initializes the z/OS UNIX kernel. The default is OMVS.

The ROOT statement defines and mounts the root file system. In this example:

- HFS is the TYPE of the file system.
- OMVS.ROOT is the file system, which is the name of an already defined HFS data set.
- The root file system has a default of read/write mode.

If an active BPXPRMxx PARMLIB member specifies `FILESYTYPE TYPE(HFS)`, or `FILESYSTYPE TYPE(ZFS)`, then the kernel services start up in full-function mode when the system is IPLed. To use the full function, you need to:

- Set up BPXPRMxx PARMLIB definitions
- Set up DFSMS
- Set up the hierarchical file systems

- Set up the security product definitions for z/OS UNIX
- Set up the users' access and their individual file systems

DFSMS manages the HFS data sets that contain the file systems. zFS data sets are VSAM linear data sets defined in aggregates.

Note: APAR OW35441 now gives you the ability to allocate PDSE and HFS data sets on unmanaged (non-SMS) volumes, if running DFSMS 1.4 or DFSMS 1.5.

BPXOINIT address space

BPXOINIT is the started procedure that runs the initialization process. BPXOINIT is also the jobname of the initialization process. BPXOINIT is shipped in SYS1.PROCLIB.

At system IPL time, kernel services are started automatically. If the OMVS parameter in the IEASYSxx PARMLIB parameter is not specified, the kernel services are started in minimum mode. If the OMVS parameter specifies one or more BPXPRMxx PARMLIB members, they are all used to configure the kernel services when the system is IPLed.

The BPXOINIT address space has two categories of functions:

1. It behaves as PID(1) of a typical UNIX system. This is the parent of /etc/rc, and it inherits orphaned children so that their processes get cleaned up using normal code in the kernel. This task is also the parent of any MVS address space that is dubbed and not created by fork or spawn. Therefore, TSO/E commands, batch jobs, and so forth have a parent PID of 1.
2. Certain functions that the kernel performs need to be able to make normal kernel calls. This address space is used for these activities, for example, mmap() and user ID alias processing.

zFS address space

zFS runs as a z/OS UNIX colony address space. There must be an entry in a BPXPRMxx parmlib member for ZFS and the ZFS PROC must be available. zFS is started by z/OS UNIX based on the FILESYSTYPE statement for ZFS in the BPXPRMxx parmlib member.

zFS can be started at IPL if the BPXPRMxx parmlib member is in the IEASYSxx parmlib member's OMVS=(xx,yy) list. It can also be started later by using the SETOMVS RESET=(xx) operator command.

4.6 z/OS HFS root

- Single HFS structure containing:
  - Root directory
  - All z/OS related products
- Simplifies installation and management of HFS
  - Two jobs to complete installation - ServerPac
    - ALLOCDS - Allocates HFS data sets
    - RESTFS - Restores the root file system - and - mounts new root and additional file systems
- RESTFS job
  - Requires submitting user ID to be superuser
    - BPX.SUPERUSER profile - Effective UID is 0
    - BPX.FILEATTR.APF - BPX.FILEATTR.PROGCTL

Figure 4-6 z/OS UNIX root install using ServerPac

ServerPac jobs

The z/OS ServerPac provides two jobs to complete the installation of the root HFS. They create a single HFS data set structure that contains the following:

- The root directory.
- All z/OS-related products placed into the ServerPac order that require z/OS UNIX System Services.

ALLOCDS job

The ALLOCDS job allocates and catalogs your new target system data sets. For z/OS orders, this job also allocates the couple data sets. Many of these data sets have unique considerations for serialization, availability, security, backup and recovery. For these considerations, see the planning and implementation books for the products you are installing. If you are installing your order on SMS-managed target system volumes, you likely performed SMS setup work earlier in the installation dialog, such as defining storage classes.

RESTFS job

The restore of the archive file is directed to the Install Directory on the driving system. Earlier in the installation process, you specified the Install Directory variable (FA00PQ04) during the Define Installation Variables function of the dialog. By default, the Install Directory is /Service. This value cannot be blank (the job will fail). If you specified a directory other than /Service, ensure that the name you use does not exceed 20 characters in length. In a multilevel directory path, the lowest level from the root is automatically created. Higher directories are

not created. If your driving system's UNIX file system is mounted read-only, the specified directory must already exist.

Understand that the target system's UNIX file system will be mounted to the Install directory during the restore. As a result, the real names of the target system's UNIX file system data sets must be unique in your environment.

This job mounts the new Root file system, and creates mount points for and mounts the additional file systems. The BPXPRMFS member of CPAC.PARMLIB is updated to reflect the file system structure.

The RESTFS job then restores those files into the Root file system.

RESTFS job requirements

The RESTFS job (and corresponding subsystem jobs) previously required the submitting user ID to have UID(0) set in its OMVS segment. Access to the BPX.SUPERUSER profile in the FACILITY class would provide the required authority without the need for UID(0). The ServerPac now sets the effective UID to zero when the user ID has access to BPX.SUPERUSER. This eliminates the need for the user to have to execute the restore of the ServerPac to have UID(0) authority.

In addition, regardless of your installation method, the user ID must be permitted read access to the FACILITY classes BPX.FILEATTR.APF and BPX.FILEATTR.PROGCTL (or BPX.FILEATTR.* if you choose to use a generic facility for both facility classes).

In order to define these FACILITY classes, you can use the following commands (which are also provided in SYS1.SAMPLIB):

```
RDEFINE FACILITY BPX.FILEATTR.APF UACC(NONE)
RDEFINE FACILITY BPX.FILEATTR.PROGCTL UACC(NONE)
SETROPTS CLASSACT(FACILITY)
SETROPTS RACLIST(FACILITY)
PERMIT BPX.FILEATTR.APF CLASS(FACILITY) ID(your_userid) ACCESS(READ)
PERMIT BPX.FILEATTR.PROGCTL CLASS(FACILITY) ID(your_userid) ACCESS(READ)
SETROPTS RACLIST(FACILITY) REFRESH
```

Note: This does not affect the pax utility. You still required UID(0) in order to execute the pax command outside of the ServerPac processing. This change is for ServerPac processing only.

RESTFS is historically the number one source of service calls. To help eliminate some of the problems, logic checking for common setup problems, new error messages, and more specific error messages for better diagnosis have been added. RESTFS calls pax and BPXISETS from within the exec to eliminate bypass checking and job elimination. A new status message indicator shows progress of the job.

4.7 zFS with z/OS V1R7

- zFS is the preferred file system
- HFS may no longer be supported in future releases and you will have to migrate the remaining HFS file systems to zFS
- Continued use of HFS will be discouraged
- Effort is needed to migrate data from HFS file systems to zFS file systems
- Tools to help with migration are needed
  - BPXWH2Z tool - ISPF based in z/OS V1R7

Figure 4-7 zFS as the preferred file system

HFS migration to zFS

In z/OS V1R7, zFS is the preferred file system. Continued use of HFS will be discouraged. Some effort is needed to migrate data from HFS file systems to zFS file systems. A migration tool is implemented in z/OS V1R7 to help with migration as needed.

The HFS physical file system is stabilized and will be removed in a future z/OS release.

Migration tool BPXWH2Z

You can use the ISPF-based tool, BPXWH2Z, to migrate HFS file systems to zFS file systems. It has a panel interface that enables you to alter the space allocation, placement, SMS classes, and data set names. With this tool, you can:

- Migrate HFS file systems (both mounted and unmounted) to zFS file systems. If the HFS being migrated is mounted, the tool automatically unmounts it and then mounts the new zFS file system on its current mount point.
- Define zFS aggregates by default to be approximately the same size as the HFS. The new allocation size can also be increased or decreased.
- Have the migration run in a TSO foreground or a UNIX background.

4.8 HFS or zFS data sets

- You can choose whether each filesystem is to be an HFS or a zFS:
  - On the data set attributes panels
  - With the new CHange DSNTYPE command
- New data set types in the installation dialog:
  - HFS, zFS, PDS, PDSE, SEQ, and VSAM

```
>---+--- CHANGE ---+---+--- DSNTYPE ---+---+--- PDS PDSE ---+---<
  |          | |          | |           |
  +--- CH -------+   +--- TYPE ------+   +--- PDSE PDS ---+
              |          | |           |
              +--- T ---------+   +--- HFS ZFS ----+
                            |           |
                            +--- ZFS HFS ----+
```

Figure 4-8 Choose between HFS or zFS data sets

ServerPac changes in z/OS V1R5

Beginning with z/OS V1R5, you can type over the Data set type field with the new value, as follows:

HFS To change a zFS data set to an HFS data set

PDS To change a PDSE data set to a PDS data set (if the PDSE was originally shipped as a PDS data set)

PDSE To change a PDS data set to a PDSE data set

ZFS To change an HFS data set to a zFS data set

For data sets that are not eligible to be changed, the Data Set Type field is read-only. The dialog does not, for example, allow you to change your order's PDSE data sets to PDS data sets, because they contain members that cannot be loaded into a PDS.

CHANGE DSNTYPE command

You can use the CHANGE DSNTYPE command to convert your shipped order's data sets to a different format:

- PDS data sets to PDSE data sets.
- PDSE data sets to PDS data sets. (This can be done only for data sets that were originally PDS data sets.)
- HFS data sets to zFS data sets.
- zFS data sets to HFS data sets.

4.9 Set data set type

```
CPPP605D  ------------ Modify System Layout ( R0150026 ) ------------------
COMMAND ==> _

Data Set Modification - Attributes


       Data set Name ==> OMVSZ15.RL000006.OMVS.ETC
        Shipped       :  OMVS.ETC

       Placement      :  C       (DLIB, Target, Catalog, or User-Defined)

       Data Set Type ==> HFS     (HFS, PDS, PDSE, SEQ, VSAM, or ZFS)
        Shipped       :  HFS

       SMS-Managed   ==> NO      (Yes or No)
       SMS-Eligible   :  YES
       SMS-Required   :  NO

      Logical Volume ==> HLB002  Shipped    :  CAT001
     Physical Volume  :  T6Z5H1
       Storage Class  :
```

Figure 4-9 Choose a data set type

ServerPac changes in z/OS V1R5

The dialog provides you with a general-purpose view and change facility for working with groups of data sets in your configuration. With this facility, you can:

- Display groups of data sets in your configuration by various attributes
- Make changes to all or some of the data sets in a displayed group
- Save lists of groups you display

When you select a data set from a data set list panel through line command A, the panel shown in Figure 4-9 is displayed. This panel shows the attributes of a specific data set.

You can then choose to make your file systems' data sets either HFS or zFS.

4.10 Choosing zFS

- HFS data set vs. VSAM LDS (aggregate) for zFS
- zFS aggregate must be preformatted
- Different FILESYSTYPEs used in BPXPRMxx
  - FILESYSTYPE TYPE(ZFS) ENTRYPOINT(IOEFSCM) ASNAME(ZFS)
- Different operands on MOUNT command
- Different security system setup
- CICS, DB2, and IMS ServerPacs assume that any necessary zFS setup has been done

Figure 4-10 Selecting zFS as the data set type

ServerPac processing for zFS

Using a zFS in place of an HFS is transparent to applications. The same utilities work for both (like pax). This support begins with z/OS V1R5.

But there are some differences, including the following:

- An HFS data set is a non-VSAM data set, while a zFS lives in a VSAM Linear Data Set (LDS).
- zFS data sets must be preformatted with the IOEAGFMT program.
- Because different programs are used by the system to process HFS and zFS files, a different FILESYSTYPE statement is required in BPXPRMxx.
- To point to the new FILESYSTYPE statement, a different operand is required on MOUNT.
- Additional security system setup is needed to use zFSs.

Note: Additional ECSA is also required to use zFS data sets.

4.11 ServerPac changes if using zFS

- ALLOCDS job formats zFS data sets you define
- RACFDRV and RACFTGT Jobs in z/OS orders:
  - Add group for DFS setup
  - Add user ID for ZFS address space
- RESTFS Job:
  - Uses appropriate TYPE on MOUNT commands depending on filesystem type
  - Puts appropriate TYPE on MOUNT parameters in BPXPRMFS
  - Add FILESYSTYPE for zFS to BPXPRMFS if needed

Figure 4-11 ServerPac changes for zFS support

ServerPac jobs for zFS processing

All zFS data sets must be formatted with IOEAGFMT. A step to format them is added to the ALLOCDS job when there is a zFS in the configuration. This support begins with z/OS V1R5.

The RACF setup for zFS is done unconditionally, on the assumption that, sooner or later, you will want to use zFS.

RESTFS job

The RESTFS job adds a FILESYSTYPE statement for zFS to BPXPRMFS if needed. It is not added unconditionally because there is a significant amount of free ECSA required to start the zFS address space.

Before using a zFS, you should review your virtual storage map and make sure there is at least 70 MB of ECSA available in addition to the ECSA normally required to run your system.

To determine the amount of available ECSA, you can use an RMF Virtual Storage Report that covers your peak workload periods (vastly preferred!) or format the GDA control block in IPCS and subtract GDA_ECSA_ALLOC from GDAECSAS to see how much ECSA is free at that particular time (and guess at the likely variations).

4.12 UNIX utilities: TSO/E commands

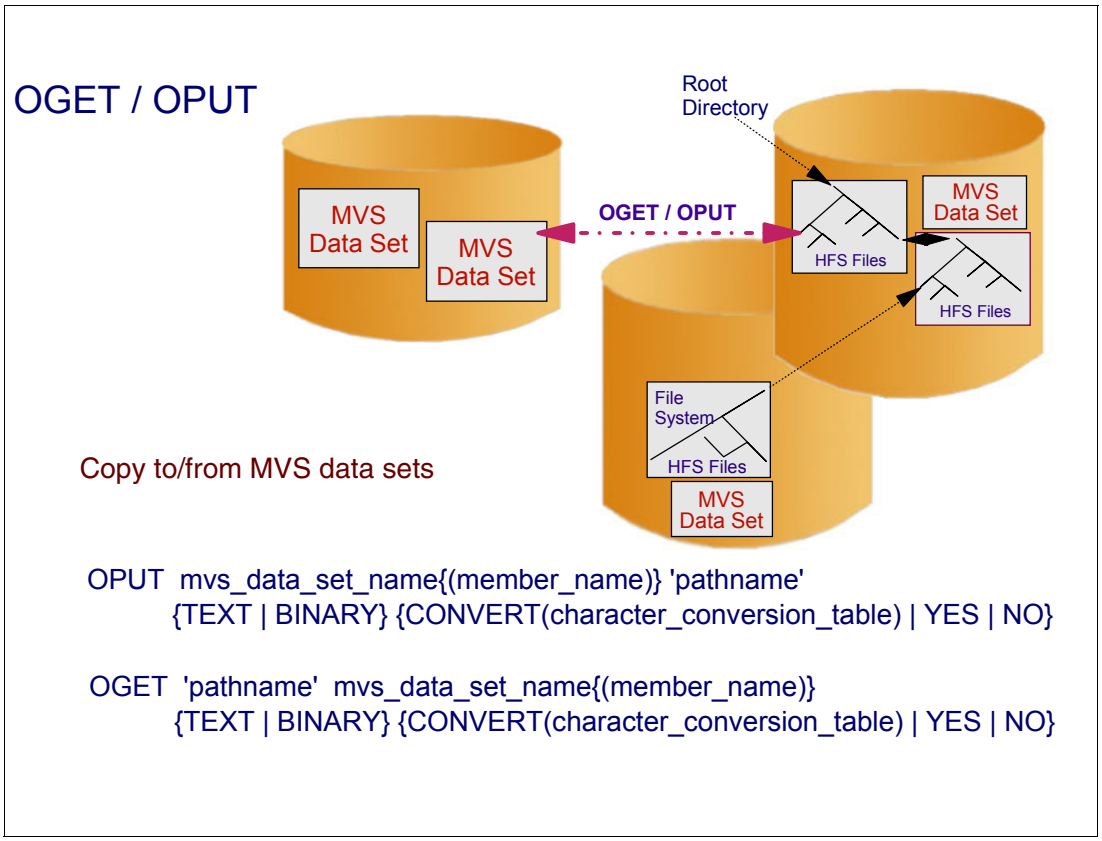


Figure 4-12 TSO/E OGET and OPUT commands

TSO commands and utilities

You can use the OGET command to copy a z/OS UNIX system file to the following:

- A member of an MVS partitioned data set (PDS or PDSE)
- An MVS sequential data set
- You can convert the data from code page 1047 to code page IBM-037 or ASCII while it is being copied. Do not use the CONVERT option when copying files that contain doublebyte data. This option is used for singlebyte data only, not for doublebyte data.

You can use the OPUT command to:

- Copy a member of an MVS partitioned data set (PDS or PDSE) to a file.
- Copy an MVS sequential data set to a file.
- You can also convert the data from code page IBM-037 or ASCII to code page IBM-1047.

pathname

Specifies the pathname of the file that is being copied to a data set. This operand is required. The pathname is:

- A relative or absolute pathname. A relative pathname is relative to the working directory of the TSO/E session (usually the HOME directory). Therefore, you should usually specify an absolute pathname.
- Up to 1023 characters long and enclosed in single quotes.

- In uppercase or lowercase characters, which are not changed by the system.

mvs_data_set_name | mvs_data_set_name(member_name)

Specifies the name of an MVS sequential data set or an MVS partitioned data set member to receive the file that is being copied. One of these two operands is required. The data set name is:

- A fully qualified name that is enclosed in single quotes, or an unqualified name
- Up to 44 characters long and is converted to uppercase letters by the system

BINARY | TEXT

Specifies whether the file being copied contains binary data or text.

- BINARY

  Specifies that the file being copied contains binary data. When you specify BINARY, OGET operates without any consideration for <newline> characters or the special characteristics of DBCS data. For example, doublebyte characters might be split between MVS data set records, or a "shift-out" state might span records.

- TEXT

  Specifies that the file being copied contains text. This is the default. If you are using a DBCS-supported terminal, you should use TEXT. It is assumed that doublebyte data in the file system includes the <newline> character in order to delineate line boundaries. Data within these lines that are delineated by <newline> characters must begin and end in the "shift-in" state.

4.13 UNIX commands to move and copy data

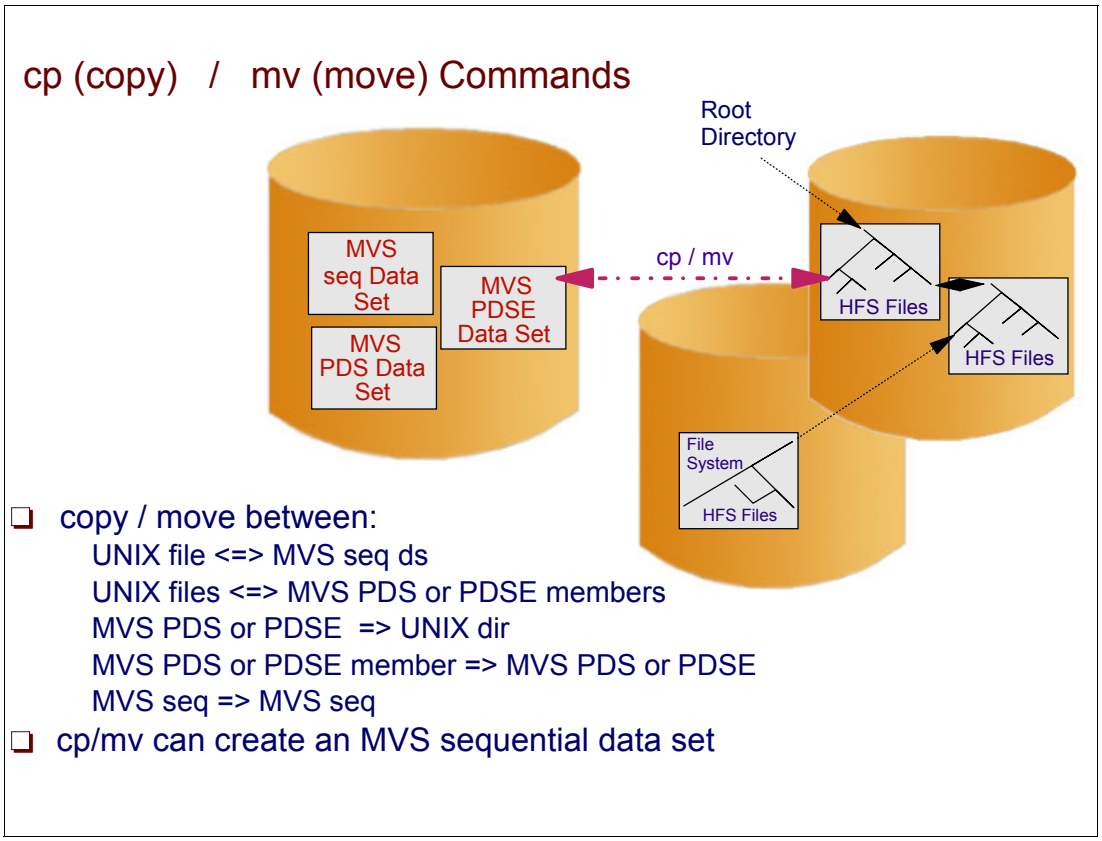


Figure 4-13 Commands to move and copy data

UNIX commands for copy and move

You can use **cp** and **mv** UNIX commands to copy or move files to and from MVS data sets. If you specify more than one file to be copied, the target (last pathname on the command line) must be either a directory or a partitioned data set. If the target is an MVS partitioned data set, the source cannot be a UNIX directory.

The cp and mv commands

You can copy and move:

- One file to another file in the working directory
- One file to a new file on another directory
- A set of directories and files to another place in your file system
- A UNIX file to an MVS data set
- An MVS data set to a file system
- An MVS data set to an MVS data set

To **cp** or **mv** to a PDS or PDSE, the data set must be allocated before the copy or move. You may *not* **cp/mv** a partitioned data set to another partitioned data set and you may *not* **cp/mv** a directory to a partitioned data set (you may use **dir /*** to accomplish copying/moving all files from the directory to a partitioned data set).

4.14 The pax and tar utilities

- Used to back up and restore files
- Example to back up a complete directory (OMVS shell)

```
pax -wf  archive_file  directory_name
pax -wf /u/sysprog/source.pax /u/sysprog/source
tso "OGET  'archive_file'  'DATA_SET_NAME' BINARY"
tso "oget '/u/sysprog/source.pax' 'SYSPROG.PAX' binary"
```

- pax and tar support for MVS data sets
  - When creating portable archives or extracting files
    - Archives can be MVS sequential or partitioned data sets
  - MVS data set are only supported as the archive file

```
pax -wf "//'user.lib(archive)'" *
pax -rf "//'user.lib(archive)'"
```

Figure 4-14 pax and tar utilities

Read and write files with the pax utility

Use **pax** to read, write and list archive files. An archive file is a single file containing one or more files and directories. Archive files can be HFS files or MVS data sets. A file stored inside an archive is called a component file; similarly, a directory stored inside an archive is called a component directory.

Note: MVS data sets cannot be specified for component files. Included with each component file and directory is recorded information such as owner and group name, permission bits, file attributes, and modification time.

You can therefore use a single archive file to transfer a directory structure from one machine to another, or to back up or restore groups of files and directories.

Archives created by pax are interchangeable with those created with the **tar** utility. Both utilities can read and create archives in the default format of the other (USTAR for pax and TAR for tar). Archives are generally named with suffixes such as .pax or .tar (or pax.Z and tar.Z for compressed files), but this is not required.

The pax utility

The **pax** utility can read and write files in CPIO ASCII format, CPIO binary format, TAR format, or USTAR format. It can read files that were written using **tar**, **cpio**, or **pax** itself. How it handles filename length and preservation of link information across the backup and restore process depends on the format you select: If you select CPIO, it behaves like the **cpio** command; and if you select TAR, it behaves like the **tar** command.

Figure 4-14 shows a **pax** example to back up a complete directory, including the subdirectories and their contents, into a data set.

In the example:

directory_name	The name of the directory you want to archive
archive_file	An absolute pathname
DATA_SET_NAME	A fully qualified data set name

The pax command

The **pax** command creates an archive file with the specified name in the current working directory.

For the **pax** command:

- The -w option writes to the archive file.
- The -f option lets you specify the name of the archive file.

The OGET command copies the archive file into the specified MVS data set.

Reading archive files

The **pax** and **tar** utilities can read an archive file directly from an MVS data set. Use the **pax** or **tar** shell command to restore the directory or file system from the archive file; all the component files are restored from the archive file. **pax** and **tar** package HFS directory trees into a single file called an “archive.” Archives are always treated as binary. Only partitioned data sets in undefined format may store an executable. Archives can be moved to other systems or directories. “Extracting files” means moving files from the archive into HFS files. This avoids OPUT of archives from MVS data sets to UNIX, and OGET of archives from UNIX to MVS data sets.

Note: When writing to a PDS member, the PDS must already exist; **pax** will automatically overwrite an existing archive.

The tar utility

tar is both a file format in the form of a type of archive bit stream and the name of the program used to handle such files. The format was created in the early days of UNIX and standardized by POSIX.1-1988 and later POSIX.1-2001. It was initially developed to be written directly to sequential I/O devices for tape backup purposes, but it is now commonly used to collect many files into one larger file, for distribution or archiving, while preserving file system information such as user and group permissions, dates, and directory structures.

tar reads and writes headers in either the original TAR format from UNIX systems or the USTAR format defined by the POSIX 1003.1 standard.

Note: The **pax** command reads and writes headers in any of the **tar** formats.

4.15 ServerPac z/OS UNIX installation

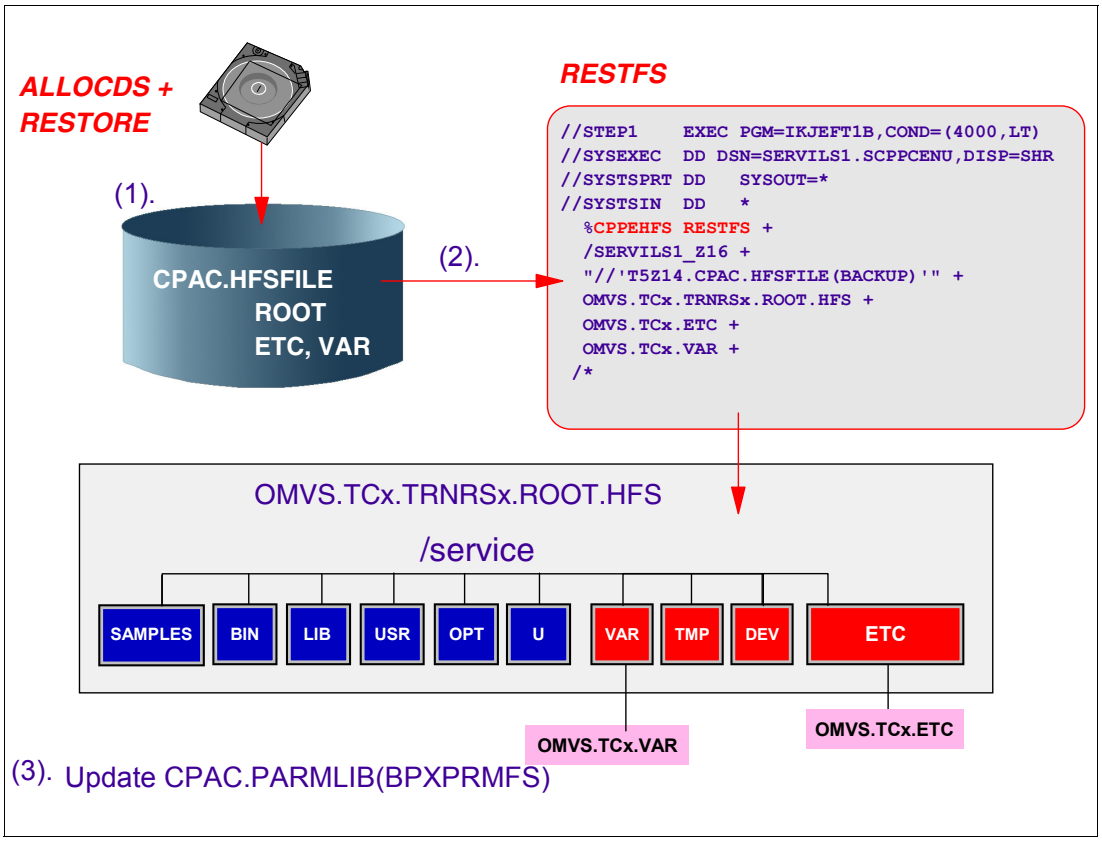


Figure 4-15 ServerPac flow for z/OS UNIX installation

Installing the root file system

For a full configuration you need to build a complete root file system.

ServerPac supplies the jobs, ALLOCDS and RESTORE, to download the PDS from tape to DASD. The other job, RESTFS uses the pax utility to restore the HFS file systems.

After running these jobs you have a complete root file system containing all the root-level directories, files, and programs.

ServerPac builds the root for you with all the features that you ordered already installed in it. They use the UNIX pax utility to compress the hierarchical format into an HFS file. It is distributed to you as a member of hlq.HFSFILE:

```
hlq.HFSFILE(BACKUP) Backup of Root File System
```

In addition to reading and writing archive files (which concatenate the contents of files and directories), pax can record file modification information such as dates, owner names, and so on. You can therefore use a single archive file to transfer a directory structure from one machine to another, or to back up or restore groups of files and directories.

(1) ALLOCDS

The ServerPac job ALLOCDS performs the following tasks for you:

- Creates ROOT HFS

- Creates ETC HFS
- Creates the following directories:

  VAR - HFS

(2) RESTFS job

The next job to run is RESTFS. This job does the following tasks for you:

- Allocates STDIN, STDOUT and STDERR files (UNIX standard outputs). An RC=4 is received if the directory is empty.
- Creates /SERVICE directory.
- Calls **pax** to restore the ROOT HFS.
- Mounts the created ROOT HFS to /SERVICE.
- Mounts the created ETC to /etc and mounts the var data sets.

Note: The RESTFS job has been combined with the RESTORE job, which restores the **pax** archive for the USS file system data sets directly from tape.

(3) Update PARMLIB

The installation process RESTFS job updates the CPAC.PARMLIB member as follows:

- Updates the BPXPRMFS with the MOUNT FILESYSTYPE section.
- Updates the BPXPRMFS with the FILESYSTYPE TYPE(....) and the corresponding entry point.

4.16 Non-volatile root file system

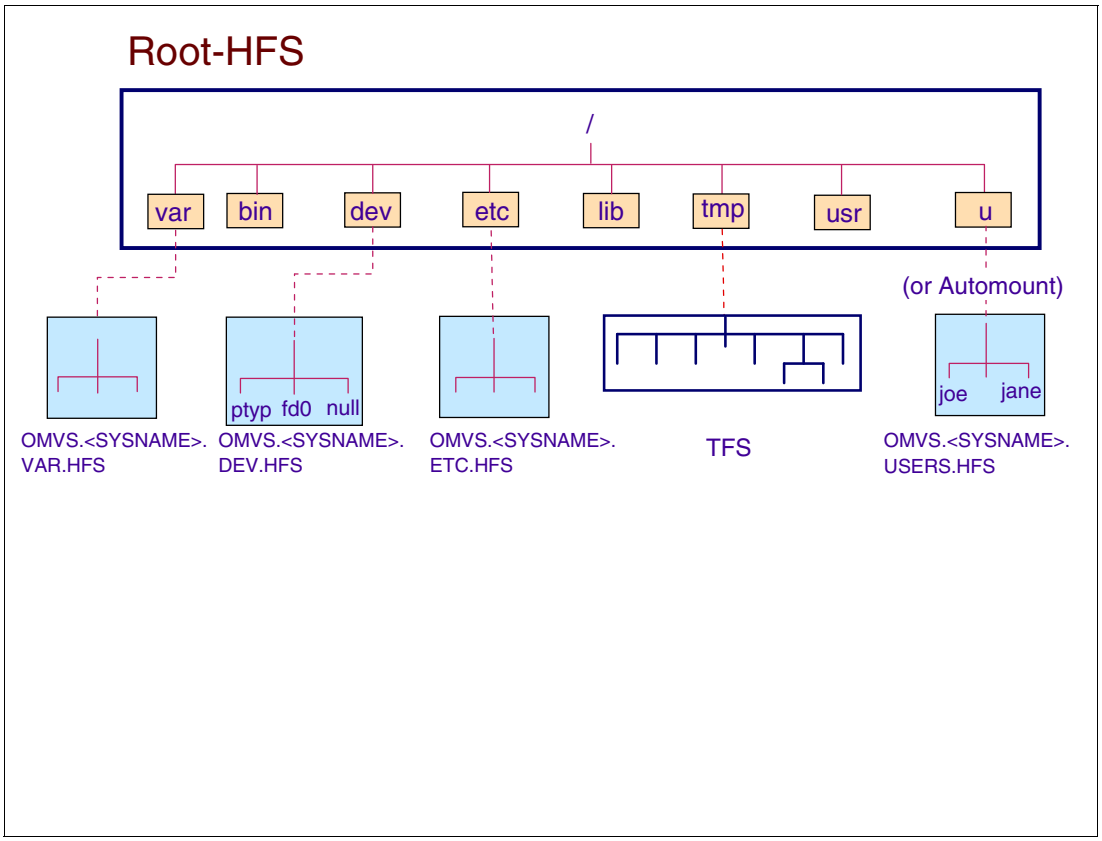


Figure 4-16 Managing the root file system

Managing the root file system

Extra planning should be done to keep file activity out of the root file system. This will protect the root and make future migration to new system levels easier.

Installation customized files (such as profiles in /etc) should be kept in their own HFS. This will protect these files from being overlaid by a future system replace.

Only file systems that are mounted in read-only mode can be shared between multiple system images. However, IBM no longer recommends mounting the root file system in read-only mode since it will cause customization problems and incorrect setup.

z/OS is built on a *system replace* basis. As new levels come out, all systems data sets, including the root HFS, will be replaced. Therefore, make a plan to get all of your tailored files out of the root and into their own HFS. Then when the system is replaced, you will only have to do minimal revising of your profiles and other user-oriented files and remount their HFSs to the new root. Keeping changes out of the root will make future changes much easier to manage.

4.17 Installation of other products

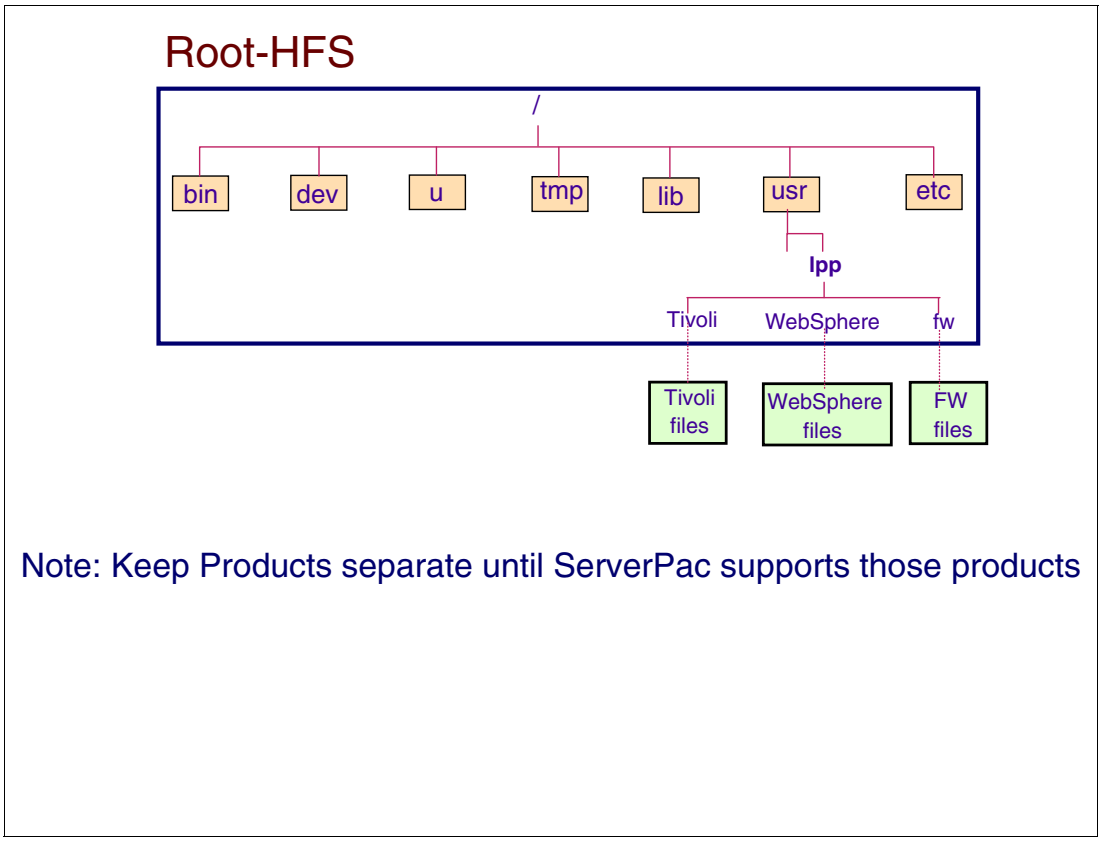


Figure 4-17 Installing products that are not part of the ServerPac

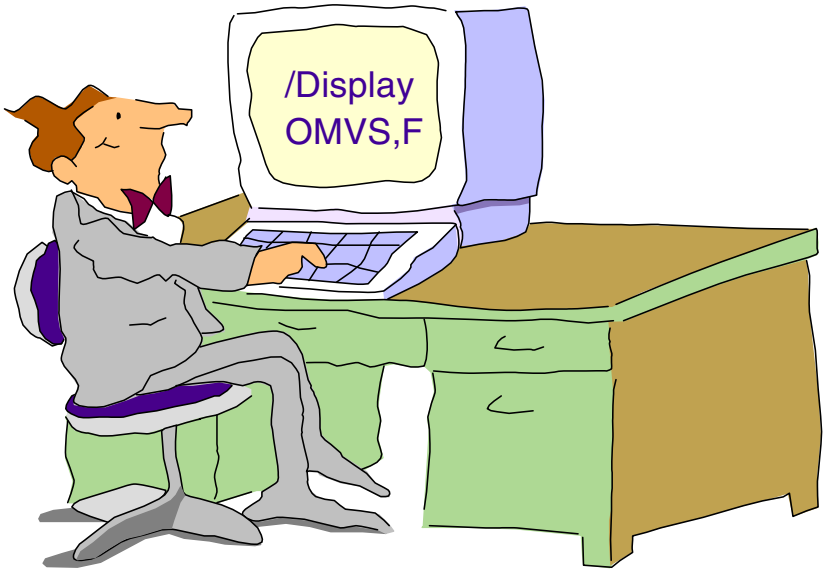
Installation of products off the root

When installing products that are not part of the ServerPac order and that install into the HFS, consider the following:

- Create new directories where the files associated with the new products will be installed.
- If possible, create a new HFS data set and mount it to the new directory. After installation of the product, all the files will reside in the new HFS data set.
- IBM has created the /usr/lpp directory and each product that puts files in the HFS structure creates a directory in lpp and then creates and mounts its own HFS at that mount point.
- Keeping new products in different HFS data sets offers better file system management while maintaining a more stable root file system. This also ensures easier maintenance when applying service.

Figure 4-18 on page 124 shows the IBM convention for managing IBM products. Directory /usr/lpp is built during the build of the root. As products (such as DCE or ICS) are built, they will create a directory in /usr/lpp and then will mount an HFS to that directory. All the product files will then be installed in a separate file system. Since the main installation vehicle for z/OS is system replace, all these directories are built in root as it is built. System replace should include these products.

4.18 z/OS UNIX System Services installation




```
BPXO044I 15.23.48 DISPLAY OMVS 971
OMVS     000E ACTIVE          OMVS=00
TYPENAME   DEVICE ----------STATUS----------- MODE QJOBNAME  QPID
BPXTFS          1 ACTIVE                      RDWR
  NAME=ROOT
  PATH=/
```

Figure 4-18 Display of root file system

Command to display the root

Before we update the provided JCL we should check how the TFS was named. It should be equal to the name we used in our first step, where we requested an unmount for the TFS.

You can get the information using the **D OMVS,F** command.

This command provides the following output:

- BPXTFS NAME=ROOT
- PATH = /
- UNIX Service is ACTIVE
- HFS is in Read/Write (RDWR) Mode

The TFS is used to get UNIX System Services up and running during IPL. Since OS/390 V2R3 it is no longer possible to start and stop z/OS UNIX System Services from OMVS.

Using the **D OMVS,F** command shows the mounted file systems; this can also be done with the UNIX ISHELL.

5

z/OS UNIX shell and utilities

This chapter provides information on how to invoke, customize, and use the z/OS UNIX shell and utilities. It includes details on the following topics:

- A brief overview about the z/OS UNIX shell
- How to invoke the shell using different methods
- Creating the required resources for the shell
- Setting up code page translation for the shell
- Shell environment variables
- Global variables
- How to pick up the region size when invoking the shell
- Where and how to set up the time zone variable
- Customizing the c89/cc compiler
- Installing books for OHELP

© Copyright IBM Corp. 2011. All rights reserved.

5.1 The z/OS UNIX shell

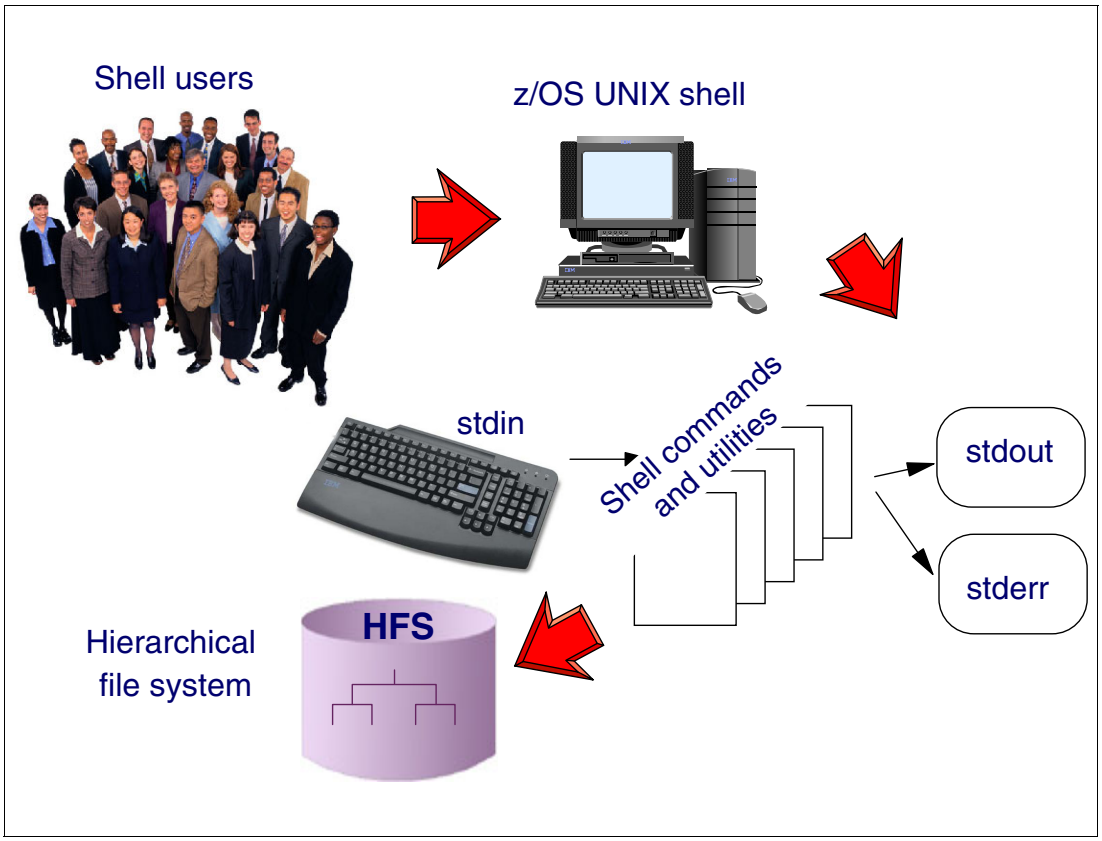


Figure 5-1 Overview of the z/OS UNIX shell

z/OS UNIX shell

The z/OS UNIX shell can be compared to TSO/E for z/OS. It is the interactive interface to z/OS UNIX. The shell feature comes with multiple commands and utilities which are in most cases the same as the ones that come with a UNIX system. The z/OS UNIX shell is based on the UNIX System V shell with some of the features from the Korn shell. The z/OS UNIX shell conforms to the POSIX 1003.2 standard and to the XPG4 standard.

POSIX 1003.2 distinguishes between a command (a directive to the shell to perform a specific task) and a utility (a program callable by name from the shell). To the end user there is no difference between a command and a utility. For the purpose of this discussion, command refers to both commands and utilities.

The shell is a command processor that can be used to:

- Invoke shell commands or utilities that request services from the system.
- Write shell scripts using the shell programming language.
- Run shell scripts, REXX EXECS, and C-language programs interactively, in the background, or in batch.

Shell commands are typically short, and they can be combined in pipes, or in shell scripts. Once a shell command begins executing it has access to three files:

```
stdin:    standard input    Default is the keyboard.
stdout:   standard output   Default is the screen.
stderr:   standard error    Default is the screen.
```

5.2 Input, output, errors with UNIX shell

- Executing commands, REXX or C programs - user has access to three files:
  - Input file - This is the input keyboard
  - Output file - Terminal screen by default
    - Or, you can specify an HFS file
  - Error file - Terminal screen for error messages
    - Or, you can specify an HFS file
- File names:
  - Input - stdin
  - Output - stdout
  - Error - stderr

Figure 5-2 Files that a user has access to in a shell session

Files used by a shell session

z/OS C/C++ programs require that stdin, stdout, and stderr be defined as either a file or a terminal. Many C functions use stdin, stdout, and stderr. However, it depends on how the application is coded whether or not it writes to stderr and stdout.

Shell command access to files

When a shell command begins running, it has access to three files:

- It reads from its standard input file. By default, standard input is the keyboard.
- It writes to its standard output file.
  - If you invoke a shell command from the shell, a C program, or a REXX program invoked from TSO READY, standard output is directed to your terminal screen by default.
  - If you invoke a shell command, REXX program, or C program from the ISPF shell, standard output cannot be directed to your terminal screen. You can specify an HFS file or use the default, a temporary file.
- It writes error messages to its standard error file.
  - If you invoke a shell command from the shell or from a C program or from a REXX program invoked from TSO READY, standard error is directed to your terminal screen by default.

– If you invoke a shell command, REXX program, or C program from the ISPF shell, standard error cannot be directed to your terminal screen. You can specify an HFS file or use the default, a temporary file.

  If the standard output or standard error file contains any data when the command completes, the file is displayed for you to browse.

Shell file names

In the shell, the names for these files are:

- stdin for the standard input file
- stdout for the standard output file
- stderr for the standard error file

5.3 Accessing the z/OS UNIX shell

- Pseudoterminal files - (pseudo-TTYs)
  - Used by users and applications to gain access to the shell
  - Pair of character special files - (master and slave)
    - Master - Used by OMVS and rlogin
    - Slave - Used by the shell to read and write terminal data
  - Naming convention:
    - Master - /dev/ptypNNNN
    - Slave - /dev/ttypNNNN
    - NNNN - (0000 to MAXPTYS value -1)

Figure 5-3 Files used when accessing the z/OS UNIX shell

Pseudoterminal files

Certain shell commands, such as **mesg**, **talk**, and **write** require pseudoterminals to have a group name of TTY. When a user logs in, or issues the OMVS command from TSO/E, the group name associated with these terminals is changed to TTY. As part of the installation, you had to define the group TTY or use the group alias support as a GROUP ID in the security data base, as shown in the following example:

```
ADDGROUP TTY (OMVS(GID(2))
```

Pseudo-TTYs

Pseudoterminals (pseudo-TTYs) are used by users and applications to gain access to the shell. A pseudo_TTY is a pair of character special files, a master file and a corresponding slave file. The master file is used by a networking application such as OMVS or rlogin. The corresponding slave file is used by the shell or the user's process to read and write terminal data. The convention for the names of the pseudo-TTY pair is:

```
/dev/ptypNNNN for the master (major 1)
/dev/ttypNNNN for the slave (major 2)
```

`NNNN` is between 0000 and one less than the MAXPTYS value in the BPXPRMxx PARMLIB member.

When a user enters the TSO/E OMVS command to initialize a shell, the system selects an available pair of these files. The pair represents the connection. The maximum number of pairs is 10000. You can specify an appropriate number of pairs in the MAXPTYS parameter.

5.4 Controlling session resources

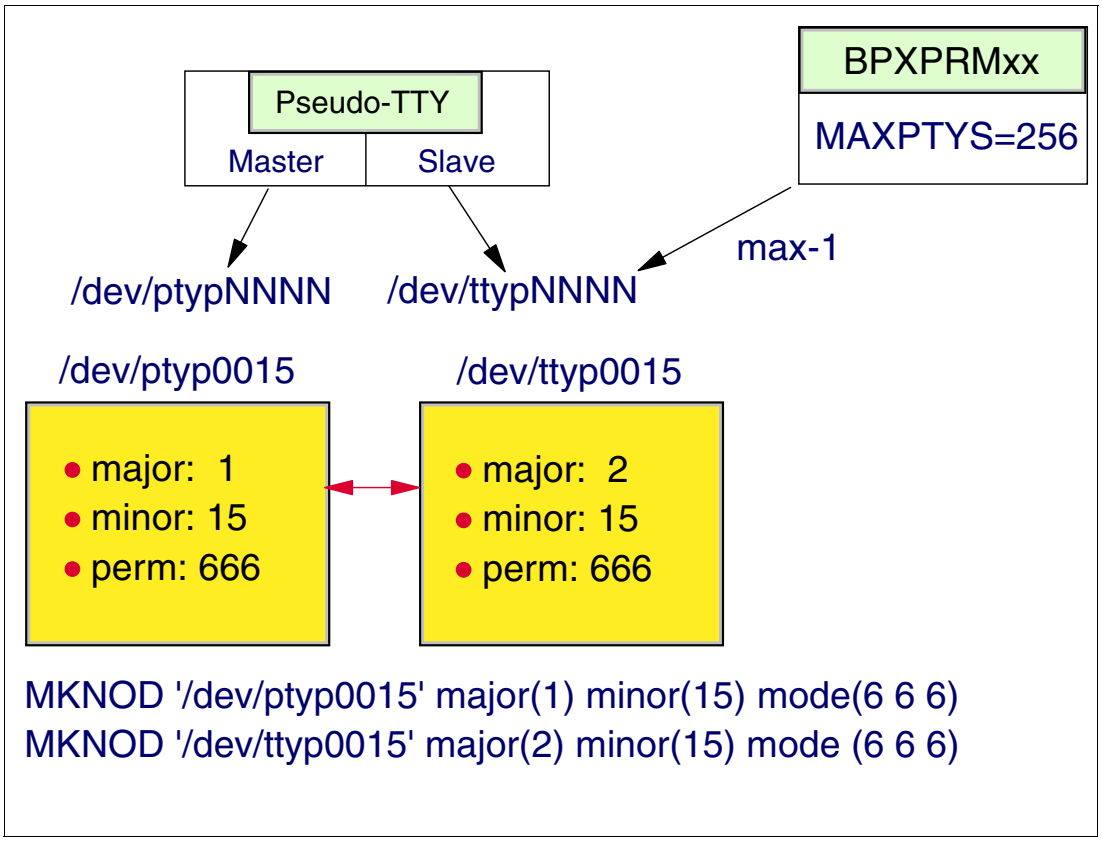


Figure 5-4 Pseudo-TTY files used by shell users

Shell user files

Pseudoterminals (pseudo-TTYs) are used by users and applications to gain access to the shell. A pseudo_TTY is a pair of character special files, a master file and a corresponding slave file. The master file is used by a networking application such as OMVS or **rlogin**. The corresponding slave file is used by the shell or the user's process to read and write terminal data.

The convention for the names of the pseudo-TTY pair is:

```
/dev/ptypNNNN for the master (major 1)
/dev/ttypNNNN for the slave (major 2)
```

The NNNN is between 0000 and one less than the MAXPTYS value in the BPXPRMxx member.

When a user enters the TSO/E OMVS command or logs in using **rlogin** or **telnet** to initialize a shell, the system selects an available pair of these files. The pair represents the connection. You can specify an appropriate number of pairs in the MAXPTYS parameter.

The MKNOD command

When using the MKNOD command, make:

- The major number 1 for the master and 2 for the slave.
- The minor number the same as the NNNN.

Note: The commands can be in a CLIST or REXX exec or entered directly in a TSO/E session or a shell session.

MAXPTYS statement in the BPXPRMxx member

The default controlling terminal can be accessed through the /dev/tty special file (major 3). This file is defined the first time the system is IPLed. The MAXPTYS statement specifies the maximum number of pseudo-TTY sessions that can be active at the same time. The range is 1 to 10000; the default and the value in BPXPRMXX is 256.

MAXPTYS lets you manage the number of interactive shell sessions. When you specify this value, each interactive session requires one pseudo-TTY pair. You should avoid specifying an arbitrarily high value for MAXPTYS. However, because each interactive user may have more than one session, it is recommended that you allow four pseudo-TTY pairs for each user (MAXIUDS * 4). The MAXPTYS value influences the number of pseudo-TTY pairs that can be defined in the file system.

5.5 Dynamic /dev

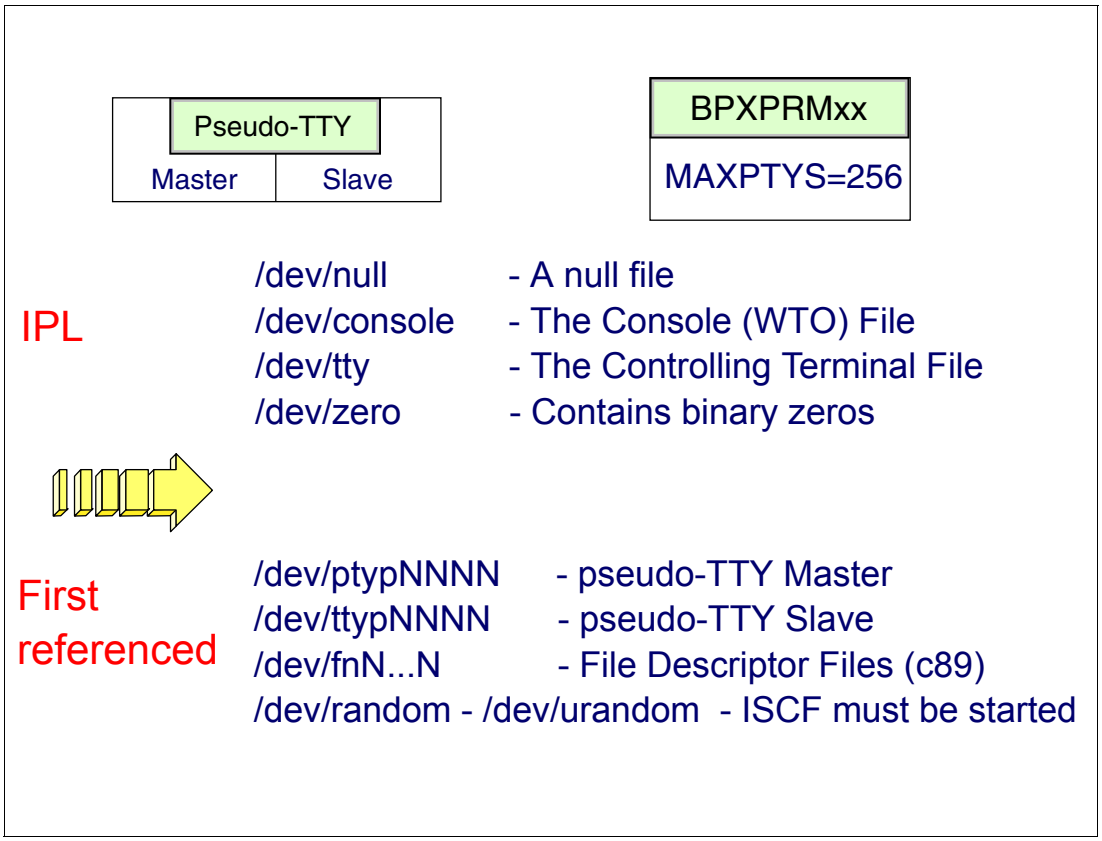


Figure 5-5 Dynamic /dev definitions

Files created dynamically at IPL

The null file, /dev/null, (major 4) is analogous to an MVS DUMMY data set. Data written to this file is discarded. The standard null file, named /dev/null, is created the first time the system is IPLed, or when referenced, if it does not exist already.

The /dev/console (major 9) file is sent to the console and displayed using a write-to-operator (WTO) within message BPXF024I. This message also contains the user ID of the process that wrote to the console.

The default controlling terminal can be accessed through the /dev/tty special file (major 3).

The zero file, /dev/zero (major 4, minor 1), is similar to /dev/null in that data written to this file is discarded, but when the file is read from, it provides an inexhaustible supply of binary zeros. The standard zero file, named /dev/zero, is created the first time the system is IPLed or when referenced, if it does not exist already.

The random number files, /dev/random and /dev/urandom (major 4, minor 2) provide cryptographically secure random output that was generated from the cryptographic hardware available on zSeries. The foundation of this random number generation is a time-variant input with a very low probability of recycling. In order to use these device files, Integrated Cryptographic Service Facility (ICSF) must be started, and a Cryptographic Coprocessor feature may be required, depending on the model of the zSeries server.

5.6 Invoking the shell via TSO/E

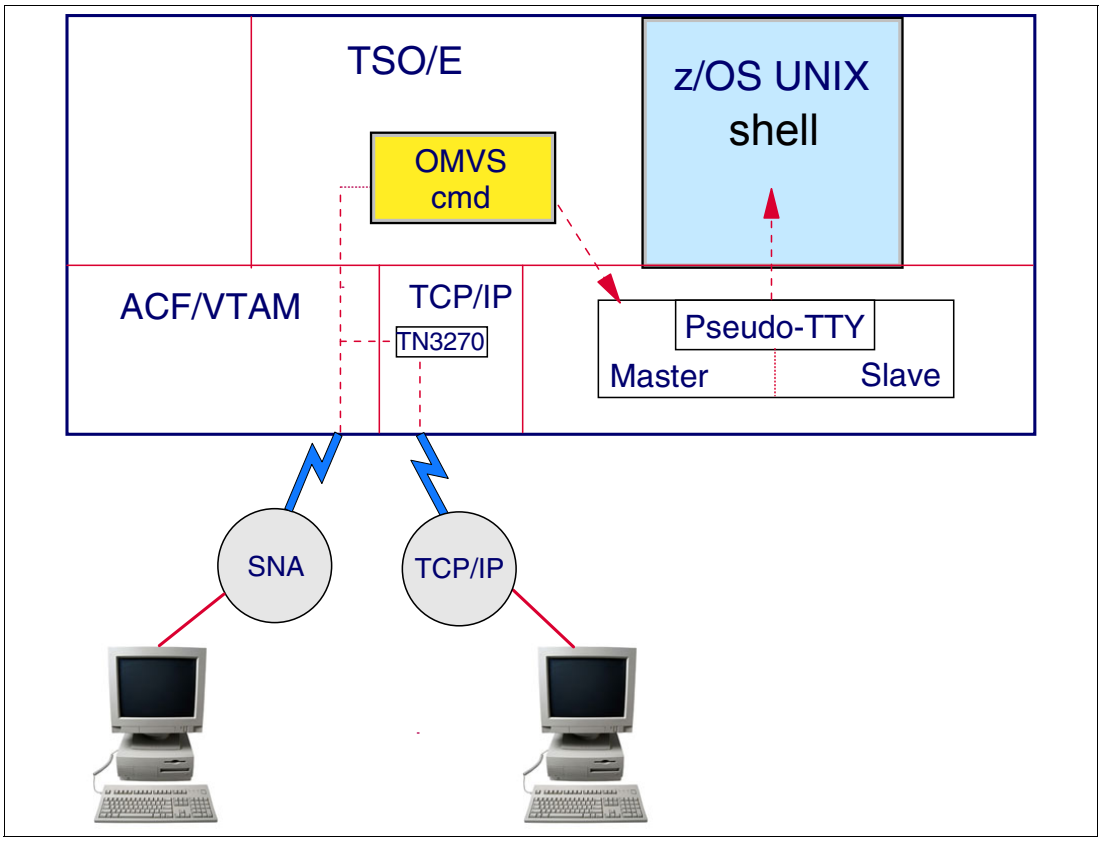


Figure 5-6 Invoking the shell from TSO/E or from a remote workstation

Accessing the shell

There are several ways that you can access the z/OS UNIX shells:

- The TSO/E OMVS command, which provides a 3270 interface
- The **`rlogin`** command, which provides an ASCII interface
- The **`telnet`** command, which provides an ASCII interface
- The **`ssh`** command, or secure shell, is an encrypted protocol and associated program intended to replace **`telnet`**.

OMVS shell

One of the methods to access the shell is by logging on to TSO/E and issuing the command OMVS. When accessing the shell from TSO/E, the users can jump back and forth between the shell and TSO/E. It is also possible to issue TSO/E commands from the shell.

The OMVS TSO/E command, together with the pseudo-TTY function, maps and transforms the 3270-oriented TSO/E terminal interface and user externals to the POSIX 1003.1 defined terminal interface expected by the POSIX 1003.2 conforming shell.

When you use the OMVS command to get into the z/OS UNIX shell, you can use ISPF to edit (OEDIT) and browse (OBROWSE) HFS files. You can switch between the shell and TSO session. You can have multiple shell sessions at the same time and toggle between the sessions using PF keys.

The shell process can run in the same address space as the user's TSO/E session.

Network shell connections

The network connection to the shell via TSO/E can be either VTAM or TCP/IP. This includes:

- Real and emulated 3270 terminals in an SNA network.
- UNIX systems and other workstations that in a TCP/IP network support the Telnet 3270 (TN3270) client function. The TN3270 client communicates with the Telnet server (TN-S) in TCP/IP on z/OS.

Pseudo-terminals

Pseudo-terminals are defined as a pair of character special files, one file designated as the master, and one file as the slave. The pseudo-TTY master process is traditionally a network server application; in z/OS UNIX the TSO/E command processor is the server application. The pseudo-TTY slave process is the user shell process.

The pseudo-TTY function resides in the z/OS UNIX kernel and consists of a master pty and a slave pty. The master pty is used by the networking application (z/OS UNIX) to communicate with the user applications, for example the shell. The slave pty is used by the shell process and associated applications. The TSO/E session reads and writes to the master, and the shell process reads and writes to the slave.

5.7 Invoking the shell via rlogin or telnet

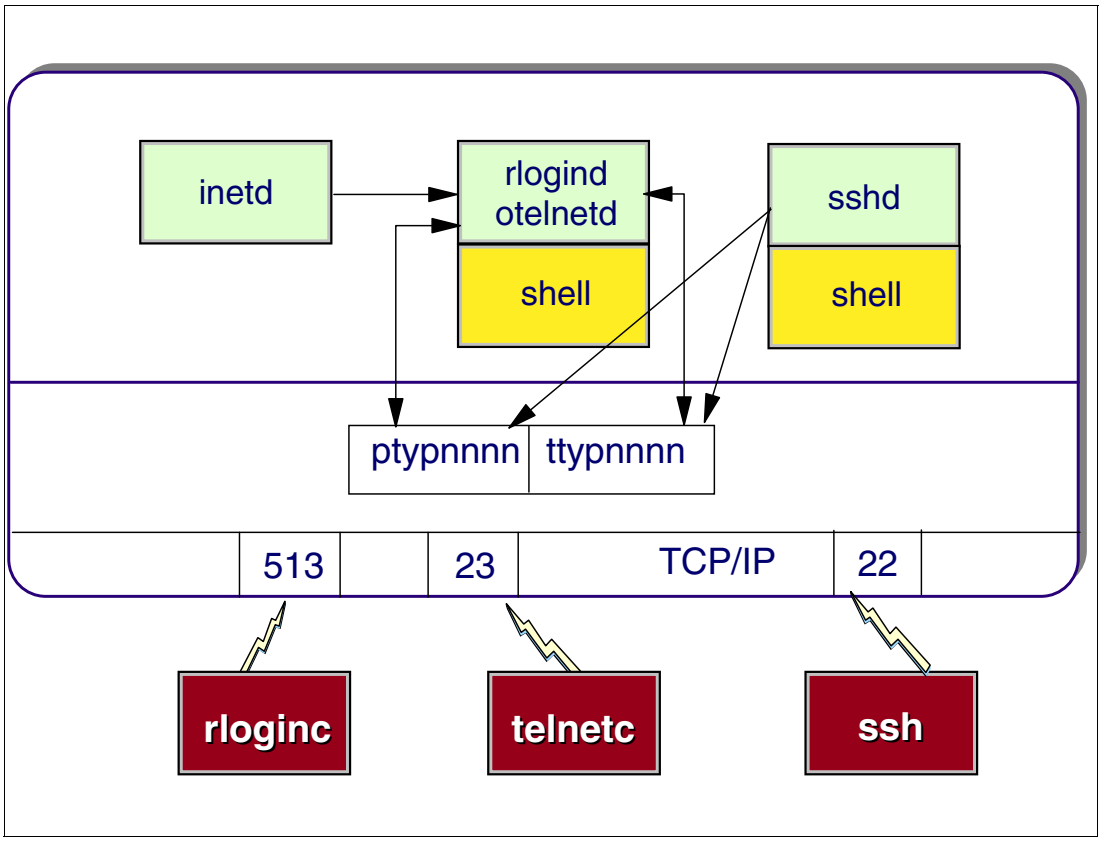


Figure 5-7 Invoking the shell from an rlogin or telnet

Accessing the shell from a remote terminal

A z/OS UNIX user can log in directly to the z/OS UNIX shell in either of the following ways:

rlogin If the inetd daemon is set up and active on the z/OS system, a workstation user with rlogin client support can use the TCP/IP network to log into the shell without going through TSO/E.

telnet Telnet support comes with the z/OS CS. It also uses the inetd daemon, which must be active and set up to recognize and receive the incoming **`telnet`** requests. You can **`telnet`** to the shell from a workstation that is connected via TCP/IP or Communications Server to the MVS system. Use the **`telnet`** command syntax supported at your site.

ssh ssh (Secure Shell) is a program to log into another computer over a network, to execute commands in a remote machine, and to move files from one machine to another. It provides strong authentication and secure communications over unsecure channels. It is intended as a replacement for rlogin, rsh, and rcp. Additionally, ssh provides secure X connections and secure forwarding of arbitrary TCP connections.

The advantage of ssh over Kerberos is that ssh does not require a central database of users, and users can use their regular password. However, ssh and Kerberos can safely coexist.

z/OS OMVS shell

A z/OS system provides asynchronous terminal support for the z/OS UNIX shell. This is different from the 3270 terminal support provided by the TSO/E OMVS command. In a UNIX system, the UNIX shell operates in non-canonical mode (often called raw mode). This means that each keystroke is transmitted from the terminal to the system. Several UNIX applications, among them a popular editor called vi, depend on raw mode support. Using the 3270-terminal interface for the z/OS UNIX shell meant that UNIX applications depending upon raw mode support could not be ported to z/OS.

Figure 5-7 shows an overview of the two methods for logging in directly to the shell. Directly logging in to the shell is the only way to get raw mode support. Both rlogin and telnet require the inetd daemon to be set up and active.

Differences between TSO/E and remote terminal

There are some differences between asynchronous terminal support (direct shell login) and 3270 terminal support (OMVS command):

- You cannot switch to TSO/E. However, you can use the TSO shell command to run a TSO/E command from your shell session.
- You cannot use the ISPF editor (this includes the oedit and TSO/E OEDIT commands, which invoke ISPF edit).

Note: The asynchronous interface does not automatically put you in raw mode. The terminal is set to operate in line mode, and only when using a utility that requires raw mode will it be changed.

5.8 rlogin and telnet access

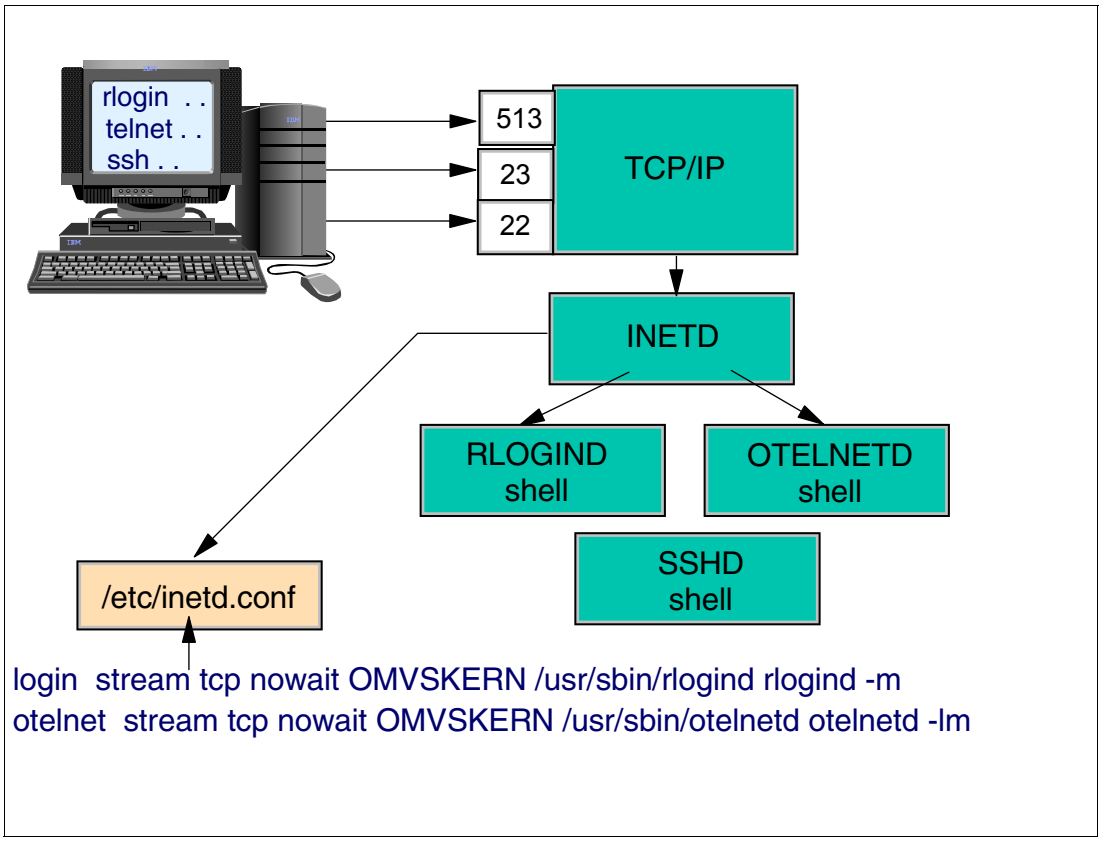


Figure 5-8 rlogin and telnet access to the shell

Entering the shell from a remote terminal

Client users issue a **`telnet`** or **`rlogin`** command in the syntax of the system they are logged on to. TCP/IP must be set up on both systems to recognize the appropriate system names and ports.

The inetd daemon provides service management for a network. For example, it starts the rlogind program whenever there is a remote login request from a workstation.

The rlogind program is the server for the remote login command **`rlogin`** commonly found on UNIX systems. It validates the remote login request and verifies the password of the target user. It starts a z/OS shell for the user and handles translation between ASCII and EBCDIC code pages as data flows between the workstation and the shell.

INETD daemon configuration file

When inetd is running and receives a request for a connection, it processes that request for the program associated with that socket. For example, if a user tries to log in from a remote system into the z/OS shell while inetd is running, inetd processes the request for connection and then issues a fork() and execl() to the rlogin program to process the rlogin request. It then goes back to monitoring for further requests for those applications that can be found as defined in the /etc/inetd.conf file as follows:

```
login stream tcp nowait OMVSKERN /usr/sbin/rlogind rlogind -m
otelnet stream tcp nowait OMVSKERN /usr/sbin/otelnetd otelnetd -lm
```

Where:

stream stream or dgram. This is the socket_type.

tcp tcp or udp. The protocol is used to further qualify the service name. Both the service name and the protocol should match an entry in the data set, /etc/services, which is part of the user's TCP/IP configuration.

nowait wait or nowait. Wait indicates the daemon is single-threaded and another request will not be serviced until the first one completes. Nowait specifies that the inetd daemon issues an accept when a connect request is received on a stream socket. If wait is specified, the inetd daemon does not issue the accept. It is the responsibility of the server to issue the accept if this is a stream socket.

z/OS UNIX daemons

The rlogin daemon or the telnet daemon is responsible for validating the user, handling ASCII-to-EBCDIC translation, and invoking the shell.

The presence of the **-m** option on the command indicates that the shell should start in the same address space as the daemon. The **-l** sets default mode for login to line mode.

A pseudo-TTY pair is assigned from the same pool to the shell session. To get multiple shell sessions the user issues another **rlogin** or **telnet** client command.

The sshd daemon

ssh is actually a client-server protocol and a suite of connectivity tools. The ssh client program is a secure remote login program—that is, a secure alternative to **rlogin** or **rsh**. The ssh suite also provides sftp, which is a secure version of ftp, and rcp rcp, a secure version of rcp. The client program requests a connection to the sshd secure remote login daemon running in the target host. The sshd daemon handles key exchange, encryption, authentication of both server and user, command execution, and data exchange. By default sshd listens to port 22 for incoming connection requests.

The sshd daemon is started independent of the inetd daemon. The daemon can be started with the following started task JCL:

```
//SSHD     PROC
//SSHD     EXEC PGM=BPXBATCH,REGION=0M,TIME=NOLIMIT,
//  PARM='PGM /usr/sbin/sshd -f /etc/ssh/sshd_config'
//STDOUT   DD SYSOUT=*,LRECL=137,RECFM=VB
//STDENV   DD PATH='/etc/ssh/stdenv',PATHOPTS=(ORDONLY)
```

The main command that should be run from a user ID having BPX.DAEMON read access is:

```
/usr/sbin/sshd -f /etc/ssh/sshd_config
```

5.9 Customizing z/OS UNIX initialization

- /etc is the "SYS1.PARMLIB" for z/OS UNIX
- Files to customize - Copy from /samples
  - /etc/init or /usr/sbin/init
    - Program that executes an initialization shell script
  - /etc/init.options
    - Initialization options file
  - /etc/rc
    - Customization commands
    - Similar to COMMANDxx for z/OS

Figure 5-9 Files needed for z/OS UNIX initialization

Files used by z/OS UNIX initialization

The files /etc/init and /usr/sbin/init are referred to synonymously as the initialization program that is run when the OMVS address space is initialized during the IPL. Copy the file /samples/init.options to /etc/init.options. At startup, the kernel services attempt to run the program /etc/init, which is an initialization program created by the user. If that program is not found, then the kernel services try to run /usr/sbin/init, the default initialization program that is run when the OMVS address space is initialized. The files are run at IPL.

Note: You can use a REXX exec in an MVS data set as an alternative to running the /etc/init initialization program. To activate the REXX exec for initialization, you must specify its name on the STARTUP_EXEC statement in the BPXPRMxx parmlib member.

The /usr/sbin/init and /etc/init programs

The /usr/sbin/init program invokes a shell to execute an initialization shell script that customizes the environment for z/OS UNIX users. When this shell script finishes or when a time interval established by /usr/sbin/init expires, kernel services become available for general batch and interactive use.

The /etc/init.options file

Before /usr/sbin/init invokes the shell to execute the system initialization shell script, it reads the file /etc/init.options for values of various options. The IBM-supplied default is in /samples/init.options. Copy this file to /etc/init.options and make the appropriate changes. If you already have /etc/init.options, then compare it to /samples/init.options and retrofit any new updates.

/etc/init.options can contain up to 25 -e option lines specifying names and values for different environment variables. /usr/sbin/init passes the resultant environment variable array to the shell that it invokes. In turn, the shell uses this array to set up an execution environment for the initialization shell script that is appropriate for the installation. TZ is an example of an environment variable that should be considered.

The /etc/rc file

You need to copy /samples/rc file to /etc/rc. It contains customization commands for z/OS UNIX System Services Application Services.

5.10 Initializing z/OS UNIX

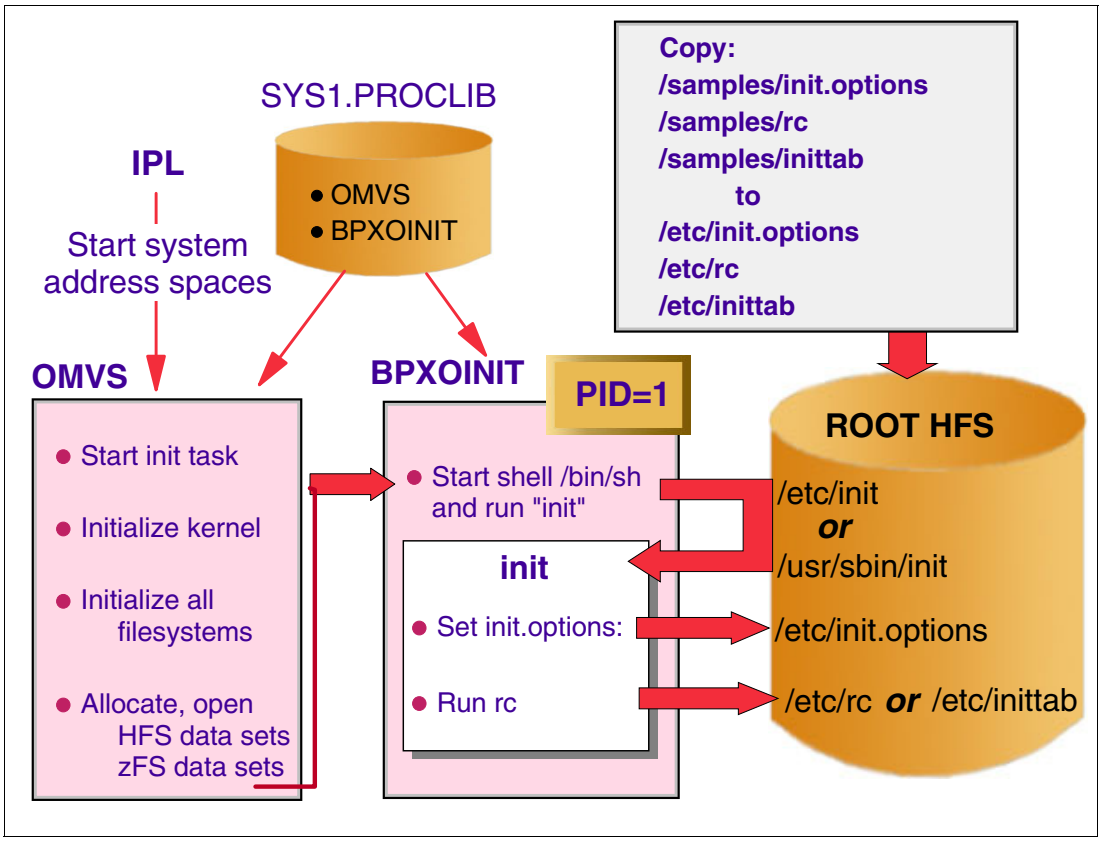


Figure 5-10 z/OS UNIX initialization processing

z/OS UNIX initialization

/usr/sbin/init and the customized /etc/init.options and /etc/rc files are run at IPL. There is no other way to invoke them explicitly. During the IPL, one of the first address spaces to start is OMVS.

The file /etc/init is referred to as the initialization program that is run when the z/OS UNIX component is started, even though the /usr/sbin/init file may really be run if no such program is found. This file contains the default initialization program shipped with the z/OS UNIX shell and utilities.

BPXOINIT

BPXOINIT is the started procedure that runs the initialization process. BPXOINIT is also the job name of the initialization process and is shipped in SYS1.PROCLIB. The STEPLIB DD statement is propagated from OMVS to BPXOINIT. If there is a STEPLIB DD statement in the BPXOINIT procedure, it is not used if a STEPLIB DD statement was specified in the OMVS procedure.

The /etc/init program

The /etc/init program invokes a shell to execute an initialization shell script that customizes the environment for the UNIX kernel. When this shell script finishes or a time interval established by /etc/init expires, z/OS UNIX becomes available for general batch and interactive use.

5.11 Environment variables

- All UNIX systems have environment variables
  - The value of an environment variable is a string of characters
- Each process has its own private set of environment variables
  - By default, when a process is created it inherits a duplicate environment of its parent process
  - Running programs can access the values of environment variables for configuration purposes
- Shell scripts and batch files use environment variables to store temporary values for reference later in the script

Figure 5-11 Environment variables in UNIX systems

Environment variables

Environment variables are global values or settings that determine the default operation of all shells and are also passed on to application programs. Environment variables contain information about your working environment. With all UNIX systems, each process has its own private set of environment variables. By default, when a UNIX process is created it inherits a duplicate environment of its parent process, except for explicit changes made by the parent when it creates the child process, for example a fork or exec function. With different UNIX systems, they do not all use the same variable names. Running programs can access the values of environment variables for configuration purposes.

The value of an environment variable is a string of characters. For a C-language program, an array of strings called the environment shall be made available when a process begins. For z/OS UNIX, environment variables are defined in files in the file structure such as /etc/init.options, /etc/rc, and /etc/profile.

Environment variable names used by the utilities in the Shell and Utilities volume of IEEE Std 1003.1-2001 consist solely of uppercase letters, digits, and the '_' (underscore) from the characters defined in Portable Character Set and do not begin with a digit. Other characters may be permitted by an implementation; processes tolerate the presence of such names. Uppercase and lowercase letters shall retain their unique identities and shall not be folded together. The name space of environment variable names containing lowercase letters is reserved for processes. Processes can define any environment variables with names from this name space without modifying the behavior of the standard utilities.

5.12 Environment variables

- A name associated with a string of characters made available to the programs that you run
- Some environment variables used by the shell are
  - PATH and TZ
- Display variables with SET command

```
PATH="/usr/lpp/Printsrv/bin:/bin:."
HOME="/u/rogers"
SHELL="/bin/sh"
STEPLIB="none"
TERM="dumb"
TZ="EST5EDT"
_BPXK_SETIBMOPT_TRANSPORT="TCPIPOE"
_BPX_TERMPATH="OMVS"
```

Figure 5-12 Environment variables

Environment variables

When a program begins, an environment is made available to it. The environment consists of strings of the form `name=value`, where `name` is the name associated with the environment variable, and its `value` is represented by the characters specified in value. UNIX systems traditionally pass information to programs through the environment variable mechanism.

SET command

Using the **`set`** command without arguments displays the names and values of all shell variables, sorted by name, in the format Variable="value". An example is:

```
PWD="/u/rogers"
RANDOM="7322"
SECONDS="2"
SHELL="/bin/sh"
STEPLIB="none"
TERM="dumb"
TZ="EST5EDT,M3.2.0,M11.1.0"
```

Shell user variables

There are global variables for all shell users and each user can override these variables with an individual set of variables. You can also change any of the values for the duration of your session (or until you change them again). You enter the name of the environment variable and equate it to a new value.

5.13 The /etc/init.options file

```
-a  9999                          timeout=180 secs (default)
-t  1                             terminate shell = yes
-sc /etc/rc                       shell script = /etc/rc
-e TZ=EST5EDT                     TZ environment variable
*e LANG=C                         LANG environment variable
*e NLSPATH=/usr/lib/nls/msg/%L/%N NLSPATH environment variable
*sh /bin/sh                       shell = /bin/sh
*e PATH=/bin                      PATH environment variable
*e SHELL=/bin/sh                  SHELL environment variable
*e LOGNAME=ROOT                   LOGNAME environment variable.
```

Figure 5-13 The /etc/init.options file

Updating the /etc/init.options file

The IBM-supplied default is located in the file /samples/init.options. Copy this file to /etc/init.options. This file will be used the next time z/OS UNIX is started.

The file /etc/init.options treats all lines in the file that do not start with a hyphen (-) as comment lines. Lines that start with a hyphen are used to specify options.

The following options can be specified:

-a The maximum time in seconds that /etc/init will wait for the initialization shell script to complete.

-t Terminate option indicating whether to terminate the script if the timeout occurs.

-sh The pathname of the initializing shell.

-sc The pathname of the initializing shell script.

-e TZ=EST5EDT The time zone environment variable is set to the time for the USA east coast (five hours east of GMT/UTC). For a detailed description of how to set the time zone variable, see “Setting up the Time Zone Variable,” in *z/OS UNIX System Services Planning*, GA22-7800.

e Environment variable options. The environment variables that are set in this file will only be valid for the initialization shell. Later figures show how environment variables can be set for the shell users.

5.14 The etc/rc file

```
# Initial setup for z/OS UNIX
export _BPX_JOBNAME='ETCRC'

# Provide z/OS UNIX Startup Diag.
set -v -x

# Setup utmpx file
>/etc/utmpx
chmod 644 /etc/utmpx

# Reset all slave tty files
chmod 666 /dev/tty*
chown 0 /dev/tty*

# Setup write, talk, mesg utilities
chgrp TTY /bin/write
chgrp TTY /bin/mesg
chgrp TTY /bin/talk
chmod 2755 /bin/write
chmod 2755 /bin/mesg
chmod 2755 /bin/talk
```

Page 1

```
# Setup uucp utility
. . .

# Invoke vi recovery
mkdir -m 777 /etc/recover
/usr/lib/exrecover

# Start AUTOMOUNT
/usr/sbin/automount

# Start the INET daemon
_BPX_JOBNAME='INETD'
  /usr/sbin/inetd /etc/inetd.conf &

sleep 5
echo /etc/rc script executed, 'date'
```

Page 2

Figure 5-14 The /etc/rc file used during initialization

Updating the /etc/rc file

A sample initialization shell script can be found in the file /samples/rc. Figure 5-14 shows the IBM-supplied /samples/.rc file. Copy this file to the /etc directory. No changes will be used until the next time z/OS UNIX is started.

The initialization script can be used to start daemons, or perform shell commands or scripts automatically each time z/OS UNIX is started. For example, the inetd daemon can be added to this script, as well as the automount command if it is to be used.

The /etc/rc file contains customization commands for z/OS UNIX System Services Application Services. The figure shows the IBM-supplied default in /samples/rc. Copy this file to /etc/rc and make the appropriate changes. If you already have an /etc/rc file, then compare it to /samples/rc and retrofit any new updates.

Customize your /etc/rc file by adding shell commands. For instance, you could add a command to start an installation-supplied daemon. The script can also invoke other scripts, for instance, an rc.tcpip script to start tcp daemons.

Export variables

Export marks each variable name so that the current shell makes it automatically available to the environment of all commands run from that shell. Exported variables are thus available in the environment to all subsequent commands.

The _BPX_JOBNAME variable sets the z/OS job name for this script to ETCRC.

```
export _BPX_JOBNAME='ETCRC'
```

Figure 5-14 shows parts of the sample provided in the /samples/rc file. The AUTOMOUNT facility is started and some lines were taken out to make it fit the figure frame.

```
/usr/sbin/automount
```

The example shows that the inetd daemon will be started automatically each time z/OS UNIX is initialized. The TCP/IP topic will have more information on the inetd start options. This _BPX_JOBNAME variable will set the z/OS job name to the INETD daemon.

```
_BPX_JOBNAME='INETD' /usr/sbin/inetd /etc/inetd.conf
```

/samples/rc file contents

```
# Initialization shell script, pathname = /etc/rc
# Initial setup for z/OS UNIX
export _BPX_JOBNAME='ETCRC'
# Provide z/OS UNIX Startup Diagnostics
set -v -x
# Setup utmpx file
>/etc/utmpx
chmod 644 /etc/utmpx
# Reset all slave tty files
chmod 666 /dev/tty*
chown 0 /dev/tty*
# Allow only file owner to remove files from /tmp
chmod 1777 /tmp
# Allow only file owner to remove files from /var
chmod 1777 /var
# Allow only file owner to remove files from /dev
chmod 1755 /dev
# Setup write, talk, mesg utilities
# chgrp TTY   /bin/write
# chgrp TTY   /bin/mesg
# chgrp TTY   /bin/talk
# chmod 2755  /bin/write
# chmod 2755  /bin/mesg
# chmod 2755  /bin/talk
# Performed at install in HOT7707
# Commented out in HOT6609 and performed in SAMPLIB job FOMISCHO
# Setup mailx utility
# No need to CHGRP /usr/mail directory
# No need to CHGRP mailx utility
# No need to CHMOD mailx to turn on SETGID
# Setup uucp utility
# chown uucp:uucpg /usr/lib/uucp
# chown uucp:uucpg /usr/lib/uucp/IBM
# chown uucp:uucpg /usr/spool/uucp
# chown uucp:uucpg /usr/spool/locks
# chown uucp:uucpg /usr/spool/uucppublic
# chown uucp:uucpg /usr/spool/uucp/.Xqtdir
# chown uucp:uucpg /usr/spool/uucp/.Sequence
# chown uucp:uucpg /usr/spool/uucp/.Status
# chown uucp:uucpg /bin/uucp
# chown uucp:uucpg /bin/uuname
```

```
# chown uucp:uucpg /bin/uustat
# chown uucp:uucpg /bin/uux
# chown uucp:uucpg /usr/lib/uucp/uucico
# chown uucp:uucpg /usr/lib/uucp/uuxqt
# chown uucp:uucpg /usr/lib/uucp/uucc
# chmod 4775 /bin/uucp
# chmod 4775 /bin/uuname
# chmod 4775 /bin/uustat
# chmod 4775 /bin/uux
# chmod 4754 /usr/lib/uucp/uucico
# chmod 4754 /usr/lib/uucp/uuxqt
# chmod 4774 /usr/lib/uucp/uucc
# Performed at install in HOT7707
# Commented out in HOT6609 and performed in SAMPLIB job FOMISCHO
# Invoke vi recovery
# mkdir -m 777 /var/tmp
# export TMP_VI="/var/tmp"
mkdir -m 777 /etc/recover
/usr/lib/exrecover
# Create TERMINFO database
# tic /usr/share/lib/terminfo/ibm.ti
# tic /usr/share/lib/terminfo/dec.ti
# tic /usr/share/lib/terminfo/wyse.ti
# tic /usr/share/lib/terminfo/dtterm.ti
# commented tic out in HOT1180 - all TERMINFO files are shipped
# Start the INET daemon for remote login activity
#_BPX_JOBNAME='INETD' /usr/sbin/inetd /etc/inetd.conf &
sleep 5
echo /etc/rc script executed, date
```

5.15 The /etc/inittab file with z/OS V1R8

- Identify system processes that can be started at system initialization
  - To identify processes that can be restarted automatically when they unexpectedly end - (RESPAWN)
- The /etc/inittab file lists system processes, commands and shell scripts to be started when z/OS UNIX initializes
  - Copy the /samples/inittab file to /etc/inittab and edit
    - Add daemons - syslogd and cron - started from /etc/rc
    - /etc/inittab file processed only once during initialization
  - To run a command again, start manually
    - From the z/OS UNIX shell or via BPXBATCH
    - Or - set _BPXK_INITTAB_RESPAWN=YES

Figure 5-15 Using the /etc/inittab file with z/OS V1R8

The /etc/inittab file

In z/OS V1R8, the /etc/inittab file is the same as used on other UNIX platforms. You have the support for /etc/inittab to allow an installation to identify system processes that can be started at system initialization; and to identify processes that can be restarted automatically when they unexpectedly end.

The /etc/inittab file lists system processes such as commands and shell scripts that are to be started when z/OS UNIX is initialized. Copy the IBM-supplied /samples/inittab file to /etc/inittab and add additional entries if required.

For example, you can add daemons such as syslogd and cron that are normally started from /etc/rc to take advantage of the respawn capability. The /etc/inittab file is processed only once during initialization. If a command entry in the file needs to be run again, it must be manually restarted, for example, from the z/OS UNIX shell or via BPXBATCH.

To restore the respawn attribute, the command can be started using the _BPXK_INITTAB_RESPAWN environment variable.

Requirement: You must have superuser authority in order to set the _BPXK_INITTAB_RESPAWN environment variable to YES.

5.16 The _BPXK_INITTAB_RESPAWN variable

- This variable specifies whether a process is to be dynamically started with the respawn attribute
  - _BPXK_INITTAB_RESPAWN=YES specifies that a process is to be started with the respawn attribute Setting the YES attribute after the process has started does not affect the setting of the respawn attribute
- If a process is started by a spawn with _BPXK_INITTAB_RESPAWN=YES (set by an export shell command, for example), the shell invokes the target program
  - The program is automatically restarted when it ends, even if it was not originally started from the /etc/inittab file

Figure 5-16 Setting the _BPXK_INITTAB_RESPAWN variable

The _BPXK_INITTAB_RESPAWN variable

The _BPXK_INITTAB_RESPAWN environment variable specifies whether a process is to be dynamically started with the respawn attribute.

YES Specifies that a process is to be started with the respawn attribute. Setting the YES attribute after the process has started does not affect the setting of the respawn attribute. If a process is started by a spawn with _BPXK_INITTAB_RESPAWN=YES (set by an export shell command, for example), the shell invokes the target program. The program will be automatically restarted when it ends, even if it was not originally started from the /etc/inittab file.

NO Disables the respawn capability of the process.

5.17 Rules for coding /etc/inittab

- Identifier: - Identifies the inittab entry
- Runlevel: - Not supported with z/OS UNIX
- Action: - Specifies how to handle the process that is specified in the command field. The supported actions are:
  - once - Start the process and do not wait for it to end
  - respawn - Process is restarted when it ends
  - respfrk - For programs that fork a child and go away
  - wait - Start a process and wait for it to end - never restarted

/samples/inittab

```
etcrc::wait:/etc/rc > /dev/console 2>&1
inetd::respfrk:/usr/sbin/inetd /etc/inetd.conf
msgend::once:/bin/echo Done processing /etc/inittab > /dev/console
:end of file
```

Figure 5-17 Rules for coding /etc/inittab

The /etc/inittab file

The /etc/inittab file lists system processes such as commands and shell scripts that are to be started when z/OS UNIX is initialized. Copy the IBM-supplied /samples/inittab file, shown in Figure 5-15, to /etc/inittab and add entries to it.

Rules for coding /etc/inittab

Each entry is delimited by a newline character. A backslash (\) preceding a newline character indicates the continuation of an entry. There are no limits (other than maximum entry size) on the number of entries in the /etc/inittab file. The maximum entry size is 1024 characters. Each entry field is only limited by the maximum entry size. The entry fields are:

Identifier Identifies the inittab entry. The identifier for a given entry can be up to 7 mixed case characters and it must be unique from other identifiers in the file. Because the identifier is used as the job name for the command to be run, it must be syntactically acceptable as a job name. The lowercase characters are converted to uppercase to be used in the job name

RunLevel Not supported for z/OS UNIX. Identifies the run levels in which this entry can be processed. Even though the RunLevel field is not supported for z/OS UNIX, it is in the inittab entry for compatibility with other UNIX implementations. The run level field can be up to 32 alphanumerical characters.

Action Specifies how to handle the process that is specified in the command field. The supported actions are:

once Starts the process and does not wait for it to end. When it ends, The process is not restarted when it ends.

respawn Starts the process as specified in the entry and does not wait for it to end. Instead, it continues scanning the /etc/inittab file. The process is restarted when it ends. When a process is spawned again, it is restarted with the same file descriptors and environment variables that it was started with originally. If a process ends due to a shutdown of all fork activity, the process is not restarted until fork activity is re-enabled. If a respawnable process ends and then ends again after being restarted within 15 minutes of its first ending, then message BPXI082D is displayed. The message identifies the process entry and asks whether to retry or ignore the error. A process identified for respawn will not be able to register as a permanent process that can survive a shutdown and restart cycle because the /etc/inittab file will be processed again during restart. **Tip:** To avoid excessive consumption of common storage, limit the number of processes started with the respawn attribute to 100 or fewer.

respfrk Starts the process as specified in the entry. Do not wait for the process to end; in other words, continue scanning the /etc/inittab file.

- If the process never issues a fork command, then this action behaves the same way as the respawn action.
- If the process issues a fork, then the respawn attribute is transferred to the forked child process and the original process is respawned when the child process ends.
- If the original process issues any additional forks, the respawn attribute is not transferred. It is only transferred the first time the fork is issued.

  Tip: This option is intended for a program that forks itself to create a child process which then continues running while the parent process ends. For example, the cron and inetd daemons are written this way. This option is not found on any other UNIX platforms.

wait Starts the process and waits for it to end. The process is not restarted when it ends.

Note: If /etc/inittab exists in your system, it is used instead of the /etc/rc file.

5.18 Customizing the OMVS command

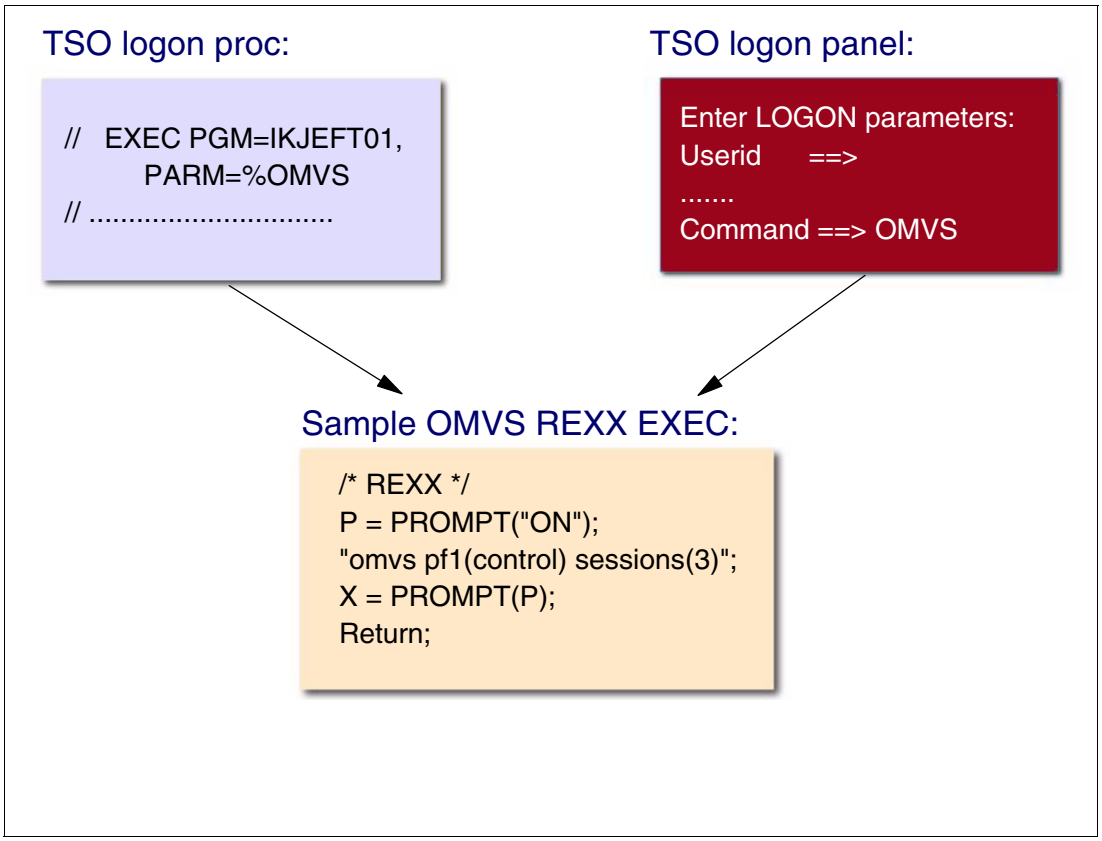


Figure 5-18 Customizing the use of the **OMVS** *command from TSO/E*

Customize the TSO command line for UNIX users

The TSO/E access to the shell can be customized by the following options:

- The logon procedure for users that want to go directly into the shell when they log on to TSO/E can be modified to issue the OMVS command.

  Select or create a TSO/E logon procedure for users who wish to invoke the shell automatically when they log on to TSO/E.

  In the logon procedure, add a PARM parameter to the EXEC statement for program IKJEFT01.

  ```
  //      EXEC  PGM=IKJEFT01, ...
  //            PARM=OMVS
  ```

- A user can then type the OMVS command on the TSO/E logon panel command line.
- To customize the OMVS command for all shell users, you can create a REXX exec with the customized options. Then specify the name of the REXX exec in the PARM parameter, instead of with the OMVS command. In the exec, for example, you can specify changes like these:
  - Use of the 3270 alarm
  - Number of sessions (default is 1)
  - The key or keys to be used for escape
  - The settings for the function keys

- The table to be used for code page conversion
- Shared address space

Using the prompt option

When your profile allows for prompting, the PROMPT function can set the prompting option on or off for interactive TSO/E commands, or it can return the type of prompting previously set. When prompting is on, execs can issue TSO/E commands that prompt the user for missing operands.

The PROMPT function can be used only in REXX execs that run in the TSO/E address space. To set the prompting option on, use the PROMPT function followed by the word `'ON'` enclosed within parentheses:

```
P = PROMPT('ON')
```

To reset prompting to a specific setting saved in variable `P`, write:

```
X = PROMPT(P)
```

Note: The time limit for a shell user is the same as the TSO/E timeout.

5.19 Shell environment variables

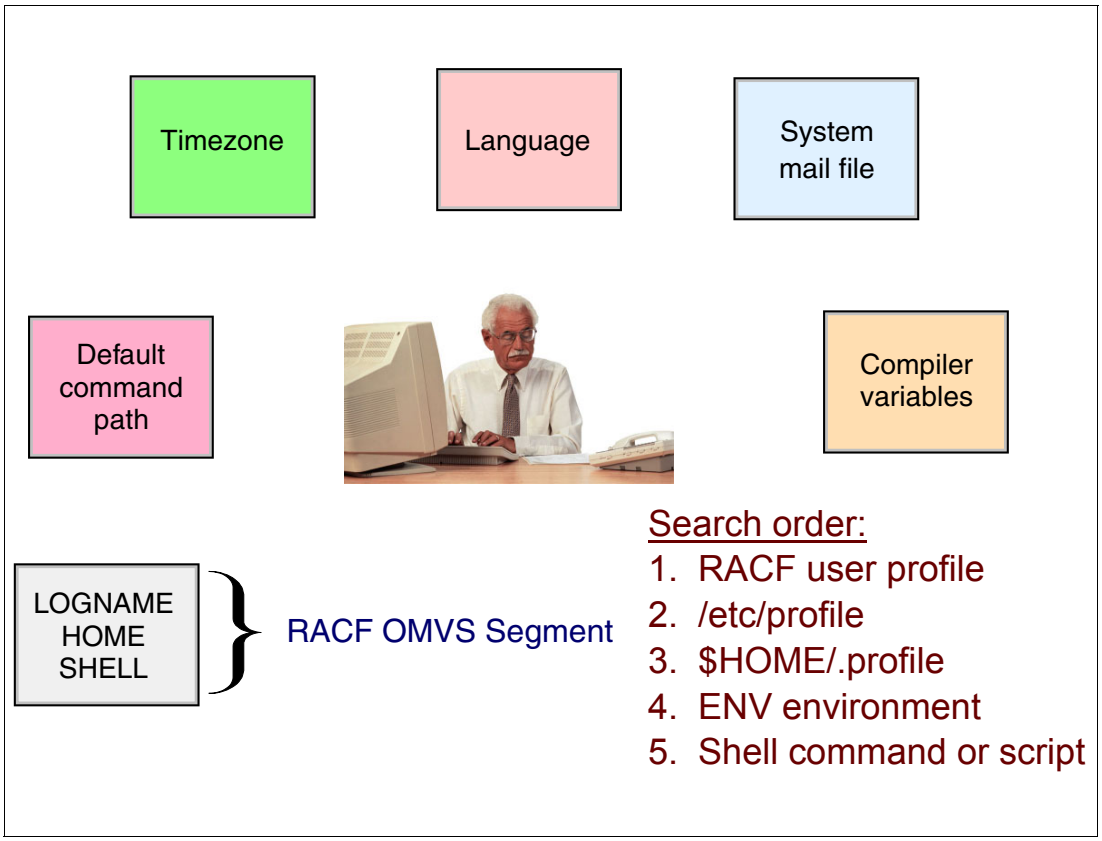


Figure 5-19 Specifying environment variables for the shell user

Setting the shell user environment

An environment variable is a variable that describes the operating environment of a process and typically includes information about the home directory, command search path, the terminal in use, and the current time zone.

Depending on the commands used to set it, an environment variable can be global (used for all processes), local (used only for the current process), or can be exported (used for the current process and for any other processes created by the current process).

Search order for environment variables

Setting an environment variable is optional. If a variable is not set, it will have no value. The following is a list of the places where environment variables can be set:

- By the system programmer:
  - RACF user profile

    Sets the LOGNAME, HOME, and SHELL environment variables from data in the RACF user profile, which was specified in the RACF ADDUSER.
  - /etc/profile

    A system-wide file that sets environment variables for all z/OS shell users. This file is only run for login shells.

- By the shell users:
  - $HOME/.profile

    Sets environment variables for individual users. This file is only run for login shells.
  - The file named in the ENV environment variable in $HOME/.profile

    This file is run for both login shells and subshells.
  - A shell command or shell script of the user.

5.20 Customizing your shell environment

- When you start the shell, these files are used to determine your settings:
  - Order of overrides:
    - /etc/profile
    - $HOME/.profile
    - A file named in ENV environment variable in: $HOME/.profile which points to a .setup file

Figure 5-20 Customizing the user shell environment

User shell environment

When you start the z/OS shell, it uses information in three files to determine your particular needs or preferences as a user. The files are accessed in this order:

- /etc/profile
- $HOME/.profile
- The file that the ENV variable specifies

Settings established in a file accessed earlier can be overwritten by the settings in a file accessed later.

/etc/profile

The /etc/profile file provides a default system-wide user environment. The system programmer may modify the variables in this file to reflect local needs (for example, the time zone or the language). If you do not have an individual user profile, the values in the /etc/profile are used during your shell session.

/$HOME/.profile

The $HOME/.profile file (where $HOME is a variable for the home directory for your individual user ID) is an individual user profile. Any values in the .profile file in your home directory that differ with those in the /etc/profile file override them during your shell session. z/OS provides a sample individual user profile. Your administrator can set up such a file for you, or you can create your own.

5.21 Global variables in /etc/profile

```
# Improve shell performance
if [ -z "$STEPLIB" ] && tty -s;
then
    export STEPLIB=none
    exec sh -L
fi
# Set the time zone as appropriate.
TZ=EST5EDT
# Set a default command path, including current working dir (CDW)
PATH=/bin:.
# Sets the path environment variables
LIBPATH=/lib:/usr/lib:.
MANPATH=/usr/man/%L
NLSPATH=/usr/lib/nls/msg/%L/%N
# Sets the language
LANG=C
# Sets the name of the system mail file and enables mail notification.
MAIL=/usr/mail/$LOGNAME
# Export the values so the system will have access to them.
export TZ PATH LIBPATH MANPATH NLSPATH MAIL LANG
# Set the default file creation mask - umask
umask 022
# Set the LOGNAME variable readonly so it is not accidentally modified.
readonly LOGNAME
```

Figure 5-21 Customizing the global /etc/profile variables for all users

Customizing /etc/profile

The /etc/profile file is the system-wide profile for the z/OS shell users. It contains environment variables and commands used by most shell users.

All shell users will get the environment variables set by this profile. Users can override some of the variables by setting the variables in their private profiles or in their shell sessions.

Note: All environment variables are written with capital letters.

Environment variables in /etc/profile

The /etc/profile file contains the environment variables and commands used by most shell users. An IBM-supplied sample is located in /samples/profile. Copy the sample to the /etc directory and make any necessary changes.

STEPLIB The use of STEPLIB=none is recommended to improve performance for shell users. Keep this setting.

TZ The timezone (TZ) is based on the Greenwich Mean Time (GMT), also called Universal Time Coordinated (UTC). Change it to your local time. The time zone used in the example is for the US East Coast (five hours west of UTC or GMT). It also shows the daylight saving time zone that will be used (EDT).

PATH This specifies the default command path where z/OS UNIX will search for the commands and programs a user is executing. If your installation will have programs in for example, /usr/bin, specify:

```
PATH=/bin:/usr/bin
```

In the PATH variable, a colon (:) separates the list of pathnames. The dot (.) represents the current working directory for a user. The order listed in the PATH variable controls the search order.

NLSPATH Specifies the pathname for message catalogs.

LANG Specifies the name of the default locale.

MAIL Specifies the pathname for mail files. Change it if you want to use another pathname for the mail files. The setting for the MAIL variable indicates that each user will use a mail file with the same name as their LOGNAME, and it will be located in the directory specified for the variable. Note that when you use a dollar sign ($) in front of a variable in a shell script, it means that you want to use the value assigned to that variable. $LOGNAME will be interpreted as the user ID of the shell user (see LOGNAME defined in RACF profile).

EXPORT The **`export`** command will export a variable to a subshell. All the environment variables defined will be valid only for the shell session (login shell) a user gets when the user invokes the shell (TSO OMVS or **`rlogin`**). Shell scripts and many shell commands will actually be executed in a separate shell created by the login shell. The shell creates child processes to run the scripts or commands. These subshells will not be aware of the variable settings that were specified in the login shell unless you export the variable specifications. To ensure that the subshell runs with the same environment, use the **`export`** command to export the variables.

UMASK The **`umask`** command was introduced in our discussion on security. The umask value of 022 will result in permission bits: rwxr-xr-x.

Note: Any changes made in /etc/profile will take effect the next time a user invokes the shell.

The contents of the /etc/profile file are actually a shell script. It contains examples of the script programming language with if-then statements, sets values for environment variables, and issues some shell commands (**`export`**, **`umask`**, **`readonly`**).

5.22 User-defined settings

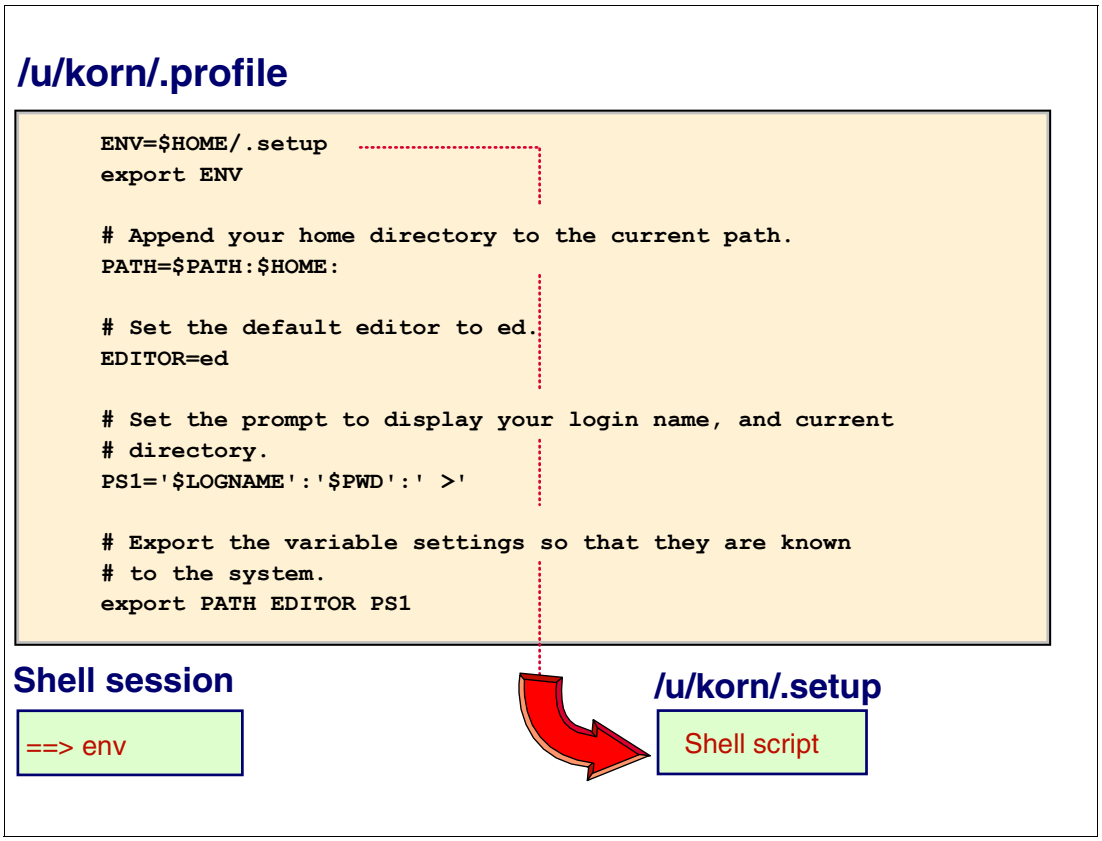


Figure 5-22 Setting the user variables by the user

Shell user environment values

Shell users can set their own values for the environment variables using any of the following methods:

- A $HOME/.profile file. A sample is provided in /samples/.profile/; it can be copied to a shell user's home directory and modified.

  Add the environment variable _BPX_SHAREAS=YES to the user profile. This will cause all shell commands that run in separate processes to create the processes in the same address space as the shell.
- The ENV variable in the $HOME/.profile can be set to a file name which is a shell script that sets environment variables.
- A shell user can use the **env** command to display the environment variables for their session, and to change any of these variables. The change will only last for the length of the session.

The .profile file

The .profile file must be located in a user's home directory. Figure 5-22 shows the contents of the sample provided by IBM. One of the variables in the .profile is called ENV, and this variable can be used to point to another file where a user can add environment variables and shell commands that the user wants to perform when the shell is invoked.

Some of the things a user may want to update in the .profile file are:

- ENV: use if the user has a separate shell script with environment variables.
- PATH: Add the user's home directory to the search path:

```
PATH=$PATH:$HOME:
```

Note how variables can be used in other variables. System programmers and system administrators need to have a $HOME/.profile file with the PATH environment set as follows:

```
PATH=$PATH:/usr/sbin:$HOME:
```

This allows the system programmers to run authorized utilities and to start daemons found in /usr/sbin, as follows:

PS1	Change the shell prompt. The default is #.
TZ	Change if the user is located in a different time zone than the system.
_BPX_SHAREAS=YES	Add this to the profile to reduce the number of address spaces used by a shell user.

The env command

If using the **env** command to set a variable, the setting will only last for the duration of the user's session. The next time the user invokes the shell, this setting is forgotten. Use the .profile file or a shell script for variables that should be the same for each session.

The system programmer should use the /etc/profile file for variables that all users should have set.

5.23 Setting the time zone

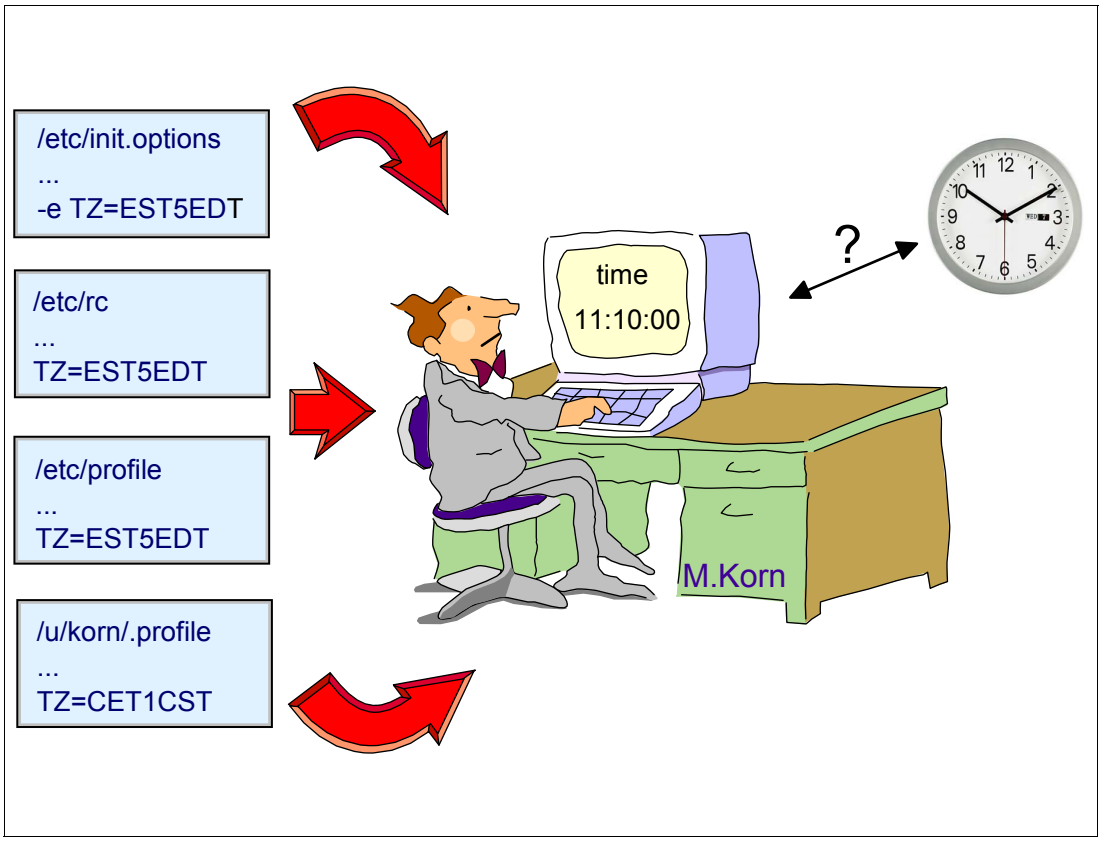


Figure 5-23 Setting the time zone variable

Setting the time zone (TZ)

The time zone is an environment variable used by the time service to set the correct time for z/OS UNIX and for applications running on z/OS UNIX. It is possible to specify a time zone variable in 4 different files. Table 5-1shows the files and when changes made in these files become active.

Table 5-1 Activation schedule for time zone variables

Files	After z/OS UNIX restart	After next user enters the shell	Immediately after change
/etc/init.options	YES	NO	NO
/etc/rc	YES	NO	NO
/etc/profile	YES	YES	NO
$HOME/.profile	YES	YES	NO

General activation order: First the settings in /etc/init.options and /etc/rc become active at kernel initialization time. Afterwards, the settings in /etc/profile are read. If a user has a .profile in his HOME directory that has its own TZ value specified, this setting will be the active one for this user if he enters the shell.

Time zones

The shell and utilities assume that the times stored in the file system and returned by the operating system are stored using the Greenwich Mean Time (GMT) or Universal Time Coordinated (UTC) as a universal reference. In the system-wide /etc/csh.login, the TZ environment variable maps that reference time to the local time specified with the variable. You can use a different time zone by setting the TZ variable in your .login file.

Setting your time zone

If you want to set your time zone to Eastern Standard Time (EST) and export it, specify:

```
setenv TZ "EST5EDT"
```

Where:

- EST is Eastern Standard Time, the local time zone.
- The standard time zone is 5 hours west of the universal reference time.
- EDT is the Eastern Daylight Savings time zone.

Recommendation: It is recommended to have a TZ setting active in /etc/init.options and in /etc/profile. If there are daemons to start via the /etc/rc file, export the TZ variable before starting the daemons. If you want the correct time on any replies inside the shell, you will have to specify the TZ variable at least in the user profiles ($HOME/.profile) if no specifications were made in the system-wide profile /etc/profile where all users are affected.

5.24 Customizing the C89/CC compilers

/etc/profile

```
# Start of c89/cc/c++ customization section
# ================================================================
# High-Level Qualifier "prefixes" for data sets used by c89/cc/c++:
# Compiler:
   export _C89_CLIB_PREFIX="CBC"
# Prelinker and runtime library:
   export _C89_PLIB_PREFIX="CEE"
#
# OS/390 system data sets:
   export _C89_SLIB_PREFIX="SYS1"
# Compile and link-edit search paths:
#   Compiler include file directories:
    export ${_CMP}_INCDIRS="/usr/include /usr/lpp/ioclib/include"
#   Link-edit archive library directories:
    export ${_CMP}_LIBDIRS="/lib /usr/lib"
# Esoteric unit for data sets:
# ================================================================
#   Unit for (unnamed) work data sets:
    export ${_CMP}_WORK_UNIT="SYSALLDA"
```

Figure 5-24 Customizing the c89/CC compiler options

Compiler customization

As we mentioned earlier, the /etc/profile file provides a default system-wide user environment. Any environment variables that are set in /etc/profile affect every shell user unless they override these settings in their own $HOME/.profile or in a setup script file that is pointed to by the ENV variable.

For each z/OS release, there is a default compiler for c89, cc, or c++ (cxx) to use. Optionally, you can choose to use a non-default compiler. This section lists the compiler choices for each release, including the default compilers; the environment variable settings for each compiler are identified.

c89/cc/c++ utilities

The c89/cc/c++ utilities use a number of environment variables. The default values are specified as comments in the /samples/csh.login file that is shipped with each release. The naming conventions for the environment variables are as follows:

- c89 begin with the prefix _C89
- cc begin with the prefix _CC
- c++ begin with the prefix _CXX

If the C/C++ Class Library DLLS are used in building your executables (the default for the c++ utility), then this will also target your executable for the same level of C/C++ Class Library.

In the section titled `'c89/cc/c++ customization section'`, remove the comment for the following export commands to tell the C/C++ compiler what the high level qualifier of the C/C++, Language Environment and z/OS target library data sets are. Check the high level qualifiers on your system to make sure they agree with these default values.

Use ISPF 3.4 and look at your target libraries to check. If they agree, there is no need to uncomment these lines because they are the default setting.

```
#export _C89_CLIB_PREFIX="CBC"
#export _C89_PLIB_PREFIX="CEE"
#export _C89_SLIB_PREFIX="SYS1"
```

Note: SYS1 refers to the high level qualifier of CSSLIB, MACLIB and MODGEN on your system.

If your high level qualifiers do not match, then make the changes here and uncomment the three export statements. See the Environment Variables section of the c89/cc/c++ command discussion in *z/OS UNIX System Services Command Reference*, SA22-7802 for more information.

If the system is not configured with an esoteric unit SYSDA, or some other esoteric unit is to be used for VIO temporary unnamed work data sets set by c89, the following environment variable needs to be set. Specifying a null value for this variable ("") results in c89 using an installation-defined default for the UNIT. The environment variable is shown being set to the default value:

```
#export _C89_WORK_UNIT="SYSDA"
```

The environment variables used by the cc utility have the same names as the ones used by c89 except that the prefix is _CC instead of _C89. Likewise, for the c++ (cxx) utility, the prefix is _CXX instead of _C89. Normally, you do not need to explicitly assign the environment variables for all three utilities. These **`eval`** commands set the variables for the other utilities, based on those set for c89.

```
#eval "export $(typeset -x | grep "^_C89_" ...
#eval "export $(typeset -x | grep "^_C89_" ...
```

5.25 Code page tables

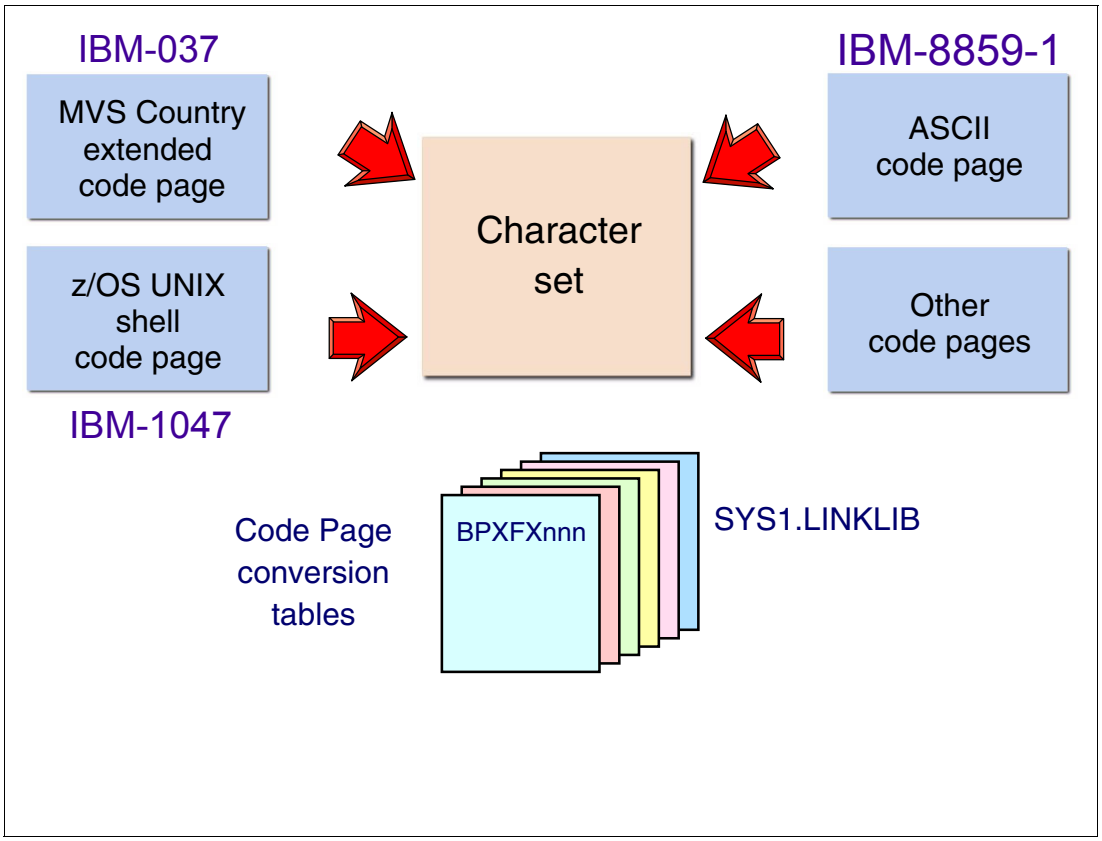


Figure 5-25 Defining code page tables

Code page tables

A code page for a character set determines the graphic characters produced for each hexadecimal code. The code page is determined by the programs and national languages being used.

If the shell is using a locale generated with code pages IBM-1047, an application programmer needs to be concerned about “variant” characters in the POSIX portable character set whose encoding may vary from other EBCDIC code pages. For example, the encodings for the square brackets do not match on code pages IBM-037 and IBM-1047.

For most countries, MVS uses one code page and the z/OS UNIX shell uses a different one. To complicate the matter, many users have workstations which use an ASCII-based code page. When moving data between these environments, the data may have to be converted between the code pages to maintain the same characters. This may result in different hexadecimal codes depending on the code pages.

Conversion tables in SYS1.LINKLIB

There are code page conversion tables located in SYS1.LINKLIB which can be used to convert between the z/OS code page and the shell code page for the national languages supported by the z/OS UNIX shell.

The source for these code page tables can be found in SYS1.SAMPLIB. They can be used as samples for creating a new table if an installation has special requirements.

5.26 Specifying a code page

TSO

```
OMVS   CONVERT((BPXFX111))
OMVS   CONVERT('SYS1.XLATE(BPXFX450)')
```

RLOGIN/TELNET

```
$
$ Issue chcp only if not using TSO/E OMVS
$
if
  test "$_BPX_TERMPATH" != "OMVS"
  then
  chcp -a IBM-850 -e IBM-500
fi
```

Figure 5-26 Specifying code pages

Specifying code pages

The OMVS command has a CONVERT option that lets you specify a conversion table for converting between code pages for the shell accesses by a TSO/E user. The table you want to specify depends on the code pages you are using in MVS and in the shell. For example, if you are using code page IBM-037 in MVS and code page IBM-1047 in the shell, specify the following when you enter the OMVS command:

```
OMVS CONVERT((BPXFX111))
```

The BPXFX111 translation table is for z/OS (code page 00037) to z/OS UNIX (code page 1047) translation. This would be used by most shell users in the U.S.

IBM has supplied many code pages for different countries with the product. If you were in Switzerland you might invoke the shell with the BPXFX450 conversion table. If you leave out the data set name, the normal z/OS search order is used to find the module.

If you are accessing the shell through the rlogin or telnet interface you do not use the **OMVS** command. You must change to the appropriate code page by issuing the **chcp** command. This could be issued by the user or be put in a profile for them.

The example in Figure 5-26 shows the code for a script that checks to be certain that **OMVS** was not issued, and then issues the **chcp** with an ASCII conversion table of IBM-850 and an EBCDIC table of IBM-500. You may also want to change the environment variables that reflect local formats at the same time.

5.27 Internationalization variables (locales)

- A locale is the subset of your environment that deals with language and cultural conventions
  - It is made up of a number of categories
  - Each is associated with an environment variable
    - Controls a specific aspect of the environment

LANG
LC_ALL
LC_COLLATE
LC_CTYPE
LC_MESSAGES
LC_MONETARY
LC_NUMERIC
LC_TIME
LC_SYNTAX
NLSPATH

Figure 5-27 Internationalization variables

Locales

A locale is the subset of your environment that deals with language and cultural conventions. It is made up of a number of categories, each of which is associated with an environment variable and controls a specific aspect of the environment. Figure 5-27 shows the categories and their spheres of influence.

Internationalization variables

Internationalization enables you to work in a cultural context that is comfortable for you through locales, character sets, and a number of special environment variables. The process of adapting an internationalized application or program, particular to a language or cultural environment, is termed localization.

Categories

To give a locale control over a category, set the corresponding variable to the name of the locale. In addition to the environment variables associated with the categories, there are two other variables that are used in conjunction with localization, LANG and LC_ALL. All of these variables affect the performance of the shell commands. The general effects apply to most commands, but certain commands such as sort, with its dependence on LC_COLLATE, require special attention to be paid to one or more of the variables; we discuss such cases in the localization section of the command.

The effects of each environment variable are as follows:

LANG Determines the international language value. Utilities and applications can use the information from the given locale to provide error messages and instructions in that locale's language. If the LC_ALL variable is not defined, any undefined variable is treated as though it contained the value of LANG.

LC_ALL Overrides the value of LANG and the values of any of the other variables starting with LC_.

LC_COLLATE Identifies the locale that controls the collating (sorting) order of characters and determines the behavior of ranges, equivalence classes, and multicharacter collating elements.

LC_CTYPE Identifies the locale that defines character classes (for example, alpha, digit, blank) and their behavior (for example, the mapping of lowercase letters to uppercase letters). This locale also determines the interpretation of sequences of bytes as characters (such as singlebyte versus doublebyte characters).

LC_MESSAGES Identifies the locale that controls the processing of affirmative and negative responses. This locale also defines the language and cultural conventions used when writing messages.

LC_MONETARY Determines the locale that controls monetary-related numeric formatting (for example, currency symbol, decimal point character, and thousands separator).

LC_NUMERIC Determines the locale that controls numeric formatting (for example, decimal point character and thousands separator).

LC_TIME Identifies the locale that determines the format of time and date strings.

LC_SYNTAX Identifies the locale that defines the encodings for the variant characters in the portable character set.

NLSPATH A localization variable that specifies where the message catalogs are to be found.

Examples of locales

For example, the following locale specifies that the z/OS shell is to look for all message catalogs in the directory /system/nlslib, where the catalog name is to be constructed from the name parameter passed to the z/OS shell with the suffix .cat.

```
NLSPATH="/system/nlslib/%N.cat"
```

Note: Substitution fields consist of a % symbol, followed by a single-letter keyword. These keywords are currently defined as follows:

```
%N   The value of the name parameter
%L   The value of the LC_MESSAGES category, or LANG, depending on how the
     catopen() function that opens this catalog is coded.
%l   The language element from the LC_MESSAGES category
%t   The territory element from the LC_MESSAGES category
%c   The codeset element from the LC_MESSAGES category
```

The following specifies that the z/OS shell should look for the requested message catalog in name, name.cat, and /nlslib/category/name.cat, where category is the value of the LC_MESSAGES or LANG category of the current locale:

```
NLSPATH=":%N.cat:/nlslib/%L/%N.cat"
```

5.28 Setting the region size

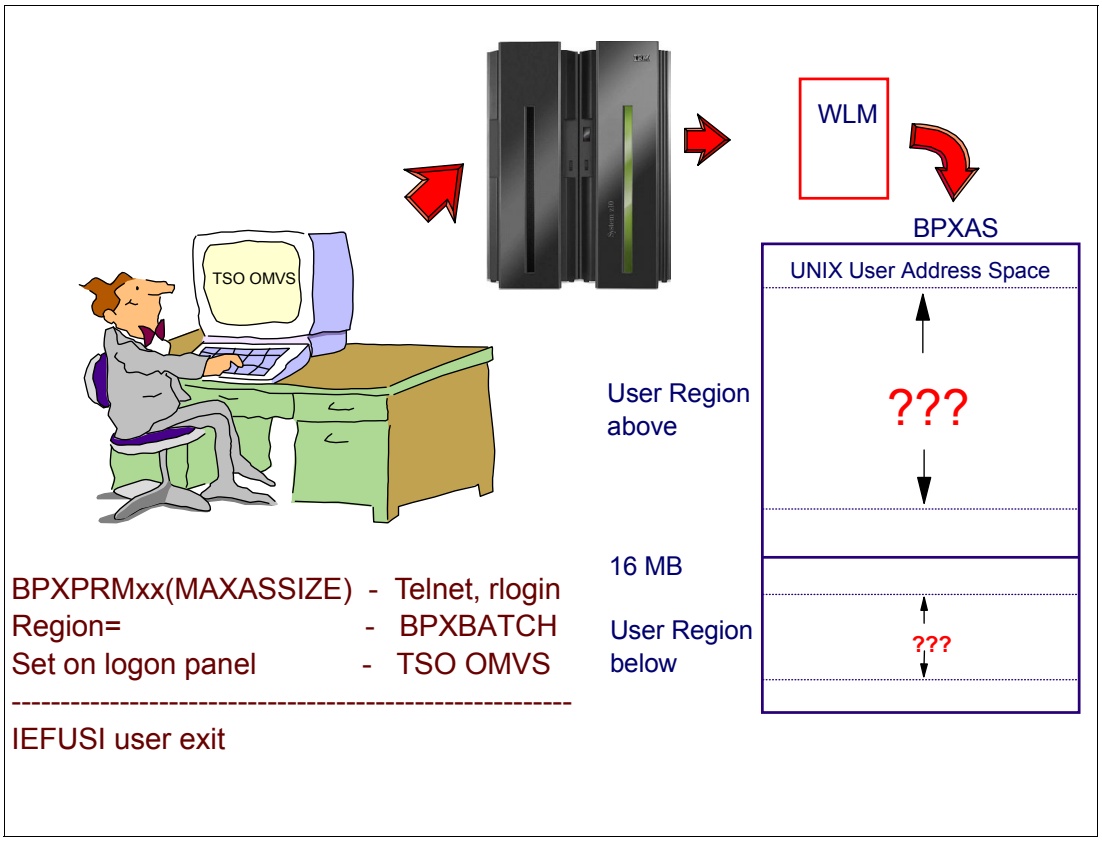


Figure 5-28 Setting the user region size default

Setting region sizes for programs

The region size describes the amount of storage within which a user is allowed to run/execute programs. This value determines what kinds of programs (depending on their size) and how many programs are executable at the same time.

Remote user logons

Telnet and rlogin users get their region size from the MAXASSIZE parameter in the SYS1.PARMLIB with the BPXPRMxx member.

BPXBATCH jobs

If BPXBATCH is used there is a BPXBATCH REGION parameter that describes the region size; however, started procedures that create z/OS UNIX processes use the REGION parameter on the EXEC statement.

TSO users logon panel

In principle, TSO users get the region size from the logon panel to their programs. If a user enters the z/OS UNIX shell by issuing a TSO OMVS command, the parent process doesn't run in the user's address space. Under control of the Work Load Manager, an address space will be created that gets the same region size as the TSO/E user itself. IEFUSI is a user exit where an installation can set the region size and region limit for all programs that run under this job step. Make sure this exit does not change the region size setting for the z/OS UNIX process.

5.29 Setting up printers for shell users

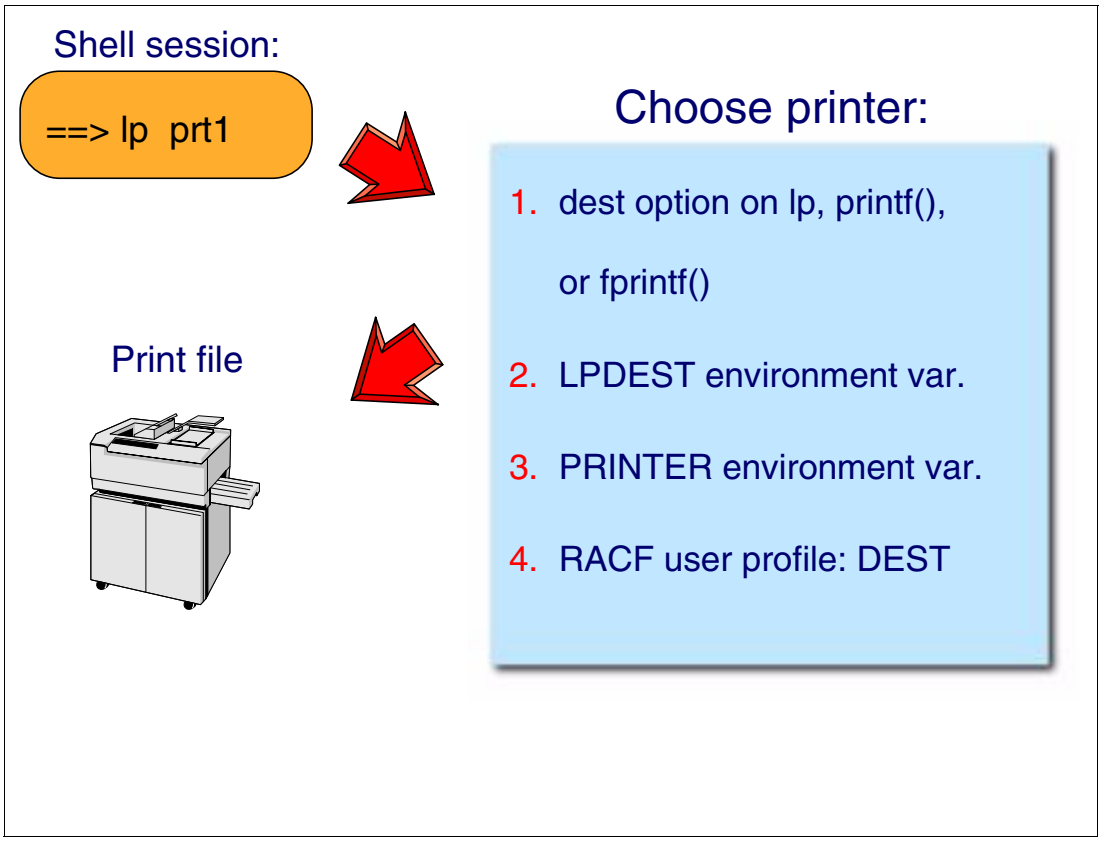


Figure 5-29 Shell user printer defaults

Setting a printer default

Each z/OS UNIX user has a number of default printers specified in different ways. The system will select a printer in the following order:

- The printer in the dest option of the **lp** shell command, or the printf() or fprintf() functions.
- The printer specified in the LPDEST environment variable.
- The printer specified in the PRINTER environment variable.
- The printer in the RACF user profile. It is specified by the DEST parameter of the RACF ADDUSER or ALTUSER command.

The z/OS Infoprint Server, available beginning with OS/390 V2R8, provides much greater support for printing than was available in previous systems.

Inform the application programmers of the destinations or symbolic names of printers you specified in JES initialization statements. The dest option of the **lp** command uses the same destinations as the DEST parameter in the OUTPUT JCL statement. If omitted, the system uses the default printer. The dest option on **lp** can be:

- LOCAL for any installation printer.
- A destination that is defined in a JES2 DESTID initialization statement.

JES controls the print separators, also called cover pages and banner pages, for SYSOUT output for all users, including z/OS UNIX users. To place a user's name and address in the print separator for forked processes, specify the user's name and address in the WORKATTR segment of the RACF user profile.

5.30 Installing books for OHELP

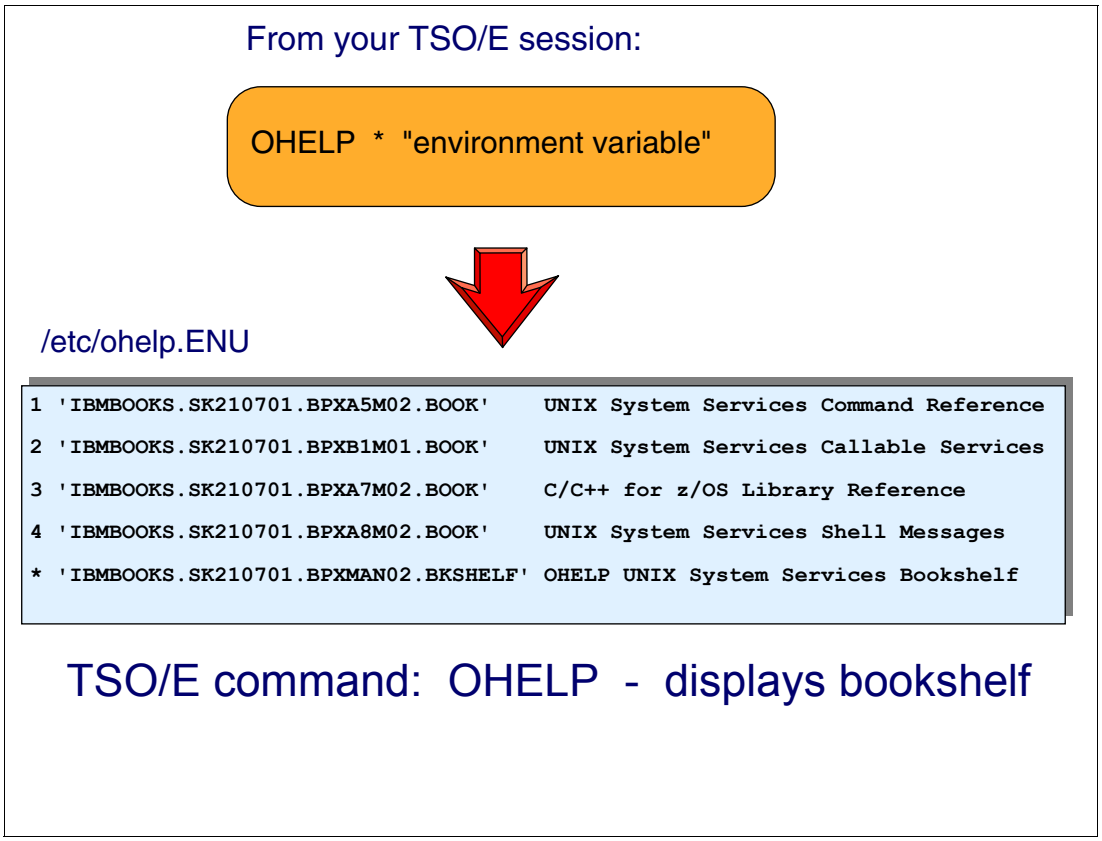


Figure 5-30 Setting the user OHELP command to access the z/OS UNIX books

Access to documentation

The TSO/E OHELP command provides online help for:

- Shell commands
- Callable services
- C functions
- Shell messages

In order to use OHELP, the following must be done:

- BookManager® READ/MVS is required.
- The z/OS UNIX bookshelf must be installed.
- An HFS file must be customized to map the OHELP reference ID with a z/OS UNIX book. Do this using the following steps:
  - Copy the sample file in /samples/ohelp.ENU to /etc/ohelp.ENU.
  - Update the file if you need to change the data set names of the online books, or if you want to add or delete books.

5.31 Using the man command

- The man command gives help information about a shell command
  - man command_name
- To produce a list of all the shell commands for editing
  - man -k edit
- To view descriptions of TSO/E commands such as mount
  - man tsomount

Figure 5-31 Using the man command to access manuals

Access to documentation from the shell

You can use the **man** command to get help information about a shell command. The **man** syntax is:

```
man command_name
```

To scroll the information in a man page, press Enter.

To end the display of a man page, type `q` and press Enter.

To search for a particular string in a system that has a list of one-line command descriptions, use the `-k` option:

```
man -k string
```

For example, to produce a list of all the shell commands for editing, you could type:

```
man -k edit
```

You can use the man command to view manual descriptions of TSO/E commands. To do this, you must prefix all commands with TSO. For example, to view a description of the **MOUNT** command, you would enter:

```
man tsomount
```

-k Searches a precomputed database of syntax lines for information on keywords.

-M -M *path* searches the directories indicated by path for help files. If **-M** is not specified, man uses the path specified in the MANPATH environment variable if set; otherwise man searches /usr/man/%L. The value of the LANG environment variable is substituted for %L in this directory and in the directories specified by MANPATH.

-w Displays only the filename of the file containing the help file.

-x Displays what files man is searching to find the help file. **-x** also displays any errors man encounters in extracting man pages from online book files.

5.32 Enabling various tools

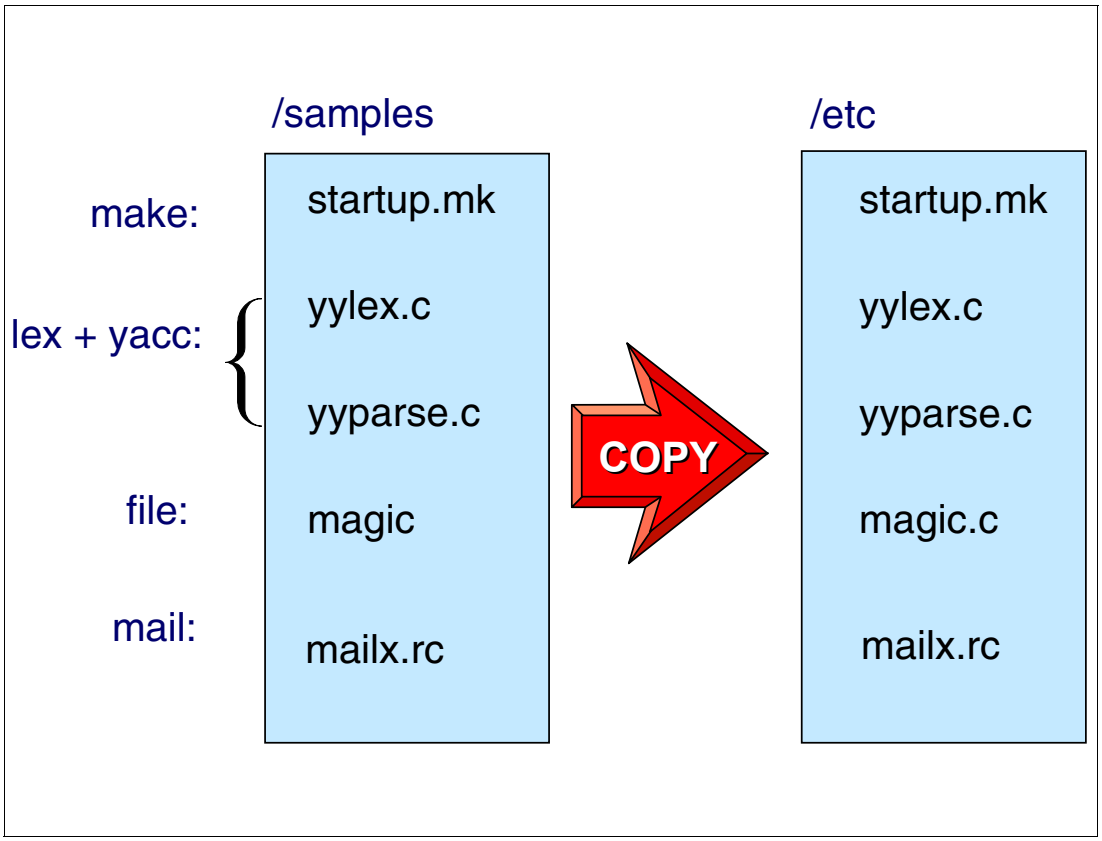


Figure 5-32 Customizing shell utilities

Shell utilities

Some of the shell utilities must be customized before they can be used:

make The make utility uses a configuration file. Copy the sample in /samples/startup.mk to /etc/startup.mk.

The make utility helps manage large applications consisting of multiple programs. When an application programmer updates a program, make will help keep all the files/programs in synchronization.

lex For the lex utility, copy the sample in /samples/yylex.c to /etc/yylex.c. This file contains the prototype lex scanner.

lex and yacc are utilities that help an application programmer to write programs. For example, input to lex and yacc describes how you want the final program to work. The output from lex and yacc is C source code. lex and yacc are often used to develop programs that analyze and interpret input, for example a compiler.

yacc For the yacc utility, copy the sample in /samples/yyparse.c to /etc/yyparse.c. This file contains the C parser template used by yacc.

file For the file utility, copy the sample in /samples/magic to /etc/magic.c. This file contains a series of templates showing different file types.

The file utility determines the format of each file by inspecting the attributes and (for an ordinary file) reading an initial part of the file. It compares each file on the

command line to templates found in a system-maintained magic file to determine their file type.

If the file is an executable, its addressing mode is determined for output. If the file is not an executable, file compares each file to templates found in a magic file to determine their file type.

The file utility then divides files that do not match a template in the magic file into text files and binary data. Then, by reading an initial segment of the text files and making an informed guess based on the contents, file further divides text files into various types such as C programs, assembler programs, files of commands to the shell, and yacc or lex programs.

mailx For the mailx utility, copy the sample in /samples/mailx.rc to /etc/mailx.rc. This file contains variable values and definitions common to all shell users.

The mailx utility program allows users to send messages to one another. The mailbox name is set up in the user's profile. It defaults to /usr/mail/$LOGNAME.

5.33 SVP for z/OS UNIX and tools

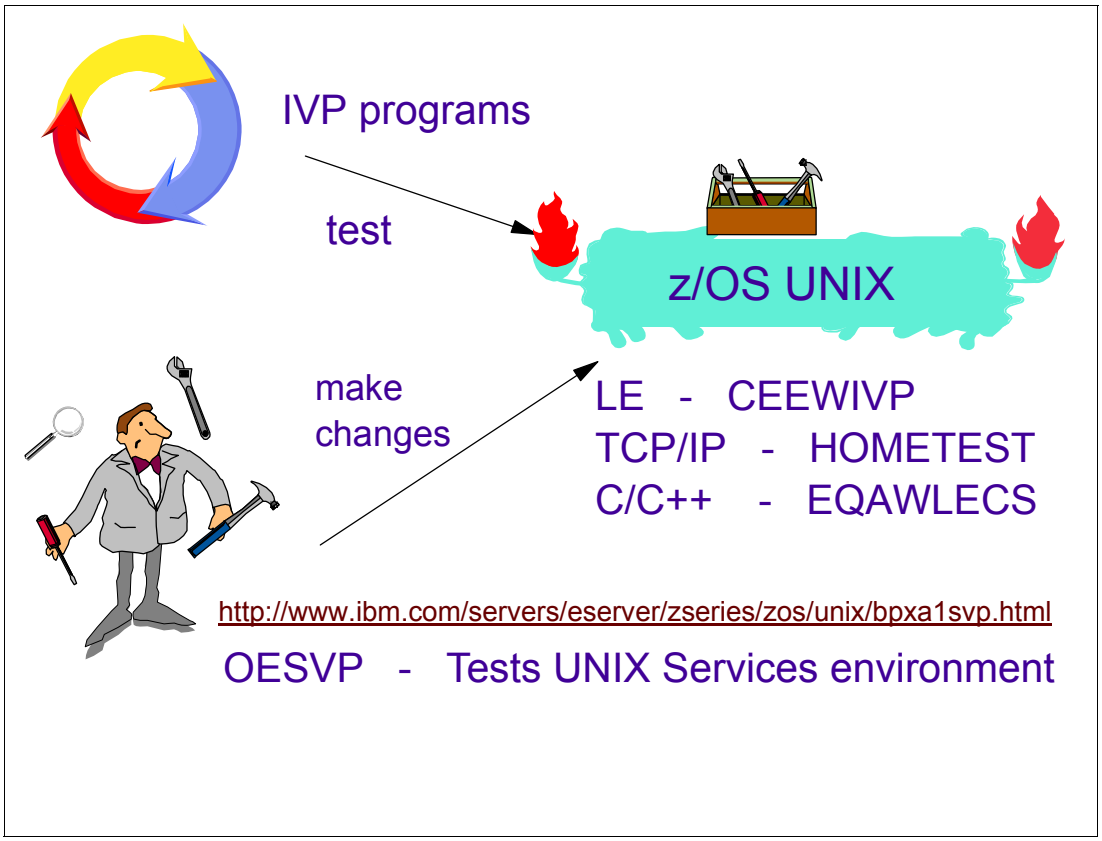


Figure 5-33 Customizing the installation verification program

Setup verification program

The Setup Verification Program (SVP) lets you check for troublesome setup errors before they cause a problem. After you have followed the instructions in *z/OS UNIX Systems Services Planning*, SC28-1890, and completed your setup and customization (including the z/OS shell and utilities), you can run the SVP.

Using the SVP, you can:

- Verify that each user has a UID and OMVS segment defined, and each group has a GID.
- Check for duplicate assignment of UIDs and GIDs.
- Verify that each user has access to and owns a home directory and has read, write, and search access to it.
- Check the permissions for several directories usually set up at installation.
- Check that files in the /dev directory are defined correctly. Reconcile the number of pseudo-ttys and file descriptor files with the BPXPRMxx definitions. On z/OS V1R7 and above, it does minimal checking in the /dev directory because the files are created by the system as needed.
- Verify that the z/OS UNIX shell will run.
- Verify that the OMVS command will run.
- Check customization for utilities. The program checks or does the following:
  - Files that have been copied from /samples to /etc

- terminfo files
- Settings for some environment variables
- Ability to compile and run a program
- Performs various other checks

After customizing the different tools and changing the environment, you have to test the new settings with the related Setup Verification Programs (SVP).

- LE environment

  To verify that Language Environment was installed properly, run CEEWIVP, which is found in your SCEESAMP library. Refer to the comments in the job for instructions, expected condition codes, and expected output.

- IBM Communications Server IP

  Verify your host name and address configuration with HOMETEST. This program will test the system configuration defined by the HOSTNAME, DOMAINORIGIN, and NSINTERADDR statements in the TCPIP.DATA data set.

- C/C++ with Debug Tool

  If you are using the C/C++ debug tools, you have to submit the following jobs in this order:

  ```
  hlq.SCBCJCL(EQAWLECS)
  hlq.SCBCJCL(EQAWLU62)
  hlq.SCBCJCL(EQAWIVP2)
  ```

- C/C++ compiler

  If C/C++ has been enabled, verify that the following C/C++ components are installed properly:

  - C/C++ compilers

    Run the jobs HLB4801F and HLB4801G from the ServerPac Installation Dialog.

  - C/C++ Host Performance Analyzer

    In order to test the C/C++ Host Performance Analyzer you can run the two jobs HLB4801M and HLB4801N from the ServerPac Installation Dialog, or find the same JCL in hlq.SCTVJCL with the names CTVFINT and CTVFUNK.

  - C/C++ IBM Open Class® Library

    To test the Open Class Library, use the jobs HTV4821R, HTV4821S and HTV4821T, which can be run from the ServerPac Installation dialog or from the hlq.SCLBJCL library with following names: CLB3JIV1, CLB3JIV2, and CLB3JIV3.

- z/OS UNIX environment and tools

  To test the z/OS UNIX environment and tools use the program oesvp, which can be downloaded from http://www-1.ibm.com/servers/eserver/zseries/zos/unix/bpxa1svp.html with the name oesvp.exec.bin.

5.34 Setup Verification Program (SVP)

```
          OS390 Unix System Services Setup Verification Program
Command ===> _

Select the checks you would like to perform:
_  Kernel
_  Sysplex
_  Users
_  Groups
_  User and group setup
_  User home directories
_  Permissions for basic directories
_  /dev directory
_  Shell environment
_  OMVS command
_  Utility customization
```

Figure 5-34 Using the Setup Verification Program

Setup Verification Program

The SVP for z/OS UNIX ISPF panels is shown Figure 5-34. During the z/OS shell test, do not interrupt the process when the indicator RUNNING/INPUT changes; instead, press PF10.

This utility requires ISPF version 4.1 or higher, as well as z/OS V1R7 or higher.

For more information, see *ServerPac: Installing Your Order* or the ServerPac Program Directory.

Setup verification program

The Setup Verification Program (SVP) lets you check for troublesome setup errors before they trip you up. After you have followed the instructions in *z/OS UNIX System Services Planning*, GA22-7800, and completed your setup and customization (including the z/OS shell and utilities), you can run the SVP.

Select a specific check

Select the specific checks you wish to run, as follows:

- Kernel

  Displays kernel configuration information. If you believe this information is not how you intended to set up your system, you should check your BPXPRMxx parmlib member and startup procedures.

- Sysplex

  Displays the names of the active systems using SYSPLEX(YES) and verifies that some of the special symlinks are set up correctly.
- Users

  Verify that each user that has a UID also has the default group assigned a GID.
- Groups

  Checks that a GID is assigned to every group.
- User and group setup

  Checks for duplicate assignment of UIDs and GIDs.
- User home directories

  Verifies that each user has access to and owns his home directory and has read, write, and search access to that directory. If automount is used for home directories, all file systems will be mounted.
- Permissions for basic directories

  This check surveys the permissions for several directories normally found on the system.
- /dev directory

  Verifies that customary files are defined correctly. This also reconciles the number of pseudo-ttys and file descriptor files with parmlib definitions.
- Shell environment

  Verifies that the shell will run.
- OMVS command

  Verifies that the OMVS command will run.
- Utility customization

  Checks for differences in some files in the /samples directory with the corresponding file in the /etc directory, that the terminfo files have been created, some environment variables are appropriately set, a program could be compiled and run, and various other checks. This verification step can be interactive. If it thinks something should be done, it will ask if you want it to do it.

Requirements for use of the SVP

To use the SVP you must satisfy the following requirements:

- You must be a superuser (UID=0) with RACF SPECIAL authority, or the equivalent.
- Your system must be at any release of z/OS.
- Your system must be at ISPF version 4.1 or higher.
- You can use any security product; RACF is not required.

6

Security customization

This chapter provides an overview of UNIX System Services security. It provides information about how to customize the security definitions for z/OS UNIX System Services. In addition to introducing the basic concepts of UNIX security, this chapter provides the details on how to:

- Define new z/OS UNIX System Services users and groups
- Change existing users and groups for z/OS UNIX System Services
- Set up security for the z/OS UNIX System Services kernel
- Set up security for z/OS UNIX System Services daemons
- Set up security for z/OS UNIX System Services products
- Define users and groups to RACF
- Manage superusers
- Define permissions bits and how they are used
- Define RACF profiles for z/OS UNIX System Services
- Protect the daemons programs
- Understand server security
- Utilize various auditing capabilities

The security configuration is an important prerequisite to enabling z/OS UNIX System Services. At least minimal security work is required just to make z/OS UNIX start. After that, the security configuration becomes increasingly more complex as users are defined and workload is brought onto the system. This chapter gives you a comprehensive view of the work required to set up z/OS UNIX in a way that is secure and in line with the needs of typical business organizations.

© Copyright IBM Corp. 2011. All rights reserved.

6.1 RACF OMVS segments

User profile

Userid	Default Group	Connect Groups		TSO	DFP	OMVS		
						UID	Home	Program
SMITH	PROG1	PROG1	PROG2	...	...	15	/u/smith	/bin/sh

Group profile

Groupid	Superior Group	Connected Users				OMVS GID
PROG1	PROGR	SMITH	BROWN	...	...	25

Group profile (no OMVS segment)

Groupid	Superior Group	Connected Users			
PROG2	PROGR	SMITH	WHITE	...	...

Figure 6-1 Defining RACF OMVS segments

RACF profile OMVS segments

The RACF user profile definition was expanded with a segment called OMVS for z/OS UNIX support. All users and programs that need access to z/OS UNIX must have a RACF user profile defined with an OMVS segment which has, as a minimum, a UID specified. A user without a UID cannot access z/OS UNIX.

RACF user profile

Within this profile is an OMVS segment that defines the user as a z/OS UNIX user. The OMVS segment has the following three fields:

UID A number from 0 to 2147483647 that identifies a z/OS UNIX user. A z/OS UNIX user must have a UID defined.

HOME The name of a directory in the file system. This directory is called the home directory and becomes the current directory when the user accesses z/OS UNIX. This field is optional.

The home directory is the current directory when a user invokes z/OS UNIX. During z/OS UNIX processing, this can be changed temporarily by using the **cd** (change directory) shell command. The command will not change the value in the RACF profile. The directory specified as home directory in the RACF profile must exist (be pre-allocated) before a user can invoke z/OS UNIX. If a home directory is not specified in RACF, the root (/) directory will be used as default.

PROGRAM The name of a program. This is the program that will be started for the user when the user begins a z/OS UNIX session. Usually this is the program name for the z/OS UNIX shell. This field is optional.

RACF group profile

The RACF group also has a segment called OMVS to define z/OS UNIX groups. It contains only one field:

GID A number from 0 to 2147483647 that identifies a z/OS UNIX group.

Example segment

The example in Figure 6-1 on page 182 shows a user profile for TSO/E user ID SMITH which is connected to two groups, PROG1 and PROG2. SMITH is defined as a z/OS UNIX user because he has a UID specified. His home directory is /u/smith and he will get into the shell when he issues the OMVS command because the name of the shell, /bin/sh, is specified as the program name.

A program that will access z/OS UNIX and run as a started task (for example, RMFGAT) or a daemon (for example, the inetd daemon, which is used for remote login (rlogin) to the shell via TCP/IP) must also be defined to RACF with a user profile and a UID specified. This type of user does not require a home directory or a program specified in the OMVS segment. The home directory and program are important for people's user IDs.

The RACF profile for a group is also extended with an OMVS segment. A z/OS UNIX group is a RACF group with a GID specified in the OMVS segment. The figure shows that group PROG1 is also a z/OS UNIX group with a GID value of 25. The group PROG2 does not have an OMVS segment and therefore is not a z/OS UNIX group.

Connected users

If a user is connected to several groups, the user has a level of group authority for each group. If you are a connected user who has the group-SPECIAL attribute, you have the authority to:

- Define new users to RACF (provided you also have the CLAUTH attribute for the USER class)
- Connect and remove users from the group
- Delegate and change group authorities and set the default UACC for all new resources belonging to members of the group
- Modify, list, and delete the group profile
- Define, delete, and list the names of the subgroups under the group
- Specify the group terminal option

6.2 z/OS UNIX UIDs and GIDs

- UID = user identifier
  - Number in range 0 - 2,147,483,647
    - But.... 0 - 16,777,215 due to `pax` protocol
  - 0 = Superuser (Root)
- GID = group identifier
  - Number in range 0 - 2,147,483,647
    - But.... 0 - 16,777,215 due to `pax` protocol
- /etc/passwd

Figure 6-2 z/OS UNIX users UIDs and GIDs

Defining UIDs and GIDs

UNIX systems have a concept of users and groups similar to RACF. A user identifier (or UID) is a number between 0 and some large number that varies between brands of UNIX. User numbers do not have to be unique and it is possible (though not recommended) for several users to share the same UID, and even be logged on at the same time. UNIX sees these users as being the same entities and they receive the same levels of authorization. A user with UID=0 is what is called a superuser ("root" on some brands of UNIX). A superuser has unlimited authority within a UNIX environment. For obvious reasons, UID=0 needs to be strictly controlled.

Users are all related to a group. Groups allow authority to be controlled in a more economical way, in that giving access to a group is a lot easier than giving access to a few hundred users individually. A group identifier (or GID) is a number between 0 and some large number that varies between brands of UNIX. Any number of users can share the same GID.

The z/OS UNIX implementation of UNIX allows UID and GID numbers up to 2,147,483,647, but due to restrictions in UNIX design, it is recommended that the maximum number used be no more than 77,777,777.

Other UNIX platform security

UNIX systems typically store their user/group information in a file called `/etc/passwd`. This file does not exist under z/OS UNIX, because RACF (a like security product) is used instead.

6.3 z/OS UNIX users and groups

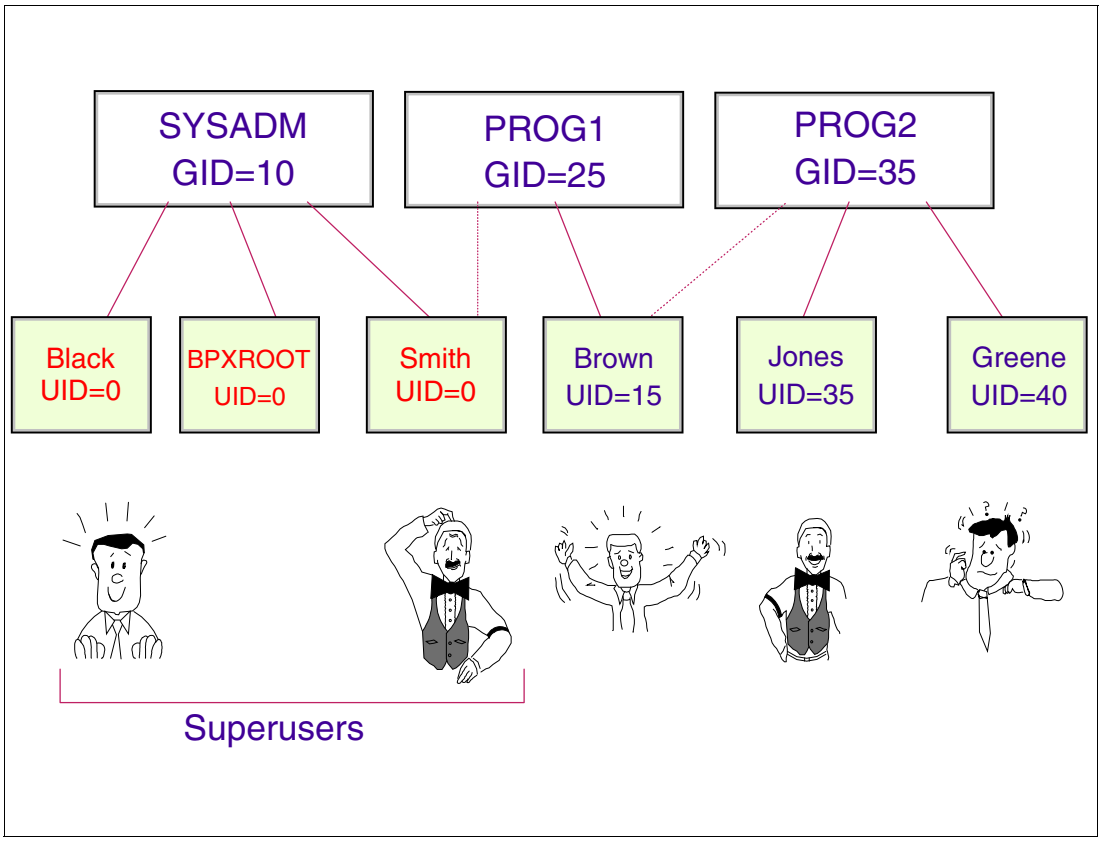


Figure 6-3 Defining z/OS UNIX users and groups

Defining z/OS UNIX users and groups

z/OS UNIX users are TSO/E user IDs with a RACF segment called OMVS defined for z/OS UNIX use. All users that want to use z/OS UNIX services must be defined as z/OS UNIX users. Similar to users in a UNIX system, z/OS UNIX users are identified by a UID (user identification). The UID has a numerical value.

There are two types of users:

- User (regular user)
  - Identified by a non-zero UID
- Superuser (authorized/privileged user). A superuser can be any of the following:
  - A z/OS UNIX user with a UID=0
  - A started procedure with a trusted or privileged attribute in the RACF started procedures table

Superusers

The concept of *superuser* comes from UNIX. Sometimes it is also referred to as *root* authority. A superuser can:

- Pass all z/OS UNIX security checks, so that the superuser can access any file in the hierarchical file system. A superuser does not get any additional authorities to access MVS/ESA resources. The authority is limited to the z/OS UNIX component.

- Manage z/OS UNIX processes and files.
- Have an unlimited number of processes running concurrently. For a started procedure, this is true only if it has a UID of 0. It is not true for a trusted or privileged process with a different UID.
- Change identity from one UID to another.
- Use the **`setrlimit`** command to increase any of the system limits for a process.

A superuser is usually a system administrator, or it can be a started procedure that is authorized by the RACF started procedures table or the RACF STARTED class.

z/OS UNIX users belong to one or more groups in the same way that TSO/E users belong to groups. A z/OS UNIX group is a RACF group with a GID defined. The GID has a numerical value.

Reminder: Multiple users may have the same UID.

BPXROOT

If the target UID is 0 and a userid is not known, the setuid service sets the MVS userid to BPXROOT, or to a userid that is specified as a PARMLIB option during installation. BPXROOT is set up during system initialization as a superuser with a UID of 0. The BPXROOT userid is not defined to the BPX.DAEMON FACILITY class profile. This special processing is necessary to prevent a superuser from gaining daemon authority.

6.4 BPXROOT user ID

- Define a superuser with a user ID of BPXROOT
  - Allows daemon processes to invoke setuid() for superusers
  - NOPASSWORD option indicates that BPXROOT is a protected user ID
- BPXPRMxx parmlib member
  - SUPERUSER(BPXROOT)

```
ADDUSER BPXROOT DFLTGRP(OMVSGRP)
OMVS(UID(0) HOME('/') PROGRAM('/bin/sh'))
  NOPASSWORD
```

Figure 6-4 Define a BPXROOT user ID

Defining BPXROOT

Define a superuser with a user ID of BPXROOT on all systems so that daemon processes can invoke setuid() for superusers. Specify UID=0, a home directory of / (root), and the program /bin/sh. BPXROOT should not have any special permission to MVS resources. This user ID is used in rare cases where a daemon process tries to change the identity of a process to superuser but does not know the MVS identity of the process. BPXROOT is the default name.

Use of BPXROOT

If the target UID is 0 and a user ID is not known, the setuid service sets the MVS user ID to BPXROOT, or to a user ID that is specified as a parmlib option during installation. BPXROOT is set up during system initialization as a superuser with a UID of 0. The BPXROOT user ID is not defined to the BPX.DAEMON profile in the FACILITY class. This special processing is necessary to prevent a superuser from gaining daemon authority.

NOPASSWORD option

The NOPASSWORD option indicates that BPXROOT is a protected user ID that cannot be used to enter the system by using a password or password phrase. The user ID will not be revoked due to invalid logon attempts.

SUPERUSER statement

On the SUPERUSER statement in the BPXPRMxx parmlib member, specify the user ID that the kernel will use when you need a user ID for UID(0):

```
SUPERUSER(BPXROOT)
```

Note: If not specified, the default is BPXROOT. In that situation, do not permit the BPXROOT user ID to BPX.DAEMON. The BPXROOT user ID is used when a daemon process invokes setuid() to change the UID to 0 and the user name has not been previously identified by getpwnam() or by the _passwd() function. This action prevents the granting of daemon authority to a superuser who is not defined to BPX.DAEMON.

6.5 Superuser with appropriate authority

- 3 ways to assign superuser authority
  - Assigning a UID of 0, which is the least desirable way
    - Okay for special administrators
  - Using the BPX.SUPERUSER resource in the RACF FACILITY class
    - Allows requesting superuser access, but you do not have the access unless you make the request
  - Using the UNIXPRIV class profiles, the preferred way
    - Profiles in the UNIXPRIV class grant RACF authorization for certain z/OS UNIX privileges

Figure 6-5 Defining superuser authority

Defining superuser authority

When you are defining users, you might want to define some of them with appropriate superuser privileges, a UID of 0. There are three ways of assigning superuser authority, as follows:

- Assigning a UID of 0, which is the least desirable way
- Using the BPX.SUPERUSER resource in the FACILITY class
- Using the UNIXPRIV class profiles, the preferred way

While some functions require a UID of 0, in most cases you can choose among the three ways. When choosing among them, try to minimize the number of user IDs (as opposed to started procedures) with a UID(0) superuser authority. To summarize the choices, UID(0) gives you access to all UNIX functions and resources, as is true for all UNIX systems.

BPX.SUPERUSER for superuser authority

However, in z/OS, RACF allows certain users to perform specific privileged functions without being defined as UID(0). BPX.SUPERUSER allows you to request that you be given such access, but you do not have the access unless you make the request.

Using UNIXPRIV for superuser authority

The UNIXPRIV class allows you to do specific privileged functions, such as mounting a file system. Both these definitions are similar to having UID(0) in that, before RACF grants access to a system resource or use of it, the system checks these definitions.

6.6 Commands for superusers

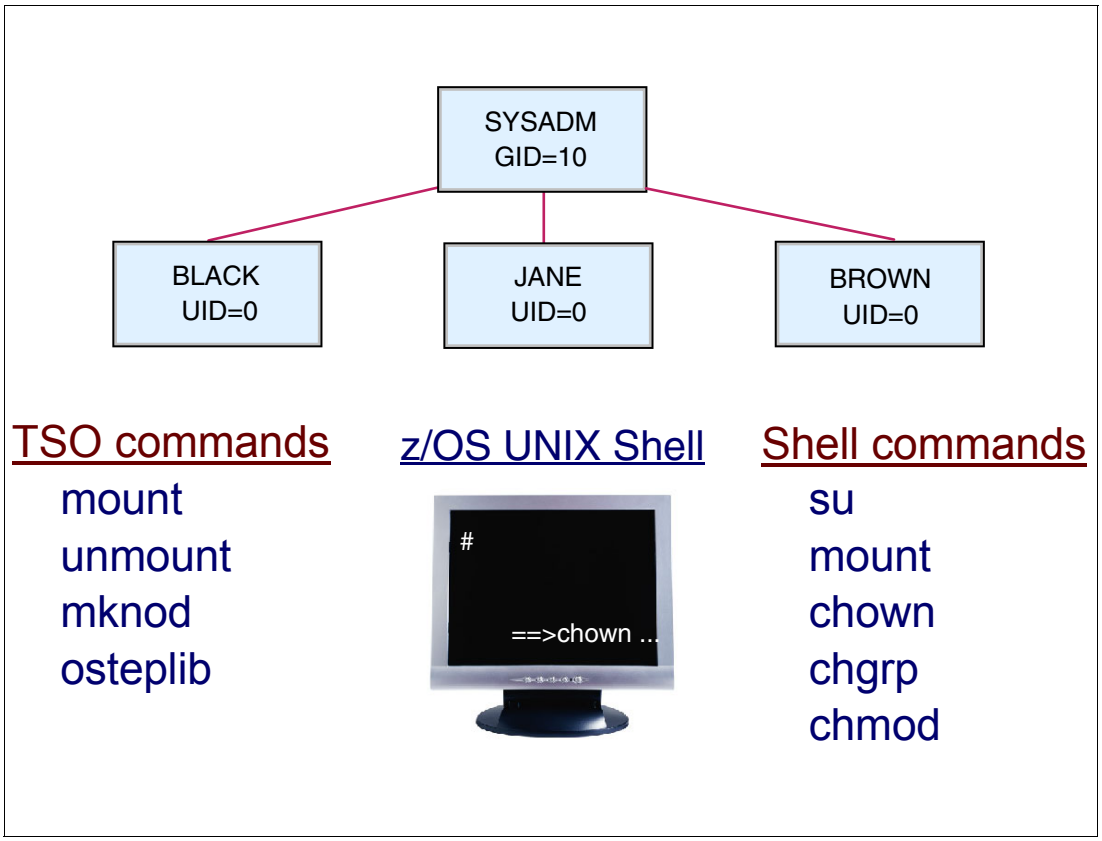


Figure 6-6 Superuser commands

Commands only for superusers

Superusers are special users in a z/OS UNIX environment and they are identified by a UID value of 0. One way of defining superusers is to set the UID to 0 in the user's OMVS segment. Using this method, the user always runs as a superuser. Multiple users can be defined with a UID of 0.

Define some system administrators as superusers by adding an OMVS segment to their user profile. Define the home directory as root (/). Add an OMVS segment to the groups to which these users are connected.

Superuser commands

There are TSO and shell commands that can only be issued by a superuser or the file owner. Several of these commands are shown in Figure 6-6.

`chown` Used to change the owner or group of a file or directory.

`chmod` Changes the access permissions

`mount` Used to mount a file system

`chgrp` Sets the group ID to group for the files and directories

`su` Starts a new shell and lets you operate in it with the privileges of a superuser or another user.

6.7 z/OS UNIX security and RACF profiles

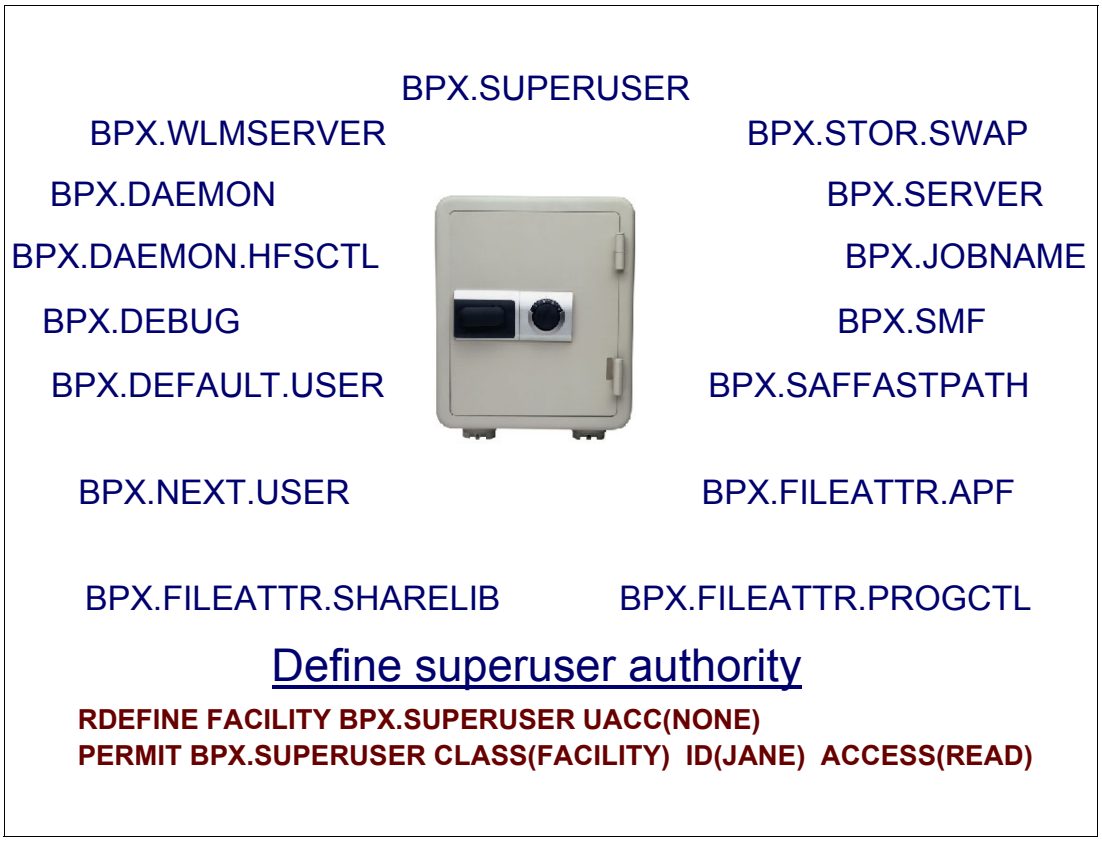


Figure 6-7 z/OS UNIX RACF profiles for security

RACF profiles to provide security for z/OS UNIX

A RACF FACILITY class profile example follows to give a non-zero user superuser authority:

```
RDEFINE FACILITY BPX.SUPERUSER UACC(NONE)
PERMIT BPX.SUPERUSER CLASS(FACILITY) ID(JANE) ACCESS(READ)
```

Using the BPX.SUPERUSER resource in the FACILITY class is another way for users to get the authority to do most of the tasks that require superuser authority. When they need to perform superuser tasks, they can switch to superuser mode using the **su** command or the “Enable superuser mode (SU)” option in the ISPF shell.

RACF FACILITY class resource names

The following are FACILITY class profiles you may want to define for z/OS UNIX:

- **BPX.DAEMON**

  BPX.DAEMON serves two functions in the z/OS UNIX environment:

  – Any superuser permitted to this profile has the daemon authority to change MVS identities via z/OS UNIX services without knowing the target user ID's password. This identity change can only occur if the target user ID has an OMVS segment defined.

    If BPX.DAEMON is not defined, then all superusers (UID=0) have daemon authority. If you want to limit which superusers have daemon authority, define this profile and permit only selected superusers to it.

- Any program loaded into an address space that requires daemon level authority must be defined to program control. If the BPX.DAEMON profile is defined, then z/OS UNIX will verify that the address space has not loaded any executables that are uncontrolled before it allows any of the following services that are controlled by z/OS UNIX to succeed:
  - seteuid
  - setuid
  - setreuid
  - pthread_security_np()
  - auth_check_resource_np()
  - _login()
  - _spawn() with user ID change
  - _password()

- **BPX.DAEMON.HFSCTL**

  Controls which users with daemon authority are allowed to load uncontrolled programs from MVS libraries into their address space.

- **BPX.DEBUG**

  Users with read access to the BPX.DEBUG FACILITY class profile can use **`ptrace`** (via dbx) to debug programs that run with APF authority or with BPX.SERVER authority.

- **BPX.DEFAULT.USER**

  Not all users and groups need to have discrete OMVS segments defined for them. For example:

  - Users who need to use sockets and do not need any other UNIX services. In the past, users could open sockets without any other special permissions.
  - Users who want to run multithreading PL/1 programs. PL1 uses some kernel services, so the OMVS segments are currently required to get dubbed.
  - Users who just want to experiment with the shell and do not have an OMVS segment defined.

  The default OMVS segments will reside in a USER profile and a GROUP profile. The names of these profiles are selected by the installation, and stored in the application data field of BPX.DEFAULT.USER.

- **BPX.FILEATTR.APF**

  Controls which users are allowed to set the APF-authorized attribute in an HFS file. This authority allows the user to create a program that will run APF-authorized. This is similar to the authority of allowing a programmer to update 'SYS1.LINKLIB' or 'SYS1.LPALIB'.

- **BPX.FILEATTR.PROGCTL**

  Controls which users are allowed to set the program-controlled attribute in an HFS file. Programs marked with this attribute can execute in server address spaces that run with a high level of authority.

- **BPX.FILEATTR.SHARELIB**

  Indicates that extra privilege is required when setting the shared library extended attribute via the chattr() callable service. This prevents the shared library region from being misused.

- **BPX.JOBNAME**

  Controls which users are allowed to set their own job names by using the _BPX_JOBNAME environment variable or the inheritance structure on spawn. Users with READ or higher permissions to this profile can define their own job names.

- **BPX.NEXT.USER**

  Enables automatic assignment of UIDs and GIDs. The APPLDATA of this profile specifies a starting value, or range of values, from which RACF will derive unused UID and GID values.

- **BPX.SAFFASTPATH**

  Enables faster security checks for file system and IPC constructs.

- **BPX.SERVER**

  Restricts the use of the **`pthread_security_np()`** service. A user with at least READ or WRITE access to the BPX.SERVER FACILITY class profile can use this service. It creates or deletes the security environment for the caller's thread.

  This profile is also used to restrict the use of the BPX1ACK service, which determines access authority to z/OS resources. Servers with authority to BPX.SERVER must run in a clean program-controlled environment. z/OS UNIX will verify that the address space has not loaded any executables that are uncontrolled before it allows any of the following services that are controlled by z/OS UNIX to succeed:

  - seteuid
  - setuid
  - setreuid
  - pthread_security_np()
  - auth_check_resource_np()
  - _login()
  - _spawn() with user ID change
  - _password()

- **BPX.SMF**

  Checks if the caller attempting to cut an SMF record is allowed to write an SMF record or test if an SMF type or subtype is being recorded.

- **BPX.SRV.userid**

  Allows users to change their UID if they have access to BPX.SRV.userid, where uuuuuuuu is the MVS user ID associated with the target UID. BPX.SRV.userid is a RACF SURROGAT FACILITY class profile.

- **BPX.STOR.SWAP**

  Controls which users can make address spaces nonswappable. Users permitted with at least READ access to BPX.STOR.SWAP can invoke the _mlockall() function to make their address space either nonswappable or swappable.

  When an application makes an address space nonswappable, it may cause additional real storage in the system to be converted to preferred storage. Because preferred storage cannot be configured offline, using this service can reduce the installation's ability to reconfigure storage in the future. Any application using this service should warn the customer about this side effect in their installation documentation.

- **BPX.SUPERUSER**

  Users with access to the BPX.SUPERUSER FACILITY class profile can switch to superuser authority (effective UID of 0).

- **BPX.WLMSERVER**

  Controls access to the WLM server functions _server_init() and _server_pwu(). A server application with read permission to this FACILITY class profile can use the server functions, as well as the WLM C language functions, to create and manage work requests.

6.8 z/OS UNIX security: BPX.SUPERUSER

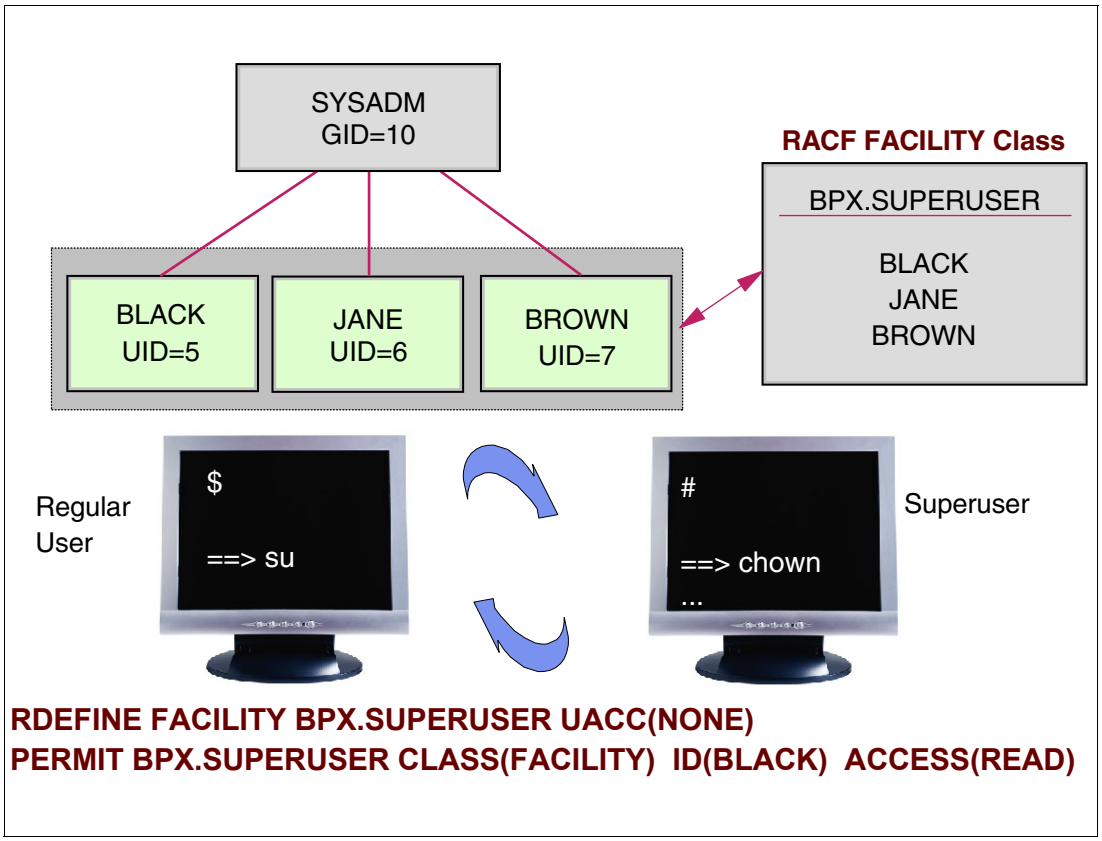


Figure 6-8 Using BPX.SUPERUSER to define superuser authority

Superuser authority with BPX.SUPERUSER profiles

There is an alternative way of defining superusers. Define system administrators in RACF with non-zero UIDs, and give them READ access to a RACF FACILITY class called BPX.SUPERUSER. Users with this authority will be able to temporarily switch to become superuser when this authority is required for administrative tasks. These users can use any of the following methods to switch to superuser:

- In the z/OS UNIX shell, use the command **su** (switch user). This command creates a subshell where the user will have superuser authority and authorized commands can be executed. When the subshell session is ended, the user returns to the first shell session as a regular user.
- Use the ISHELL command to enter the z/OS UNIX ISPF Shell. Select the option to switch to superuser state. The user will then have superuser authority until the user exits the ISHELL environment.
- After gaining superuser authority in the ISHELL, you can do a split screen in ISPF and enter the OMVS command. The z/OS UNIX shell that is started inherits the superuser authority set up in the ISHELL.

The su command

Enter the shell using the OMVS command and then issue the **su** command with no operands. This creates a nested shell that runs with superuser authority. With the **su** command, you can switch to any user ID, including the superuser. A user can switch to superuser authority (with an effective UID of 0, if the user is permitted to the BPX.SUPERUSER resource in the

FACILITY class within RACF. Either the ISPF shell or the **su** shell command can be used for switching to superuser authority.

If you do not specify a user ID, the **su** command changes your authorization to that of the superuser. If you specify a user ID, **su** changes your authorization to that of the specified user ID.

When you change the user ID by specifying a user ID and password or password phrase, you assume the MVS identity of the new user ID even if the user ID has UID 0.

If you use the **-s** option on the **su** command, you will not be prompted for a password. Use this option if you have access to the BPX.SRV.userid SURROGAT class profile. The userid is the MVS user ID associated with the target UID.

To return to your own user ID, type:

```
exit
```

This returns you to the shell in which you entered the **su** command.

chown command

The **chown** command sets the user ID (UID) to the owner for the files and directories named by pathname arguments. The owner can be a user name from the user data base, or it can be a numeric user ID. (If a numeric owner exists as a user name in the user data base, the user ID number associated with that user name is used.)

If you include a group name (that is, if you specify owner followed immediately by a colon (:) and then group with no intervening spaces, such as owner:group) **chown** also sets the group ID (GID) to the group for the files and directories named. The group can be a group name from the security facility group data base, or it can be a numeric group ID. If a numeric group exists as a group name in the group data base, the group ID number associated with that group is used. If there is no change to the GID, then specify -1 (or do not specify the :group).

6.9 z/OS UNIX superuser granularity

- Superuser security exposure
  - Can execute all commands
  - Access all HFS files in the system
  - Applicable to every *superuser*
- RACF UNIXPRIV class
  - Give a user superuser function for specific task

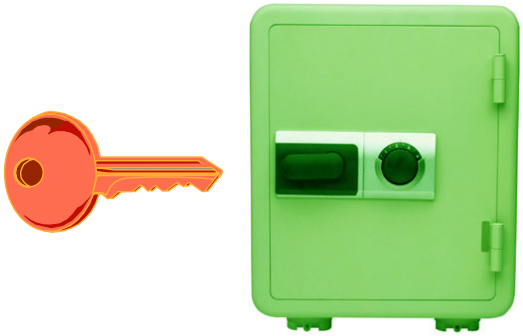


Figure 6-9 Adding superuser granularity for access

Superuser authority

A superuser can:

- Pass all security checks, so that the superuser can access any file in the file system.
- Manage processes.
- Have an unlimited number of processes running concurrently. For a started procedure, this is true only if it has a UID of 0. It is not true for a trusted or privileged process with a different UID.
- Change identity from one UID to another.
- Use setrlimit to increase any of the system limits for a process.

RACF UNIXPRIV class

You can define profiles in the UNIXPRIV class to grant RACF authorization for certain z/OS UNIX privileges. By defining profiles in the UNIXPRIV class, you can specifically grant certain superuser privileges with a high degree of granularity to users who do not have superuser authority. This allows you to minimize the number of assignments of superuser authority at your installation and reduces your security risk.

Resource names in the UNIXPRIV class are associated with z/OS UNIX privileges. You must define profiles in the UNIXPRIV class protecting these resources in order to use RACF authorization to grant z/OS UNIX privileges. The UNIXPRIV class must be active and

SETROPTS RACLIST must be in effect for the UNIXPRIV class. Global access checking is not used for authorization checking to UNIXPRIV resources.

Assigning superuser authority using the UNIXPRIV class

You may choose to assign the UID of 0 to multiple RACF user IDs. However, you should seek to minimize the assignment of superuser authority in your installation. You can accomplish this by setting z/OS UNIX user limits and by managing superuser privileges through UNIXPRIV profiles.

This RACF profile allows a normal user to become a superuser for only a specific superuser function.

6.10 Resource names: UNIXPRIV

```
❑ CHOWN.UNRESTRICTED            -  NONE
❑ FILE.GROUPOWNER.SETGID        -  NONE
❑ RESTRICTED.FILESYS.ACCESS     -  NONE
❑ SUPERUSER.FILESYS      - READ - UPDATE - CONTROL
❑ SUPERUSER.FILESYS.CHOWN       -  READ
❑ SUPERUSER.FILESYS.MOUNT       -  READ - UPDATE
❑ SUPERUSER.FILESYS.PFSCTL      -  READ
❑ SUPERUSER.QUIESCE             -  READ - UPDATE
❑ SUPERUSER.IPC.RMID            -  READ
❑ SUPERUSER.PROCESS.GETPSENT    -  READ
❑ SUPERUSER.PROCESS.KILL        -  READ
❑ SUPERUSER.PROCESS.PTRACE      -  READ
❑ SUPERUSER.SETPRIORITY         -  READ
❑ SUPERUSER.FILESYS.VREGISTER   -  READ
❑ SHARED.IDS                    -  NONE
❑ SUPERUSER.FILESYS.ACLOVERRIDE -  NONE
❑ RESTRICTED.FILESYS.ACCESS     -  NONE
❑ SUPERUSER.FILESYS.CHANGEPERMS -  READ
```

Figure 6-10 Profiles in the RACF UNIXPRIV class

UNIXPRIV RACF profiles

You can define profiles in the UNIXPRIV class to grant RACF authorization for certain z/OS UNIX privileges. These privileges are automatically granted to all users with z/OS UNIX superuser authority. By defining profiles in the UNIXPRIV class, you can specifically grant certain superuser privileges, with a high degree of granularity, to users who do not have superuser authority. This allows you to minimize the number of assignments of superuser authority at your installation and reduces your security risk.

Resource names for profiles

Resource names in the UNIXPRIV class are associated with z/OS UNIX privileges. You must define profiles in the UNIXPRIV class protecting these resources in order to use RACF authorization to grant z/OS UNIX privileges. The UNIXPRIV class must be active and SETROPTS RACLIST must be in effect for the UNIXPRIV class. Global access checking is not used for authorization checking to UNIXPRIV resources. The following profiles are:

- CHOWN.UNRESTRICTED

  ACCESS required (NONE). Allows all users to use the **chown** command to transfer ownership of their own files.

- FILE.GROUPOWNER.SETGID

  Specifies that a directory's set-gid bit is used to determine the group owner of any new objects created within the directory.

- SUPERUSER.FILESYS

  ACCESS required (READ). Allows you to read any HFS file, and to read or search any HFS directory.

  ACCESS required (UPDATE). Allows you to write to any HFS file, and includes privileges of READ access.

  ACCESS required (CONTROL or higher). Allows you to write to any HFS directory, and includes privileges of UPDATE access.

- SUPERUSER.FILESYS.CHOWN

  ACCESS required (READ). Allows you to use the **chown** command to change ownership of any file.

- SUPERUSER.FILESYS.MOUNT

  ACCESS required (READ). Allows you to issue the TSO/E MOUNT command or the **mount** shell command with the nosetuid option. Also allows you to unmount a file system with the TSO/E UNMOUNT command or the **unmount** shell command mounted with the **nosetuid** command. Users permitted to this profile can use the **chmount** shell command to change the mount attributes of a specified file system.

  ACCESS required (UPDATE). Allows user to issue the TSO/E MOUNT command or the **mount** shell command with the setuid option. Also allows user to issue the TSO/E UNMOUNT command or the **unmount** shell command with the setuid option. Users permitted to this profile can issue the **chmount** shell command on a file system that is mounted with the setuid option.

- SPERUSER.FILESYS.QUIESCE

  ACCESS required (READ). Allows users to issue **quiesce** and **unquiesce** commands for a file system mounted with the nosetuid option.

  ACCESS required (UPDATE). Allows users to issue **quiesce** and **unquiesce** commands for a file system mounted with the setuid option.

- SUPERUSER.FILESYS.PFSCTL

  ACCESS required (READ). Allows users to use the **pfsctl()** callable service.

- SUPERUSER.FILESYS.VREGISTER

  ACCESS required (READ). Allows a server to use the **vreg()** callable service to register as a VFS file server.

- SUPERUSER.IPC.RMID

  ACCESS required (READ). Allows users to issue the **ipcrm** command to release IPC resources.

- SUPERUSER.PROCESS.GETPSENT

  ACCESS required (READ). Allows users to use the **w_getpsent** callable service to receive data for any process.

- SUPERUSER.PROCESS.KILL

  ACCESS required (READ). Allows users to use the **kill()** callable service to send signals to any process.

- SUPERUSER.PROCESS.PTRACE

  ACCESS required (READ). Allows users to use the **ptrace()** function through the dbx debugger to trace any process. Allows users of the **ps** command to output information on all processes. This is the default behavior of ps on most UNIX platforms.

- SUPERUSER.SETPRIORITY

  ACCESS required (READ). Allows users to increase their own priority.

- SHARED.IDS

  Allows users to assign UID and GID values that are not unique.

- SUPERUSER.FILESYS.ACLOVERRIDE

  Specifies that ACL contents override the access that was granted by SUPERUSER.FILESYS. Can be overridden for specific users or groups. The user or group must have the same access that would be required to SUPERUSER.FILESYS while accessing the file.

- SUPERUSER.FILESYS.CHANGEPERMS

  Allows users to use the **`chmod`** command to change the permission bits of any file and to use the **`setfacl`** command to manage access control lists for any file.

6.11 z/OS UNIX UNIXPRIV class profiles

RACF Class - UNIXPRIV

RDEFINE UNIXPRIV resource-name UACC( )

SETROPTS CLASSACT(UNIXPRIV)

RDEFINE UNIXPRIV SUPERUSER.*FILESYS.MOUNT*
UACC(NONE)

PERMIT SUPERUSER.*FILESYS.MOUNT*
CLASS(UNIXPRIV) ID(SMITH) ACCESS(READ)

SETROPTS RACLIST(UNIXPRIV)

Figure 6-11 Defining superuser authority with the UNIXPRIV class

Defining profiles in the UNIXPRIV class

You can reduce the number of people who have superuser authority at you installation by defining profiles in the UNIXPRIV class that grant RACF authorization for certain z/OS UNIX privileges.

Normally, these privileges are automatically defined for all users who are defined with z/OS UNIX superuser authority. But you can use UNIXPRIV to grant certain superuser privileges, with a high degree of granularity, to users who do not have superuser authority.

UNIXPRIV class example

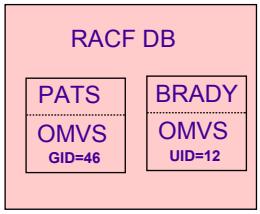
For example, if users have READ access to SUPERUSER.FILESYS.MOUNT, they can issue a **mount** and **unmount** command without being a defined superuser with all superuser capabilities, as follows:

```
RDEFINE UNIXPRIV SUPERUSER.FILESYS.MOUNT UACC(NONE)
PERMIT  SUPERUSER.FILESYS.MOUNT CLASS(UNIXPRIV) ID(SMITH) ACCESS(READ)
SETROPTS  RACLIST(UNIXPRIV) REFRESH
```

Now user SMITH (UID=35) can issue mounts, which is a superuser function. This is the only superuser function user SMITH can do.

6.12 Assigning UIDs

- Make UIDs unique, or users end up 'sharing' files
- Assign manually - ADDUSER OMVS(UID(37850))
  - Use employee serial number, if you can - or
- **SHARED.IDS** profile UNIXPRIV class - **z/OS V1R4**
  - Acts as a system-wide switch to prevent assignment of an ID that is already in use
- Enable **shared UID prevention**



RDEFINE UNIXPRIV SHARED.IDS UACC(NONE)
SETROPTS RACLIST(UNIXPRIV) REFRESH

ADDUSER MARCY OMVS(UID(12))

IRR52174I Incorrect UID 12. This value is already in use by BRADY.

ADDUSER (HARRY MARY) OMVS(UID(14))

IRR52185I The same UID cannot be assigned to more than one user.

Figure 6-12 Assigning UIDs with shared UID prevention

Assigning UIDs

z/OS UNIX allows multiple users to have the same UID. Assigning the same UID to multiple user IDs allows each user to access all of the resources associated with the other users of that shared user ID. The shared access includes not only z/OS UNIX resources such as files, but also includes the possibility that one user could access z/OS resources of the other users that are normally considered to be outside the scope of z/OS UNIX.

Note: This function was added in z/OS V1R4.

You can assign a z/OS UNIX user identifier (UID) to a RACF user by specifying a UID value in the OMVS segment of the RACF user profile. This can be an employee serial number or some other unique number for each user.

When assigning a UID to a user, make sure that the user is connected to at least one group that has an assigned GID. This group should be either the user's default group or one that the user specifies during logon or on the batch job. A user with a UID and a current connect group with a GID can use z/OS UNIX functions and access z/OS UNIX files based on the assigned UID and GID values. If a UID and a GID are not available as described, the user cannot use z/OS UNIX functions.

Shared UID prevention option

In order to prevent several users from having the same UID number, a new RACF SHARED.IDS profile has been introduced in the UNIXPRIV class. This new profile acts as a system-wide switch to prevent assignment of a UID that is already in use. The use of the SHARED.IDS profile requires AIM stage 2 or 3. To enable shared UID prevention, it is necessary to define a new SHARED.IDS profile in the UNIXPRIV class, as follows:

```
RDEFINE UNIXPRIV SHARED.IDS UACC(NONE)
SETROPTS RACLIST(UNIXPRIV) REFRESH
```

SHARED.IDS examples

Once the SHARED.IDS profile has been defined and the UNIXPRIV class refreshed, it will not allow a UID to be assigned if the UID is already in use.

As shown in Figure 6-12 on page 202, UID 12 is not assigned to user MARCY because in the RACF database this UID is assigned to user BRADY. Also, users HARRY and MARY can not be assigned the same UID 14.

The same is true for GIDs; it will not allow a GID to be shared between different groups.

Note: The use of this new functionality does not affect pre-existing shared UIDs. They remain as shared once you install the new support. If you want to eliminate sharing of the same UID, you must clean them up separately. The release provides a new IRRICE report to find the shared UIDs.

6.13 Shared UID prevention

- New SHARED keyword
  - OMVS segment of the ADDUSER, ALTUSER, ADDGROUP, and ALTGROUP commands

PERMIT SHARED.IDS CLASS(UNIXPRIV) ID(UNIXGUY) ACC(READ)
SETROPTS RACLIST(UNIXPRIV) REFRESH

RACF DB
BPXOINIT
OMVS
UID=0

UNIXGUY

AU OMVSKERN OMVS(UID(0) SHARED)
AG (G1 G2 G3) OMVS(GID(9) SHARED)

HARRY

AU MYBUDDY OMVS(UID(0) SHARED)
IRR52175I You are not authorized to specify the SHARED keyword.

To specify the SHARED operand, you must have the SPECIAL attribute or at least READ authority to the SHARED.IDS profile in the UNIXPRIV class

Figure 6-13 Using the SHARED keyword to allow duplicate UID assignment

Allowing duplicate user IDs

You may want to assign the same UID to multiple user IDs if these user IDs are used by the same person or persons. It may also be necessary to assign multiple users a UID of 0 (superuser authority). When doing this, it is important to remember that a superuser is implicitly a trusted user who has the potential of using UID(0) to access all z/OS resources.

Even if the SHARED.IDS profile is defined, you may still require some UIDs to be shared and others not to be shared. For example, you may require multiple superusers with a UID(0). It is possible to do this using the new SHARED keyword in the OMVS segment of the ADDUSER, ALTUSER, ADDGROUP, and ALTGROUP commands.

To allow an administrator to assign a non-unique UID or GID using the SHARED keyword, you must grant that administrator at least READ access to the SHARED.IDS profile and be at the z/OS V1R4 level or above, as follows:

```
PERMIT SHARED.IDS CLASS(UNIXPRIV) ID(UNIXGUY) ACCESS(READ)
SETROPTS RACLIST(UNIXPRIV) REFRESH
```

Once user ID UNIXGUY has at least READ access to the SHARED.IDS profile, UNIXGUY can assign the same UID or GID to multiple users, using the SHARED KEYWORD, as follows: ADDUSER OMVSKERN OMVS(UID(0) SHARED)

Note: To specify the SHARED operand, you must have the SPECIAL attribute or at least READ authority to the SHARED.IDS profile in the UNIXPRIV class.

6.14 Automatic UID and GID assignment

- New AUTOUID and AUTOGID keywords
  - OMVS segment of the ADDUSER and ALTUSER
  - OMVS segment of the ADDGROUP and ALTGROUP
- Derived values are guaranteed to be unique
- Automatic assignment by RACF with **z/OS V1R4**

ADDUSER MARY OMVS(AUTOUID)

IRR52177I User MARY was assigned an OMVS UID value of 4646

ADDGROUP PAYR OMVS(AUTOGID)

IRR52177I Group PAYR was assigned an OMVS GID value of 105

Figure 6-14 Assigning UIDs and GIDs automatically

Specifying automatic assignment of UIDs and GIDs

You can assign a z/OS UNIX user identifier (UID) and group identifier (GID) to a RACF user by specifying a UID value in the OMVS segment of the RACF user profile or by using the AUTOUID and AUTOGID keywords.

UIDs and GIDs can be assigned automatically by RACF to new users, making it easier to manage the process of assigning UIDs and GIDs to users. Previously, this was a manual process and guaranteed the uniqueness of the UID and GID for every user.

Note: When assigning a UID to a user, make sure that the user is connected to at least one group that has an assigned GID. This group should be either the user's default group or one that the user specifies during logon or on the batch job. A user with a UID and a current connect group with a GID can use z/OS UNIX functions and access z/OS UNIX files based on the assigned UID and GID values. If a UID and a GID are not available as described, the user cannot use z/OS UNIX functions.

Using a new AUTOUID keyword with the ADDUSER and ALTUSER commands, an unused UID will be assigned to the new or modified user. Using the AUTOGID keyword on the ADDGROUP and ALTGROUP commands, a GID will be assigned automatically to the new or modified group.

Note: This function was added in z/OS V1R4.

6.15 Automatic assignment requirements

- AIM stage 2 or 3 is required - **IRR52182I** message
- SHARED.IDS **must** be defined - **IRR52183I** message
  - The **SHARED.IDS** profile is defined as follows:
    - **RDEFINE UNIXPRIV SHARED.IDS UACC(NONE)**
    - **SHARED.IDS** - Allows assigning UID/GID values that are unique. It acts as a system-wide switch to prevent assignment of an ID which is already in use
- The **BPX.NEXT.USER** Facility class profile **must** be defined and RACLISTed - **IRR52179I** message
  - **RDEFINE FACILITY BPX.NEXT.USER APPLDATA(UID/GID)**
  - **SETROPTS RACLIST(FACILITY) REFRESH**

Figure 6-15 Requirements for automatic assignment of UIDs and GIDs

Application identity mapping

Application identity mapping (AIM) must be implemented if you wish to define and use SHARED.IDS. If the RACF command detects that the SHARED.IDS profile is defined, but the RACF database is not at least at AIM stage 2, the command fails and message IRR52176I is issued. Refer to *z/OS Security Server RACF System Programmer's Guide*, SA22-7681 for more information on using the IRRIRA00 utility to advance to AIM stage 2.

Automatic assignment requirements

A SHARED.IDS profile allows users to assign UID and GID values that are not unique.

The use of automatic UID/GID requires the following:

- You can convert your RACF database to stage 3 of application identity mapping using the IRRIRA00 conversion utility. It converts an existing RACF database to application identity mapping functionality using a four-stage approach. Install AIM stage 2 or 3, otherwise an IRR52182I message is issued and the automatic assignment attempt fails with the following message:

  ```
  IRR52182I Automatic UID assignment requires application identity mapping to
  be implemented.
  ```

- A SHARED.IDS profile must be defined, otherwise, an IRR52183I message is issued and the attempt fails with the following message:

  ```
  IRR52183I Use of automatic UID assignment requires SHARED.IDS to be
  implemented.
  ```

- The SHARED.IDS must be defined as follows:

  RDEFINE UNIXPRIV SHARED.IDS UACC(NONE)

The BPX.NEXT.USER profile

RACF can automatically generate a unique ID value in the OMVS segment upon your request. This is done by defining a profile called BPX.NEXT.USER in the FACILITY class by specifying the following:

- The AUTOUID operand of the ADDUSER and ALTUSER commands
- The AUTOGID operand of the ADDGROUP and ALTGROUP commands
- The BPX.NEXT.USER facility class profile must be defined and RACLISTed. Otherwise, an IRR52179I message will be issued and the attempt fails with the following message:

  ```
  IRR52179I The BPX.NEXT.USER profile must be defined before you can use
  automatic UID assignment.
  ```

6.16 Automatic assignment examples

Profile examples

```
RDEFINE FACILITY BPX.NEXT.USER APPLDATA('5-70000/3-30000')

      ADDUSER USERA OMVS(AUTOUID)
IRR52177I User USERA was assigned an OMVS UID value of 5.

      ADDUSER USERB OMVS(AUTOUID)
IRR52177I User USERB was assigned an OMVS UID value of 6.
```

Other Examples of APPLDATA

```
RDEFINE FACILITY BPX.NEXT.USER APPLDATA('1/0')
RALTER FACILITY BPX.NEXT.USER APPLDATA('2001/201')
RDEFINE FACILITY BPX.NEXT.USER APPLDATA('NOAUTO/3000')
```

Figure 6-16 Examples of assigning UIDs and GIDs automatically

Automatic assignment using APPLDATA

APPLDATA consists of 2 qualifiers separated by a forward slash (/). The qualifier on the left of the slash character specifies the starting UID value or range of UID values. The qualifier on the right of the slash character specifies the starting GID value or range of GID values. Qualifiers can be null or specified as `'NOAUTO'` to prevent automatic assignment of UIDs or GIDs.

The starting value is the value RACF attempts to use in ID assignment, after determining that the ID is not in use. If it is in use, the value is incremented until an appropriate value is found.

The maximum value valid in the APPLDATA specification is 16,777,215. If this value is reached or a candidate UID/GID value has been exhausted for the specified range, subsequent automatic ID assignment attempts fail and message IRR52181I is issued.

Profile examples

In the following example, we have defined the APPLDATA for a range of values from 5 to 70000 for UIDs and from 3 to 30000 for GIDs. USERA and USERB are added using the automatic assignment of UID. The range of automatic UID assignment starts with 5, so USERA is assigned to UID(5), which was free. UID(6) and UID(7) were already assigned before we started our example, so the next free UID is 8. USERB is assigned to UID(8).

```
RDEFINE FACILITY BPX.NEXT.USER APPLDATA('5-70000/3-30000')

ADDUSER USERA OMVS(AUTOUID)
```

```
IRR52177I User USERA was assigned an OMVS UID value of 5.

ADDUSER USERB OMVS(AUTOUID)
IRR52177I User USERB was assigned an OMVS UID value of 6.
```

RACF assigns the UID or GID

RACF extracts the APPLDATA from the BPX.NEXT.USER and parses out the starting value. It checks whether it is already in use and if so, the value is incremented and checked again until an unused value is found. Once a free value is found, it assigns the value to the user or group and replaces the APPLDATA with the new starting value, which is the next potential value or the end of the range.

In our example, that means that if UID(6) is free after UID(5) is assigned to USERA, RACF will start checking from UID(7) in the next assignment. But you can change the APPLDATA and modify the starting value. The APPLDATA can be changed using the following command:

```
RALTER FACILITY BPX.NEXT.USER APPLDATA('2000/500')
```

Note: Automatic assignment of UIDs and GIDs fails if you specify a list of users to be defined with the same name or if you specify the `SHARED` keyword. Also, `AUTOUID` or `AUTOGID` is ignored if UID or GID is also specified.

Other examples of APPLDATA

Here are some examples of correct and incorrect APPLDATA specifications:

Good data
1/0
1-50000/1-20000
NOAUTO/100000
/100000
10000-20000/NOAUTO
10000-20000/

Bad data
123B
/
2147483648 /* higher than max UID value */
555/1000-900

If you have an incorrect specification and attempt to use AUTOUID on an ADDUSER command, the following message is issued:

```
IRR52187I Incorrect APPLDATA syntax for the BPX.NEXT.USER profile.
```

6.17 Automatic assignment with RRSF

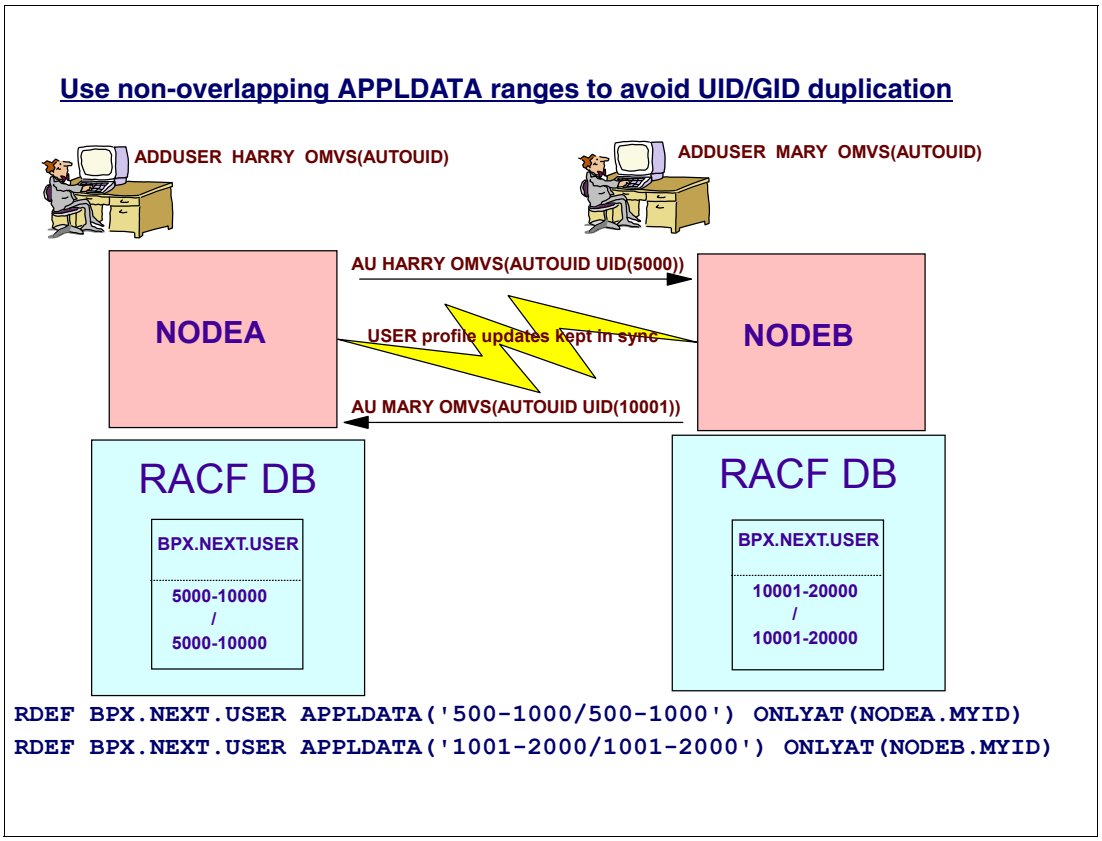


Figure 6-17 Automatic assignment with RRSF

RACF remote sharing facility (RRSF)

The RACF remote sharing facility allows RACF to communicate via APPC with other MVS systems that use RACF, allowing you to maintain remote RACF databases. RRSF extends the RACF operating environment beyond the traditional single host and shared DASD environments, to an environment made up of RRSF nodes that are capable of communicating with one another. This support provides administration of multiple RACF databases from anywhere in the RRSF network.

Use nonoverlapping APPLDATA ranges

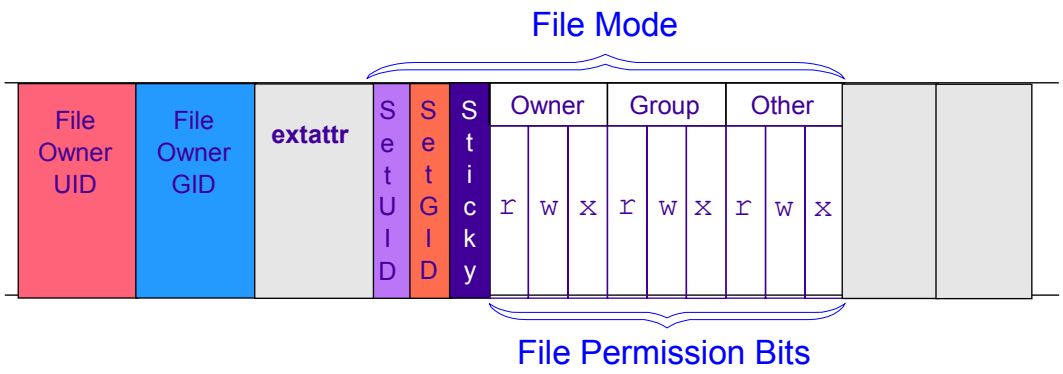
In an RRSF configuration, shown in Figure 6-17, use nonoverlapping APPLDATA ranges to avoid UID and GID duplications.

An additional concern is to make RACF automatically suppress propagation of internal updates. This can be done by specifying the `ONLYAT` keyword to manage the BPX.NEXT.USER profile, as follows:

```
RDEFINE BPX.NEXT.USER APPLDATA('5000-10000/5000-10000') ONLYAT(NODEA.MYID)
RDEFINE BPX.NEXT.USER APPLDATA('10001-20000/10001-20000') ONLYAT(NODEB.MYID)
```

For more information about automatic assignment in a RRSF configuration, refer to *z/OS Security Server RACF Security Administrator's Guide*, SA22-7683.

6.18 z/OS UNIX security: File security packet



Access	Permission for File	Permission for Directory
Read (**r**)	Permission to read or print the contents.	Permission to read, but not search, the contents.
Write (**w**)	Permission to change, add to, or delete from the contents.	Permission to change, add, or delete directory entries.
Execute (**x**)	Permission to run the file. This permission is used for executable files.	Permission to search the directory.

Figure 6-18 File security packet definitions

File security packet (FSP)

Each z/OS UNIX file and directory has a file security packet (FSP) associated with it to control access. The FSP is created when a file or directory is created. It is stored in the file system for the life of the file or directory, until the file or directory is deleted, at which time the FSP is also deleted. The FSP consists of:

- File owner UID
- File owner GID
- File mode

Extended attributes

Another section of the FSP, which is specific to the z/OS UNIX implementation, is called extended attributes (extattr). It contains flags to mark HFS program files as APF-authorized and program controlled. The **`extattr`** shell command is used to manipulate these bits.

File Mode

The file mode consists of:

SetUID This bit only relates to executable files. If on, it causes the UID of the user executing the file to be set to the file's UID.

SetGID This bit only relates to executable files. If on, it causes the GID of the user executing the file to be set to the file's GID.

Sticky Bit This bit only relates to executable files. If on, it causes the file to be retained in memory for performance reasons. The implementation of this varies between

platforms. In z/OS UNIX, it means programs are loaded from LPA (or LNKLST as per normal MVS program search) instead of an HFS file. For a directory, the sticky bit causes UNIX to permit files in a directory or subdirectories to be deleted or renamed only by the owner of the file, or by the owner of the directory, or by a superuser.

Sticky bit for directories

Using the **mkdir**, MKDIR, or **chmod** command, you can set the sticky bit on a directory to control permission to remove or rename files or subdirectories in the directory. When the bit is set, a user can remove or rename a file or remove a subdirectory only if one of these is true:

- The user owns the file or subdirectory.
- The user owns the directory.
- The user has superuser authority.

If you use the rmdir, rename, rm, or mv utility to work with a file, and you receive a message that you are attempting an “operation not permitted,” check to see if the sticky bit is set for the directory the file resides in.

Permission bits

The file mode also has the file permission bits, consisting of:

- Owner read/write/execute permissions
- Group read/write/execute permissions
- Other (or all users) read/write/execute permissions

where:

r Read (r) access to both files and directories

w Write (w) access to both files and directories

x Execute (x) has a different meaning for files and directories, as follows:

- For an executable file, an access of x means that the user can execute the file.

- For a directory, an access of x means the user can search the directory.

Both read (r) and execute (x) are required in order to execute a shell script. To access HFS files, a user needs the following:

- Search (x) permission to all the directories in the pathname of files the user wants to access.
- Write permission to directories where the user will be creating new files and directories.
- Read and/or write permission, as appropriate, to files for access.
- Execute (x) permission for an executable file.

Note: In z/OS UNIX, these three permissions are not hierarchical. For example, a user with write permission who does *not* have read permission, can only write over existing data or add data to a file, and cannot look at the contents of the file or print the file. Similarly, write and read permission does not allow a user to execute a file or search a directory.

6.19 Octal values for permission bits

0	---	No access
1	--x	Execute-only
2	-w-	Write-only
3	-wx	Write and execute
4	r--	Read-only
5	r-x	Read and execute
6	rw-	Read and write
7	rwx	Read, write and execute

XXX
421

Permission Bit Examples:

700	owner(7=rwx)	group(0=---)	other(0=---)
755	owner(7=rwx)	group(5=r-x)	other(5=r-x)

- To change the permission bits for a file:
  - The ISPF shell
  - The chmod and setfacl commands
  - The chmod() function in a program

Figure 6-19 Octal values of permission bit settings

Permission bits in octal

z/OS UNIX commands typically allow the definition of permission bit settings using octal notation. It is a simpler way to describe the permission bit string.

The octal numbers relate to the bit positions as follows:

```
Dec     Bin     Map
===     ===     ===
0   =   000  =  ---
1   =   001  =  --x
2   =   010  =  -w-
3   =   011  =  -wx
4   =   100  =  r--
5   =   101  =  r-x
6   =   110  =  rw-
7   =   111  =  rwx
```

where:

Dec Decimal representation of the octal value

Bin Binary representation of the octal value

Map Map of permission bits according to binary positions

Permission bit settings

UNIX commands such as **chmod** accept octal notation when referring to permission bit settings.

A permission bit setting of 700 is good for a user's private files, allowing the owner full access while denying access to others.

A permission bit setting of 755 is good for a user to let other people access their files (read and execute), but not update them.

Changing permission bit settings

To change the permission bits for a file, use one of the following:

- The ISPF shell.
- The **chmod** command. You can use it to change individual bits without affecting the other bits. You can also use the **setfacl** command to change permission bits.
- The **chmod()** function in a program. The function changes all permission bits to the values in the mode argument.

6.20 Data set security versus file security

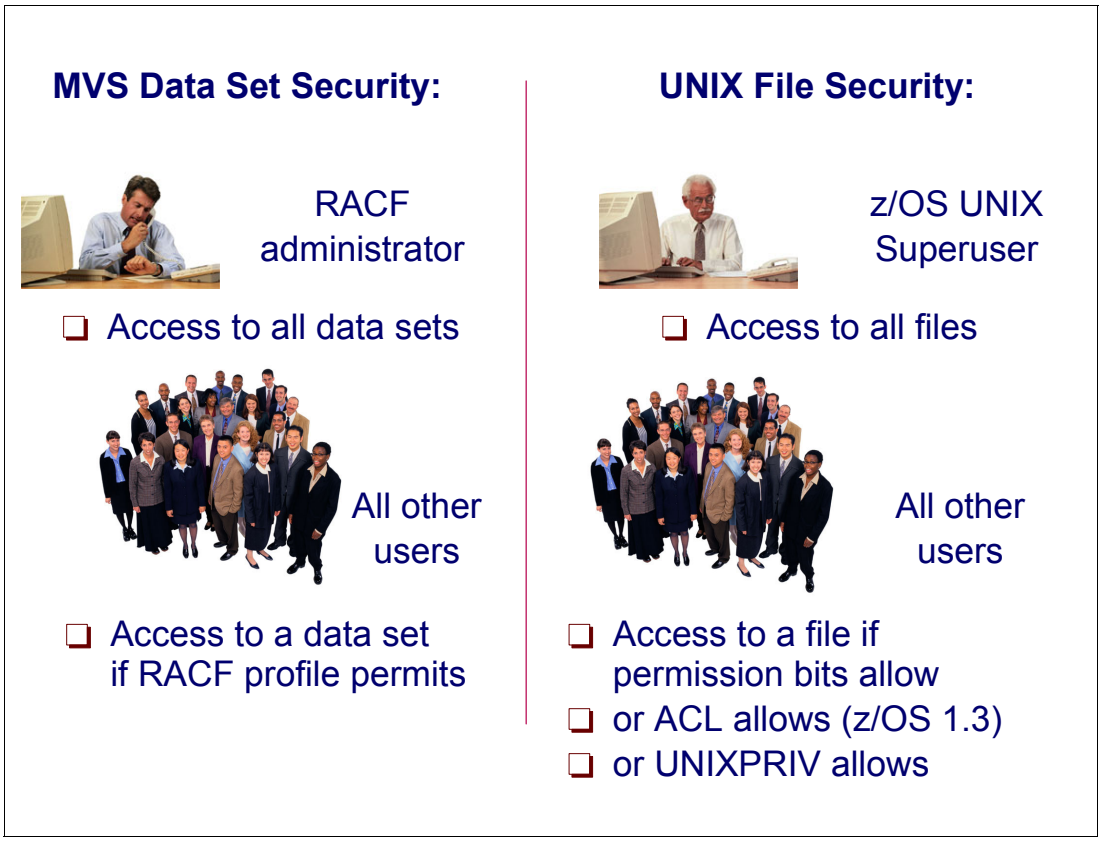


Figure 6-20 MVS data set security versus file system security

MVS security versus z/OS UNIX security

Security processing within z/OS UNIX differs in many ways from standard security processing in MVS. MVS resources like users and data are protected by RACF profiles stored in the RACF database. RACF refers to the profiles when deciding which users should be permitted to protected system resources. Security administration is done with RACF commands or RACF ISPF panels.

z/OS UNIX users are defined as MVS users and they are administrated by RACF profiles. The security information for files and directories in a hierarchical file system is stored within the file system itself in a file security packet (FSP). HFS files and directories are protected by permission bit information which is kept in the FSP. Administration of file security is performed by using z/OS UNIX shell commands, or ISHELL menu options.

The user administration is similar for regular MVS users and z/OS UNIX users. Every user must present a password when logging on to the system. z/OS UNIX uses a UID and GID for each user and this information is stored in RACF profiles together with the user ID and password information. The concept of a superuser in z/OS UNIX is similar to a RACF security administrator.

z/OS UNIX users do not work with data sets, they work with files and directories. z/OS UNIX users do not have to be aware that their data is located physically in an HFS data set. All they see is the hierarchical file structure made up of multiple mounted HFS data sets. The FSPs are maintained by z/OS UNIX commands. RACF data set profiles cannot be used to protect z/OS UNIX files and directories.

6.21 z/OS UNIX user's security environment

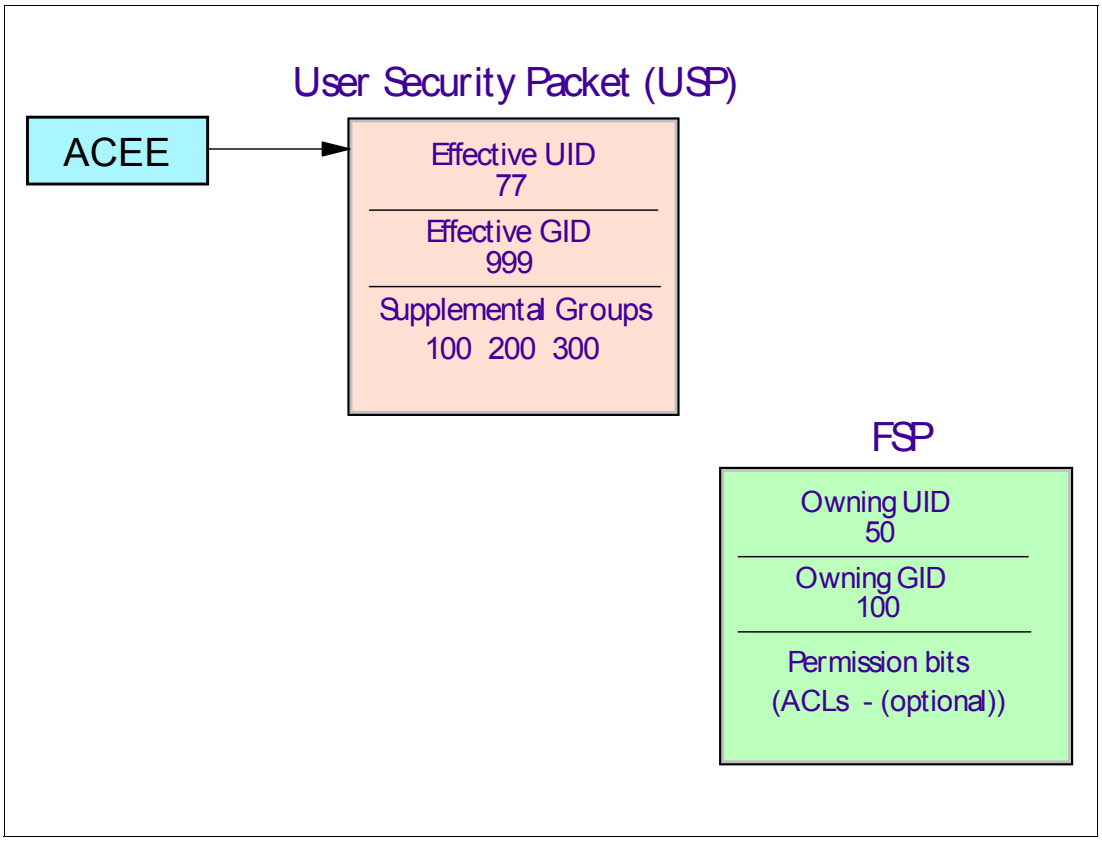


Figure 6-21 A z/OS UNIX user's file security environment

User's security environment

Authorization checking for z/OS UNIX files and directories uses the control blocks shown in Figure 6-21, and RACF makes the following checks:

- The accessor environment element (ACEE) is a control block that contains a description of the current user's security environment, including the user ID, current connect group, user attributes, and group authorities. An ACEE is constructed during user identification and verification by RACF.
- The effective UID and effective GID of the process are used in determining access decisions. The only exception is that if file access is being tested, rather than requested, the real UID and GID are used instead of the effective UID and GID. The real and effective IDs are generally the same for a process, but if a set-uid or set-gid program is executed, they can be different.

FSP and security flow

Permission bit information is stored in the file security packet (FSP) within each file and directory. (ACLs may also be stored with the file.) Permission bits allow you to specify read authority, write authority, or search authority for a directory. They also allow specification of read, write, or execute authority for a file. Because there are three sets of bits, separate authorities can be specified for the owner of the file or directory, the owning group, and everyone else (like RACF's universal access authority, or UACC). The owner is represented by a UID. The owning group is represented by a GID. Access checking compares the user's UID and GID to the ones stored in the FSP.

Security check

The security check and flow is as follows:

- Security information, such as the owner's UID-GID and the permission bits for a file, is kept in a 64-byte area called the file security packet (FSP), which is mapped by IRRPIFSP. The FSP is the security-related section of a file's attributes.
- The FSP is created by a SAF call from the PFS when a file is created. Some of the information is taken from the current security environment, and some of it is passed as parameters.
- The PFS stores the FSP with the attributes of the file.
- When an access check is to be done, the PFS calls SAF with the type of check that is being requested, the audit_structure from the current call, and the file's FSP. SAF passes these to the security product, which extracts user information from the current security environment and compares it against the access control that is stored within the FSP. The audit_structure is used primarily for any auditing that may be necessary.

6.22 Access checking flows

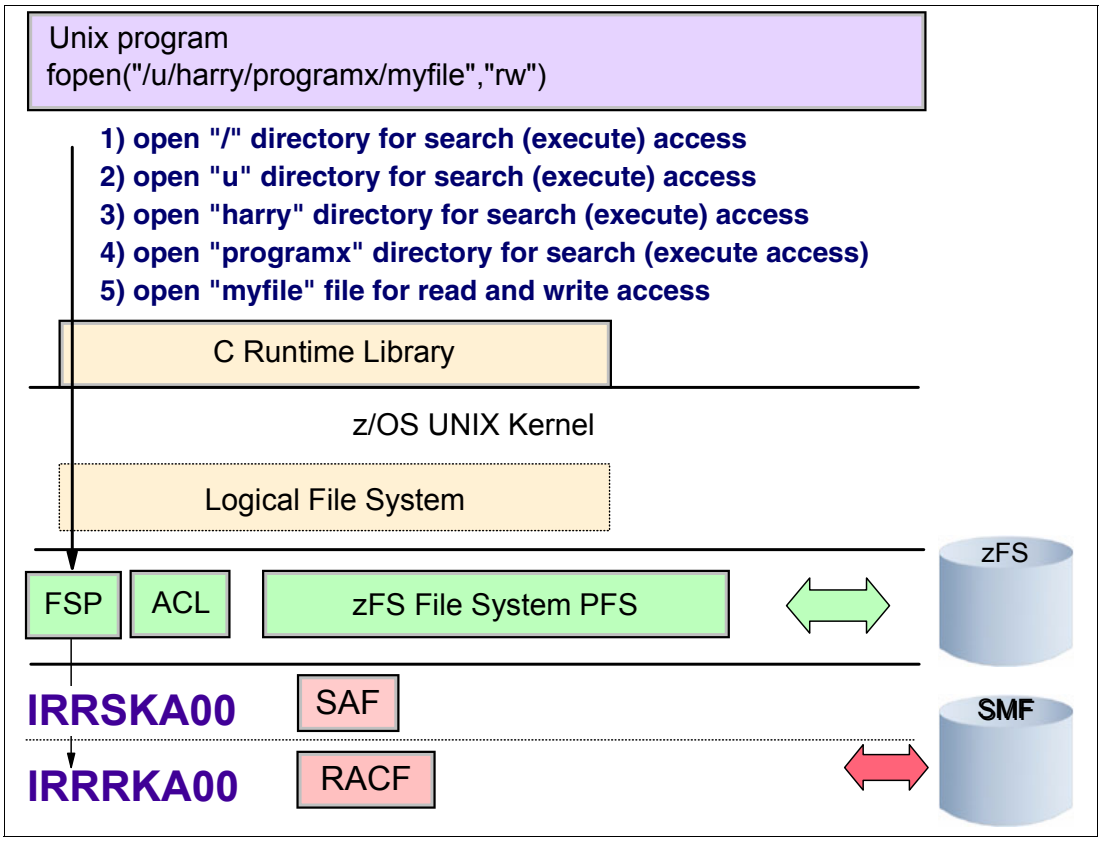


Figure 6-22 Access checking flow for access to a file

Security check flow

Figure 6-22 shows the access checking flow from a UNIX program—or the access could be from a z/OS UNIX shell user—to the security product, as follows:

- The z/OS UNIX kernel calls the file system, and the file system calls the security product.
- The kernel calls the file system iteratively for each directory component of the path name (one is required in order to locate the next).
- The base file name is retrieved.
- For each directory lookup, the file system calls the security product to make sure the user has search authority.
- The security product is called to ensure the user has the requested access to the base file.

This is the basic architecture of every UNIX file system.

6.23 File authorization checking flow

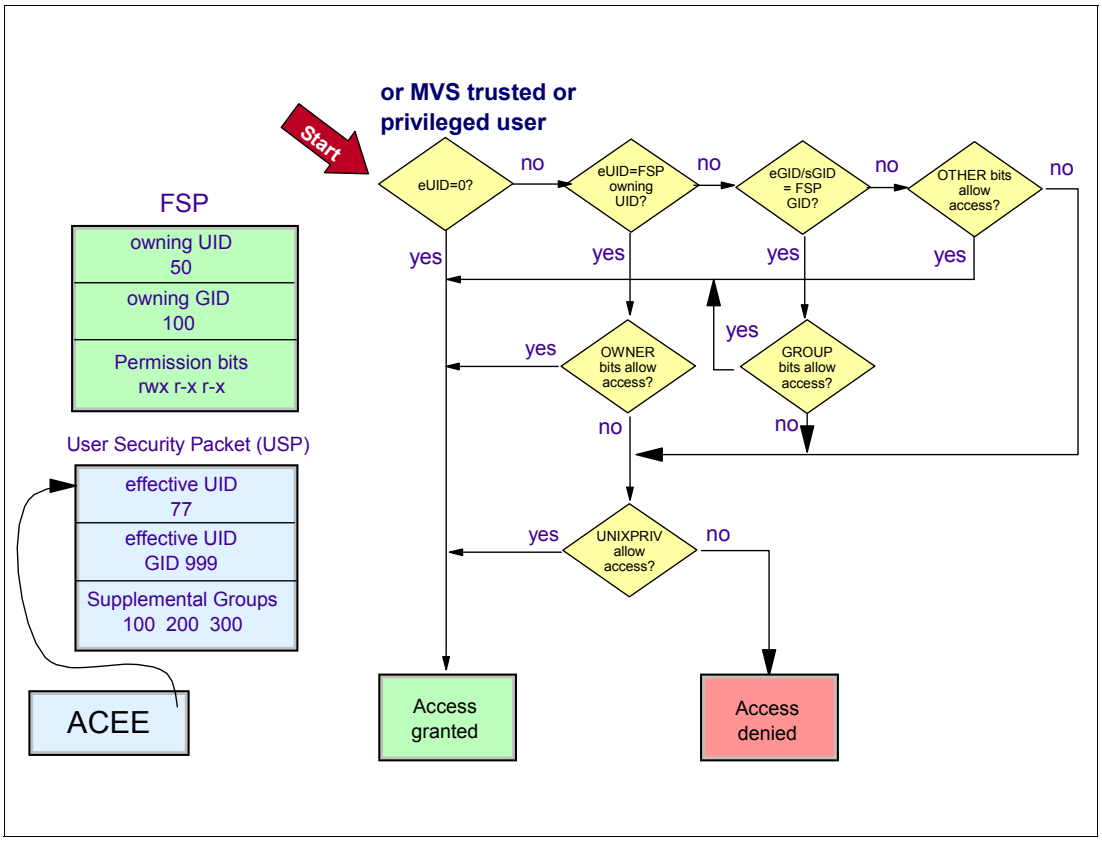


Figure 6-23 Authorization checking with z/OS UNIX

Authorization checking flow

Authorization checking is done for all directories and files (including special files) in the file system. z/OS UNIX calls RACF to perform the authorization checking and passes RACF the FSP (file security packet), and the CRED (security credentials).

Figure 6-23 shows the sequence of authorization checks, as follows:

- A superuser (UID of zero) is allowed access to all resources.
- If the effective UID of the process (the accessor) equals the UID of the file, RACF uses the owner permissions in the FSP to either allow or deny access.
- If the effective GID of the process equals the GID of the file, RACF uses the group permissions in the FSP to either allow or deny access. If RACF list-of-groups checking is active (SETROPTS GRPLIST), RACF will look at the user's connect groups that have a GID for a group that matches the GID of the file. If it finds a matching GID, RACF will allow or deny access based on the group permissions specified in the FSP. Note that if a user is connected to more that 300 z/OS UNIX groups, only the first 300 will be used.
- If the effective UID or GID of the process does not match the file UID or GID, then the other permission bits will determine access.

6.24 POSIX standard and UNIX ACLs

- In the POSIX standard, two different ACLs are referenced as follows:
- **Base ACL entries** are permission bits (owner, group, other) - It refers to the FSP
- **Extended ACL entries** are ACL entries for individual users or groups, such as the permission bits that are stored with the file, not in RACF profiles

Figure 6-24 The POSIX standard for ACL support

z/OS UNIX support for ACLs

ACLs have existed on various UNIX platforms for many years, but with variations in the interfaces. ACL support in z/OS V1R3 is based on a POSIX standard that was never implemented when z/OS UNIX was first introduced as OpenEdition.

In the POSIX standard, two different ACLs are referenced as follows:

- Base ACL entries

  These entries are permission bits (owner, group, other). This refers to the FSP. You can change the permissions using **`chmod`** or **`setfacl`**. They are not physically part of the ACL, although you can use **`setfacl`** to change them and **`getfacl`** to display them.

- Extended ACL entries

  These entries are ACL entries for individual users or groups, such as the permission bits that are stored with the file, not in RACF profiles. Extended ACL entries, like the permission bits, are stored with the file, not in RACF profiles. Each ACL type (access, file default, directory default) can contain up to 1024 extended ACL entries. Each extended ACL entry specifies a qualifier to indicate whether the entry pertains to a user or a group, the actual UID or GID itself, and the permissions being granted or denied by this entry. The allowable permissions are read, write, and execute. As with other UNIX commands, **`setfacl`** allows the use of either names or numbers when referring to users and groups.

Note: This function was added in z/OS V1R3.

6.25 Limitations of current permission bits

- Can only specify permissions for file owner (user), group owner, and everybody else (other)
- Cannot permit/restrict access to specific users and groups - lots of customer requirements for capability
- Access Control Lists introduced in z/OS V1R3
  - To manage files using UNIX setfacl/chmod commands, or the ISHELL, a user must be either:
    - UID(0)
    - The file owner
    - Have READ access to the UNIXPRIV class profile SUPERUSER.FILESYS.CHANGEPERMS
  - Requirements are the same for changing the permission bits

Figure 6-25 z/OS UNIX limitation with POSIX standard on ACLs

ACLs for greater granularity

Access control lists (ACLs) were introduced in z/OS V1R3 as a way to provide a greater granularity for access to z/OS UNIX files and directories. ACLs are based on a POSIX standard that was never approved, and other UNIX implementations.

z/OS V1R3 provides support for ACLs to control access to files and directories for:

- Specific individual user or users (UID)
- Specific group or groups (GID)

To create an ACL for a file, you must have one of the following security access controls:

- Be the file owner
- Be permitted to the BPX.SUPERUSER RACF profile
- Have superuser authority (UID=0)
- Have READ access to the SUPERUSER.FILESYS.CHANGEPERMS profile in the UNIXPRIV class

6.26 FSPs and ACLs

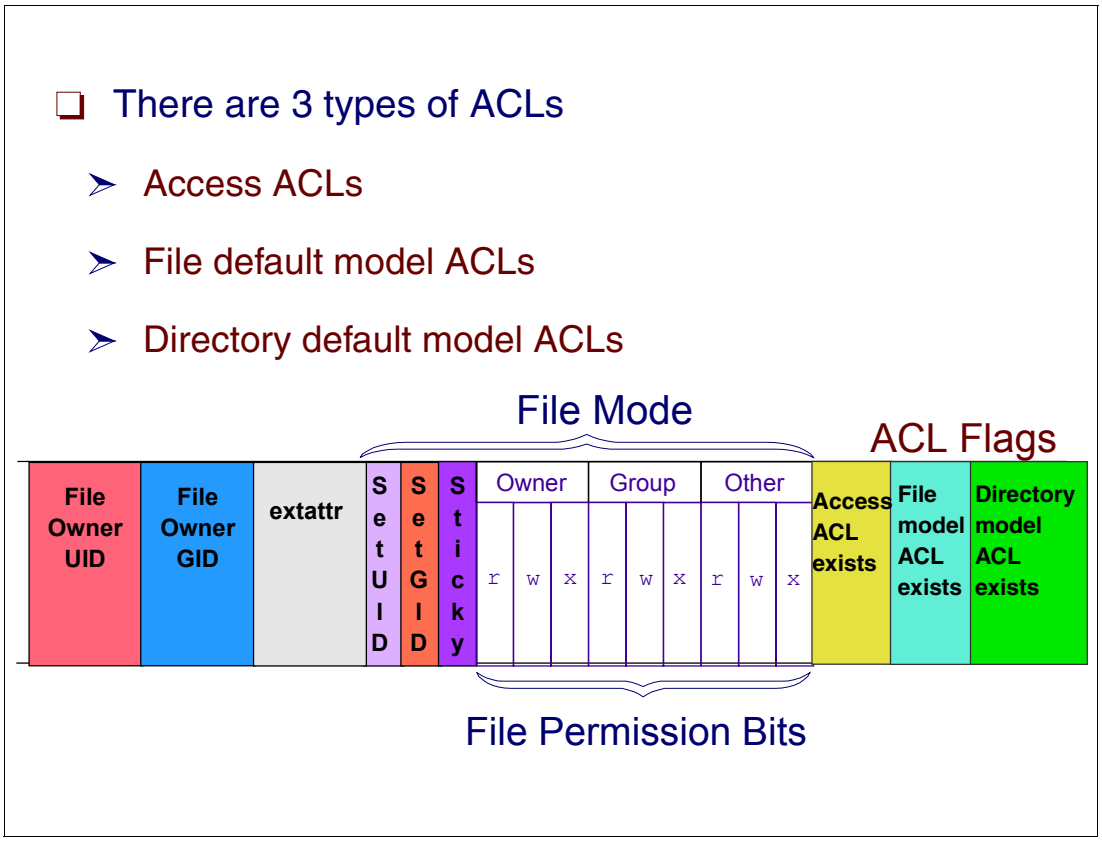


Figure 6-26 File security packet and ACL support

ACL types

Use access control lists (ACLs) to control access to files and directories by individual user (UID) and group (GID). ACLs are used in conjunction with permission bits. They are created, modified, and deleted using the **`setfacl`** shell command. To display them, use the **`getfacl`** shell command. You can also use the ISHELL interface to define and display ACLs.

ACLs are used together with the permission bits in the FSP to control the access to z/OS UNIX files and directories by individual users (UIDs) and groups (GIDs).

To reduce administrative overhead, three types of ACLs (extended ACLs) are defined, giving the capability to inherit ACLs to newly created files and directories as follows:

Access ACLs This type of ACL is used to provide protection for a file system object (specific for a file or directory).

File default ACLs This type is a model ACL that is inherited by files created within the parent directory. The file default ACL is copied to a newly created file as its access ACL. It is also copied to a newly created subdirectory as its file default ACL.

Directory default ACLs This type is a model ACL that is inherited by subdirectories created within the parent directory. The directory default ACL is copied to a newly created subdirectory as both its access ACL and directory default ACL.

6.27 Access control list table

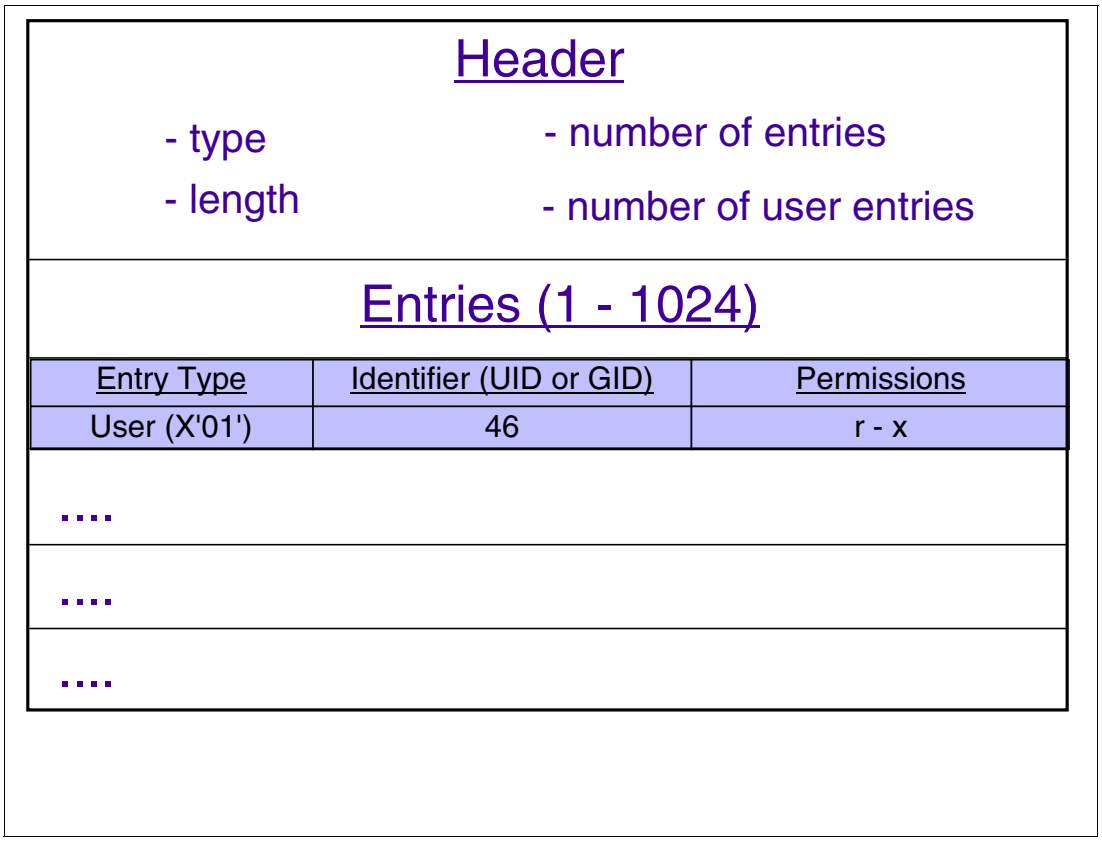


Figure 6-27 ACL table that contains ACL entries after creation

ACL table structure

An ACL is mapped by the SAF IRRPFACL macro, as shown in Figure 6-27, where the set of user entries is followed by the set of group entries.

The entries are sorted in ascending order by UID and GID to help optimize the access checking algorithm. The table consists of a list of entries (with a maximum of 1024) where every entry has information about the type (user or group), identifier (UID or GID), and permissions (read, write, and execute) to apply to a file or directory.

Extended ACL entries are ACL entries for individual users or groups. Like the permission bits, they are stored with the file, not in RACF profiles.

Note: ACLs are supported by HFS, zFS, and TFS. It is possible that other physical file systems will eventually support z/OS ACLs. Consult your file system documentation to see if ACLs are supported.

6.28 File authorization check summary

- RACF uses the following to determine whether the user is authorized to access the file with the requested access level:
  - The user's security environment - (ACEE and USP)
  - The permission bits - (FSP)
  - The access ACL - (FSP and ACL table)
  - The following UNIXPRIV class profiles
    - SUPERUSER.FILESYS
    - RESTRICTED.FILESYS.ACCESS
    - SUPERUSER.FILESYS.ACLOVERRIDE

ACL entries are used only if the RACF FSSEC class is active

SETROPTS CLASSACT(FSSEC)

Figure 6-28 Authorization checks made for access to files and directories

RACF checking summary

Authorization checking for z/OS UNIX files and directories is done by RACF, which makes the following checks:

- The accessor environment element (ACEE) is a control block that contains a description of the current user's security environment, including user ID, current connect group, user attributes, and group authorities. An ACEE is constructed during user identification and verification.
- RESTRICTED.FILESYS.ACCESS

  Specifies that RESTRICTED users cannot gain file access by virtue of the “other” permission bits.
- SUPERUSER.FILESYS.ACLOVERRIDE

  If no group bits access is allowed and the FSSEC class is active, and an ACL exists, and there is an ACL entry for any of the user's supplemental GIDs, then the permission bits of that ACL entry are checked. If at least one matching ACL entry was found for the GID, or any of the supplemental GIDs, then processing continues with the ACLOVERRIDE checking.
- SUPERUSER.FILESYS

  If no group ACL matches, then if the UNIXPRIV class is active, the SUPERUSER.FILESYS access is checked.

6.29 Profiles in UNIXPRIV class

- Grant authorization for certain UNIX privileges

RDEFINE UNIXPRIV SUPERUSER.FILESYS UACC(NONE)
PERMIT SUPERUSER.FILESYS CLASS(UNIXPRIV) ID(user|group) ACC(READ)

- SUPERUSER.FILESYS - ACC(.....)
  - **READ** - Allows a user to read any local file, and to read or search any local directory
  - **UPDATE** - Allows a user to write to any local file, and includes privileges of READ access
  - **CONTROL/ALTER** - Allows a user to write to any local directory, and includes privileges of UPDATE access

Figure 6-29 SUPERUSER.FILESYS profile in UNIXPRIV class

UNIXPRIV class profile for authorization checking

The UNIXPRIV class provided the capability to assign specific superuser functions to a user or group when you give a user or group either a:

- UID of 0
- BPX.SUPERUSER profile

Either of these gives a user or group access to all UNIX functions and resources. A BPX.SUPERUSER profile allows you to request that you be given such access, but you do not have the access unless you make the request. So, instead of giving a user or group access to all functions a superuser has, the UNIXPRIV class provides profiles that allow access to specific superuser functions.

The SUPERUSER.FILESYS profile in the UNIXPRIV class has three access levels that allow access to z/OS UNIX files as follows:

READ	Allows a user to read any local file, and to read or search any local directory.
UPDATE	Allows a user to write to any local file, and includes privileges of READ access.
CONTROL/ALTER	Allows a user to write to any local directory, and includes privileges of UPDATE access.

6.30 Profiles in UNIXPRIV class (2)

New with z/OS V1R3

- **RESTRICTED.FILESYS.ACCESS** - This profile in the UNIXPRIV class controls the access to filesystem resources for restricted users based on the “other” permission bits
- **SUPERUSER.FILESYS.ACLOVERRIDE** - This profile allows RACF to force the use of the ACL authorizations to override a user’s SUPERUSER.FILESYS profile authority
- **SUPERUSER.FILESYS.CHANGEPERMS** - This profile allows users to use the **chmod** command to change the permission bits of any file and to use the **setfacl** command to manage access control lists for any file

Figure 6-30 UNIXPRIV profiles introduced with z/OS V1R3

Profiles for authorization checking

Resource names in the UNIXPRIV class are associated with z/OS UNIX privileges. You must define profiles in the UNIXPRIV class protecting these resources in order to use RACF authorization to grant z/OS UNIX privileges. The UNIXPRIV class must be active and SETROPTS RACLIST must be in effect for the UNIXPRIV class. Global access checking is not used for authorization checking to UNIXPRIV resources.

RESTRICTED.FILESYS.ACCESS profile

This profile specifies that RESTRICTED users cannot gain file access by virtue of the “other” permission bits. Checking for this new profile RESTRICTED.FILESYS.ACCESS is done for RESTRICTED users regardless of whether an ACL exists, so this function can be exploited whether you plan to use ACLs or not. You can define the profile as follows:

```
RDEFINE UNIXPRIV RESTRICTED.FILESYS.ACCESS UACC(NONE)
SETROPTS RACLIST(UNIXPRIV) REFRESH
```

SUPERUSER.FILESYS.ACLOVERRIDE profile

Any user who is not a superuser with UID(0) or the file owner, and who is denied access through the ACL, can still access a file system resource if the user has sufficient authority to the SUPERUSER.FILESYS resource in the UNIXPRIV class. Therefore, to prevent this, you can force RACF to use your ACL authorizations to override a user's SUPERUSER.FILESYS authority by defining the following profiles:

```
RDEFINE UNIXPRIV SUPERUSER.FILESYS.ACLOVERRIDE UACC(NONE)
```

```
SETROPTS RACLIST(UNIXPRIV) RACLIST
```

SUPERUSER.FILESYS.CHANGEPERMS profile

As an enhancement to superuser granularity, when using the **chmod** command, a RACF service (IRRSCF00) has been updated to check the caller's authorization to the resource SUPERUSER.FILESYS.CHANGEPERMS in the UNIXPRIV class if the caller's user ID is not either:

- UID(0)
- The owner of the file
- BPX.SUPERUSER

If the user executing the **chmod** command has at least READ authority to the resource, the user is authorized to change the file mode in the same manner as a user having UID(0).

This profile allows users to use the **chmod** command to change the permission bits of any file and to use the **setfacl** command to manage access control lists for any file.

6.31 RACF RESTRICTED attribute

- Defined through ADDUSER or ALTUSER
  - ALTUSER RSTDUSR RESTRICTED
- Restricted user IDs cannot access protected resources they are not specifically authorized to access
- Access authorization for restricted user IDs bypasses global access checking
- In addition, the UACC of a resource and an ID(*) entry on the access list are not used to enable a restricted user ID to gain access

The RESTRICTED attribute does not prevent users from gaining access to z/OS UNIX file system resources

----- Access allowed through "other" permissions -----

Figure 6-31 Assigning the RESTRICTED attribute with RACF

Defining restricted users

You can define a restricted user ID by assigning the RESTRICTED attribute through the ADDUSER or ALTUSER commands, as follows:

```
ALTUSER RSTDUSR RESTRICTED
```

User IDs with the RESTRICTED attribute cannot access protected resources they are not specifically authorized to access. Access authorization for restricted user IDs bypasses global access checking. In addition, the UACC of a resource and an ID(*) entry on the access list are not used to enable a restricted user ID to gain access.

However, the RESTRICTED attribute has no effect when a user accesses a z/OS UNIX file system resource; the file's “other” permission bits can allow access to users who are not explicitly authorized. To ensure that restricted users do not gain access to z/OS UNIX file system resources through other bits, use the new UNIXPRIV profile, RESTRICTED FILESYS.ACCESS.

This is a solution to a problem many installations face when they connect the mainframe to the Internet. There are RACF user IDs assigned to "any user who comes in from the Web who asks to look at sensitive information." If the installation has not yet set the UACCs to all our sensitive data sets and resources to NONE, then such users could conceivably read sensitive data. If these user IDs are set as RESTRICTED, they can only access data to which they have been explicitly permitted.

6.32 z/OS UNIX file access checking

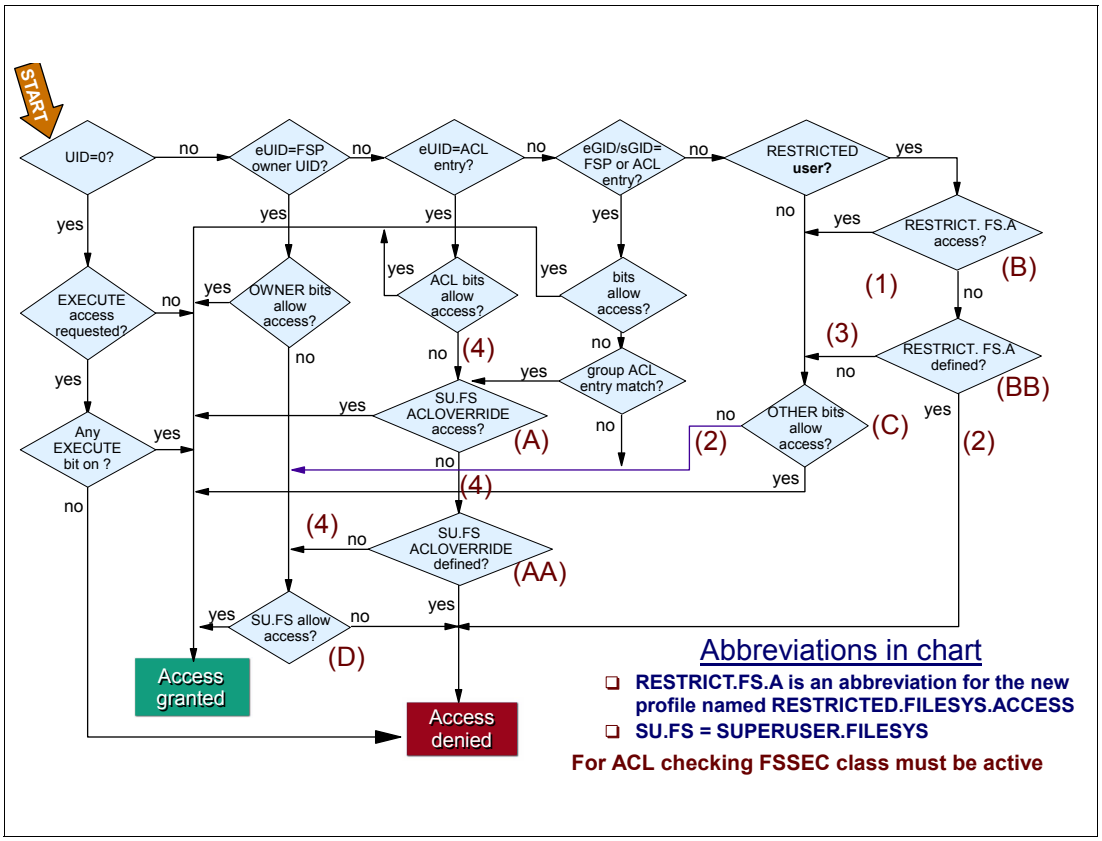


Figure 6-32 Authorization checking after introduction of UNIXPRIV profiles

Authorization checking flow

The effective UID and effective GID of the process are used in determining access decisions. The only exception is that if file access is being tested, rather than requested, the real UID and GID are used instead of the effective UID and GID. The real and effective IDs are generally the same for a process, but if a set-uid or set-gid program is executed, they can be different.

Check UID

If the user is not system and is not a superuser, the permission bits and ACL (if one exists, and if the FSSEC class is active) for the file are checked to see if the access requested is allowed.

- If the selected UID matches the owner UID of the file, the owner permission bits are checked.
- If the UIDs don't match, the user ACL entries are checked. If the selected UID matches an ACL entry, the ACL entry bits are checked.

Check GID

If a matching ACL entry was not found for the user, the group bits and the group ACL entries are checked. The selected GID and supplemental GIDs are checked against the file owner GID and the group ACL entries until a match is found that grants the requested access, or until all the GIDs have been checked.

- If the GID matches the file owner GID, the file's “group” permission bits are checked. If the group bits allow the requested access, then access is granted.
- If any of the user's supplemental GIDs match the file owner GID, the file's group permission bits are checked. If the group bits allow the requested access, then access is granted.

Check for ACLs

If no group bits access is allowed and the FSSEC class is active, and an ACL exists, and there is an ACL entry for any of the user's supplemental GIDs, then the permission bits of that ACL entry are checked. If at least one matching ACL entry was found for the GID, or any of the supplemental GIDs, then processing continues with the ACLOVERRIDE checking.

If no group ACL matches, then if the UNIXPRIV class is active, the SUPERUSER.FILESYS access is checked.

Check SUPERUSER.FILESYS.ACLOVERRIDE

SUPERUSER.FILESYS.ACLOVERRIDE is checked only when a user's access was denied by a matching ACL entry based on the user's UID or one of the user's GIDs. If the user's access was denied by the file's permission bits, SUPERUSER.FILESYS is checked.

See “Access checking with ACLs (1)” on page 233 and “Access checking with ACLs (2)” on page 234.

Check RESTRICTED.FILESYS.ACCESS

If no match was found, the other permission bits are checked, unless the user has the RESTRICTED attribute, the UNIXPRIV class is active, the resource named RESTRICTED.FILESYS.ACCESS is protected, and the user does not have at least READ access.

See “RESTRICTED user profile” on page 231 and “Restricted user access checking” on page 232.

6.33 RESTRICTED user profile

- Profile - **RESTRICTED.FILESYS.ACCESS**
  - If not defined, then the 'other' bits are treated like UACC and ID(*) for RESTRICTED users during RACF profile checking **(3 on access flow)**
  - If defined, RESTRICTED users cannot be granted file access via the 'other' bits and file access is denied

(BB on access flow)

RDEFINE UNIXPRIV RESTRICTED.FILESYS.ACCESS UACC(NONE)
SETROPTS RACLIST(UNIXPRIV) REFRESH

Figure 6-33 How the RESTRICTED user profile works

Checking RESTRICTED user profile

Users with the RESTRICTED attribute cannot access protected resources they are not specifically authorized to access. However, the RESTRICTED attribute has no effect when a user accesses a z/OS UNIX file system resource; therefore, if you do not define the RESTRICTED.FILESYS.ACCESS profile, then the file's “other” permission bits can allow access to users who are not explicitly authorized.

To ensure that restricted users do not gain access to z/OS UNIX file system resources through other bits, you must define the RACF profiles as follows:

```
RDEFINE UNIXPRIV RESTRICTED.FILESYS.ACCESS UACC(NONE)
SETROPTS RACLIST(UNIXPRIV) REFRESH
```

Note: Do not attempt to deny access to certain restricted users by defining this resource with UACC(READ) and then permitting those users with access of NONE. The UACC of a resource cannot be used to allow access when the user is restricted.

6.34 Restricted user access checking

- For exception cases, permit the RESTRICTED user (or one of its groups) to RESTRICTED.FILESYS.ACCESS **(1 on access flow)**
  - PERMIT RESTRICTED.FILESYS.ACCESS CLASS(UNIXPRIV) ID(RSTDUSR) ACCESS(READ)

    (B on access flow)
  - SETROPTS RACLIST(UNIXPRIV) REFRESH
  - This does not grant the user access to any files - It just allows the 'other' bits to be used in access decisions for this user **(C on access flow)**
- SUPERUSER.FILESYS still applies to RESTRICTED users regardless **(2 on access flow)** of the existence of RESTRICTED.FILESYS.ACCESS **(D on access flow)**

Figure 6-34 How the RESTRICTED user profile is used on the authority checking

Checking RESTRICTED user profile

If needed, grant exceptions to certain restricted users to allow them to gain access based on the file's other bits. Add those users, or one of their groups, to the access list with READ authority.

This does not grant the user access to any files. It just allows the other bits to be used in access decisions for this user.

Note: SUPERUSER.FILESYS still applies to RESTRICTED users regardless of the existence of the RESTRICTED.FILESYS.ACCESS profile.

6.35 Access checking with ACLs (1)

- Profile - **SUPERUSER.FILESYS.ACLOVERRIDE**
  - Any user - not a superuser with UID(0) or file owner - and is denied access through the ACL can still access a file system resource if having sufficient authority to **SUPERUSER.FILESYS** resource in UNIXPRIV class **(4 on access flow)**
  - Therefore, to prevent this, you can force RACF to use your ACL authorizations to override a user's **SUPERUSER.FILESYS** authority by defining:

    (AA on access flow)

RDEFINE UNIXPRIV SUPERUSER.FILESYS.ACLOVERRIDE UACC(NONE)

SETROPTS RACLIST(UNIXPRIV) REFRESH

Figure 6-35 Access checking with the ACLOVERRIDE profile

SUPERUSER.FILESYS.ACLOVERRIDE profile

Any user who is not a superuser with UID(0) or the file owner, and who is denied access through the ACL, can still access a file system resource if the user has sufficient authority to the SUPERUSER.FILESYS resource in the UNIXPRIV class. Therefore, to prevent this, you can force RACF to use your ACL authorizations to override a user's SUPERUSER.FILESYS authority by defining the following profiles:

```
RDEFINE UNIXPRIV SUPERUSER.FILESYS.ACLOVERRIDE UACC(NONE)
SETROPTS RACLIST(UNIXPRIV) RACLIST
```

Figure 6-32 on page 229 shows the algorithm used by RACF in order to do authorization checking that now includes checking for profiles in the UNIXPRIV class for SUPER.FILESYS.ACLOVERRIDE, as shown at (AA).

Note: This describes the relationship between the existing SUPERUSER.FILESYS profile and the new SUPERUSER.FILESYS.ACLOVERRIDE profile. Either profile could get checked for a file; it depends upon the presence of an ACL for the file, and the contents of the ACL for granting access.

6.36 Access checking with ACLs (2)

- For exception cases, permit the user or group to SUPERUSER.FILESYS.ACLOVERRIDE with whatever access level would have been required for SUPERUSER.FILESYS **(A on access flow)**

PERMIT SUPERUSER.FILESYS.ACLOVERRIDE CLASS(UNIXPRIV) ID(ADMIN) ACCESS(READ) **(grants access)**

SETROPTS RACLIST(UNIXPRIV) REFRESH

SUPERUSER.FILESYS authority will still be required when an ACL does not exist for the file

(4 on access flow)

Figure 6-36 Access authority checking with the ACLOVERRIDE profile

Access checking with ACLs

For exception cases, permit the user or group to SUPERUSER.FILESYS.ACLOVERRIDE with whatever access level would have been required for SUPERUSER.FILESYS as follows:

```
PERMIT SUPERUSER.FILESYS.ACLOVERRIDE CLASS(UNIXPRIV) ID(ADMIN) ACCESS(READ)
SETROPTS RACLIST(UNIXPRIV) REFRESH
```

SUPERUSER.FILESYS authority is still checked when an ACL does not exist for the file. This should be done for administrators for whom you want total file access authority. That is, you do not want anyone to deny them access to a given file or directory by defining an ACL entry for them with limited, or no permission bit access.

This check is made at check (A) in the flow shown in the Figure 6-32 on page 229.

SUPERUSER.FILESYS.ACLOVERRIDE is checked only when a user's access was denied by a matching ACL entry based on the user's UID or one of the user's GIDs. If the user's access was denied by the file's permission bits, SUPERUSER.FILESYS is checked.

Important: The intent of these new profiles is to allow ACLs to behave as much as possible like RACF profile access lists. The new profiles are provided to avoid changing the default behavior since that could introduce compatibility issues with previous releases.

6.37 Create ACLs

- Use OMVS shell command - **setfacl**
- Use ISHELL panels
- 3 types of ACLs
  - Access ACL - Directory default ACL - File default ACL
- ACL inheritance
  - Can establish default (or 'model') ACLs on a directory or a file
    - They will get automatically applied to new files or directories created within the directory
    - Separate default ACL used for files and (sub) directories
  - Default ACLs can reduce administrative overhead

Figure 6-37 Types of ACLs that can be created

Creating ACLs for files and directories

ACLs have existed on various UNIX platforms for many years, but with variations in the interfaces. ACL support in z/OSV1R3 is based on a POSIX standard (that was never approved) and other UNIX implementations. In the POSIX standard, two different ACLs are referenced as follows:

- Base ACL entries are permission bits (owner, group, other). It refers to the FSP.
- Extended ACL entries are ACL entries for individual users or groups, such as the permission bits that are stored with the file, not in RACF profiles.

Access control lists are introduced in z/OS V1R3 as a way to provide a greater granularity for access to z/OS UNIX files and directories. z/OS V1R3 provides support for ACLs to control access to files and directories by individual user (UID) and group (GID). ACLs are now created and checked by RACF. ACLs are created, modified, and deleted by using either of the following:

- `setfacl` shell command
- ISHELL interface

ACL inheritance

With the introduction of ACL support, when new files and directories are created in a file system, ACL inheritance is the process of automatically associating an ACL with a newly created object without requiring administrative action. ACL inheritance associates an ACL with the newly created file, myfile, without requiring administrative action. However, it is not

always necessary to apply ACLs on every file or directory within a subtree. If you have a requirement to grant access to an entire subtree (for example, a subtree specific to a given application), then access can be established at the top directory. If a given user or group does not have search access to the top directory, then no files within the subtree will be accessible, regardless of the permission bit settings or ACL contents associated with these files. The user or group will still need permission to the files within the directory subtree where appropriate. If this is already granted by the “group” or “other” bits, then no ACLs are necessary below the top directory.

Note: When defining ACLs, it is recommended to place ACLs on directories, rather than on each file in a directory.

Default or model ACLs

The phrases “default ACL” and “model ACL” are used interchangeably throughout z/OS UNIX documentation. Other systems that support ACLs have default ACLs that are essentially the same as the directory default ACLs in z/OS UNIX. According to the X/Open UNIX 95 specification, additional access control mechanisms may only restrict the access permissions that are defined by the file permission bits. They cannot grant additional access permissions. Because z/OS ACLs can grant and restrict access, the use of ACLs is not UNIX 95-compliant.

To create an ACL for a file, you must have one of the following security access controls:

- Be the file owner
- BPX.SUPERUSER
- Have superuser authority (UID=0)
- Have READ access to the SUPERUSER.FILESYS.CHANGEPERMS profile in the UNIXPRIV class

Note: The RACF UNIXPRIV class was introduced in OS/390 V2R8. It allows you to define profiles in the UNIXPRIV class to grant RACF authorization for certain z/OS UNIX privileges. By defining profiles in the UNIXPRIV class, you can grant specific superuser privileges to users who do not have superuser authority (UID=0). This allows you to minimize the number of assignments of superuser authority at your installation and reduces your security risk.

To activate the use of ACLs in z/OS UNIX file authority checks, the following RACF command needs to be run to activate the new RACF class FSSEC:

```
SETROPTS CLASSACT(FSSEC)
```

6.38 ACL types

- **Access ACLs** This type of ACL is used to provide protection for a file system object (specific for a file or directory)
- **File default ACLs** This type is a model ACL that is inherited by files created within the parent directory. The file default ACL is copied to a newly created file as its access ACL. It is also copied to a newly created subdirectory as its file default ACL
- **Directory default ACLs** This type is a model ACL that is inherited by subdirectories created within the parent directory. The directory default ACL is copied to a newly created subdirectory as both its access ACL and directory default ACL

Figure 6-38 Types of ACLs that can be created

Types of ACLs

To reduce administrative overhead, three types of ACLs (extended ACLs) are defined in order to have the capability to inherit ACLs to newly created files and directories, as follows:

Access ACLs This type of ACL is used to provide protection for a file system object (specific for a file or directory).

File default ACLs This type is a model ACL that is inherited by files created within the parent directory. The file default ACL is copied to a newly created file as its access ACL. It is also copied to a newly created subdirectory as its file default ACL.

Directory default ACLs This type is a model ACL that is inherited by subdirectories created within the parent directory. The directory default ACL is copied to a newly created subdirectory as both its access ACL and directory default ACL.

6.39 OMVS shell commands for ACLs

- **setfacl -** The setfacl command sets, modifies, and deletes an ACL definition for a file or directory. setfacl has the following syntax:
  - setfacl [–ahqv] -s entries [path ...]
  - setfacl [–ahqv] -S file [path ...]
  - setfacl [–ahqv] -D type [...][path ...]
  - setfacl [–ahqv] -m|M|x|X EntryOrFile [...][path ...]
- **getfacl -** The getfacl command obtains and displays an ACL entry for a requested file or directory. It has the following syntax:
  - getfacl [–acdfhmos][-e user] file

Figure 6-39 OMVS shell commands to create and display ACLs

z/OS UNIX commands to create and display ACLs

The new shell commands, **`setfacl`** and **`getfacl`** are used to create, modify, and display ACL entries specified by the path.

Use ACLs to control access to files and directories by individual user and group. ACLs are used in conjunction with *permission bits*. They are created, modified, and deleted using the **`setfacl`** shell command. To display them, use the **`getfacl`** shell command. You can also use the ISHELL interface to define and display ACLs.

Among the options you can specify for **`setfacl`** are the following:

- **`-s`** option: Replaces the contents of an ACL with the entries specified on the command line. It requires that the base permissions be specified. The base permissions are specified similarly to extended ACL entries, except that there is no user or group name qualifier.
- **`-m`** option: Modifies ACL entries, or adds them if they do not exist.
- **`-D`** option: Specifies that the access ACL is to be deleted. When a file is deleted, its ACL is automatically deleted; there is no extra administrative effort required.

Recommendation: Use the -m option instead of the -s option when creating ACLs. Creating ACLs using the ISHELL is probably preferred. See “Using the ISHELL panel” on page 245 through “Access ACL after creation” on page 253.

6.40 Create ACLs for a specific directory

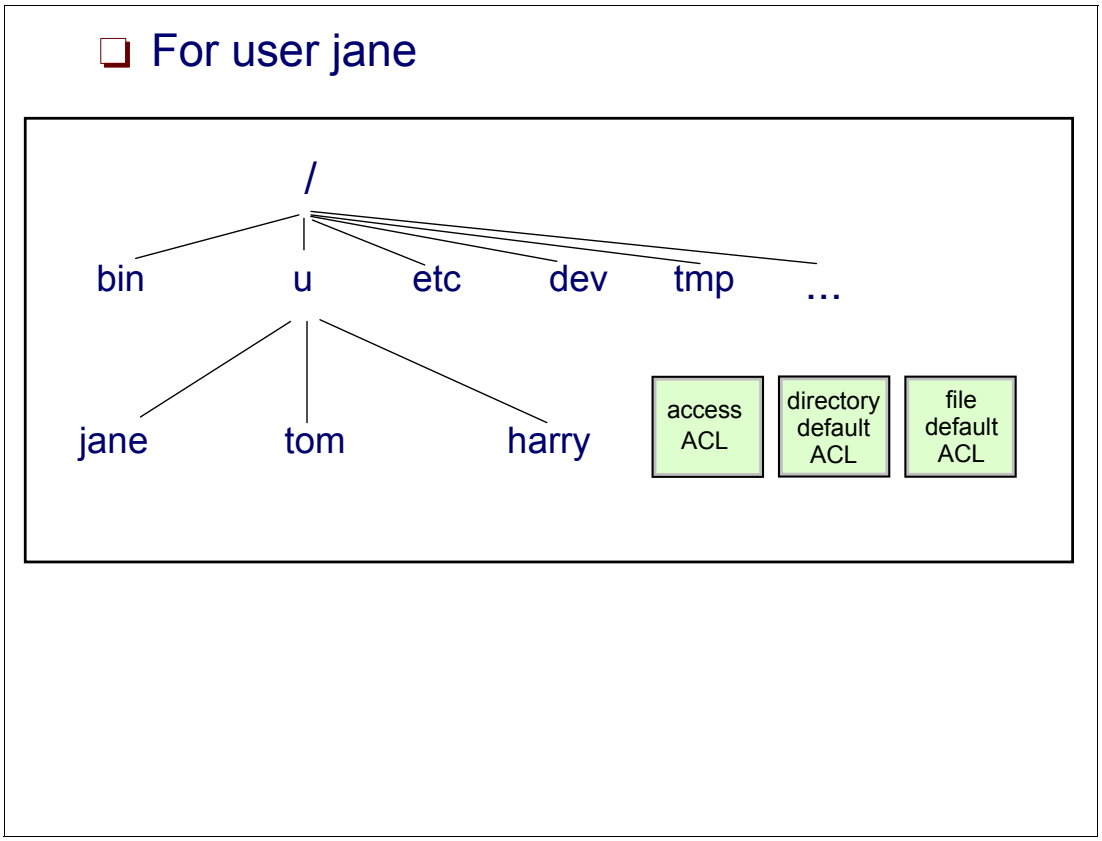


Figure 6-40 Creating ACLs for a specific directory HARRY

Example of creating ACLs

The next few figures show how to create the three ACLs shown in Figure 6-40 that give user JANE access to directory HARRY and any directories and files defined under directory HARRY.

You can use an access ACL on a directory to grant search access only to those users and groups who should have file access. The access ACL of the directory might have been automatically created as the result of a directory default ACL on its parent. Make sure that the “other” and perhaps the “group” search permission bit is off for the parent directory.

To minimize the impact to performance, keep ACLs as small as possible, and permit groups to files instead of individual users. The path length of the access check will increase with the size of an ACL, but will be smaller than the associated checking would be for a RACF profile with the same number of entries in its access list.

6.41 Create an access ACL

- Create an access ACL - using **setfacl -m option**
  - For directory harry - create an access ACL that gives user ID JANE rwx access to directory harry
    - setfacl **-m** "u:jane:rwx" harry

```
ROGERS @ SC65:/u>ls -al
total 152
dr-xr-xr-x  11 HAIMO    NOGROUP        0 Aug  2 10:45 .
drwxr-xr-x  48 HAIMO    SYS1       24576 Jul 25 14:44 ..
drwx------+  2 HARRY    SYS1        8192 Aug  2 10:44 harry
drwx------   2 JANE     SYS1        8192 Aug  2 10:44 jane
drwxr-xr-x   2 HAIMO    SYS1        8192 Jun 28 12:23 ldapsrv
drwx------   2 HAIMO    SYS1        8192 Aug  1 11:02 rogers
drwxr-xr-x   2 HAIMO    SYS1        8192 Nov 15  2001 syslogd
drwx------   3 HAIMO    SYS1        8192 May 26 11:03 user1
```

Figure 6-41 Create an access ACL for directory HARRY

Creating an access ACL

When you are setting the access ACL, the ACL entries must consist of three required base ACL entries that correspond to the file permission bits. The ACL entries must also consist of zero or more extended ACL entries, which will allow a greater level of granularity when controlling access. The permissions for base entries must be in absolute form. As shown in Figure 6-41, issuing the **`ls -al`** command, the **+** sign following the permission bits for directory HARRY indicates that an ACL exists for that directory.

Using the -m option

With the **`setfacl`** command, you use the **`-m`** option to create an ACL. The base ACL (permission bits) are indicated by omitting user or group qualifiers.

This command creates an access ACL that gives user ID JANE rwx access to directory HARRY:

```
ROGERS @ SC65:/u>setfacl -m “u:jane:rwx” harry
```

6.42 Display the access ACL

❑ Display the access ACL - using **getfacl**

➢ **-a** Displays the access ACL entries

```
ROGERS @ SC65:/u>getfacl -a harry
#file:  harry/
#owner: HARRY
#group: SYS1
user::rwx     <=== The owner's permission bit setting
group::---    <=== The group's permission bit setting
other::---    <=== Permission bit setting if neither user nor group
user:JANE:rwx
```

Figure 6-42 Displaying the created ACL using the **`getfacl`** *command*

Displaying the defined ACL

The **`getfacl`** command displays the comment header, base ACL entries, and extended ACL entries, if there are any, for each file that is specified. It also resolves symbolic links. You can specify whether to display access, file default, or directory default. You can also change the default display format. The output can be used as input to **`setfacl`**.

-a option for getfacl

This option displays the access ACL entries. This is the default if **`-a`**, **`-d`**, or **`-f`** is not specified. Figure 6-42 displays the ACL entry just created for directory HARRY.

6.43 Create a directory default ACL

- Create a directory default ACL
  - A model ACL that is inherited by subdirectories created within the parent directory
  - The directory inherits the model ACL as its directory default ACL and as its access ACL
    - setfacl -m "d:u:jane:rwx" harry

```
ROGERS @ SC65:/u>getfacl -d harry
#file:  harry/
#owner: HARRY
#group: SYS1
default:user:JANE:rwx
```

Figure 6-43 Creating a directory default ACL for directory HARRY

Creating a directory default ACL

Directory default ACLs are model ACLs that are inherited by subdirectories created within the parent directory. The directory inherits the model ACL as its directory default ACL, and as its access ACL when a new directory is created under directory HARRY.

To facilitate management of ACLs, you can define a default ACL in a directory; it will then be automatically inherited by an object, as follows:

- The directory default ACL is copied to a newly created subdirectory as both its access ACL and directory default ACL. You can modify or delete inherited ACLs later.

For directory HARRY, the following command creates a directory default ACL that gives user JANE rwx access in a directory default ACL for directory HARRY:

```
ROGERS @ SC65:/u>setfacl -m "d:u:jane:rwx" harry
```

6.44 Create a file default ACL

- Create a file default ACL
  - A model ACL that is inherited by files created within the parent directory
  - The file inherits the model ACL as its access ACL
  - Directories also inherit the file default ACL as their file default ACL
    - setfacl -m "f:u:jane:r--" harry

```
ROGERS @ SC65:/u>getfacl -f harry
#file:  harry/
#owner: HARRY
#group: SYS1
fdefault:user:JANE:r--
```

Figure 6-44 Create a file default ACL for directory HARRY

Creating a file default ACL

File default ACLs are model ACLs that are inherited by files created within the parent directory. The file inherits the model ACL as its access ACL. Directories also inherit the file default ACL as their file default ACL.

To facilitate management of ACLs, you can define a default ACL in a directory; it will then be automatically inherited by an object, as follows:

- The file default ACL is copied to a newly created file as its access ACL. It is also copied to a newly created subdirectory as its file default ACL.

For directory HARRY, the following command creates a file default ACL that gives user JANE r-- access in a file default ACL for directory HARRY:

```
ROGERS @ SC65:/u>setfacl -m "f:u:jane:r--" harry
```

6.45 Creating all ACL types

setfacl -s "u::rwx,g::---,o::---,u:jane:rwx,d:u:jane:rwx,f:u:jane:r--" harry

```
ROGERS @ SC65:/u>getfacl -adf harry
#file:  harry/
#owner: HARRY
#group: SYS1
user::rwx
group::---
other::---
user:JANE:rwx
fdefault:user:JANE:r--
default:user:JANE:rwx
```

Figure 6-45 Creating all three ACLs and specifying the permission settings

Creating all ACLs using the -s option

The **-s** option replaces the contents of an ACL with the entries specified on the command line. It requires that the base permissions be specified. The base permissions are specified similarly to extended ACL entries, except that there is no user or group name qualifier.

If you want to create all three ACLs for the directory, you can issue just one command for ACLs, and by using **-s**, you can specify the current permission bit settings as follows:

```
setfacl -s "u::rwx,g::---,o::---,u:jane:rwx,d:u:jane:rwx,f:u:jane:r--" harry
```

To display the ACLs just created by the **setfacl** command, issue the following command:

```
getfacl -adf harry
```

6.46 Using the ISHELL panel

```
  File  Directory  Special_file  Tools  File_systems  Options  Setup  Help
                      UNIX System Services ISPF Shell
Command ===>

Enter a pathname and do one of these:

    - Press Enter.
    - Select an action bar choice.
    - Specify an action code or command on the command line.

Return to this panel to work with a different pathname.
                                                              More:     +
   /u

EUID=0
```

Figure 6-46 Using the ISHELL panel to create ACLs

Create an ACL using the ISHELL

With the ISPF shell (ISHELL), a user or system programmer can use ISPF dialogs instead of shell commands to perform many tasks, especially those related to file systems and files. An ordinary user can use the ISPF shell to work with:

- Directories
- Regular files
- FIFO special files
- Symbolic links, including external links

You can also run shell commands, REXX programs, and C programs from the ISPF shell. The ISPF shell can direct stdout and stderr only to an HFS file, not to your terminal.

Note: Features to improve the ISHELL functionality were added on z/OS V1R3. The ISHELL is a user interface that allows users to work with menus rather than sometimes cryptic commands. In z/OS V1R3, many new features that have been requested over the years to improve ISHELL functionality and panel navigation are implemented.

These new features are a significant upgrade to the ISHELL, and many of them are part of the Directory List option, which lists files in a particular directory. Other enhancements include sorting and highlighting support, as well as easy ways to access information. The effective user ID(EUID) is displayed on the panel so you can know the authority you have at any particular moment. In the figure, EUID=0 indicates that the current UID that is accessing the ISHELL is a superuser.

6.47 Create an access ACL using ISHELL

```
  File  Directory  Special_file  Commands  Help
                                Directory List
Command ===> _____________________________________________________________

Select one or more files with / or action codes.  If / is used also select an
action from the action bar otherwise your default action will be used.  Select
with S to use your default action.  Cursor select can also be used for quick
navigation.  See help for details.
EUID=0   /u/
  Type  Perm  Changed-EST5EDT   Owner      ------Size  Filename    Row 1 of 8
_ Dir    555  2003-02-24 09:42  HAIMO               0  .
_ Dir    700  2003-02-24 09:41  HAIMO            8192  rogers
_ Dir    755  2003-02-20 09:15  HAIMO           24576  ..
_ Dir    700  2002-09-13 14:47  HAIMO            8192  tom
a Dir    700  2002-08-02 15:40  HARRY            8192  harry
_ Dir    700  2002-08-02 10:44  JANE             8192  jane
_ Dir    755  2002-06-28 12:23  HAIMO            8192  ldapsrv
_ Dir    755  2001-11-15 14:35  HAIMO            8192  syslogd
```

- Use "a" as an action character to access the File attributes
- Create an ACL for user jane to access directory harry

Figure 6-47 Creating an access ACL using the ISHELL

Create an access ACL using the ISHELL

Figure 6-47 shows the Directory List that is displayed once you have entered the ISHELL, typed **/u** on the command line, and pressed Enter.

Placing an “**a**” action code for directory HARRY results in the File Attribute panel being displayed. This is the first step to create ACLs.

In the following sections we describe how to create an ACL that gives user JANE access to directory HARRY.

6.48 File attributes panel for /u/harry

```
  File  Directory  Special_file  Commands  Help
_
    Edit  Help
C
                Display File Attributes
S
a   Pathname : /u/harry                                   s used also select an
w                                   More:     +            will be used.  Select
n   File type . . . . . . : Directory                      so be used for quick
E   Permissions . . . . . : 700
    Access control list . : 0                              ilename    Row 1 of 8
_   File size . . . . . . : 8192
_   File owner  . . . . . : HARRY(10103)                   ogers
_   Group owner . . . . . : SYS1(2)                        .
_   Last modified . . . . : 2002-08-02 15:40:01            om
a   Last changed  . . . . : 2002-08-02 15:40:01            arry
_   Last accessed . . . . : 2002-08-02 15:40:30            ane
_   Created . . . . . . . : 2002-08-02 10:44:16            dapsrv
_   Link count  . . . . . : 3                              yslogd
     F1=Help         F3=Exit         F4=Name
     F7=Backward     F8=Forward     F12=Cancel
```

- Current Access control list shows a 0
- To create an ACL, place cursor under Edit and press Enter

Figure 6-48 Display File Attribute panel showing no ACL exists

Working in the ISHELL

If you scroll forward twice, you can see if a Directory Default ACL or File Default ACL is defined, as shown in Figure 6-49. These two fields (Directory Default and File Default ACL) only apply to directory files.

6.49 File attributes panel showing ACLs

```
_  File  Directory  Special_file  Commands  Help
_
_    Edit  Help
C    ____________________________________________________
                  Display File Attributes
S
a    Pathname : /u/harry
w                                          More:   -
n    Major device  . . . . : 0
E    Minor device  . . . . : 0
     File format . . . . . : NA
_    Shared AS . . . . . . : -
_    APF authorized  . . . : -
_    Program controlled  . : -
_    Shared library  . . . : -
a    Char Set ID/Text flag : 00000 OFF
_    Directory default ACL : 0
_    File default ACL  . . : 0
_    Seclabel  . . . . . . : ____________________
      F1=Help         F3=Exit         F4=Name
      F7=Backward     F8=Forward     F12=Cancel
```

Figure 6-49 Display file attributes panel showing ACLs

Panel to display ACLs

Figure 6-49 shows the Directory default ACL and the File default ACL for the pathname /u/harry.

For directory /u/harry, the Directory default ACL and File default ACL fields both show zeros, indicating that no directory or file ACL exists.

6.50 Select option to create an access ACL

```
  File  Directory  Special_file  Commands  Help
_
     Edit  Help
C
     8_ 1. Mode fields...
        2. Owning user...                                  s used also select an
S       3. Owning group...                                  will be used.  Select
a       4. User auditing...            More:     +         so be used for quick
w       5. Auditor auditing...
n       6. File format...
E       7. Extended attributes...                          ilename   Row 1 of 10
        8. Access contol list...
_       9. Directory default ACL...                        ogers
_      10. File default ACL...                             .
_                                   14:27:11               artr2
a    Last changed  . . . . : 2003-01-16 14:27:11           c63
_    Last accessed . . . . : 2002-11-13 18:58:58           om
_    Created . . . . . . . : 2001-07-13 10:56:15           arry
_    Link count  . . . . . : 14                            ane
_     F1=Help         F3=Exit         F4=Name              dapsrv
_     F7=Backward     F8=Forward     F12=Cancel            yslogd
```

Figure 6-50 Select Option 8 to create the access ACL

Select option for ACL creation

Specify either option 8, 9, or 10, then press Enter; you now have access to modify, add, or delete the ACL entries for the access ACL, directory default ACL, and the file default ACL, respectively.

6.51 Create an access ACL

```
  File  Directory  Special_file  Commands  Help
_
  Edit  Help
C
                Display File Attributes
S                                                  s used also select an
a  Pathname : /u/harry                              will be used.  Select

                  Access Control List: Access
Command ===> _                                        Scroll ===> PAGE

Type over read, write or execute permissions to make a change.
Clear the value to reset it, anything else will set it.
To delete, place a D in the S column for an entry or use command D * for
all entries. Use commands SORT ID or SORT NAME to reorder the table.

Option:     2  1. Add group  2. Add user   3. Copy       4. Replace

S           ID  Name      Read  Write  Execute  Type
****************************** Bottom of data ******************************
```

- Type a 2 on the Option line to create ACL - press Enter

Figure 6-51 Creating the access ACL for directory HARRY

Window to create the ACL

Figure 6-51 shows the panel used to create an ACL for a user ID to give access to a file or directory. In this example, you are going to give user JANE access to directory HARRY. To do this, place a 2 on the Option line and press Enter.

6.52 Add an access ACL

```
  File  Directory  Special_file  Commands  Help
_ ┌──────────────────────────────────────────────────────────────┐
  │  Edit  Help                                                  │
C │ ────────────────────────────────────────────────────────────
  │                Display File Attributes                       │
S │                                                              │
a │ Pathname : /u/harry                                          │
  └──────────────────────────────────────────────────────────────┘
  ┌──────────────────────────────────────────────────────────────┐
  │                 Access Control List: Access                  │
C │ ┌──────────────────────────────────┐  __________ Scroll ===> PAGE
  │ │              Add an ACL Entry    │
T │ │                                  │  o make a change.
C │ │ Enter new User ACL               │  l set it.
T │ │                                  │  try or use command D * for
a │ │ Permissions:                     │   to reorder the table.
  │ │ /  Read                          │
O │ │ /  Write                         │  opy       4. Replace
  │ │ /  Execute                       │
S │ │                                  │    Type
* │ │ Name or ID   jane_____           │  ****************************
  │ │                                  │
  │ │  F1=Help      F3=Exit     F12=Cancel
  │ └──────────────────────────────────┘
  └──────────────────────────────────────────────────────────────┘
```

- ❑ Place a / next to each permission required
- ❑ Enter name of user ID - press Enter

Figure 6-52 The Add an ACL Entry window is used to create the ACL

Window to choose permission bits for the ACL access

Use the panel shown in Figure 6-52 to specify the permission bit settings that you require to give user JANE access to directory HARRY. Give user JANE `rwx` (read, write and execute) access to directory HARRY, as shown in the Pathname field in the figure. the panel in Figure 6-53 shows the newly created ACL after you press Enter.

6.53 Access ACL after creation

```
  File  Directory  Special_file  Commands  Help
_ ┌──────────────────────────────────────────────┐ ──────────────────────
  │  Edit  Help                                  │
C │ ─────────────────────────────────────────────────────────────────────
  │            Display File Attributes           │
S │                                              │ s used also select an
a │ Pathname : /u/harry                          │  will be used.  Select
  ┌──────────────────────────────────────────────────────────────────────┐
  │                Access Control List: Access                Row 1 to 1 of 1
  │ Command ===> ____________________________________________ Scroll ===> PAGE
  │
  │ Type over read, write or execute permissions to make a change.
  │ Clear the value to reset it, anything else will set it.
  │ To delete, place a D in the S column for an entry or use command D * for
  │ all entries. Use commands SORT ID or SORT NAME to reorder the table.
  │
  │ Option:    _  1. Add group  2. Add user   3. Copy       4. Replace
  │
  │ S           ID  Name      Read  Write  Execute  Type
  │ _        10102  JANE       R      W       X     User
  │ ***************************** Bottom of data ******************************
```

Figure 6-53 Display of access ACL after creation

Display of access ACL using the ISHELL

The ACL list is ordered with the group ACLs before the user ACLs. Initially the group and user ACLs are ordered by name. This order can be changed with the command **`sort id`** or **`sort name`**. This list will only show UIDs and GIDs that have associated names. The **`setfacl`** shell utility must be used to manage entries with UIDs and GIDs that are not defined on this system.

The panel shows the newly created access ACL that gives user JANE rwx access to directory /u/harry.

You can use D in the selection field (S) to delete the ACL entry.

The END command (F3) will exit the ACL list and save any changes. The SAVE command can also be used to save changes but will not exit the ACL list.

6.54 ACL inheritance: New directory/new file

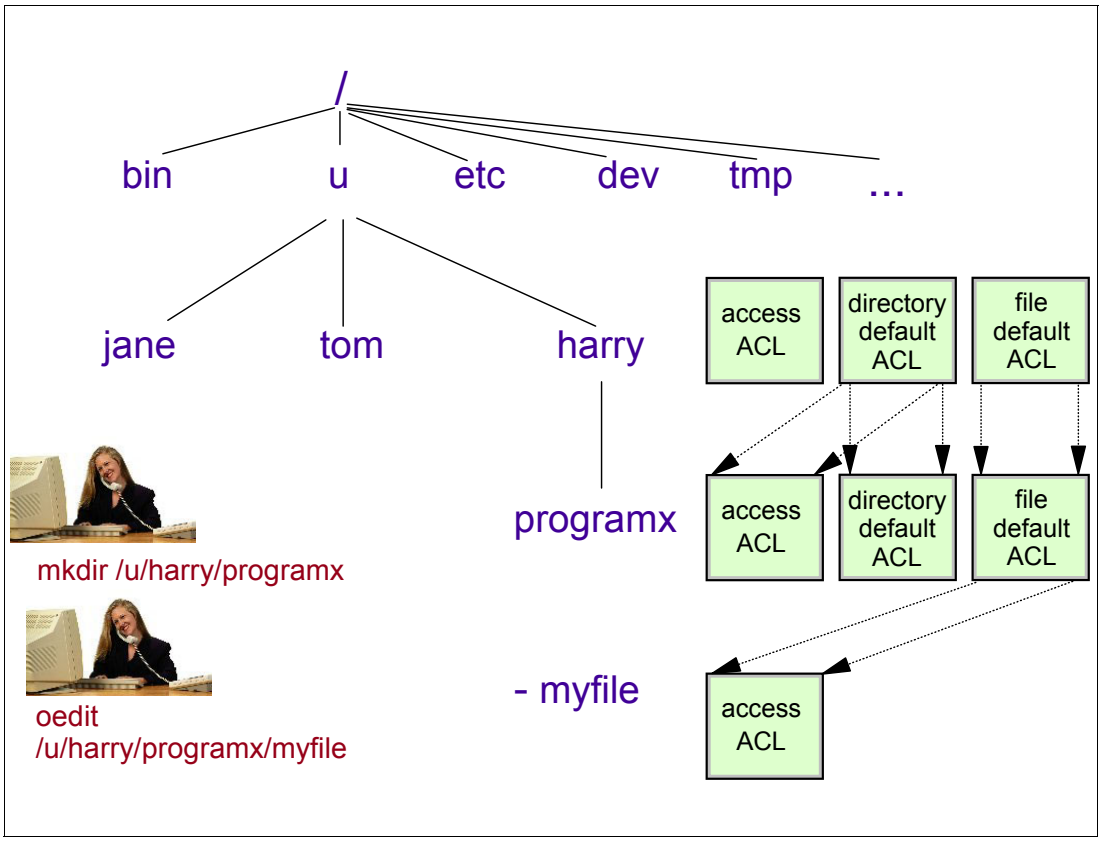


Figure 6-54 ACL inheritance

ACL inheritance

The directory structure is changed, as shown in Figure 6-54, by issuing the following command:

```
mkdir /u/harry/programx
```

The new directory, programx, inherits an access ACL from the directory default ACL of directory HARRY, and inherits the directory default ACL and file default ACL from directory HARRY.

ACL inheritance, as shown in the figure, associates an ACL with the newly created file, *myfile*, without requiring administrative action. However, it is not always necessary to apply ACLs on every file or directory within a subtree. If you have a requirement to grant access to an entire subtree (for example, a subtree specific to a given application), then access can be established at the top directory. If a given user or group does not have search access to the top directory, then no files within the subtree will be accessible, regardless of the permission bit settings or ACL contents associated with these files. The user or group will still need permission to the files within the directory subtree where appropriate. If this is already granted by the “group” or “other” bits, then no ACLs are necessary below the top directory.

Note: When defining ACLs, it is recommended to place ACLs on directories, rather than on each file in a directory.

6.55 Multilevel security with z/OS V1R5

- Support is added allowing SECLABELs to be associated with file system resources and users to provide greater restrictiveness than is possible with POSIX permissions alone
  - ALL systems must be z/OS V1R5 or above
  - Requires use of zFS for ROOT and /dev resources
  - Requires use of zFS for all file systems mounted for readwrite access

Figure 6-55 Using multilevel security beginning with z/OS V1R5

Multilevel security (MLS)

Traditionally, access to z/OS UNIX resources is based on POSIX permissions. Beginning with z/OS V1R5, if the SECLABEL class active, additional security is provided by authorization checks that are performed for security labels, in addition to POSIX permissions. Security labels are used to maintain multiple levels of security within a system. By assigning a security label to a resource, the security administrator can prevent the movement of data from one level of security to another within the z/OS UNIX environment.

When the SECLABEL class is active, security labels can be set on z/OS UNIX resources.

zFS root

zFS and HFS can both participate in shared sysplexes. However, only zFS supports security labels. Therefore, in a multilevel-secure environment, you must use zFS file systems instead of HFS file systems. See *z/OS Planning for Multilevel Security*, GA22-7509 for more information on multilevel security and migrating your HFS version root to a zFS version root with security labels.

When an HFS file system or zFS aggregate is created, the file system root will be assigned the security label that is specified in the RACF data set profile that covers the data set name. If a security label is not specified or if a data set profile does not exist, then a security label will not be assigned to the file system root.

6.56 Multilevel security (MLS)

- Multilevel security is a security policy that allows:
  - Classification of data and users based on:
    - Hierarchical security levels combined with
    - Non-hierarchical security categories
- Multilevel-secure security policy has two primary goals, as follows:
  - First, the controls must prevent unauthorized individuals from accessing information at a higher classification than their authorization
  - Second, the controls must prevent individuals from declassifying information

Figure 6-56 Understanding multilevel security

Classifying data

Security classification of users and data allows installations to impose additional access controls on sensitive resources. Each user and each resource can have a security classification in its profile. You can choose among the following:

- A security level (SECLEVEL) is an installation-defined name that corresponds to a numerical security level (the higher the number, the higher the security level).
- A security category (CATEGORY) is an installation-defined name that corresponds to a department or an area within an organization in which the users have similar security requirements.
- A security label (SECLABEL) is an installation-defined name that corresponds to a security level and zero or more security categories.

Multilevel-secure environment

The security policy that you implement in a multilevel-secure environment has as its key feature a system of access controls that not only prevents individuals from accessing information at a classification for which they are not authorized, but also prevents individuals from declassifying information. The system must protect resources of different levels of sensitivity.

6.57 MLS support for z/OS UNIX

- SECLABELs for z/OS UNIX processes and sockets
- SECLABELs for z/OS UNIX files and directories
- SECLABELs for z/OS UNIX interprocess communications (IPC)
- zFS supports security labels for MLS
  - Externals for MLS are contained in other elements/components
  - (RACF, shell, z/OS UNIX APIs, etc.)
  - zFS supports low level functions for MLS

Figure 6-57 MLS support in z/OS UNIX

SECLABELS for z/OS UNIX processes and sockets

This function has been modified to allow SECLABELs to be defined as follows:

- For workstations (allowing for both reading and writing)
- To support the z/OS UNIX environment where a user may enter the system from a remote IP address using an application such as rlogin
- To associate SECLABELs to IP addresses

SECLABELS for z/OS UNIX files and directories

RACF assigns a user's SECLABEL to a new file or directory when it is created. The SECLABEL cannot be changed. Use the z/OS UNIX command **`chlabel`** to create the SECLABEL. The SECLABEL can be changed by copying the file to a directory with a different SECLABEL. Because subdirectories have the same SECLABEL as the parent directory, files in a directory will have the same SECLABEL as the directory.

SECLABELS for z/OS UNIX interprocess communication (IPC)

When creating an IPC security packet (ISP), if the SECLABEL class is active, RACF will copy the process SECLABEL (if one exists) into the ISP. Later, when checking access to the IPC for a subsequent connection, RACF will reject the request if the current process does not have a SECLABEL or the SECLABEL does not match. Once a SECLABEL has been assigned to an IPC object, there is no way to change it.

Access checking for IPC objects will be treated as EQUALMAC checking, meaning that SECLABELs must be equivalent, unless the resource's SECLABEL is SYSMULTI, or the accessor's SECLABEL is SYSMULTI.

zFS support for SECLABELs

The zSeries file system (zFS) supports security labels. A zFS file system is contained in a VSAM linear data set. The security label of the root within each zFS file system data set is determined at the time the file system aggregate (container) is allocated, from the security label of the profile in the DATASET class that covers the aggregate. If the SECLABEL class is active when allocating the aggregate, all file systems subsequently created within that aggregate contain a root with the security label that is specified in the profile for that aggregate. If no profile exists for the aggregate, or if it exists but does not specify a security label, and if the MLFSOBJ option is not active, the root for any file systems within that aggregate has no security label. If the MLFSOBJ option is active, requiring security labels on all file system objects, the user's security label is assigned to the root of the file system.

zFS stores security labels in the FSP in the metadata. zFS calls RACF for security label processing and also can do some checking on its own.

MLS support

z/OS V1R5 support includes support for MLS with the following components:

- RACF
- z/OS UNIX System Services
- zFS
- TCP/IP
- JES2 (not JES3)
- SDSF
- DFSMS - MLS SECLABELs in ACS Routines
- DB2 V8 - Security labels on rows in tables

6.58 Mandatory access control (MAC)

- A MAC check compares the security labels of the subject and object and grants the subject access to the object
  - A subject can read an object if the subject's security label dominates the object's security label
  - A subject can write to an object if the object's security label dominates the subject's security label
  - A subject cannot write to an object whose security label the subject's security label dominates, unless the security labels are equivalent - not allowed to write down
  - A subject can both read and write an object only if the subject's and object's security labels are equivalent

Figure 6-58 Mandatory access control (MAC) checking

MAC checking

Mandatory access control is based on the theory of *dominance*, and is achieved through the use of security labels.

Dominance

One security label dominates a second security label when the following two conditions are true:

- The security level that defines the first security label is greater than or equal to the security level that defines the second security label.
- The set of security categories that defines the first security label includes the set of security categories that defines the second security label.

MAC checking overview

A mandatory access check compares the security labels of the subject and object and grants the subject access to the object as follows:

- A subject can read an object if the subject's security label dominates the object's security label.
- A subject can write to an object if the object's security label dominates the subject's security label. A subject cannot write to an object whose security label the subject's security label dominates, unless the security labels are equivalent; we say that the subject is not allowed to write down.

- A subject can both read and write an object only if the subject's and object's security labels are equivalent.

To ensure that a user does not declassify data, a subject can read from and write to (alter) only an object with an equivalent security label; however, a subject can copy information from an object having a security label that does not dominate the security label of the subject to an object having the subject's current security label. Or, for objects that support write-only processing, such as z/OS UNIX files, to an object with a security label that dominates the subject's security label.

6.59 Discretionary access control (DAC)

- Once the user passes a MAC check, a discretionary check follows
- A DAC check ensures that the user is identified as having a "need to know" for the requested resource
  - DAC uses other access control information, such as:
    - Access control list in the profile protecting a resource
    - z/OS UNIX access control (permissions, the access control list, and the UNIXPRIV class)

MAC check occurs first, then DAC
Or DAC only, if the SECLABEL class is not active

Figure 6-59 Discretionary access control (DAC) checking

Discretionary access control checking

Discretionary access control is the principle of restricting access to objects based on the identity of the subject (the user or the group to which the user belongs). Discretionary access control is implemented using access control lists. A resource profile contains an access control list that identifies the users who can access the resource and the authority (such as read or update) each user is allowed in referencing the resource. The security administrator defines a profile for each object (a resource or group of resources), and updates the access control list for the profile. This type of control is discretionary in the sense that subjects can manipulate it, because the owner of a resource, in addition to the security administrator, can identify who can access the resource and with what authority.

Once the user passes the mandatory access check, a discretionary check follows. The discretionary access check ensures that the user is identified as having a “need to know” for the requested resource. The discretionary access check uses other access control information, such as the access control list in the profile protecting a resource, or z/OS UNIX access control (permissions, the access control list, and the UNIXPRIV class).

6.60 SECLABELs and MAC

SETROPTS CLASSACT(SECLABEL) RACLIST(SECLABEL)

- PURPLE could be a SECLABEL name indicating SECLEVEL secret for categories PROJECTA, PROJECTB, and PROJECTC
- GREEN could be a SECLABEL name indicating SECLEVEL sensitive for categories PROJECTA and PROJECTB
- RED could be a SECLABEL name indicating SECLEVEL unclassified for category PROJECTC

SECLABEL	SECLEVEL	CATEGORY (Not required)
PURPLE	SECRET	PROJECTA, PROJECTB, PROJECTC
GREEN	SENSITIVE	PROJECTA, PROJECTB
RED	UNCLASSIFIED	PROJECTC

Figure 6-60 SECLABELs with categories and MAC checking

SECLABELs with categories and MAC checking

A security label establishes an association between a RACF security level and a set of zero or more RACF security categories. For example, the system shown in Figure 6-60 might have three security levels: unclassified, sensitive, and secret; and three security categories: Project A, Project B, and Project C. Then, PURPLE could be a security label name indicating Secret for Project A, Project B, and Project C. COLUMBIA could be a security label name meaning Sensitive for Project A and Project B; UNION could be a security label name indicating unclassified for Project C.

Defining SECLABELs

The security administrator defines two profiles in the RACF SECDATA resource class that define the security levels and security categories for the system:

- The SECLEVEL profile contains a member for each hierarchical security level in the system.
- The CATEGORY profile contains a member for each non-hierarchical category in the system.

SECLEVEL The hierarchical security level defines the degree of sensitivity of the data. “SECRET,” “SENSITIVE,” and “UNCLASSIFIED” are examples of levels you could define. You might define “SECRET” to be a security level of 30, “SENSITIVE” to be a level of 20, and “UNCLASSIFIED” to be a level of 10. The security administrator can define up to 254 security levels.

CATEGORY The non-hierarchical categories further qualify the access capability. The security administrator can define zero or more categories that correspond to some grouping arrangement in the installation. “PROJECTA,” “PROJECTB,” and “PROJECTC” could be categories defined.

Security labels After defining the SECLEVEL and CATEGORY profiles, the security administrator defines a profile in the SECLABEL resource class for each security label. Each security label name must be unique. Each SECLABEL profile specifies the particular combination of a SECLEVEL member and zero or more members of the CATEGORY profile that applies to the security label. You do not need to define a security label for every possible combination of level and category.

SECLABEL dominance with categories

One security label dominates a second security label when the following two conditions are true:

- The security level that defines the first security label is greater than or equal to the security level that defines the second security label.
- The set of security categories that define the first security label includes the set of security categories that defines the second security label.

Two security labels are said to be disjoint or incompatible if neither dominates the other because they have incompatible sets of categories. For example, if SECLABEL GREEN has categories PROJECTA and PROJECTB, and SECLABEL RED has categories PROJECTC, neither contains all the categories of the other, so neither dominates the other and they are disjoint.

6.61 Special SECLABELs and definitions

- SYSHIGH - SYSLOW - SYSNONE
- SYSMULTI - (New with MLS in z/OS V1R5)
- Defining SECLABELs

```
RDEFINE SECLABEL security-label
SECLEVEL(seclevel-name)
ADDCATEGORY(category-1 category-2 ...)

PERMIT security-label CLASS(SECLABEL)
ACCESS(READ) ID(user-id-1 user-id-2 ...)

SETROPTS CLASSACT(SECLABEL)
RACLIST(SECLABEL)
          or
SETROPTS RACLIST(SECLABEL) REFRESH
```

Figure 6-61 Special SECLABELs and their definitions

System SECLABELs

At IPL time, RACF dynamically creates system SECLABELs that cannot be defined, modified, or deleted by any user, but can be assigned to users and resources. The following SECLABELs are defined:

SYSHIGH SYSHIGH is created at the highest security level defined to RACF and includes all defined categories. Resources with SYSHIGH can be accessed only by users with SYSHIGH; users with SYSHIGH have access to all other SECLABELs. If a higher level or additional categories are defined, SYSHIGH automatically assumes the new values when the SECLABEL class is refreshed.

SYSLOW SYSLOW is created at the lowest security level defined to RACF and includes no categories. Resources with SYSLOW can be accessed by users with any valid SECLABEL; users with SYSLOW can access only resources that are SYSLOW. If a lower level is defined, SYSLOW automatically assumes the new value when the SECLABEL class is refreshed.

SYSNONE SYSNONE is assigned to resources that have no security-relevant data. MAC verification considers SYSNONE to be equivalent to any user's SECLABEL, automatically allowing access. Assigning SYSNONE to a user has the same effect as assigning SYSLOW to the user.

SYSMULTI The SYSMULTI SECLABEL is new with z/OS V1R5. The SYSMULTI security label is equivalent to any other security label.

6.62 SYSMULTI SECLABEL

- SYSMULTI
  - Compares as "equivalent" to any other defined SECLABEL for MAC decisions
  - Intended for daemons and servers that can accept connections from users running at different classification levels (SECLABELs) and properly mediate data access
  - UNIX directories (often, not always, root in a file system) that can have subdirectories of different SECLABELs

Figure 6-62 SYSMULTI SECLABEL

SYSMULTI SECLABEL

SYSMULTI can be assigned in cases where any classification of data could be processed. It compares as equivalent to any other defined SECLABEL for MAC decisions.

It is intended for the following types of functions:

- Daemons and servers that can accept connections from users running at different classification levels (SECLABELs) and properly mediate data access
- UNIX directories (often, not always, root in a file system) that can have subdirectories of different SECLABELs

It generally should not be assigned to real users, nor to a server that is not designed to handle multiple SECLABELs.

Assigning SYSMULTI to directories

If a directory has been assigned a security label, then new files and directories created within that directory will inherit a security label as follows:

- If the parent directory is assigned a security label of SYSMULTI, the new file or directory will be assigned the security label of the user. If the user has no security label, then one is assigned to the new object.
- If the parent directory is assigned a security label other than SYSMULTI, the new file or directory is assigned the same security label as the parent directory.

6.63 z/OS UNIX and SECLABELs

- chlabel command
  - This new shell command chlabel can be used to set security labels for UNIX files and directories
    - Requires RACF SPECIAL authorization
  - Recommended do this command before MLS activated
- Once SECLABEL has been set, it cannot be changed
  - chlabel [–cqR] [–h|–L] seclabel pathname
    - By specifying the -R parameter, labels are set “recursively” on file subdirectories

Note: Only the zFS file systems supports the setting of SECLABELs

Figure 6-63 Defining SECLABELs with z/OS UNIX

Defining SECLABELs

The **`chlabel`** shell command allows a security administrator to assign a security label to a file system object that does not have one.

If a z/OS UNIX file, directory, or symbolic link was created in a zFS file system without being assigned a security label (for example, if the SECLABEL class was not active when the file, directory, or symbolic link was created), the security administrator can assign a security label to it using the **`chlabel`** shell command.

chlabel command

`chlabel` sets the multilevel security label of the files and directories specified by pathname. Setting the seclabel is only allowed if the user has RACF SPECIAL authority, and no seclabel currently exists on the resource. Once a seclabel is set, it cannot be changed.

Note: zFS file systems support the chlabel utility, which allows the setting of an initial security label on a file or directory. Use this utility to set security labels on allocated zFS files and directories *after* they have been created.

6.64 Understanding UMASK

- Set the default file creation mask: umask 022
  - Default mask is: 022 000 010 010
- New file created with permission bits: 777

	rwx	rwx	rwx
File	111	111	111
UMASK	000	010	010
	111	101	101

- File permission bits (with UMASK): 755

Figure 6-64 Understanding the UMASK

Defining the UMASK

When a file is created, it is assigned initial access permissions. If you want to control the permissions that a program can set when it creates a file or directory, you can set a file mode creation mask using the **umask** command.

The user can set this file mode creation mask for one shell session by entering the **umask** command interactively, or you can make the **umask** command part of your login. When you set the mask, you are setting a limit on allowable permissions: You are implicitly specifying which permissions are not to be set, even though the calling program may allow those permissions. When a file or directory is created, the permissions set by the program are adjusted by the umask value: the final permissions set a the program's permissions minus what the umask values restrict.

To use the **umask** command for a single session, enter:

```
umask mode
```

To create a mask that sets read-write-execute permission *on* for the owner of the file and *off* for everyone else, enter:

```
umask 077
```

6.65 Displaying the UMASK

- Symbolic and Octal UMASK notation

```
ROGERS @ SC47:/etc>umask
0022
ROGERS @ SC47:/etc>umask -S
u=rwx,g=rx,o=rx
ROGERS @ SC47:/etc>umask u=rwx,go=r
ROGERS @ SC47:/etc>umask
0033
ROGERS @ SC47:/etc>
 ===>
```

Figure 6-65 Displaying the UMASK

Displaying the UMASK

The umask sets the file-creation permission-code mask of the invoking process to the given mode.

The mode may be specified in symbolic (rwx) or octal format. The symbolic form specifies what permissions are allowed. The octal form specifies what permissions are disallowed.

The file-creation permission-code mask (often called the umask) modifies the default (initial) permissions for any file created by the process. The umask specifies the permissions which are not to be allowed.

If the bit is turned off in the umask, a process can set it on when it creates a file. If you specify:

```
umask a=rx
```

you have allowed files to be created with read and execute access for all users. If you were to look at the mask, it would be 222. The write bit is set, because write is not allowed. If you want to permit created files to have read, write, and execute access, then set umask to 000. If you call **umask** without a mode argument, umask displays the current umask.

6.66 Default permissions and UMASK

==> **umask 022** Changes defaults for a user

Command	Default Permission	Final settings after umask
mkdir	rwx rwx rwx	rwx r-x r-x
MKDIR	rwx r-x r-x	rwx r-x r-x
JCL, no PATHOPTS	--- --- ---	--- --- ---
OEDIT	rwx --- ---	rwx --- ---
vi editor	rw- rw- rw-	rw- r-- r--
ed editor	rw- rw- rw-	rw- r-- r--
Redirection (>)	rw- rw- rw-	rw- r-- r--
cp	output = input	output = input
OCOPY	--- --- ---	--- --- ---
OPUT/OPUTX	rw- --- --- (text) rwx --- --- (binary)	rw- --- --- (text) rwx --- --- (binary)

Figure 6-66 Default permissions and the UMASK

Default permissions

The system assigns default permission bits for files and directories at creation time. The settings depend on the type of command or facility that is used, and in some cases, on the type of file that is created.

A user can change the default setting when a file is created by using the **umask** shell command. The values set by the **umask** command will last for the length of the user's session, or the command can be part of the user's login so that the user always has the same default permissions.

PATHDISP indicates how MVS should handle the file when the job step ends normally or abnormally. This performs the same function as the DISP parameter for a data set.

umask

In /etc/profile, umask sets the default file creation mask. Using a umask of 022 causes a file created with mode 777 to have permissions of 755. The creator cannot set the group write or other write bits on in the file mode field, because the mask sets them off.

Changing the default permission settings

The umask service changes the process's file creation mask. This mask controls file permission bits that are set whenever the process creates a file. File permission bits that are turned on in the file creation mask are turned off in the file permission bits of files that are created by the process. For example, if a call to the open service, BPX1OPN, specifies a “mode” argument with file permission bits, the process's file creation mask affects that argument. Bits that are on in the mask are turned off in the mode argument, and therefore in the mode of the created file.

If you use the MKDIR TSO/E command to create a directory, the permission bits will be different than if you use the shell command **mkdir** to do the same thing.

Each user can influence these defaults for his or her shell session by using the **umask** command. **umask** sets the file-creation permission-code mask of the invoking process to the given mode. You can specify the mode in any of the formats recognized by **chmod**.

As stated previously, the file-creation permission-code mask determines the default permissions for any file created by the process. For example, a file created by the **vi** command has the permissions specified by the umask unless the **vi** command specifies explicit permissions itself.

Redirecting command output to a file

Commands entered at the command line typically use the three standard files (STDIN, STDOUT, and STDERR), but you can redirect the output for a command to a file you name. If you redirect output to a file that does not already exist, the system creates the file automatically.

6.67 Example of creating a new file

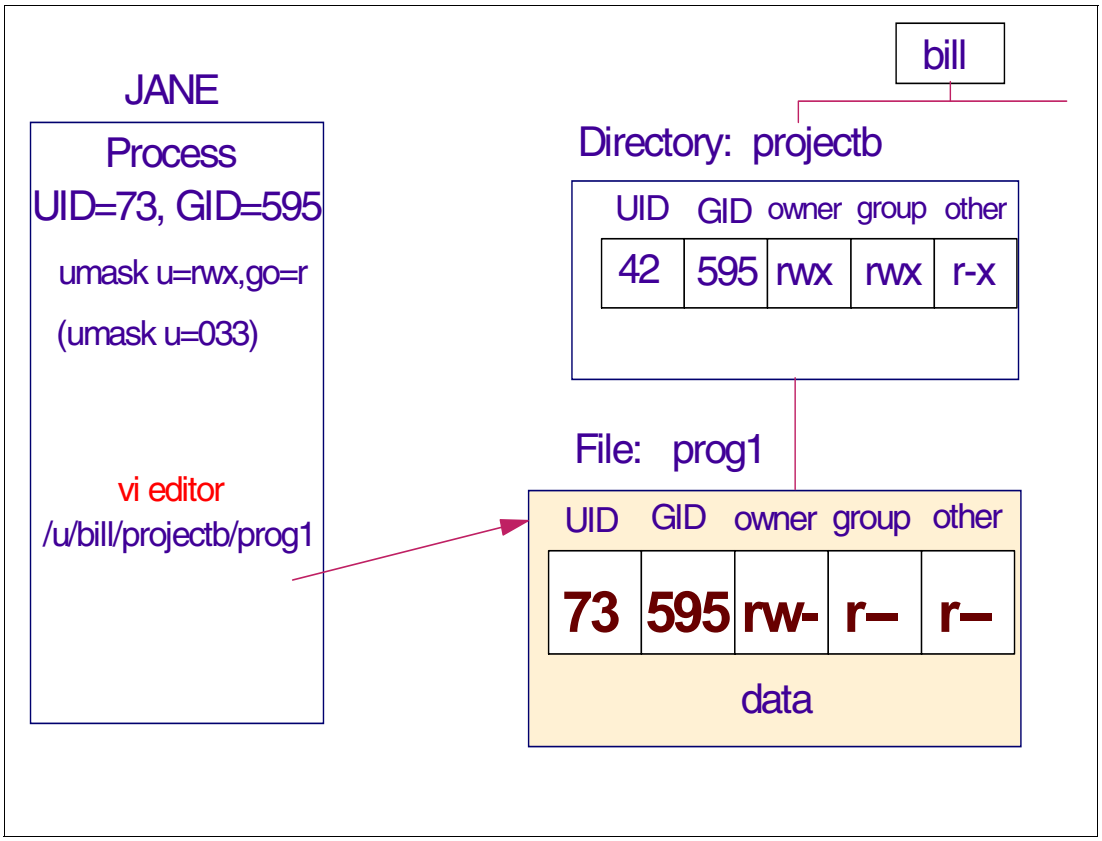


Figure 6-67 Example of creating a new file

Creating a new file

In this example the user JANE (UID 73) is creating a new file. A file could be created in various ways. It could be copied from another file, it could be created with an editor, it could be moved, it could be created using JCL, and so on.

Security for a file is specified in the file security packet (FSP) which is a part of the attributes of the file. Each file has an FSP. The FSP is created when the file is created and is deleted when the file is deleted.

The contents of the FSP are determined as follows:

- The owning UID of the new file is taken from the effective UID (EUID) of the process that creates the file.
- The owning GID of the new file is taken from the owning GID of the directory for the new file.
- The file permission bits are set by the process that creates the file. The setting of the permission bits can be different, depending on the method used to create the file.

Effect of the UMASK

In this example, the user has set a umask to specify what the permissions should be for all files that the user creates. To do this, the user JANE issued the following shell command:

```
umask u=rwx,go=r
```

This will give the owner of the file `r`, `w`, and `x` access, while group and others will get `r` access.

Settings for the new file

As illustrated in Figure 6-67, when the new file is created:

- The UID of the file is set to 73.
- The GID is set to 595.
- The file permission bits are set according to the umask as follows:
  - The file owner has `rw`, which mean read and write access.
  - Users in the group with a GID of 595 have `r--`, which means read access.
  - All other users (also known as world) have `r--`, which means read access.

In order to create a new file, the user must have write access to the directory for the new file.

6.68 Can user JOE access the file

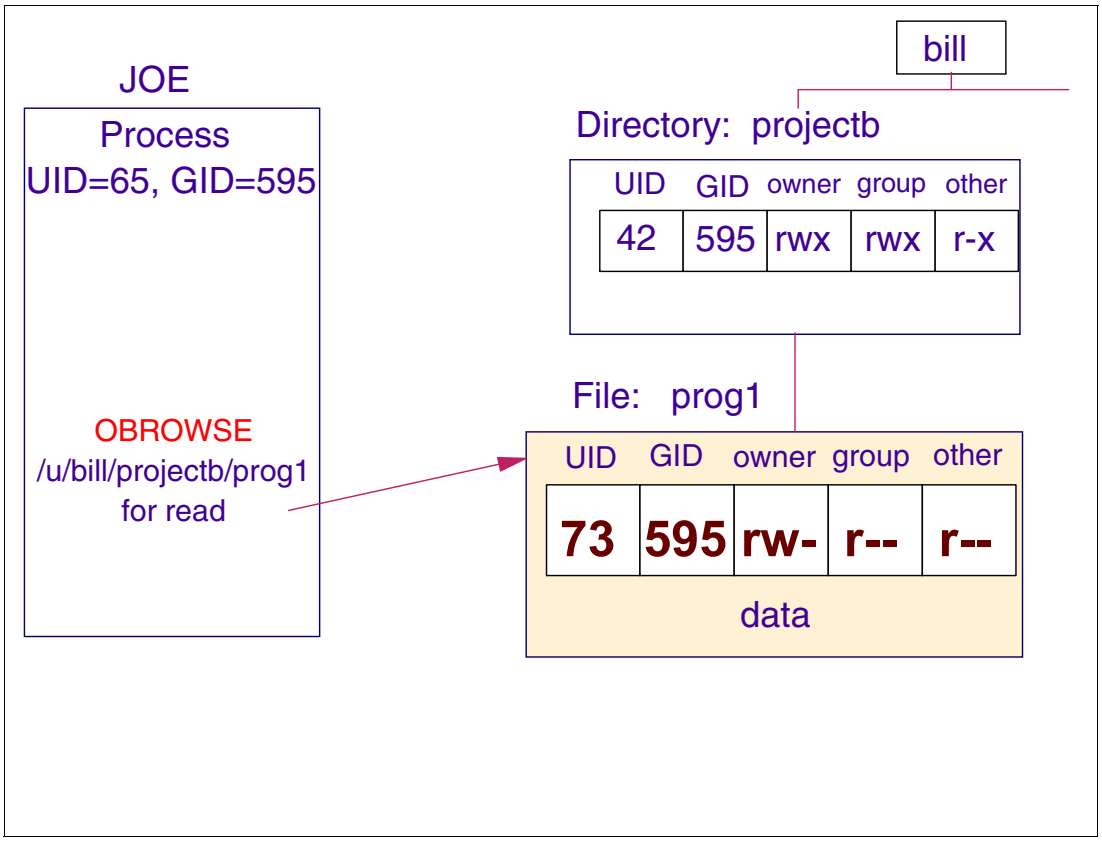


Figure 6-68 Can a user access the newly created file

User access to the new file

In this example, another user who has a UID of 65 wants read access to the file /u/bill/projectb/prog1.

This user is not a superuser and is not a privileged or trusted started procedure.

The UID of the process, 65, does not match the file UID.

The GID of the process does match the file GID, and therefore the group permissions apply. The user wants read access and the permissions are `r--`, so the user is allowed access.

If the effective GID of the process did not match the file GID, RACF would have looked at the user's list of groups (assuming that RACF list of groups is active) for a group that has a GID that is the same as the file GID.

In order to access a file, the user must also have read and search permission to all directories in the path.

6.69 Can user ANN copy the file

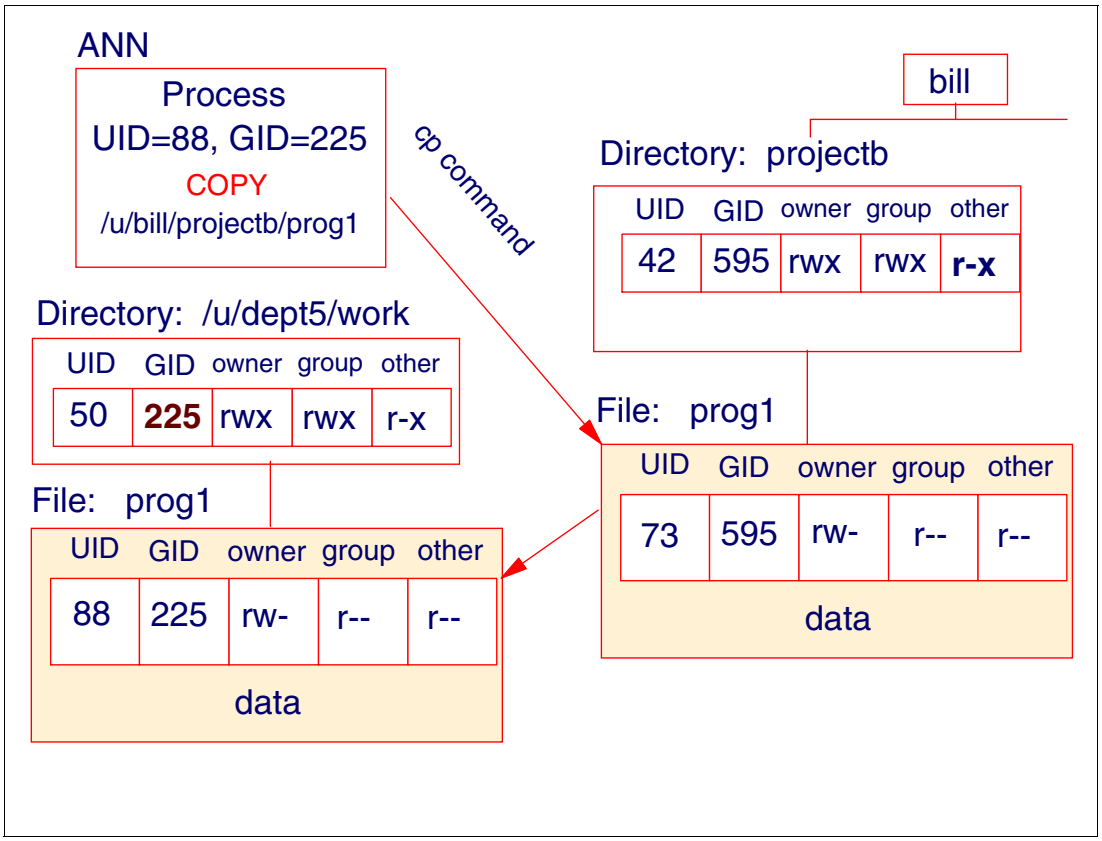


Figure 6-69 Can a user copy the newly created file

Access to copy the new file

In this example, a user called ANN with a UID=88 wants to copy the file that user JANE created. The user is not a superuser, and is not a privileged or trusted started proc. This user is not the owner of the file and none of this user's groups have a GID that match the GID of the file.

In this case, access will be determined by the permissions for other users. Since the permission for other users is read access (`r--`), the user is allowed to copy the file.

User ANN does not have a umask set and the file permissions will be determined by the defaults for the **cp** command which was the method used for the copy. The new file will get the following attributes:

- The UID will be set to the UID of the person who copied the file. This will be 88.
- The GID will be GID of the directory where the file will be placed, which is 225.
- The file permission bits will be copied from the file *prog1* since this is the default for the **cp** command.

6.70 Setting file permissions

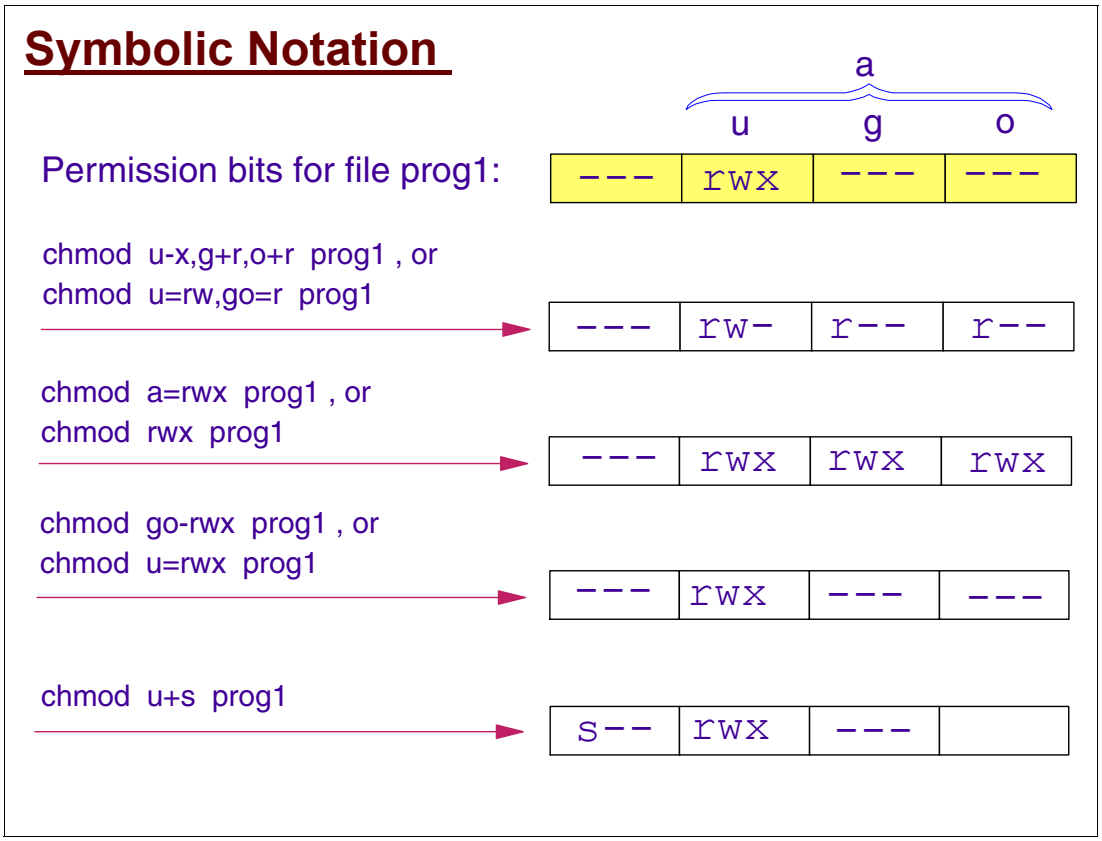
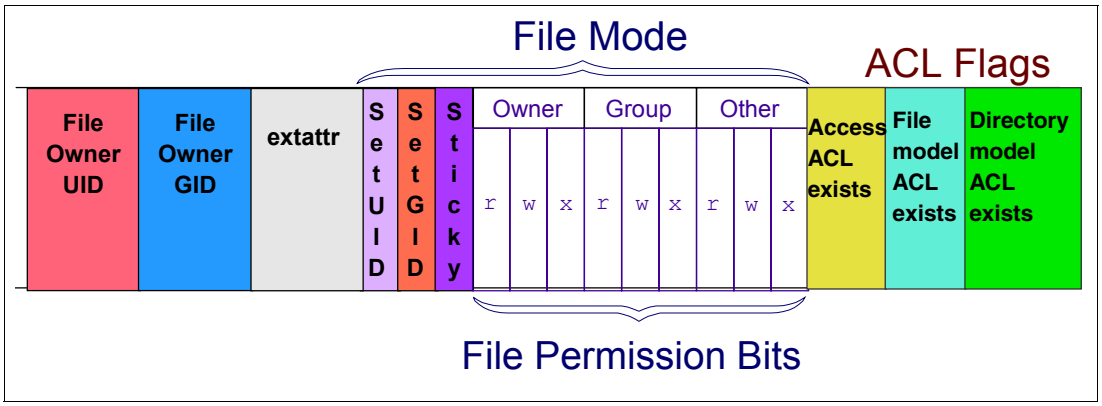


Figure 6-70 Commands to set permission bits

Commands to change permission bits

A file's mode mask includes the file permission bits, the SetUID bit, the SetGID bit, and the sticky bit. The file security packet (FSP) has the setuid, setgid, and sticky bits.



The chmod command

The initial permission bit for the file prog1 is shown in the box at the top right. The **chmod** command is used to make a change to the file mode mask of a file or directory, as follows:

- The z/OS UNIX shell command **chmod u-x,g+r,o+r** deletes execute (x) from the owner (u for user) permissions, adds read (r) to the group (g) permissions, and adds read (r) to the other (o) permissions.
- The same effect can be achieved with **chmod u=rw,go=r** which sets the owner (u) mask to read/write (rw), and sets the group and other (go) mask to read (r). When the equal sign is used, it turns on the bits specified and turns off all other bits.
- The command **chmod a=rwx** sets on the read, write, and execute bits for all (a) users, which includes the owner, group, and other.
- An equivalent command is **chmod rwx** in which the a (all users) is implied.
- In the command **chmod go-rwx**, rwx is turned off for group and other.
- An alternative form **chmod u=rwx** sets rwx on for the owner (u) mask, and turns off all other bits.
- The command **chmod u+s** shows how to turn on the SetUID bit. The **s** stands for set, and the **u** stands for UID. If we wanted to turn on the SetGID bit, we would use **chmod g+s**. Turning on the sticky bit would be achieved using **chmod +t**.

We cannot use RACF commands or panels to set the permissions. We use the z/OS UNIX shell commands. This is the same as AIX and UNIX. An alternative to the **chmod** command is to use the ISHELL menus. They may be more user-friendly for people who are not familiar with UNIX, and they provide help information.

Note: **chmod** can only be used by the file owner or a superuser.

6.71 Setting file permissions

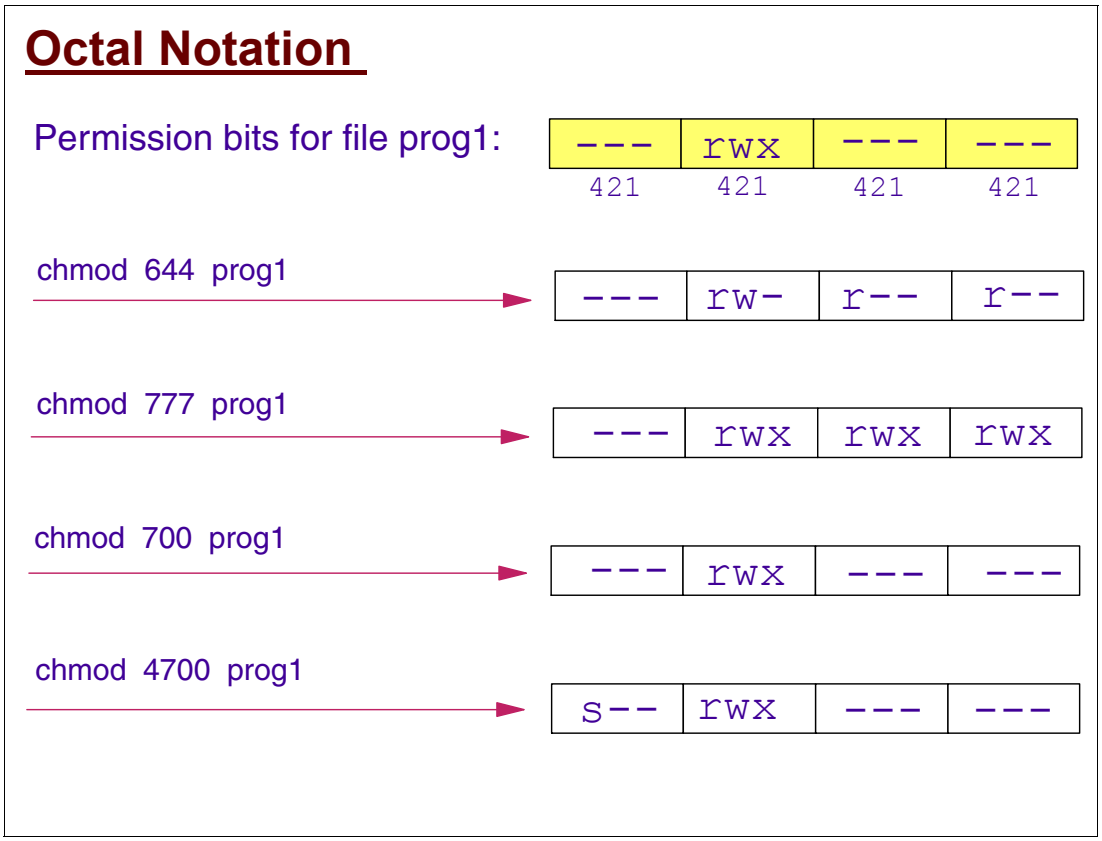


Figure 6-71 Commands to set the permission bits

Command to sets bits using octal notation

Octal notation can be used on the **chmod** command instead of the symbolic notation. With octal notation, each set of three bits is represented in a single octal digit. For example, a permission of rwx would be represented as the octal digit 7 which is the sum of the 4 for read (r), the 2 for write (w), and 1 for execute/search (x), as follows:

- In the command **chmod 644** the octal 6 sets read and write (4+2) for the file owner, and sets read (4) for group and other users.
- The command **chmod 777** sets on read/write/execute (4+2+1) for the owner, group, and other users.
- The command **chmod 700** sets on the read, write, and execute bits (4+2+1) for the owner, and gives no access to group and other users.
- In the last command **chmod 4700** we see how to set the set UID, set GID, and sticky bits. This is done by using four octal digits, where the first digit represents the set UID, set GID, and sticky bits. Here, SetUID is the left-most bit (4), SetGID is the middle bit (2), and the sticky bit is the right-most bit (1).

Note: To specify permissions for a file or directory, you use at least a three digit octal number, omitting the digit in the first position. When you specify three digits instead of four, the first digit describes owner permissions, the second digit describes group permissions, and the third digit describes permissions for all others.

6.72 List file and directory information

```
ROGERS @ SC65:/u/rogers>ls -l /usr/man/C
total 136
drwxr-xr-x   2 HAIMO    OMVSGRP     8192 May 29  2002 IBM
drwxr-xr-x   3 HAIMO    OMVSGRP     8192 Jun 10  2002 cat1
drwxrwxr-x   3 HAIMO    OMVSGRP     8192 May 29  2002 cat8
drwxr-xr-x   3 HAIMO    OMVSGRP     8192 May 29  2002 man1
-rw-rw-r--   2 HAIMO    OMVSGRP    33887 May 29  2002 whatis
```

File type
File permissions
Links
Owner userid
Owner groupid
File size
Date and time
Name

Figure 6-72 Listing the file and directory permission settings

Command to list permission bit settings

The `ls` command can be used to list important information about files and directories. If the `ls` command is used for a directory, it will display this information for all the files in the directory as shown in Figure 6-72.

Although there are 25 different options for the `ls` command, we only show the output for one option, `-l`, which stands for long. The file type in position 1 in the output indicates the type of file that is displayed on this line:

- **-** Regular file
- **d** Directory
- **b** Block special file (not supported for z/OS UNIX)
- **c** Character special file (for example, device)
- **l** Symbolic link
- **e** External link
- **p** FIFO special file (also known as a named pipe)
- **s** Socket file type

Positions 2 through 10 represent the file permissions for the owner, group, and other users.

6.73 Introducing daemons

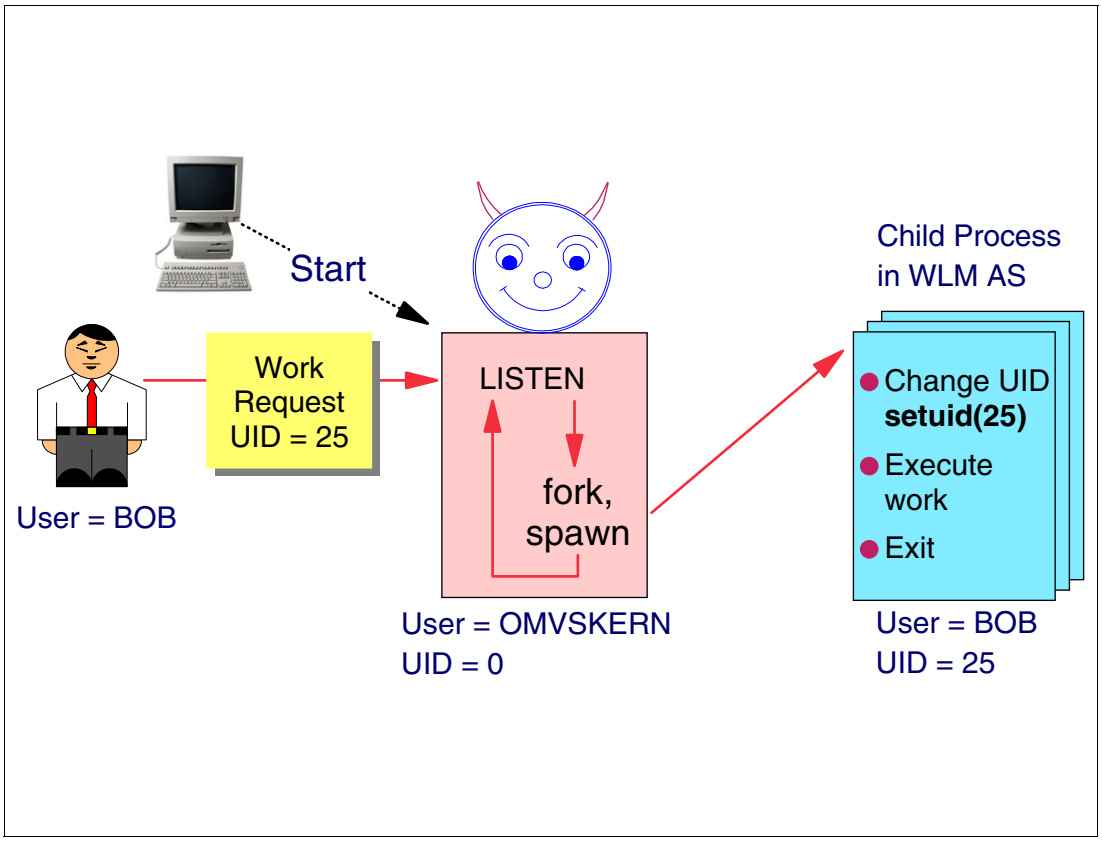


Figure 6-73 Introducing z/OS UNIX daemons

Introducing z/OS UNIX daemons

A daemon process is a process that runs in the background environment and is not associated with any particular terminal or user. Daemons run authorized (superuser authority) and can issue authorized functions like the following to change the identity of a user's process:

- setuid()
- seteuid()
- setreuid
- pthread_security_np()
- auth_check_resource_np()
- _login()
- _spawn() with user ID change
- _password()

Note: The spawn() service is allowed to create a new image under a specific user ID that is different from that of the invoker. When an invoker with appropriate privileges specifies a username on the _BPX_USERID environment variable or in the inheritance structure (INHEUSERID), the resulting image runs under the associated MVS user identity.

Daemon authority

Daemon authority is required only when a program does a setuid(), seteuid(), setreuid(), or spawn() user ID to change the current UID without first having issued a __passwd() call to the

target user ID. In order to change the MVS identity without knowing the target user ID's password, the caller of these services must be a superuser. Additionally, if a BPX.DAEMON FACILITY class profile is defined and the FACILITY class is active, the caller must be permitted to use this profile. If a program comes from a controlled library and knows the target UID's password, it can change the UID without having daemon authority.

You can run daemons with regular UNIX security or with z/OS UNIX security. A z/OS UNIX daemon could also be described as a classical server process. Initially, the daemon process is started by an external command or event. Once started, the daemon *listens* for work requests from clients. When a request is received, the daemon will note the UID of the requester, and then fork a child process to carry out the request. Before executing the request, the daemon uses a special SYSCALL **setuid** to reset the security environment to match that of the requester.

For administrators, controlling daemons requires some extra considerations:

- How and when is a daemon process started (or restarted if it fails)?
- Daemons often need initialization options customized to installation requirements.
- Daemons have the ability to issue **setuid()**. Access to this type of power needs to be controlled, by controlling which programs can be daemons.
- A special user security profile BPXROOT must be created in order to support some daemon operations.

Note: The important point about the **setuid** instruction is that, in a z/OS environment, it will reset the whole security profile of the forked address space. The UID will be set to the requester's UID and the current RACF user ID information (the ACEE) will be changed to BOB to complement the UID. The requester's task therefore runs with access to both the UNIX and z/OS resources (data sets) owned by BOB.

Daemon authority for the kernel

Give daemon authority to the kernel. Most daemons that inherit their identities from the kernel address space are started from /etc/rc. To authorize the OMVSKERN user ID for the daemon FACILITY class profile, issue:

```
PERMIT BPX.DAEMON CLASS(FACILITY) ID(OMVSKERN) ACCESS(READ)
SETROPTS RACLIST(FACILITY) REFRESH
```

Add user BPXROOT

In order for daemon processes to be able to invoke **setuid()** for superusers, define a superuser with a user ID of BPXROOT on all systems. To define the BPXROOT user ID, issue:

```
ADDUSER BPXROOT DFLTGRP(OMVSGRP) OMVS(UID(0) HOME('/')
                  PROGRAM('/bin/sh')  PASSWORD(A4B3C3D1)
```

Exactly how requests are passed to a daemon depends on the type of daemon. For TCP/IP-based daemons like the telnet or rlogin daemons, the user request will be passed over a socket connection from the IP network. For the cron daemon, the request is passed via a parameter area and a cross memory post to the daemon.

6.74 z/OS UNIX daemons

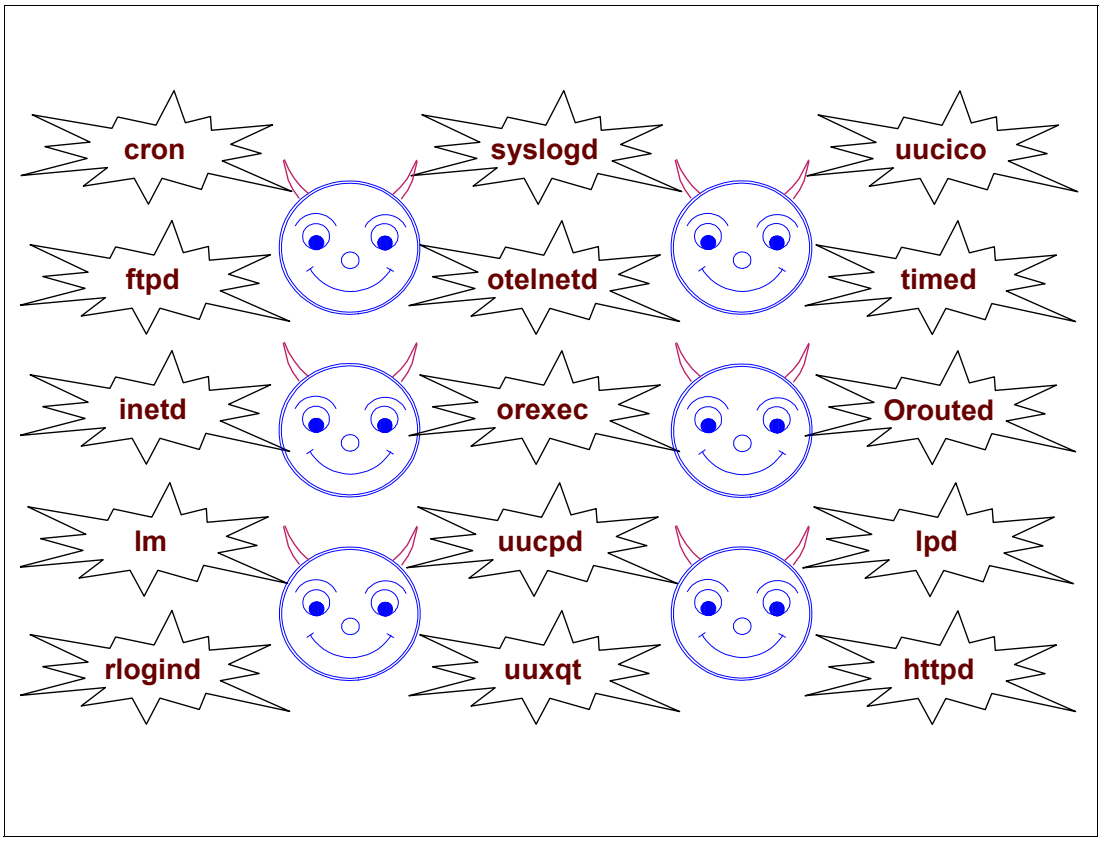


Figure 6-74 The z/OS UNIX daemons

z/OS UNIX daemons

A daemon is a long-lived process that runs unattended to perform continuous or periodic system-wide functions, such as network control. Some daemons are triggered automatically to perform their task; others operate periodically. Daemons typically encountered on z/OS UNIX systems include:

cron The batch scheduler, cron is a clock daemon that runs commands at specified dates and times. You can specify regularly scheduled commands by using the **`crontab`** command. Jobs that are to be run only once can be submitted using the **`at`** or **`batch`** commands. cron runs commands with priorities and limits set by a queuedefs file.

ftpd The file transfer daemon supplied with z/OS Communications Server.

inetd The Internet daemon. The inetd daemon provides service management for a network. It starts the rlogind program or otelnetd program whenever there is either a remote login request or a remote telnet login from a workstation.

lm The CS login monitor. The login monitor is a daemon that starts the login process in the shell for logins initiated by Communications Server (CS). CS nodes then forward service requests to the login monitor, which listens for the requests on a port number assigned to the lm service.

rlogind The remote login daemon. The rlogind program is the server for the remote login command **`rlogin`**. It validates the remote login request and verifies the password of the target user. It starts a z/OS UNIX shell for the user and handles

translation between ASCII and EBCDIC code pages as data flows between the workstation and the shell.

syslogd The syslog daemon supplied with z/OS Communications Server. Syslog daemon (syslogd) is a server process that has to be started as one of the first processes in your z/OS UNIX environment. Other servers and stack components use syslogd for logging purposes and can also send trace information to syslogd.

otelnetd The remote logon daemon supplied with z/OS Communications Server.

orexecd The remote execution protocol daemon supplied with z/OS Communications Server. The Remote Execution Protocol Daemon (REXECD) is the server for the REXEC routine. REXECD provides remote execution facilities with authentication based on user names and passwords.

uucpd The UNIX-to-UNIX copy program daemon. The uucpd daemon is used to communicate with any UNIX system that is running a version of the UNIX-to-UNIX copy program. UUCP functions are used to automatically transfer files and requests for command execution from one UUCP system to another, usually in batch mode at particular times. Other daemons associated with uucp include:

- **uucico** Processes uucp and uux file transfer requests.
- **uuxqt** Runs commands from other systems.

Orouted The Orouted daemon is supplied with z/OS Communications Server. The route daemon is a server that implements the Routing Information Protocol (RIP) (RFC 1058). It provides an alternative to the static TCP/IP gateway definitions.

lpd The line printer daemon supplied with z/OS Communications Server. The line printer daemon enables printers from any TCP/IP host that is attached to the MVS spooling system.

timed The time daemon supplied with z/OS Communications Server. The time daemon provides clients with UTC time. Network stations without a time chip obtain clocks from this daemon.

httpd The http daemon supplied with WebSphere.

6.75 UNIX-level security for daemons

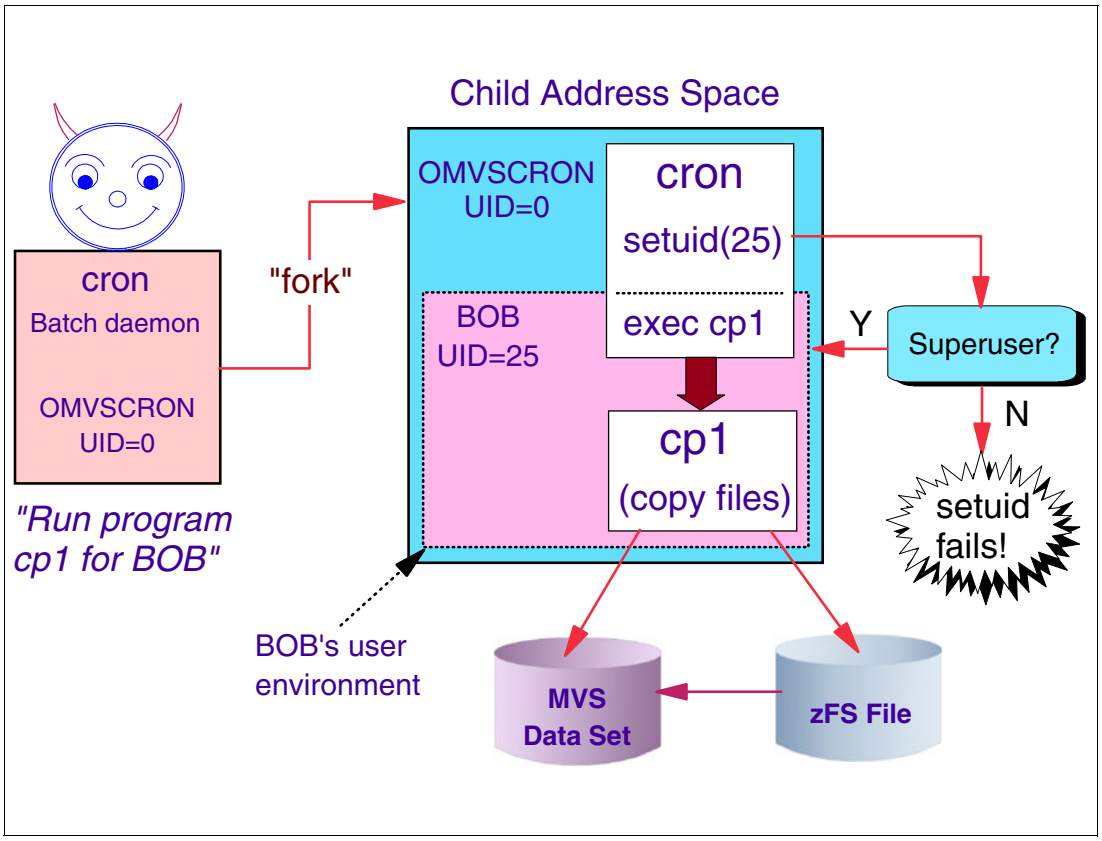


Figure 6-75 UNIX-level security for daemons

UNIX-level security for daemons

You can run daemons with UNIX-level security or with z/OS UNIX security. If the BPX.DAEMON (or BPX.SERVER) facility class is not defined, your system has UNIX-level security. In this case, the system is less secure. This level of security is for installations where superuser authority has been granted to system programmers. These individuals already have permission to access critical data sets such as PARMLIB, PROCLIB, and LINKLIB. These system programmers have total authority over a system.

Daemons are processes that perform services for other users. In order to do this, a daemon must be able to change its identity temporarily to the identity of the user it will perform work for. The functions **`setuid()`** and **`seteuid()`** are authorized functions which will change the identity of a z/OS UNIX user or program.

The daemon program clones itself using a **`fork()`** SYSCALL. The cloned child copy issues the **`setuid()`** request to initialize the security environment for the requester. The child issues **`exec()`** to pass control to the real request program to be executed. Superuser authority is required to use these functions which requires authority of UID=0. All daemon programs must execute as superusers, and all superusers are allowed to use the **`setuid()`** and **`seteuid()`** functions to change the identity of a process to any other UID. Their z/OS identity will be changed to the one corresponding to the UID value; in the example, the cron daemon changes its identity to UID=25, which is the z/OS user ID BOB.

Note: If the target UID=0, and the target user ID is not known, the kernel sets default user ID=BPXROOT.

6.76 z/OS UNIX security: BPX.DAEMON

- Provides more control over superusers
- Provides a higher level of security
- Daemon's identity must be permitted to FACILITY class
  - RDEFINE FACILITY BPX.DAEMON UACC(NONE)
  - PERMIT BPX.DAEMON CLASS(FACILITY) ID(userid) ACC(READ)
- All programs running in the address space
  - Must be loaded from a library controlled by a security product

Figure 6-76 z/OS UNIX security with daemons

BPX.DAEMON profiles

If the DAEMON (or BPX.SERVER) FACILITY class is defined, your system has z/OS UNIX security. Your system can exercise more control over your superusers.

This level of security is for customers with very strict security requirements who need to have some superusers maintaining the file system but want to have greater control over the z/OS resources that these users can access. Although BPX.DAEMON provides some additional control over the capabilities of a superuser, a superuser should still be regarded as a privileged user because of the full range of privileges the superuser is granted.

The additional control that BPX.DAEMON provides involves the use of kernel services such as **`setuid()`** that change a caller's z/OS user identity. With BPX.DAEMON defined, a superuser process can successfully run these services if the following are true:

- The caller's user identity has been permitted to the BPX.DAEMON FACILITY class profile.
- All programs running in the address space have been loaded from a library controlled by a security product. A library identified to RACF Program Control is an example. Individual files in the HFS can be identified as controlled programs.

Kernel services that change a caller's z/OS user identity require the target z/OS user identity to have an OMVS segment defined. If you want to maintain this extra level of control at your installation, you will have to choose which daemons to permit to the BPX.DAEMON FACILITY class. You will also have to choose the users to whom you give the OMVS security profile segments.

6.77 RACF program control

- Any program loaded into an A/S with daemon authority
  - Must be a controlled program
- Define programs to Program Control
- z/OS daemons reside in the HFS and are controlled
- User defined daemons
  - Specific daemon program names or *

```
RDEFINE  PROGRAM   *  UACC(READ)  ADDMEM +
            ('SYS1.LINKLIB'/'******'/NOPADCHK +
            'CEE.SCEERUN'/RLTPAK/NOPADCHK +
            'SYS1.SEZALOAD'//NOPADCHK )
SETROPTS  WHEN(PROGRAM)  REFRESH
```

Figure 6-77 Defining RACF program control

RACF program control for daemons

The purpose of protecting load modules is to provide installations with the ability to control who can execute what programs and to treat those programs as assets.

You protect individual load modules (programs) by creating a profile for the program in the PROGRAM general resource class. A program protected profile in the PROGRAM class is called a controlled program.

The name of the profile can be complete, in which case the profile protects only one program, or the name of the profile can end with an asterisk (*), in which case the profile can protect more than one program. For example, a profile named ABC* protects all programs that begin with ABC, unless a more specific profile exists. In this way you can find which programs are causing the environment (such as PADS checking) to not work.

The profile for a controlled program must also include the name of the program library that contains the program and the volume serial number of the volume containing the program library. The profile can also contain a standard access list of users and groups and their associated access authorities.

Program access to data sets

Program access to data sets (PADS) allows an authorized user or group of users to access specified data sets with the user's authority to execute a certain program. That is, some users can access specified data sets at a specified access level only while executing a certain program (and the program access is restricted to controlled programs).

To set up program access to data sets, create a conditional access list for the data set profile protecting the data sets. To do this, specify `WHEN(PROGRAM(program-name))` with the `ID` and `ACCESS` operands on the `PERMIT` command. Specifying the `WHEN(PROGRAM)` operand requires that the user or group specified must be running the specified program to receive the specified access.

PADCHK and NOPADCHK operands

Choosing between the `PADCHK` and `NOPADCHK` operands: With the `ADDMEM` operand of the **`RDEFINE`** and **`RALTER`** commands, you can also specify `PADCHK` or `NOPADCHK` as follows:

NOPADCHK `NOPADCHK` means that RACF does not perform the program-accessed data checks for the program. The program is loaded and has access to any currently opened program-accessed data sets, even though the user ID/program combination is not in the conditional access list. `NOPADCHK` allows an installation to define entire libraries of modules (such as the PL/I transient routines or ISPF) as controlled programs without having to give each of these modules explicit access to many program-accessed data sets. Use `NOPADCHK` if you trust the programs to access only data they should.

PADCHK `PADCHK` (the default) means that RACF checks for program-accessed data sets that are already open before executing the program. If there are any open program-accessed data sets, RACF ensures, before it allows this program to be loaded, that this user ID/program combination is in the conditional access list of each data set.

6.78 z/OS UNIX-level security for daemons

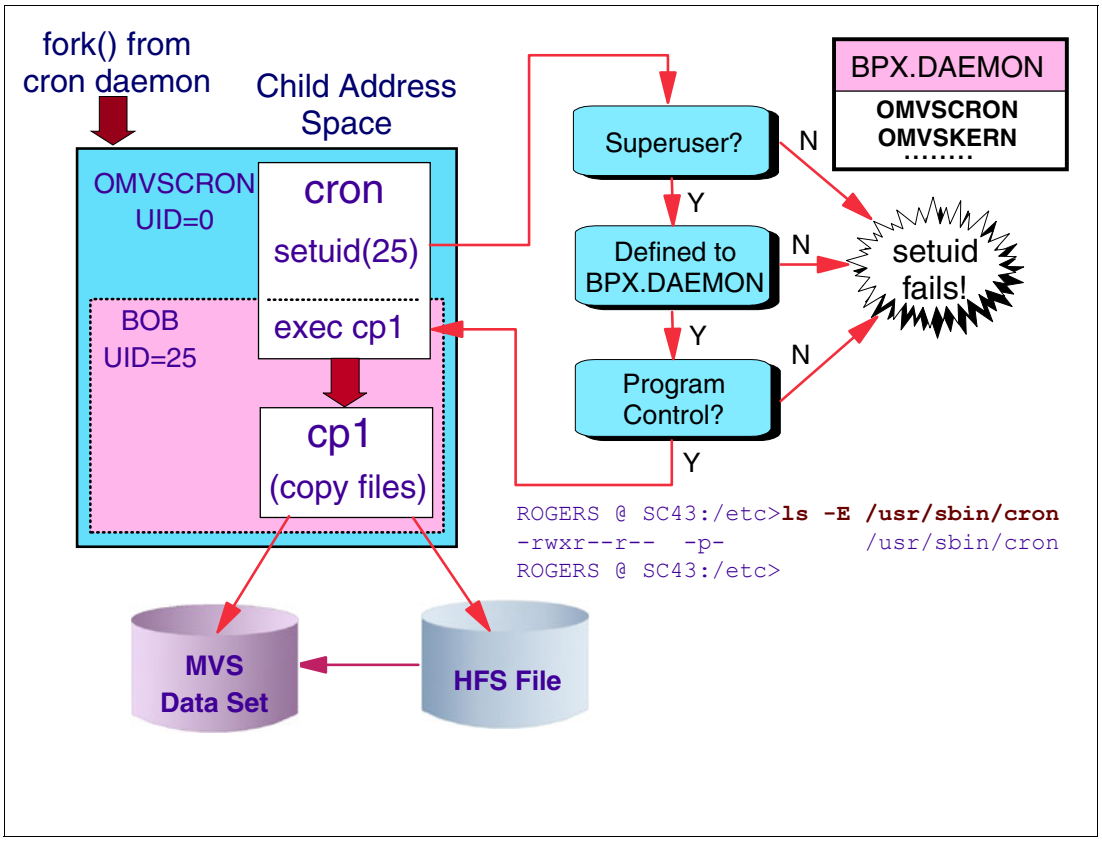


Figure 6-78 z/OS UNIX security with daemons

Example using the cron daemon

cron is a clock daemon that runs commands at specified dates and times.

Figure 6-78 shows the cron daemon running a shell script for the user ID BOB (UID=25). The script will copy HFS files to an MVS data set. Before cron can run the script, it will fork a new process and set the identity of this process to UID=25 and the MVS identity to BOB. This will ensure that the script can be run successfully with BOB's shell environment and BOB's access to his MVS data sets. When the job is done, the cron child process will end, and cron will not have any access to BOB's MVS data sets.

To obtain a higher level of security in a z/OS system with daemons, the RACF FACILITY class called BPX.DAEMON can be used to control the use of the **`setuid/seteuid`** functions. Only the superusers permitted to the BPX.DAEMON class will be allowed to use the **`setuid/seteuid`** functions. In addition, there is a program control requirement.

When **`setuid()`** SYSCALL is issued, the caller (daemon) program, and all other programs currently loaded in the address space, must have been loaded from a z/OS data set with the RACF Program Control activated, meaning they must be controlled programs. As it is the cloned child daemon program that issues the request, it will inherit the contents of its address space from the parent daemon via fork.

This solution enables an installation to have some superusers which have authority to perform system maintenance (for example, to manage the hierarchical file system), while other special superusers (daemon user IDs) are allowed to change the identity of a process.

6.79 Start options for daemons

/etc/rc
or
any shell script

```
_BPX_JOBNAME='INETD' /usr/sbin/inetd /etc/inetd.conf &
```

Can use INITTAB

MVS
Started
Task

```
//INETD    PROC
//INETD    EXEC  PGM=BPXBATCH,REGION=30M,TIME=NOLIMIT,
//             PARM='PGM /usr/sbin/inetd  /etc/inetd.conf'
//* STDIN and  STDOUT are both defaulted to /dev/null
//STDERR  DD  PATH='/etc/log',PATHOPTS=(OWRONLY,OCREAT,OAPPEND),
//             PATHMODE=(SIRUSR,SIWUSR,SIRGRP,SIWGRP)
```

Figure 6-79 Start options for daemons

Start options for daemons

In a z/OS system, there are several ways of starting and restarting daemons. The method used depends on the level of control the installation has chosen for daemons. The daemon programs are installed in /usr/sbin. Daemons can be started using the following methods:

- To be started automatically when kernel is started, place the start options in the HFS file called /etc/rc. The initialization of z/OS UNIX includes running the commands in /etc/rc. The _BPX_JOBNAME environment variable assigns a job name to the daemon.
- As a cataloged procedure using the BPXBATCH program to invoke the daemon program.

If daemons need to be stopped, the `kill` command is typically used. Some daemons may have their own specific method of shutdown.

You should have appropriate procedures and directions in place to restart these daemons in case of failure. Started procedures are one way to do this; other methods may be more attractive depending on your automation strategy.

6.80 Define daemon security

1. Define daemon user ID as superuser

```
ADDUSER  OMVSCRON  DFLTGRP(OMVSGRP) OMVS(UID(0) HOME('/') PROGRAM('/bin/sh'))
```

2. Define BPX.DAEMON in the facility class

```
RDEFINE  FACILITY BPX.DAEMON  UACC(NONE)
```

3. Permit daemon user ID to BPX.DAEMON class

```
PERMIT  BPX.DAEMON  CLASS(FACILITY)  ID(OMVSCRON)  ACCESS(READ)
```

4. Protect program libraries

```
RDEFINE  PROGRAM  *  ADDMEM('SYS1.LINKLIB'//NOPADCHK)  UACC(READ)
```

5. Activate RACF program control

```
SETROPTS WHEN(PROGRAM)  REFRESH
SETROPTS FACILITY(RACLIST)  REFRESH
```

Figure 6-80 Defining daemon security

Defining daemon security

The following steps describe how to define security for a daemon:

1. Define a user ID for the daemon which is a superuser with UID=0, for example OMVSCRON:

   ```
   ADDUSER OMVSCRON DFLTGRP(OMVSGRP) OMVS(UID(0) HOME('/') PROGRAM('/bin/sh'))
   ```

2. Define the BPX.DAEMON FACILITY class in RACF:

   ```
   RDEFINE FACILITY BPX.DAEMON UACC(NONE)
   ```

   Activate the class if this is the first RACF FACILITY class:

   ```
   SETROPTS CLASSACT(FACILITY) GENERIC(FACILITY) AUDIT(FACILITY)
   SETROPTS RACLIST(FACILITY)
   ```

3. Permit the daemon user ID to the BPX.DAEMON class:

   ```
   PERMIT BPX.DAEMON CLASS(FACILITY) ID(OMVSCRON) ACCESS(READ)
   ```

4. Protect the program libraries that need to be protected from unauthorized updates. The ADDSD command creates data set profiles for the data sets. You should protect against unauthorized updates so that nobody can replace a daemon program with a fake daemon program. If these profiles are already defined, this step can be skipped.

   ```
   ADDSD 'SYS1.LINKLIB' UACC(READ)
   ADDSD 'SYS1.SCEERUN' UACC(READ)
   ADDSD 'SYS1.SEZALOAD' UACC(READ)
   ```

Mark the data sets as controlled libraries:

```
RDEFINE PROGRAM * ADDMEM('SYS1.LINKLIB'//NOPADCHK +
                         'SYS1.SCEERUN'//NOPADCHK +
                         'SYS1.SEZALOAD'//NOPADCHK +
```

Or, mark the daemon program as controlled instead of the whole library:

```
RDEFINE PROGRAM CRON ADDMEM('SYS1.LINKLIB'//NOPADCHK)
UACC(READ) AUDIT(ALL)
```

5. Activate RACF program control:

   Place the PROGRAM profile in storage:

```
SETROPTS WHEN(PROGRAM) REFRESH
```

Starting daemon programs

In many cases, a daemon program is started from the kernel and will inherit the kernel user ID, OMVSKERN. This example shows that it can have a separate user ID as long as the user ID is defined as a superuser. This superuser must be defined with a UID=0 in RACF, which means that this user cannot become a superuser by using the **su** command.

> **Note:** This user ID should not have a TSO/E segment defined; only the OMVS segment is needed.

The name BPX.DAEMON must be used. No substitutions for this name are allowed. UACC(NONE) is recommended. If this is the first RACF FACILITY class defined in RACF, the SETROPTS command must be used to activate the class.

You can choose to protect a whole program library or individual load modules (members) in a library. The daemon program must reside in a program-controlled MVS partitioned data set, or in an HFS file with the extended attribute turned on via the **extattr +p** command. Both the program library for the daemons (for example, SYS1.LINKLIB) and the C run-time library must be protected.

An installation has a choice of either protecting all programs in a program library or individual programs. To protect all members in a data set, specify PROGRAM *.

> **Note:** Libraries in the LNKLIST concatenation are opened during IPL, and the programs in them are available to anyone unless the program name is defined as a controlled program.

When specifying program control for a program, you can choose to specify `UACC(READ)` or `UACC(NONE)`. `UACC(READ)` should be sufficient to protect the daemon program.

6.81 Auditing options for z/OS UNIX

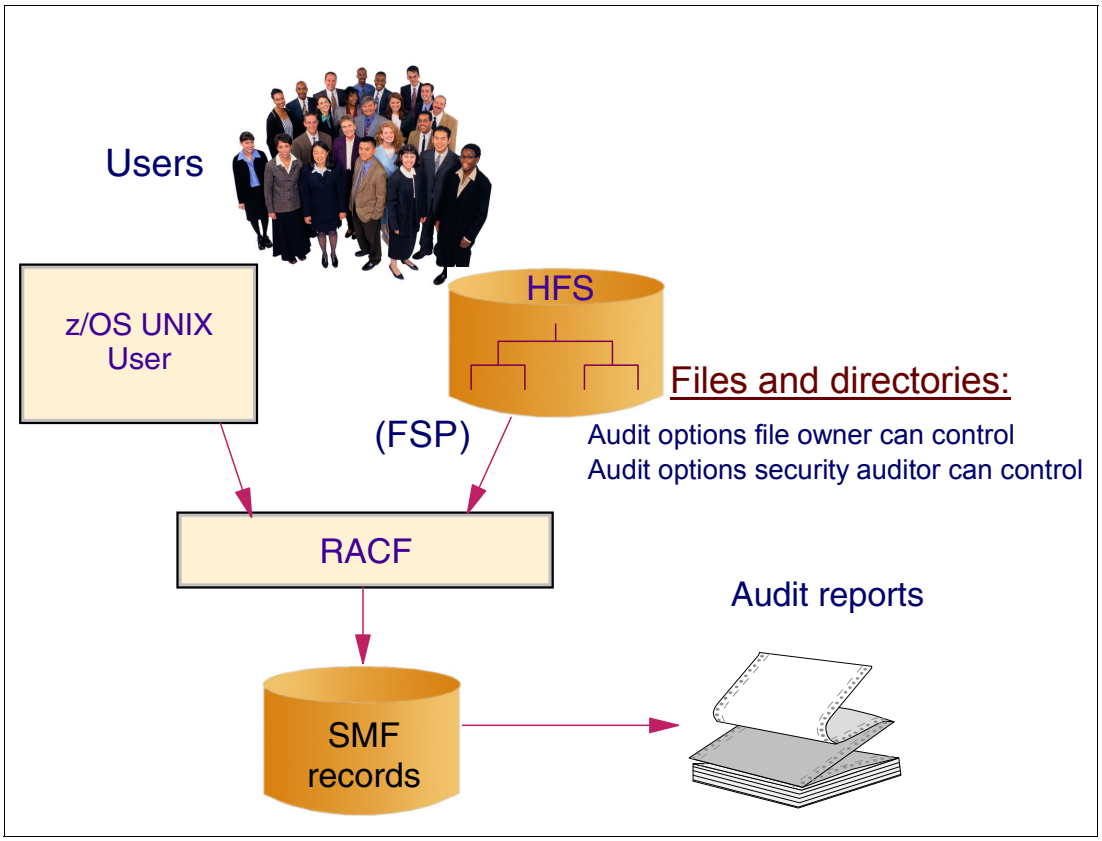


Figure 6-81 Auditing options for z/OS UNIX

z/OS UNIX auditing options

Every file and directory has security information, which consists of:

- File access permissions
- UID and GID of the file
- Audit options that the file owner can control
- Audit options that the security auditor can control

The security auditor uses reports formatted from RACF system management facilities (SMF) records to check successful and failing accesses to z/OS UNIX resources. An SMF record can be written at each point where the system makes security decisions.

Six classes are used to control auditing of z/OS UNIX security events. These classes have no profiles. They do not have to be active to control auditing.

The security administrator or the file owners can also specify auditing at the file level in the file system.

RACF provides utilities for unloading SMF data (IRRADU00) and data from the RACF database (IRRDBU00), which can be used as input in audit reports.

6.82 File-based auditing

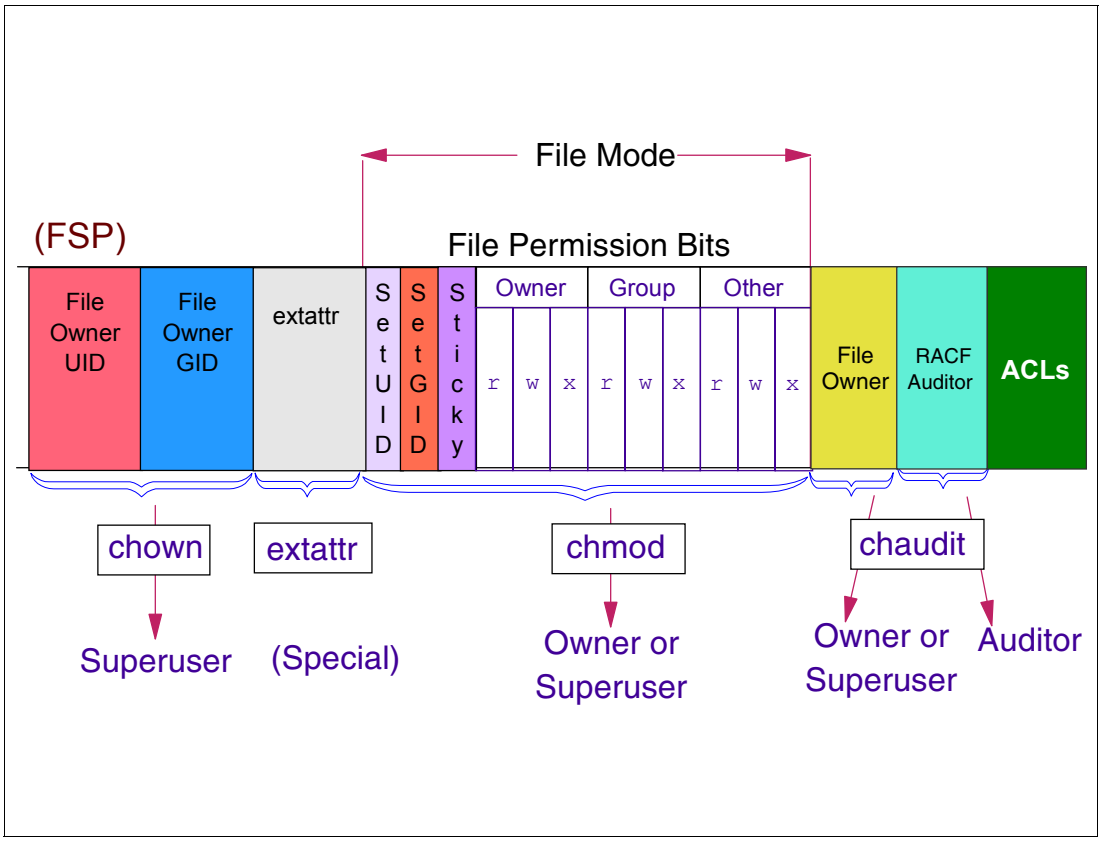


Figure 6-82 Auditing command for file access and the FSP

File-based auditing options

Auditing for file access is specified in the file security packet (FSP) with the **chaudit** command. Only a file owner or a security auditor can specify if auditing is turned on or off, and when audit records should be written for a directory or a file. There are two separate sets of auditing flags:

- Auditing set by the file owner (and superuser)
- Auditing set by the RACF AUDITOR

Audit records are written based on the combined owner and auditor settings. Auditing is set for read, write, and execute (search for directories) for the following kinds of accesses:

- Successful accesses
- Failures, that is, access violations
- All, which is both successes and failures
- None

Assigning audit options

When a file or a directory is created, default audit options are assigned. Different defaults are set for users and auditors. The same audit option is used no matter what kind of access is attempted (read, write, or execute). If auditing is not specified for a file, the defaults are:

- For owner auditing - audit failed accesses

- For RACF AUDITOR auditing - no auditing

When a file is created, these are the default audit options:

- User audit options: for all access types, audit_access_failed
- Auditor audit options: for all access types, don't_audit

Changing audit options

To change the audit options, you must use **chaudit**, a z/OS UNIX shell command, or the ISHELL menus. There are restrictions on who can change these options.

- For user audit options, you must be the owner of the file.
- For auditor audit options, you must have the RACF AUDITOR attribute. You can then change the auditor audit options for any file in the file system.

The default file-level audit options control the auditing of directory and file accesses. These defaults will only be used for a particular class (DIRSRCH, DIRACC, or FSOBJ) if SETROPTS LOGOPTIONS(DEFAULT(class)) has been issued for that class.

The syntax of the **chaudit** command is similar to the syntax of the **chmod** command, which you use to set the security of the file. Use r,w,x to specify whether you want to audit reads, writes, and/or execute accesses.

6.83 Audit z/OS UNIX events

- RACF classes for auditing:

DIRSRCH	Directory searches
DIRACC	Directory accesses
FSOBJ	All file and directory accesses
FSSEC	Change of FSP
PROCESS	Change of process UID/GID
PROCACT	Functions that look at data from other processes
IPCOBJ	Specifies auditing options for IPC accesses

Controlled by:

SETROPTS LOGOPTIONS
or
SETROPTS AUDIT

SETROPTS LOGOPTIONS(FAILURES(DIRSRCH,DIRACC))
SETROPTS AUDIT(FSOBJ,PROCESS,IPCOBJ)

Figure 6-83 RACF classes used for auditing

RACF classes for auditing

RACF provides the following classes for auditing z/OS UNIX events:

DIRSRCH Controls auditing of directory searches.

DIRACC Controls auditing of access checks for read/write access to directories.

FSOBJ Controls auditing of all access checks for file system objects except directory searches via SETROPTS LOGOPTIONS and controls auditing of creation and deletion of file system objects via SETROPTS AUDIT.

FSSEC Controls auditing of changes to the security data (FSP) for file system objects.

PROCESS Controls auditing of changes to the UIDs and GIDs of processes and changing of the thread limit via the SETROPTS LOGOPTIONS, and controls auditing of dubbing, undubbing, and server registration of processes via SETROPTS AUDIT.

PROCAT Controls auditing of functions that look at data from or affect other processes.

IPCOBJ Specifies auditing options for IPC accesses and access checks for objects and changes to UIDs, GIDs, and modes. For access control and for z/OS UNIX user identifier (UID), z/OS UNIX group identifier (GID), and mode changes, use SETROPTS LOGOPTIONS. For object create and delete, use SETROPTS AUDIT.

Controlling auditing with commands

Auditing can be controlled by using the commands SETROPTS LOGOPTIONS, and SETROPTS AUDIT, as follows:

- SETROPTS LOGOPTIONS(auditing_level(class_name)) audits access attempts to the resources in the specified class according to the auditing level specified and can be used for all classes.
- SETROPTS AUDIT(class_name) specifies the names of the classes that RACF should audit. The AUDIT option can be used for the classes FSOBJ, IPCOBJ, and PROCESS.
- SETROPTS LOGOPTIONS(DEFAULT) indicates you want no class auditing and only file-level auditing (use **`chaudit`** to specify).

Audit records are always written when:

- A user who is not defined as a z/OS UNIX user tries to dub a process.
- A user who is not a superuser tries to mount or unmount a file system.
- A user tries to change a home directory.
- A user tries to remove a file, hard link, or directory.
- A user tries to rename a file, hard link, symbolic link, or directory.
- A user creates a hard link.

Note: There is no option to turn off these audit records.

The auditing levels for LOGOPTIONS are:

ALWAYS All access attempts to resources protected by the class are audited.

NEVER No access attempts to resources protected by the class are audited (all auditing is suppressed).

SUCCESSES All successful access attempts to resources protected by the class are audited.

FAILURES All failed access attempts to resources protected by the class are audited.

DEFAULT Auditing is controlled by the auditing bits in the FSP for z/OS UNIX files and directories.

6.84 Chaudit command

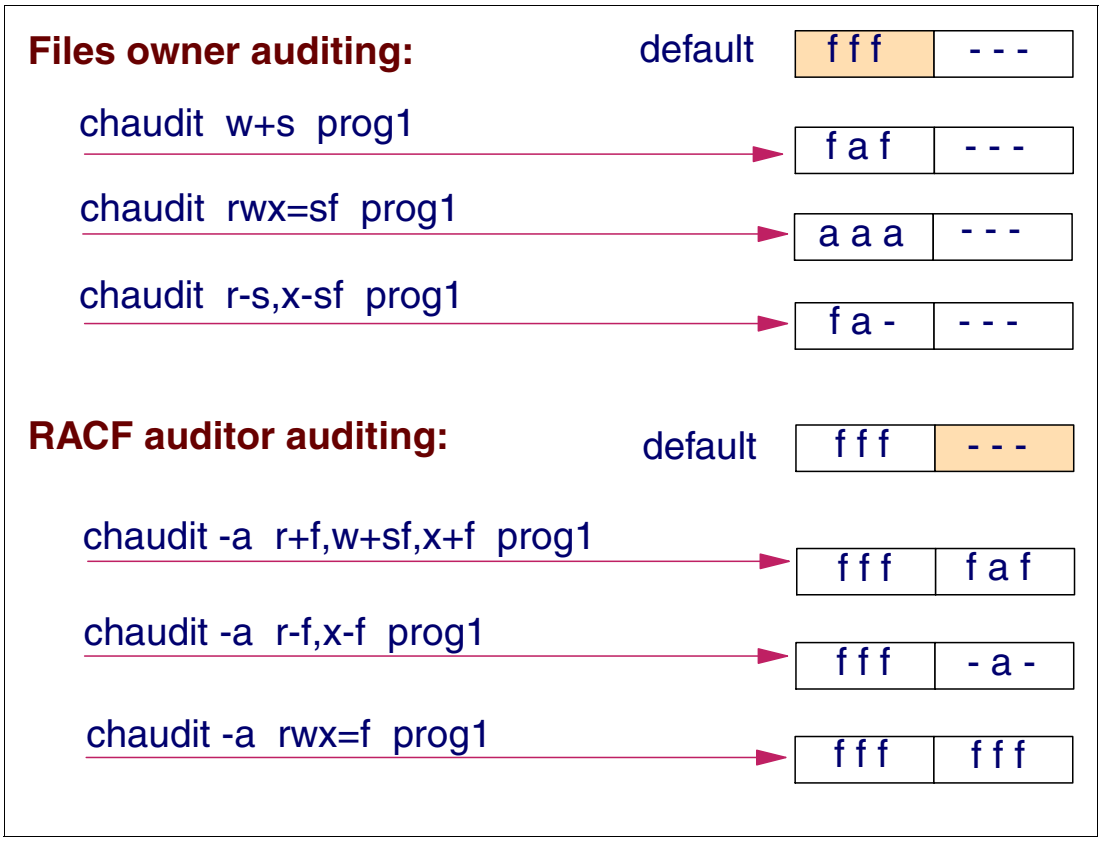


Figure 6-84 **`chaudit`** *command to set auditing options*

Chaudit command

The **`chaudit`** command changes the audit attributes of the specified files or directories. Audit attributes determine whether or not accesses to a file are audited by the system authorization facility (SAF) interface.

> **Note:** **`chaudit`** can be used only by the file owner or a superuser for non-auditor-requested audit attributes. It takes a user with auditor authority to change the auditor-requested audit attributes.

Command examples

The top part of Figure 6-84 shows examples of the **`chaudit`** command usage by the file owner (or superuser). The default audit settings are shown in the upper right corner of the figure, then the command is applied as follows:

- **`chaudit w+s prog1`** adds (`+`) auditing for successful accesses (`s`) for write accesses (`w`).
- **`chaudit rwx=sf prog1`** specifies that all (`a`) accesses, that is both successes (`s`) and failures (`f`), are to be audited for reads, writes, and executes.
- **`chaudit r-s,x-sf prog1`** says to stop (`-`) auditing successes (`s`) for read (`r`), and stop (`-`) auditing both successes (`s`) and failures (`f`) for execute (`x`) access. The same effect can be achieved with the command **`chaudit r=f,x= prog1`**.

Command examples

Examples of the **chaudit** command usage by the RACF Auditor are shown in the lower part of Figure 6-84. The default audit settings shown first, then the command is applied as follows:

- **chaudit -a r+f,w+s,x+f prog1** adds auditing of successes (s) and failures (f) for write access, and specifies to write an audit record whenever an access failure (f) occurs for read or execute accesses.
- **chaudit -a r-f,x-f prog1** turns off (-) auditing for failures (f) for read and execute accesses.
- **chaudit -a rwx=f prog1** turns on auditing for unsuccessful (f) read, write, and execute accesses.

Note that the auditor includes the option **-a** when issuing the **chaudit** command, and that the auditor can only set the audit flags in the auditor's section of the FSP.

The audit condition part of a symbolic mode is any combination of the following:

s Audit on successful access if the audit attribute is on.

f Audit on failed access if the audit attribute is on.

The following command changes the file prog1 so that all successful and unsuccessful file accesses are audited:

```
chaudit rwx=sf prog1
```

6.85 List audit information for files

```
ROGERS @ SC65:/u/rogers>ls -W /usr/man/C
total 136
drwxr-xr-x  fff---  2 HAIMO    OMVSGRP     8192 May 29  2002 IBM
drwxr-xr-x  fff---  3 HAIMO    OMVSGRP     8192 Jun 10  2002 cat1
drwxrwxr-x  fff---  3 HAIMO    OMVSGRP     8192 May 29  2002 cat8
drwxr-xr-x  fff---  3 HAIMO    OMVSGRP     8192 May 29  2002 man1
-rw-rw-r--  fff---  2 HAIMO    OMVSGRP    33887 May 29  2002 whatis
```

File auditing

Figure 6-85 Listing auditing options for specified files

Command to list audit information for files and directories

The `ls` command with the `-W` option is used to display the auditing that is in effect for a file. In this example, since we have done the `ls` for a directory, all the files in the directory are displayed.

The auditing options are displayed to the right of the file permissions. The left three characters are for owner-set auditing, and the right three characters are for auditor-set auditing.

There are four possible characters for each of these columns:

- No auditing (`-`)
- Successful accesses (`s`)
- Access failures (`f`)
- All accesses (`a`)

6.86 Auditing reports

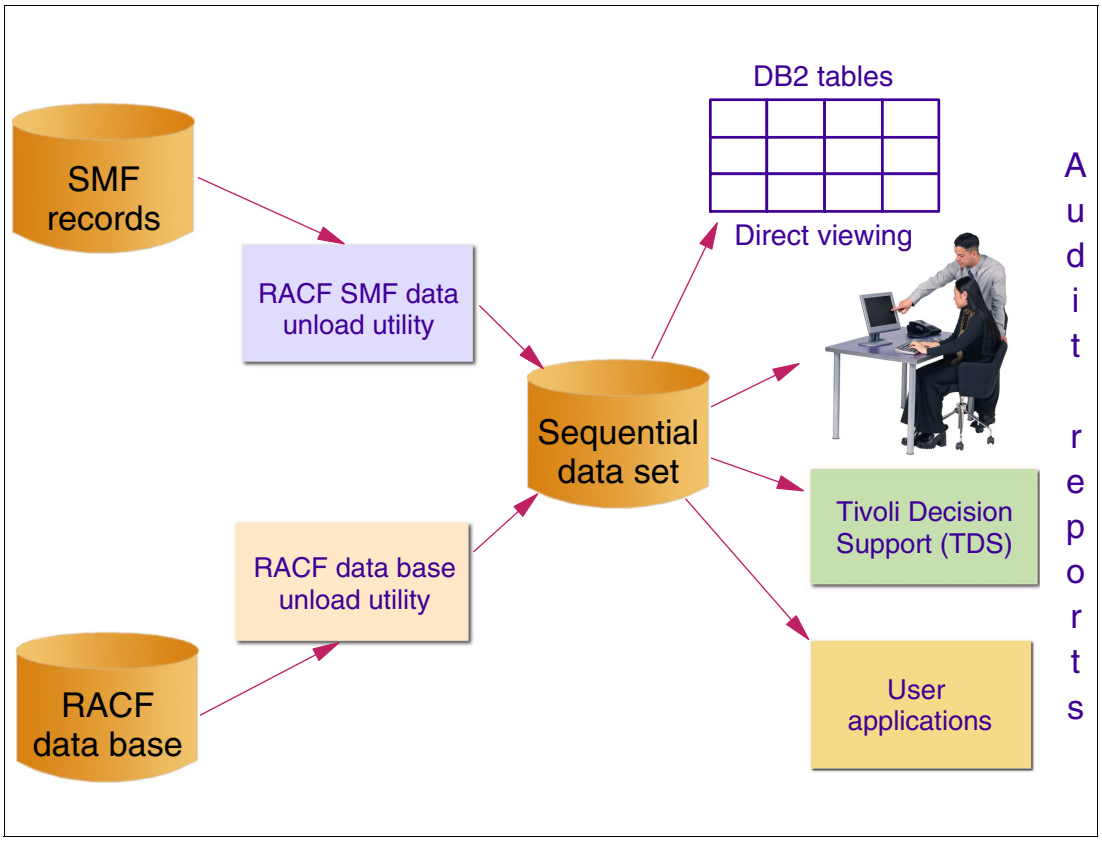


Figure 6-86 Creating auditing reports

Audit reports

In a z/OS UNIX environment which uses RACF as its access control program, the security auditor has two main tasks to perform:

- Verify that the RACF profiles have the proper contents (OMVS segments in the user and group profiles, and logging options in particular).
- Use the security logs to follow up on detected violations, and to detect abnormal behavior by authorized users.

The audit information can be quite extensive and is found in the RACF database, the RACF security log, and in SMF records. RACF provides two utilities for unloading security data and placing the output in a sequential data set:

- The RACF database unload utility, IRRDBU00 can be used to unload a user's OMVS segment.
- The RACF SMF data unload utility, IRRADU00 can be used to unload the relevant audit records.

The output from these utilities can be used in several ways: viewed directly, used as input for installation-written programs, and manipulated with sort/merge utilities. It can also be uploaded to a database manager (for example, DB2) to process complex inquiries and create installation-tailored reports. SYS1.SAMPLIB contains examples of how to create DB2 tables, and load RACF data produced from the RACF unload utilities, into the DB2 tables. It also has database query examples.

6.87 Maintain z/OS UNIX-level security

- Check purchased applications
- Determine auditing requirements for installation
- Set up rules for:
  - Sharing data in files
  - Specifying permission bits
- Protect all HFS data sets with UACC(NONE)

Figure 6-87 Checking products for z/OS UNIX-level security

Checking programs for security

To maintain the security available from the system, take these steps:

- Check any purchased program to make sure that it will not lower the system security level. If you cannot be sure of a program, do not put it on the system.
- Determine the auditing requirements for the installation. Control auditing data with the RACF SETROPTS command and the **chaudit** shell command.
- Set up rules for:
  - Sharing data in files
  - Specifying permission bits when creating files or using the **chmod** shell command or **chmod()** function
- Protect all HFS data sets with a RACF profile that specifies `UACC(NONE)`. Only administrators with responsibility for creating, restoring, or dumping HFS data sets should be permitted to this profile.

To maintain security, require z/OS UNIX users to set the permission bits for their own files to deny access to any user but the file owner.

Each user is responsible for protecting their own data. However, data that other users need to access must have the appropriate permission bits set for group or other.

6.88 Setting up z/OS UNIX (1)

- Determine GID numbering strategy
- Define new groups
  - TTY
  - OMVSGRP
- Add OMVS segments to existing groups
  - You
  - Any potential z/OS UNIX users

Figure 6-88 Tasks needed to set up z/OS UNIX

Setting up security check list

There is no predefined numbering scheme for z/OS UNIX GIDs. Therefore, you must decide what is suitable for your system when allocating GIDs.

Once a scheme has been chosen, you need to set up some groups. It is recommended that you set up at least two new groups: one for z/OS UNIX and another for TTY. The recommended group name for z/OS UNIX is OMVSGRP, but this can be changed to something else if it is not suitable on your system. The other group, TTY is recommended to be called that name—otherwise changes will be required in BPXPRMxx to support the different name.

You will need to add OMVS segments to some existing groups. For example, you (as the person doing the OMVS configuration work) will need a group OMVS segment (GID). Then you need to consider who else will need to use z/OS UNIX and ensure that they, too, have OMVS segments.

Some software detects that z/OS UNIX is active, and if so will want to use it. Two examples are TCP/IP and RMFGAT.

6.89 Setting up z/OS UNIX (2)

- Determine UID numbering strategy
- Define new users
  - OMVSKERN
  - BPXROOT
- Add OMVS segments to existing users
  - You
  - Any potential z/OS UNIX users

Figure 6-89 Tasks needed to set up z/OS UNIX

Setting up security check list

There is no predefined numbering scheme for z/OS UNIX UIDs other than UID=0 means Superuser. Therefore, you must decide what is suitable for your system when allocating UIDs.

Once a scheme has been chosen, you need to set up some user IDs. It is recommended that you set up at least two new user IDs: OMVSKERN and BPXROOT. The OMVSKERN user ID is used for the OMVS and BPXOINIT address spaces. The BPXROOT user ID is used when a daemon sets UID=0, but the user ID is not known. These recommended user ID names can be changed to something else if they are not suitable on your system.

You will need to add OMVS segments to some existing user IDs. For example, you (as the person doing the OMVS configuration work) will need a user ID OMVS segment (UID/HOME/PROGRAM). Then you need to consider who else will need to use z/OS UNIX and ensure that they, too, have OMVS segments.

Some software detects that z/OS UNIX is active, and if so will want to use it. Two examples are TCP/IP and RMFGAT.

6.90 Setting up z/OS UNIX (3)

- Define cataloged procedure (STC) controls
  - OMVS
  - BPXOINIT
  - BPXAS
  - ZFS

Figure 6-90 Tasks needed to set up z/OS UNIX

Setting up security check list

To activate z/OS UNIX, z/OS UNIX itself needs to have some identification (user ID/group) assigned to it. This is done either by the RACF started procedures table (ICHRIN03) or the STARTED class profiles.

Both the OMVS and BPXOINIT cataloged (or started) procedures (or started tasks - STCs) need to be defined. Only the OMVS procedure should be defined with the `TRUSTED` attribute.

When programs issue fork or spawn, the BPXAS PROC found in SYS1.PROCLIB is used to provide a new address space. For a fork, the system copies one process, called the parent process, into a new process, called the child process. Then it places the child process in a new address space. The forked address space is provided by workload manager (WLM), which uses the BPXAS PROC to create the address spaces.

zFS runs as a z/OS UNIX colony address space. There must be an entry in a BPXPRMxx parmlib member for ZFS and the ZFS procedure (PROC) must be available. zFS is started by z/OS UNIX based on the FILESYSTYPE statement for ZFS in the BPXPRMxx parmlib member.

6.91 Setting up z/OS UNIX (4)

- Any program loaded into an A/S with daemon authority
  - Must be a controlled program
- Define programs to Program Control
- z/OS daemons reside in the HFS and are controlled
- User defined daemons
  - Specific daemon program names or *

```
RDEFINE  PROGRAM   *  UACC(READ)  ADDMEM +
            ('SYS1.LINKLIB'/'******'/NOPADCHK +
            'CEE.SCEERUN'/RLTPAK/NOPADCHK +
            'SYS1.SEZALOAD'//NOPADCHK)
SETROPTS  WHEN(PROGRAM)  REFRESH
```

Figure 6-91 Tasks needed to set up z/OS UNIX

Setting up a security check list

All programs loaded into an address space that requires daemon authority need to be marked as controlled (for example, user programs that are loaded and any run-time library modules that are loaded). All modules loaded from LPA are considered to be controlled. The BPX.DAEMON requires z/OS UNIX load libraries to be program-controlled. The data sets listed in Figure 6-91 are the main ones required to be program-controlled when setting up z/OS UNIX. These data sets are:

- LINKLIB - z/OS (including z/OS UNIX)
- SCEERUN - Language Environment
- SEZALOAD - TCP/IP

Daemons shipped by z/OS reside in the HFS and are marked program-controlled, so you do not need to define them. For example, suppose you have a daemon named server1. The file /bin/server1 would have the sticky bit on. Member SERVER1 would reside in SYS1.LINKLIB and be defined as program-controlled:

```
RDEFINE PROGRAM SERVER1 ADDMEM('SYS1.LINKLIB'/'******'/NOPADCHK) + UACC(READ)
SETROPTS WHEN(PROGRAM) REFRESH
```

Note: '******' (six asterisks surrounded by single quotes) specifies the current SYSRES volume.

6.92 Setting up z/OS UNIX (5)

- Define BPX.DAEMON
- Define BPX.SUPERUSER or UNIXPRIV (R8)
- Define other BPX.* profiles if required

Figure 6-92 Tasks needed to set up z/OS UNIX

Setting up security check list

Of the BPX.* FACILITY class profiles, it is strongly recommended that at least BPX.DAEMON and BPX.SUPERUSER be implemented. BPX.DAEMON restricts access to the following services:

- seteuid (BPX1SEU service)
- setuid (BPX1SUI service)
- setreuid (BPX1SRU service)
- spawn (BPX1SPN service with a change in user ID requested.
- pthread_security_np()
- auth_check_resource_np()
- _login()
- _password()

BPX.SUPERUSER allows users with access to the BPX.SUPERUSER FACILITY class profile to switch to superuser authority (effective UID of 0).

You can define profiles in the UNIXPRIV class to grant RACF authorization for certain z/OS UNIX privileges. These privileges are automatically granted to all users with z/OS UNIX superuser authority. By defining profiles in the UNIXPRIV class, you can specifically grant certain superuser privileges with a high degree of granularity to users who do not have superuser authority. This allows you to minimize the number of assignments of superuser authority at your installation and reduces your security risk.

6.93 RACF definitions for zFS

- RACF definitions required for Distributed File Service
  - SMB and DFS support
    - Not required to use zFS
- Additional definitions for zFS
  - RDEFINE STARTED ZFS.** STDATA(USER(DFS))
  - SETROPTS RACLIST(STARTED) REFRESH

```
ADDGROUP DFSGRP SUPGROUP(SYS1) OMVS(GID(2))
ADDUSER DFS OMVS(HOME(/opt/dfslocal/home/dfscntl)
UID(0))
     DFLTGRP(DFSGRP) AUTHORITY(CREATE) UACC(NONE)
RDEFINE STARTED DFS.** STDATA(USER(DFS))
RDEFINE STARTED DFSCM.** STDATA(USER(DFS))
RDEFINE STARTED ZFS.** STDATA(USER(DFS)GROUP(DFSGRP
TRUSTED(YES))
   SETROPTS RACLIST(STARTED)
   SETROPTS RACLIST(STARTED) REFRESH
```

Figure 6-93 RACF definitions for zFS

zFS RACF definitions

To define DFS to RACF you must create the following definitions with these exact names. Even if you use none of the functions of DFS, you can use the following DFS definitions for zFS:

- Define DFSGRP as a group.
- Define DFS as a user.
- Define ZFS as a started task.

For RACF, use the following commands:

```
ADDGROUP DFSGRP SUPGROUP(SYS1) OMVS(GID(2))
ADDUSER DFS OMVS(HOME(/opt/dfslocal/home/dfscntl) UID(0))
     DFLTGRP(DFSGRP) AUTHORITY(CREATE) UACC(NONE)
RDEFINE STARTED ZFS.** STDATA(USER(DFS) GROUP(DFSGRP) TRUSTED(YES))
SETROPTS RACLIST(STARTED) REFRESH
```

Note: A user ID other than DFS can be used to run the ZFS started task if it is defined with the same RACF characteristics as shown for the DFS user ID.

6.94 UNIXPRIV class with z/OS V1R3 and zFS

- Some zfsadm commands require superuser authority
  - This is true also for other zFS commands and utilities
- zFS with z/OS V1R3 supports:
  - SUPERUSER.FILESYS.PFSCTL profile - UNIXPRIV
  - Allows a zFS administrator to have just READ authority to this UNIXPRIV profile resource
    - Commands that modify zFS file systems or aggregates
    - uid 0 not required with this access

```
RDEFINE UNIXPRIV SUPERUSER.FILESYS.PFSCTL UACC(NONE)
PERMIT SUPERUSER.FILESYS.PFSCTL CLASS(UNIXPRIV) ID(ROGERS) ACCESS(READ)
RDEFINE UNIXPRIV SUPERUSER.FILESYS.MOUNT UACC(NONE)
PERMIT SUPERUSER.FILESYS.MOUNT CLASS(UNIXPRIV) ID(ROGERS) ACCESS(UPDATE)
```

Figure 6-94 UNIXPRIV class changes for zFS

Authorization for administrators for zFS commands

zFS with z/OS V1R3 supports the SUPERUSER.FILESYS.PFSCTL profile of the UNIXPRIV class. This makes it possible for a zFS administrator to have just READ authority to this UNIXPRIV profile resource, SUPERUSER.FILESYS.PFSCTL, rather than requiring a UID of 0 for **`zfsadm`** commands that modify zFS file systems or aggregates. The same is true for the other zFS commands and utilities, so zFS administrators do not need a UID of 0.

To allow the zFS administrator to mount and unmount file systems, permit update access to another profile, SUPERUSER.FILESYS.MOUNT, in class UNIXPRIV.

> **Note:** UPDATE access is needed if the user needs to **`mount`**, **`chmount`**, or **`unmount`** file systems with the `setuid` option; otherwise, READ access is sufficient.

UNIXPRIV authorization is invoked by creating the needed resources in the UNIXPRIV class and then giving users READ authority to it, as follows:

```
SETROPTS CLASSACT(UNIXPRIV)
SETROPTS RACLIST(UNIXPRIV)
RDEFINE UNIXPRIV SUPERUSER.FILESYS.PFSCTL UACC(NONE)
PERMIT SUPERUSER.FILESYS.PFSCTL CLASS(UNIXPRIV) ID(ROGERS) ACCESS(READ)
RDEFINE UNIXPRIV SUPERUSER.FILESYS.MOUNT UACC(NONE)
PERMIT SUPERUSER.FILESYS.MOUNT CLASS(UNIXPRIV) ID(ROGERS) ACCESS(UPDATE)
SETROPTS RACLIST(UNIXPRIV) REFRESH
```

6.95 List current user IDs with the ISHELL

```
❑ ISHELL - Setup - (2). User list
--------------------------------------------------------------------------
                                User List                Row 622 to 645 of 927
Command ===> ____________________________________________________________

User ID          UID  Group     Home Directory        Initial Program
PUBLIC           998  SYS1      /                     /bin/sh
PUBUSER           -1  ________  ____________________  ____________________
RACF              -1  ________  ____________________  ____________________
RACHEL            -1  ________  ____________________  ____________________
RALPHB           477  SYS1      /u/ralphb             /bin/sh
RALPHR            -1  ________  ____________________  ____________________
RAMA           87674  SYS1      /                     /bin/sh
RANIERI            0  SYS1      /                     /bin/sh
RAYNO             -1  ________  ____________________  ____________________
RBORGES           -1  ________  ____________________  ____________________
RCHIANG           -1  ________  ____________________  ____________________
RCONWAY            0  SYS1      /etc                  /bin/sh
RC63               0  SYS1      /u/rc63               /bin/sh
RC64               0  SYS1      /                     /bin/sh
RC65               0  SYS1      /                     /bin/sh
REYO           10052  SYS1      /                     /bin/sh
```

Figure 6-95 List the currently defined z/OS UNIX users

List z/OS UNIX users

A system programmer can use the ISPF shell to perform tasks such as listing the current users defined in the OMVS segment. This requires superuser authority.

You can use the ISHELL to do the following:

- List files in a directory
- Create, delete, or rename directories, files, and special files
- Browse files
- Edit files
- Copy files
- Display file attributes
- Search files for text strings
- Compare files or directories
- Run executable files
- Display the attributes and contents of a symbolic link (symlink)
- Mount and unmount a hierarchical file system (HFS)
- Create a hierarchical file system (HFS)
- Set up character special files
- Set up standard directories for a root file system
- Set up existing users and groups for z/OS UNIX System Services access

6.96 The BPXBATCH utility

- BPXBATCH is an MVS utility that you can use to run shell commands or shell scripts
  - Can run executable files through the MVS batch environment
- Invoke BPXBATCH as follows:
  - In JCL
  - From the TSO/E READY prompt
  - From TSO CLISTs and REXX execs
  - From a program

Figure 6-96 Using the BPXBATCH utility

The BPXBATCH utility

BPXBATCH is an MVS utility that you can use to run shell commands or shell scripts and to run executable files through the MVS batch environment. You can invoke BPXBATCH as follows:

- In JCL - Use one of the following:

```
EXEC PGM=BPXBATCH,PARM='SH program-name'
EXEC PGM=BPXBATCH,PARM='PGM program-name'
```

- From the TSO/E READY prompt
- From TSO CLISTs and REXX execs - Use one of the following:

```
BPXBATCH SH program-name
BPXBATCH PGM program-name
```

- From a program

BPXBATCH has logic to detect when it is run from JCL. If the BPXBATCH program is running as the only program on the job step task level, it sets up stdin, stdout, and stderr and execs the requested program. If BPXBATCH is not running as the only program at the job step task level, the requested program will run as the second step of a JES batch address space from JCL in batch. If run from any other environment, the requested program will run in a WLM initiator in the OMVS subsys category.

6.97 The BPXBATCH job

```
//OMVSPGM   JOB   USER=userid
//OMVSEXEC  EXEC   PGM=BPXBATCH,PARM='pgm /cprog a1 a2'
//STDOUT      DD   PATH='/dir1/dir2/std.output',
//                 PATHOPTS=(OWRONLY,OCREATE),
//                 PATHMODE=(SIRWXV),
//                 PATHDISP=KEEP
//STDERR      DD   PATH='/dir1/dir2/std.error',
//                 PATHOPTS=(OWRONLY,OCREATE),
//                 PATHMODE=(SIRWXV),
//                 PATHDISP=KEEP
/*
```

Figure 6-97 Sample JCL for a BPXBATCH job

Using BPXBATCH

When you are using BPXBATCH to run a program, you typically pass the program a file that sets the environment variables. If you do not pass an environment variable file when running a program with BPXBATCH or if the HOME and LOGNAME variables are not set in the environment variable file, those two variables are set from your logon RACF profile. LOGNAME is set to the user name and HOME is set to the initial working directory from the RACF profile.

With BPXBATCH, you can allocate the MVS standard files stdin, stdout, and stderr as HFS files for passing input. If you do allocate these files, they must be HFS files. You can also allocate MVS data sets or HFS text files containing environment variables (stdenv). If you do not allocate them, stdin, stdout, stderr, and stdenv default to /dev/null. Allocate the standard files using the data definition PATH keyword options, or standard data definition options for MVS data sets.

The BPXBATCH default for stderr is the same file defined for stdout. For example, if you define stdout to be /tmp/output1 and do not define stderr, then both printf() and perror() output is directed to /tmp/output1.

For BPXBATCH, you can define stdin, stdout, and stderr using one of the following:

- The TSO/E ALLOCATE command, using the ddnames STDIN, STDOUT, and STDERR
- A JCL DD statement with the PATH operand, using the ddnames STDIN, STDOUT, and STDERR

- Redirection, which, in a shell, is a method of associating files with the input or output of commands.

STDIN and STDOUT

STDIN is an input stream from which data is retrieved. Standard input is normally associated with the keyboard, but if redirection or piping is used, the standard input can be a file or the output from a command.

STDOUT is the output stream to which data is directed. Standard output is normally associated with the console, but if redirection or piping is used, the standard output can be a file or the input to a command. See also standard error.

6.98 BPXBATCH and shell commands

‘

```
//GGIBPXBA   JOB   (55,500,,999),'MVS  ',CLASS=A,
// MSGCLASS=R,REGION=0K,NOTIFY=&SYSUID
//EXECBPX   EXEC PGM=BPXBATCH,REGION=8M,
//   PARM='SH ls /usr/lib'
//*STDIN   DD  PATH='/stdin-file-pathname',
//*            PATHOPTS=(ORDONLY)
//STDOUT   DD  PATH='/u/ggi/bin/mystd.out',
//             PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
//             PATHMODE=SIRWXU
//STDERR   DD  PATH='/u/ggi/bin/mystd.err',
//             PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
//             PATHMODE=SIRWXU
//STDENV   DD  *
TZ=EST5EDT
LANG=C
PATH=/bin:/usr/lpp/java/J1.4/bin:.

//*STDENV   DD  PATH='/etc/setting.envvars',
//*             PATHOPTS=ORDONLY
```

Figure 6-98 Using shell commands in BPXBATCH JCL

The BPXBATCH utility

BPXBATCH makes it easy for you to run shell scripts, REXX execs, and executable files that reside in hierarchical file system (HFS) files through the MVS job control language (JCL). If you do most of your work from TSO/E, using BPXBATCH saves you the trouble of going into the shell to run your scripts and executable files. The format of BPXBATCH for JCL and TSO/E is as follows:

The format of BPXBATCH for JCL and TSO/E is as follows:

JCL: EXEC PGM=BPXBATCH,PARM='SH | PGM program_name'

TSO/E: BPXBATCH SH | PGM program_name

In addition, BPXBATCH can be used in a REXX exec to call a shell script as shown in the following example:

```
"BPXBATCH SH "shellcmd

   IF RC ^= 0 Then
      DO
        Say ' OSHELL RC = ' RC
```

You can allocate STDIN, STDOUT, and STDERR as files, using the PATH operand, and redirect the messages to HFS files.

The command to execute the shell script or program is located in the PARM='...' section. You can either specify the PARM='...' or the //STDIN DD statements. The default when PARM='...' is not specified is SH.

SH Specifies that the shell designated in your TSO/E user ID's security product profile is to be started and is to run shell commands or scripts provided from STDIN or the specified program_name.

PGM Specifies that the specified program_name is to be run as a called program from a shell environment.

For environment settings, use the //STDENV DD statement. This can be specified as JCLIN or as a path pointing to an HFS file.

The STDERR file

The STDERR file is the output stream to which error messages or diagnostic messages are sent. See also standard input, standard output.

The STDENV file

The BPXBATCH utility also uses the STDENV file to allow you to pass environment variables to the program that is being invoked. This can be useful when not using the shell, such as when using the PGM parameter.

7

zFS file systems

The z/OS Distributed File Service zSeries File System (zFS) is a z/OS UNIX System Services (z/OS UNIX) file system that can be used in addition to the hierarchical file system (HFS). zFS file systems contain files and directories that can be accessed with z/OS UNIX application programming interfaces (APIs). These file systems can support access control lists (ACLs). zFS file systems can be mounted into the z/OS UNIX hierarchy along with other local (or remote) file system types (for example, HFS, TFS, AUTOMNT, and NFS).

This chapter describes the structure of zFS hierarchical file systems and how to customize it to your requirements. zFS is part of the Distributed File Service base element of z/OS. Before using the zFS support, you must install the z/OS release, the Distributed File Service, and the other base elements of z/OS using the appropriate release documentation.

zFS does not replace HFS, rather it is complementary to HFS. zFS can be used for all levels of the z/OS UNIX System Services hierarchy (including the root file system) when all members are at the z/OS V1R7 level. Because zFS has higher performance characteristics than HFS and is the strategic file system, HFS might not be supported in future releases, which will cause you to migrate the remaining HFS file systems to zFS.

zFS and HFS can both participate in a shared sysplex environment. zFS supports a shared file system capability in a multisystem sysplex environment. The term shared file system environment refers to a sysplex that has a BPXPRMxx specification of SYSPLEX(YES). That is, users in a sysplex can access zFS data that is owned by another system in the sysplex. For full sysplex support, zFS must be running on all systems in the sysplex in a shared file system environment and all zFS file systems must be compatibility mode file systems (that is, they cannot be file systems in multi-file system aggregates). zFS muti-file system aggregates are not supported by automount.

In addition to providing an introduction to zFS concepts, this chapter provides details on how to:

- Manage and create a zFS hierarchical file system.
- Use zFS commands to display information.
- Perform space management for a zFS hierarchical file system.

© Copyright IBM Corp. 2011. All rights reserved.

7.1 zSeries File System (zFS)

- zFS is the strategic UNIX Systems Services file system for z/OS
- The Hierarchical File System (HFS) functionality has been stabilized
  - HFS is expected to continue shipping as part of the operating system and will be supported in accordance with the terms of a customer's applicable support agreement
- IBM intends to continue enhancing zFS functionality, including RAS and performance capabilities, in future z/OS releases
  - All requirements for UNIX file services are expected to be addressed in the context of zFS only

Figure 7-1 zFS replacing HFS

zFS file systems

zFS is complementary to HFS. zFS can be used for all levels of the z/OS UNIX System Services hierarchy (including the root file system) when all members are at the z/OS V1R7 level. Because zFS has higher performance characteristics than HFS and is now considered the strategic file system, HFS may no longer be supported in future releases and you will have to migrate the remaining HFS file systems to zFS.

HFS support

Support for the HFS file system has been stabilized and you are encouraged to migrate to the zFS file system for better performance, but IBM has not announced removal of support for the HFS file system.

zFS functionality

The zSeries File System (zFS) is the strategic z/OS UNIX System Services file system for z/OS. IBM has enhanced zFS function in z/OS V1R7 so that you can use zFS file systems at all levels within the file hierarchy.

The migration to zFS file systems needs to be well planned since it will take significant effort to migrate all data from HFS file systems to zFS file systems.

Unavailability of the data for read/write access must be planned for, as well as the space for two file systems that are about the same size.

7.2 zFS compatibility mode aggregate

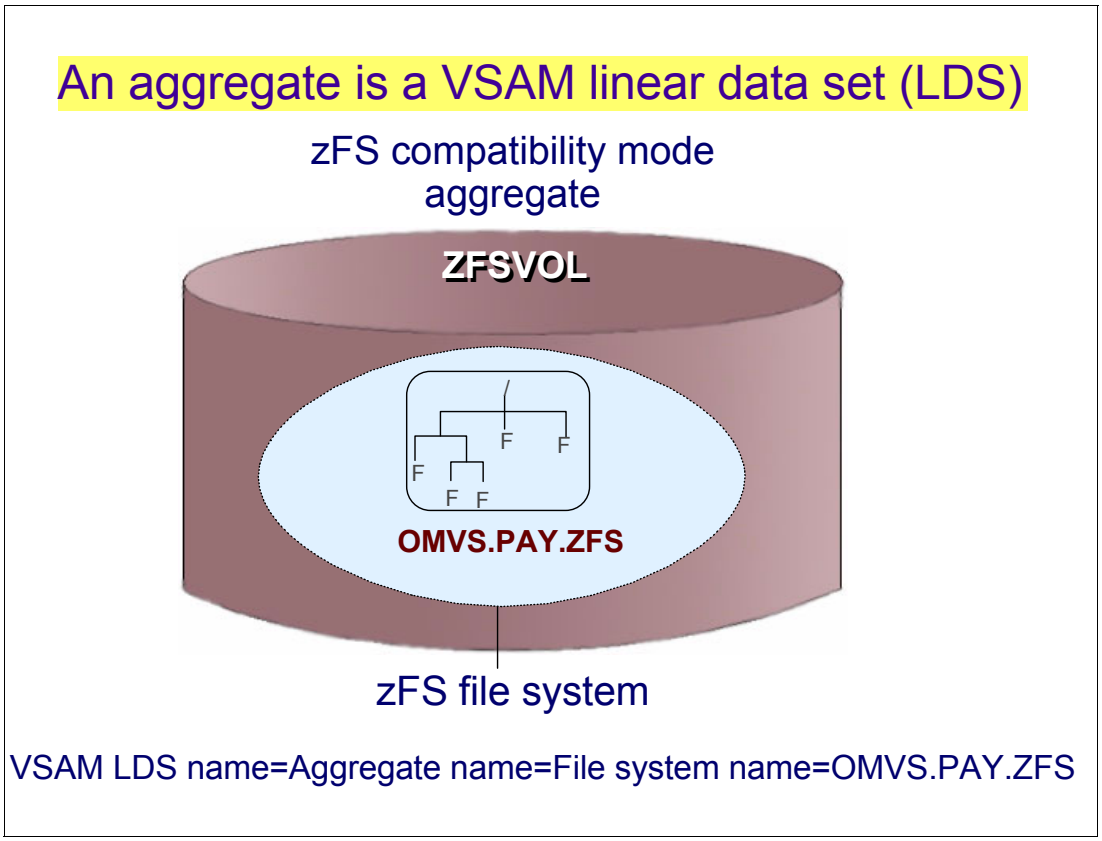


Figure 7-2 zFS compatibility mode aggregate

zFS aggregates

A zFS aggregate is a data set that contains zFS file systems. The aggregate is a VSAM Linear Data Set (VSAM LDS) and is a container that can contain one or more zFS file systems. An aggregate can only have one VSAM LDS, but it can contain an unlimited number of file systems. The name of the aggregate is the same as the VSAM LDS name. Sufficient space must be available on the volume or volumes, because multiple volumes may be specified on the DEFINE of the VSAM LDS. DFSMS decides when to allocate on these volumes during any extension of a primary allocation. VSAM LDSs greater than 4 GB may be specified by using the extended format and extended addressability capability in the data class of the data set.

zFS compatibility aggregate

A compatibility mode aggregate can contain only one zFS file system, making this type of aggregate more like an HFS file system. This is flagged in the aggregate when it is created. The name of the file system is the same as the name of the aggregate, which is the same as the VSAM LDS cluster name.

The file system size (called a quota) in a compatibility mode aggregate is set to the size of the aggregate. Compatibility mode aggregates are more like an HFS data set, except that they are VSAM linear data sets instead of HFS data sets.

7.3 BPXPRMxx definitions for zFS

- BPXPRMxx FILESYSTYPE statement
  - Defines zFS as a physical file system (PFS)

```
FILESYSTYPE TYPE(ZFS)
  ENTRYPOINT(IOEFSCM)
  ASNAME(ZFS)
```

```
//ZFS      PROC REGSIZE=0M
//ZFSGO EXEC PGM=BPXVCLNY,REGION=&REGSIZE,TIME=1440
//*STEPLIB  DD DISP=SHR,DSN=IOE.SIOELMOD
//IOEZPRM DD DSN=IOE.PARMLIB(IOEFSPRM),DISP=SHR
// PEND
```

Figure 7-3 zFS definitions in PARMLIB and PROCLIB

Defining zFS as a PFS

Physical file systems are sometimes initialized in an address space called a colony address space. You can think of these address spaces as extensions of the kernel address space. The NFS Client and DFS Client physical file systems must be set up in a colony address space because they need to use socket sessions to talk to their remote servers, and this cannot be done from the kernel. Whether or not a PFS runs in a colony address space is controlled by the ASNAME parameter of the FILESYSTYPE statement for the PFS in the BPXPRMxx member of the SYS1.PARMLIB concatenation.

zFS is the newest colony address space and is defined as follows:

```
FILESYSTYPE TYPE(ZFS)
   ENTRYPOINT(IOEFSCM)
   ASNAME(ZFS)
```

zFS procedure

To set up a physical file system in a colony address space, create a cataloged procedure in SYS1.PROCLIB.

Note: The name of the procedure must match the name specified on an ASNAME operand on the FILESYSTYPE statement in BPXPRMxx that starts physical file systems in this colony address space.

7.4 zFS colony address space

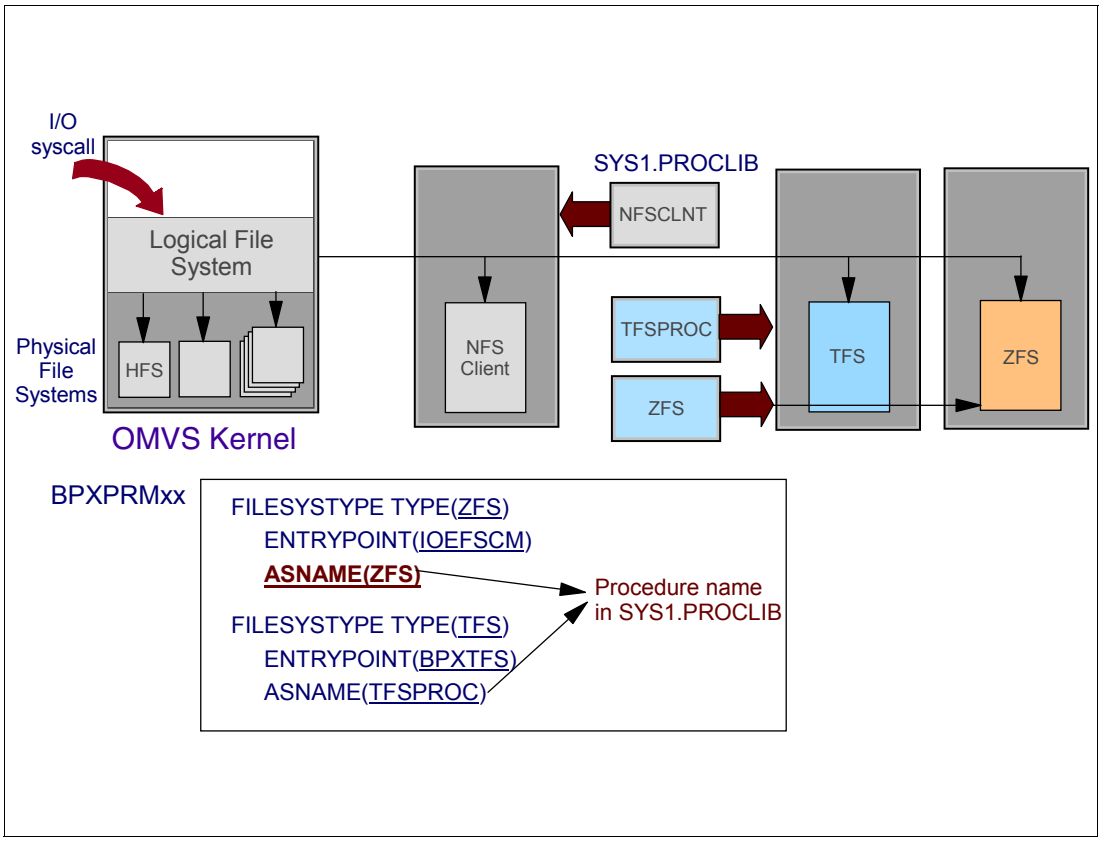


Figure 7-4 zFS colony address space

zFS colony address space

zFS runs in a z/OS UNIX System Services (USS) colony address space. A colony address space is an address space that is separate from the USS address space. HFS runs inside the USS address space and zFS runs in its own address space, as shown in Figure 7-4.

The colony address space runs outside of JES control and does not have to be stopped if JES has to be stopped, which facilitates planned shutdowns of individual systems in a sysplex that has shared file systems.

Other z/OS UNIX colony address spaces

The NFS Client, TFS, and zFS physical file systems support running outside of JES. The following information may help you to decide whether to move these z/OS UNIX colonies outside of JES. The DFS Client PFS does not support being started outside of JES.

Started procedures

z/OS UNIX colony address spaces are started procedures. If you do not want to run them under JES, you will need to change any DD SYSOUT= data sets that are specified in these procedures. These must be changed because SYSOUT data sets are only supported under JES. There are three ways you can change these data sets.

7.5 HFS data sets and zFS data sets

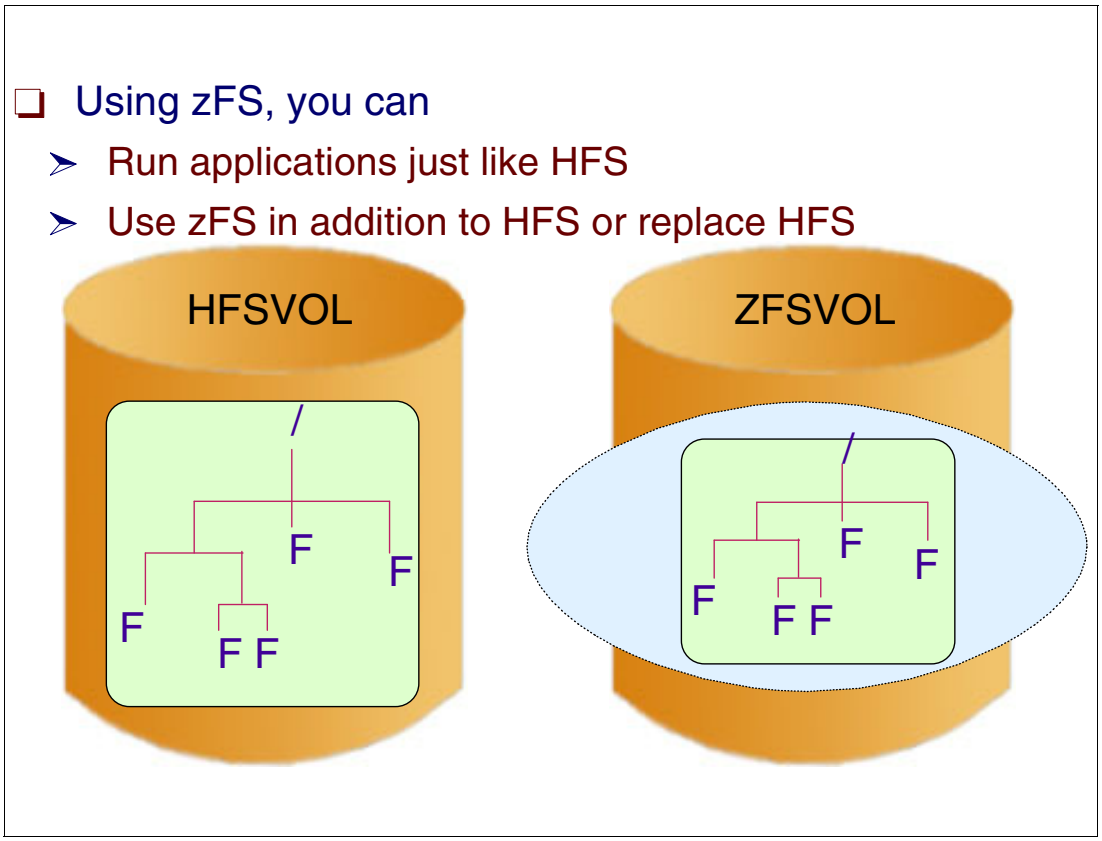


Figure 7-5 HFS data sets and zFS data sets

HFS data sets

A z/OS UNIX hierarchical file system is contained in a data set type called HFS. An HFS data set can reside on an SMS-managed volume or a non SMS-managed volume. HFS data sets can reside with other z/OS data sets on SMS-managed volumes and non SMS-managed volumes. Multiple systems can share an HFS data set if it is mounted in read-only mode.

An HFS data set can have up to 123 extents, and the maximum size of the data set is one physical volume. For a 3390-Model 3, the maximum size is 2.838 GB. HFS data sets only be accessed by z/OS UNIX.

zFS data sets

The z/OS Distributed File Service (DFS) zSeries File System (zFS) is a z/OS UNIX file system that can be used in addition to, or to replace an HFS file system. zFS provides significant performance gains in accessing files approaching 8K in size that are frequently accessed and updated. The access performance of smaller files is equivalent to that of HFS.

zFS file systems contain files and directories that can be accessed with the z/OS hierarchical file system application programming interfaces on the z/OS operating system.

7.6 zFS utilities and commands

- **IOEAGFMT** - utility program to format an aggregate
- **IOEAGSLV** - utility program to scan an aggregate and report inconsistencies
- **IOEZADM** - utility program that allows:
  - zfsadm commands to be issued using JCL
  - Running from within a TSO/E environment
- **zfsadm** - z/OS UNIX shell command
- Installation of zFS consideration for zfsadm commands
  - ln -s /usr/lpp/dfs/global/bin/zfsadm /bin/zfsadm

Figure 7-6 zFS utilities and commands

IOEAGFMT utility

The IOEAGFMT utility is used to format an existing VSAM LDS as a zFS aggregate. All zFS aggregates must be formatted before use (including HFS compatibility mode aggregates). The utility can be run even if the ZFS PFS is not active on the system.

IOEAGSLV utility

This utility scans an aggregate and reports inconsistencies. Aggregates can be verified, recovered (that is, the log is replayed), or salvaged (that is, the aggregate is repaired). This utility is known as the Salvager.

zFS provides a recovery mechanism that uses a zFS file system log to verify or correct the structure of an aggregate. This recovery is invoked by an operator command, **`ioeagslv`**.

IOEZADM utility

This utility program is specified in batch JCL and allows the zFS **`zfsadm`** command to be issued, for example:

```
//USERIDA  JOB ,'zfsadm attach',
//         CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1)
//AGGRINFO EXEC   PGM=IOEZADM,REGION=0M,
// PARM=('attach -aggregate OMVS.PRV.AGGR001.LDS0001')
// ...................
```

The zfsadm command

This command is an administrative command that can be used from the UNIX shell to customize and display the zFS environment.

Note: Although zfsadm and IOEZADM are physically different modules, they contain identical code. Whenever we reference zfsadm as a zFS administration command in this book, we mean both **zfsadm** and IOEZADM. Therefore, IOEZADM can be used as a TSO/E command; it performs the same functions as the **zfsadm** command.

zFS installation consideration

Verify that during your z/OS Distributed File Service installation a symbolic link was created for access to the **zfsadm** commands. If for some reason this was not done, issue the following command to create a symbolic link in the /bin directory:

```
#> ln -s /usr/lpp/dfs/global/bin/zfsadm /bin/zfsadm
```

7.7 zfsadm command

```
zfsadm attach      Attach an aggregate
zfsadm apropos     Display first line of help entry
zfsadm detach      Detach an aggregate
zfsadm grow        Grow an aggregate
zfsadm aggrinfo    Obtain information on an attached aggregate
zfsadm clone       Clone a filesystem
zfsadm clonesys    Clone multiple filesystems
zfsadm create      Create a filesystem
zfsadm delete      Delete a filesystem
zfsadm define      Create a VSAM linear data set aggregate
zfsadm format      Format an aggregate
zfsadm help        Get help on commands
zfsadm lsaggr      List all currently attached aggregates
zfsadm lsfs        List all file systems on an aggregate or all
zfsadm lsquota     List filesystem information
zfsadm quiesce     Quiesce an aggregate and all file systems
zfsadm rename      Rename a file system
zfsadm setquota    Set the quota for a file system
zfsadm unquiesce   Make the aggregate and all file systems available
zfsadm config      Change value of zFS configuration (IOEFSPRM) in memory
zfsadm configquery Query the current value of zFS configuration option
zfsadm query       Query or reset the performance counters
```

Figure 7-7 The **zfsadm** *command and its subcommands*

zfsadm command

The zfsadm commands have the same general structure:

```
command {-option1 argument...| -option2 {argument1 | argument2}..}
[-optional_information]
```

A command consists of the command suite (**zfsadm** in the figure and a subcommand such as **detach**). The command suite and the command name must be separated by a space. The command suite specifies the group of related commands.

zFS provides utility programs and z/OS UNIX commands to assist in the customization of the aggregates and file systems. These utilities and commands are to be used by system administrators.

The **zfsadm** command and the IOEZADM utility program can be used to manage file systems and aggregates. The **zfsadm** command can be run as a UNIX shell command from:

- A z/OS UNIX System Services shell (OMVS) or a (z/OS UNIX) telnet session
- A batch job using the BPXBATCH utility program
- TSO foreground or in batch mode using z/OS UNIX APIs (SYSCALL commands, Callable Services)

zFS file systems can be created using JCL, or by using the **zfsadm** command. Figure 7-7 shows the **zfsadm** command with its subcommands.

7.8 Allocate Linear VSAM data set

- zfsadm command - zfsadm define

```
$> zfsadm define -a OMVS.CMP01.ZFS -volumes totzf1 -cylinders 10 1
IOEZ00248E VSAM linear dataset OMVS.CMP01.ZFS successfully created.
```

- Allocate using IDCAMS in JCL

```
//RFRZAL  JOB (999,POK),'R F',CLASS=A,MSGCLASS=U,NOTIFY=&SYSUID,
//        REGION=0M
//STEP1   EXEC PGM=IDCAMS
//SYSPRINT DD  SYSOUT=*
//DISK     DD  DISP=OLD,UNIT=3390,VOL=SER=ZFSVOL
//SYSIN    DD  *
  DEFINE CLUSTER  -
         (NAME (OMVS.CMP01.ZFS) VOL(ZFSVOL) -
         LINEAR CYL(200 0) SHAREOPTIONS(3))
```

Figure 7-8 How to allocate a linear VSAM data set

Creating a zFS aggregate

A zFS aggregate is created by defining a VSAM linear data set (LDS) and then formatting that VSAM LDS as an aggregate. This is done once for each zFS aggregate. You cannot assign more than one VSAM LDS per aggregate. An aggregate can contain one or more zFS file systems. A zFS file system is equivalent to an HFS file system.

Using JCL

The VSAM LDS is allocated with the VSAM utility program IDCAMS, as shown in Figure 7-8. The JCL shows the allocation of both types of aggregates.

Using the zfsadm command

The **`zfsadm define`** command defines a VSAM LDS. The VSAM LDS is available to be formatted as a zFS aggregate. The command creates a DEFINE CLUSTER command string for a VSAM LDS with SHAREOPTIONS(2) and passes it to the IDCAMS utility. If a failure occurs, the **`zfsadm define`** command may display additional messages from IDCAMS indicating the reason for the failure.

Note: The issuer of the **`zfsadm define`** command requires sufficient authority to create the VSAM LDS.

7.9 Create the aggregate from ISHELL

```
❑ File_systems - Option 4 - New zFS

File  Directory  Special_file  Tools  File_systems  Options  Setup  Help
_
                 Create a zFS Aggregate and File System
C
 Enter the fields as required then press Enter.
E
 Aggregate name  . . . . . ROGERS.TEST.ZFS
 Owning User . . . . . . . ROGERS    (Number or user name)
 Owning Group  . . . . . . SYS1      (Number or group name)
 Permissions . . . . . . . 750   (3 digits, each 0-7)
 Primary cylinders . . . . 20
R Secondary cylinders . . . 10
 Storage class . . . . . . ________
 Management class  . . . . ________
 Data class  . . . . . . . ________
 Volume names  . . . . . . SBOX77  _______  _______  _______  _______
                           _______  _______  _______  _______  _______
                           _______  _______  _______  _______  _______
E

  F1=Help       F3=Exit       F6=Keyshelp  F12=Cancel
```

Figure 7-9 How to allocate a zFS aggregate from the ISHELL

Allocate zFS aggregate from the ISHELL

The panel shown in Figure 7-9, new with z/OS V1R4, allows you to allocate a zFS aggregate from the ISHELL.

Once you enter the ISHELL, place the cursor under File_systems and press Enter. Then select Option **4** (New zFS) and the panel shown in the figure appears. Type in the information to create the aggregate. When you specify the parameters, you now have a linear VSAM data set.

Specify the appropriate parameters to allocate a data set for an aggregate, format the aggregate, and create a file system in that aggregate. The aggregate is initially created as a compatibility mode. The file system defined in the aggregate is the same name as the aggregate and data set. See *z/OS Distributed File Service zSeries File System Administration*, SC24-5989 for detailed information on zFS aggregates, file systems, and their attributes.

Format of aggregate

This ISHELL panel function also formats the aggregate and creates the file system; it just needs to be mounted before it can be used.

7.10 Format VSAM space - create aggregate

- Stand alone utility to format allocated space:
  - IOEAGFMT format utility
- zfsadm command to format the allocated space
  - #> **zfsadm format -a OMVS.CMP01.ZFS -compat -owner 316 -p o755**
- Does not need zFS colony address space to be active
- Does not use the IOEFSPRM configuration file - or IOEPRMxx parmlib member

zfsadm format -aggregate name
[-initialempty blocks] [-size blocks] [-logsize blocks]
[-owner {uid | name}] [-group {gid | name}]
[-perms decimal | octal | hex_number] [-grow blocks]
[-system system name] [-compat] [-overwrite] [-newauditfid] [-level]

Figure 7-10 Formatting a zFS aggregate

Formatting aggregates

The VSAM linear data set must be formatted to be used as a zFS aggregate. Two options are available for formatting an aggregate and they use the same parameters:

- The IOEAGFMT format utility
- The **zfsadm format** command, as follows:

```
zfsadm format -aggregate name
     [-initialempty blocks] [-size blocks] [-logsize blocks]
     [-owner {uid | name}] [-group {gid | name}]
     [-perms decimal | octal | hex_number] [-grow blocks]
     [-system system name] [-compat] [-overwrite] [-newauditfid] [-level]#>

zfsadm format -a OMVS.CMP01.ZFS -compat -owner 316 -p o755
IOEZ00077I HFS-compatibility aggregate OMVS.CMP01.ZFS has been successfully
created
```

The formatting of aggregates can be done without the colony address space (the zFS address space) being active. The IOEFSPRM file, or the BPXPRMxx PARMLIB member, are not required for the format to take place.

Creating aggregates and file systems

A zFS file system is created in a zFS aggregate (which is a VSAM linear data set). When using compatibility mode aggregates, the aggregate and the file system are created at the same time during the format.

Formatting space

This utility formats the primary allocation and, if requested by using the `-size` parameter (with a value greater than the primary allocation), a single *extension*. An extension is a single call to the Media Manager to extend the data set to the size specified in the `-size` parameter. It is completely independent of the number of cylinders specified in the secondary allocation when the data set was defined (the secondary allocation could have been 0).

After you have allocated the space for an aggregate, the default size is the number of 8 K blocks that fits into the primary allocation. You can specify a **`-size`** option giving the number of 8 K blocks for the aggregate. If you specify a number that is less than (or equal to) the number of blocks that fit into the primary allocation, the primary allocation size is used. If you specify a number that is larger than the number of 8 K blocks that fit into the primary allocation, the VSAM LDS is extended to the size specified. This occurs during its initial formatting.

When an aggregate is initially formatted using the IOEAGFMT format utility or the **`zfsadm format`** command, the formatting takes place as follows:

- The default size formatted is the number of blocks that will fit into the primary allocation.
- Using the **`-size`** parameter, if the number of blocks to be formatted is less than the default, it is rounded up to the default.
- If a number greater than the default is specified, a single extent of the VSAM LDS is attempted after the primary allocation is formatted.

7.11 Format the aggregate

- JCL can be used to format aggregates
  - PARM parameters must be lower case
  - -compat required for compatibility aggregates

A compatibility mode aggregate is formatted with this JCL:

```
//ROGERSA  JOB (999,POK),'R F',CLASS=A,MSGCLASS=U,NOTIFY=&SYSUID,
// REGION=0M
//STEP1   EXEC PGM=IOEAGFMT,
//          PARM=(' -aggregate ROGERS.AAA.ZFS -compat ')
//SYSPRINT DD  SYSOUT=*
//STDOUT   DD  SYSOUT=*
//STDERR   DD  SYSOUT=*
//CEEDUMP  DD  SYSOUT=*
//
```

Figure 7-11 JCL to format an aggregate

Formatting aggregates using JCL

This is a batch utility that formats a VSAM Linear Data Set to become an HFS compatibility mode aggregate or a multifile system aggregate.

Figure 7-11 shows the JCL for formatting compatibility mode aggregates.

The following parameters are possible:

-aggregate name Specifies the name of the data set to format. This is also the aggregate name. The aggregate name is always translated to upper case.

-compat Indicates that a compatibility mode aggregate should be created. This means that in addition to formatting the VSAM LDS as a zFS aggregate, a zFS file system by the same name (the aggregate name) is created and its quota is set to the size of the available blocks on the aggregate.

-group gid | name Specifies the group owner for the root directory of the file system. This is used with the -compat option; otherwise it is ignored. It can be specified as a z/OS group name or as a gid. The default is the gid of the issuer of ioeagfmt. If only the -owner name is specified, the group is that owner's default group. If only the -owner uid is specified, the group is the issuer's group.

-grow blocks Specifies the number of 8 K blocks that zFS will use as the increment for extension when the -size option specifies a size greater than the primary allocation.

-help Prints the online help for this command. All other valid options specified with this option are ignored.

-initialempty blocks Specifies the number of 8 K blocks that will be left empty at the beginning of the aggregate. The default is 1. If you specify 0, you will get 1 block. This option is not normally specified.

-level Prints the level of the **ioeagfmt** command. This is useful when you are diagnosing a problem. All other valid options specified with -level are ignored.

-logsize blocks Specifies the size in 8 K blocks of the log. The default is 1% of the aggregate size. However, the logsize will never be smaller than 14 blocks and it will never be larger than 16,384 blocks (128 megabytes). This is normally sufficient. However, a small aggregate that is grown to be very large will still have a small log. You might want to specify a larger log if you expect the aggregate to grow very large.

-newauditfid Specifies that the aggregate should be formatted with the zFS auditfid and stored in the aggregate.

-overwrite Required if you are reformatting an existing aggregate. Use this option with caution; it destroys any existing data. This option is not usually specified.

-owner uid | name Specifies the owner for the root directory of the file system. This is used with the -compat option; otherwise it is ignored. It can be specified as a z/OS user ID or as a uid. The default is the uid of the issuer of **ioeagfmt**.

-perms number Specifies the permissions for the root directory of the file system. This is used with the -compat option, otherwise it is ignored. The number can be specified as octal (for example, o755), as hexadecimal (for example, x1ED), or as decimal (for example, 493). The default is o755 (owner read-write-execute, group read-execute, other read-execute).

-size blocks Specifies the number of 8 K blocks that should be formatted to form the zFS aggregate. The default is the number of blocks that will fit in the primary allocation of the VSAM Linear Data Set (LDS). If a number less than the default is specified, it is rounded up to the default. If a number greater than the default is specified, a single extend of the VSAM LDS is attempted after the primary allocation is formatted unless the -grow option is specified. In that case, multiple extensions of the amount specified in the -grow option will be attempted until the -size is satisfied. The size can be rounded up to a control area (CA) boundary by DFSMS. It is not necessary to specify a secondary allocation size on the DEFINE of the VSAM LDS for this extension to occur. Space must be available on the volume.

7.12 Ioeagfmt successful messages

- The following messages are displayed in the SYSPRINT data set of the JCL job
- To use the file system, it must be mounted

```
IOEZ00004I Formatting to 8K block number 1800 for primary extent of ROGERS.AAA.ZFS.
IOEZ00005I Primary extent loaded successfully for ROGERS.AAA.ZFS.
IOEZ00535I *** Using initialempty value of 1.
*** Using 1799 (8192-byte) blocks
*** Defaulting to 17 log blocks(maximum of 1 concurrent transactions).
IOEZ00327I Done.  ROGERS.AAA.ZFS is now a zFS aggregate.
IOEZ00048I Detaching aggregate ROGERS.AAA.ZFS
IOEZ00071I Attaching aggregate ROGERS.AAA.ZFS to create HFS-compatible file system
IOEZ00074I Creating file system of size 14207K, owner id 0, group id 2, permissions
IOEZ00048I Detaching aggregate ROGERS.AAA.ZFS
IOEZ00077I HFS-compatibility aggregate ROGERS.AAA.ZFS has been successfully created
```

Figure 7-12 zFS formatting messages

Formatting messages

When you format a compatibility mode aggregate (output of the JCL shown in the figure), the following processing is done:

- The aggregate is attached and then detached.
- A file system is created for the aggregate with the same name as the aggregate.

The ioeagfmt utility

The **ioeagfmt** utility is used to format an existing VSAM LDS as a zFS aggregate. All zFS aggregates must be formatted before use (including HFS compatibility mode aggregates). You can run **ioeagfmt** even if the zFS PFS is not active on the system. The size of the aggregate is as many 8 K blocks as fit in the primary allocation of the VSAM LDS or as specified in the -size option. The -size option can cause one additional extension to occur during formatting. To extend it further, use the **zfsadm grow** command. If -overwrite is specified, all existing primary and secondary allocations are formatted and the size includes all of that space. If the VSAM LDS has a SHAREOPTIONS value of other than 3, **ioeagfmt** will change it to SHAREOPTIONS 3 during format.

Using a batch job

For a batch job, the **ioeagfmt** options are specified in the EXEC PARM as a single subparameter (a single character string enclosed in apostrophes with no commas separating the options). You cannot put the ending apostrophe in column 72. If it needs to go to the next line, use a continuation character in column 72 (continuing in column 16 with the ending apostrophe on the second line). Remember that a JCL EXEC PARM is limited to 100 characters.

Privilege required

The user must have ALTER authority to the VSAM LDS or must be UID 0 or have READ authority to the SUPERUSER.FILESYS.PFSCTL profile in the z/OS UNIXPRIV class. In fact, UPDATE authority to the VSAM LDS is sufficient for format, but zFS will not be able to set the zFS bit in the catalog unless the issuer has ALTER authority.

If you are changing the owner or group to something other than the issuer or you are changing the permissions to other than the default, you need UID 0 or READ authority to the SUPERUSER.FILESYS.PFSCTL profile in the z/OS UNIXPRIV class.

7.13 Grow an aggregate

- Aggregates can be grown via a z/OS UNIX command:
  - **zfsadm grow** command
- Grow the size of an aggregate
  - Size 0 - Indicates to use secondary extent allocation

```
#> zfsadm grow -aggregate OMVS.CMP02.ZFS -size 0
IOEZ00173I Aggregate OMVS.CMP02.ZFS successfully grown
OMVS.CMP02.ZFS (R/W COMP): 61598 K free out of total 62072 (2000 reserved)
```

Figure 7-13 How to grow the size of an aggregate

Grow an aggregate

The format utility formats the primary allocation and, if requested by using the **-size** parameter (with a value greater than the primary allocation), a single extension. An extension is a single call to the Media Manager to extend the data set to the size specified in the **-size** parameter. It is completely independent of the number of cylinders specified in the secondary allocation when the data set was defined (the secondary allocation could have been 0).

If a compatibility mode aggregate becomes full, the administrator can grow the aggregate (that is, cause an additional allocation to occur and format it to be part of the aggregate). This is accomplished with the **zfsadm grow** command. There must be space available on the volume(s) to extend the aggregate's VSAM linear data set. The size specified on the **zfsadm grow** command must be larger than the current size of the aggregate. In the example in Figure 7-13, we used **-size 0**. Specifying a size of 0 indicates to use the secondary allocation.

Grow example

For example, suppose a two-cylinder (primary allocation, 3390) aggregate has a total of 180 8K blocks and a (potential) secondary allocation of one cylinder. 180 8K blocks is 1440K bytes. A **zfsadm aggrinfo** command for this aggregate might show 1440K. **zfsadm grow** does this by calling DFSMS to allocate the additional DASD space. DFSMS requires a number of reserved blocks. Therefore, specify a few blocks larger than the current size before an allocation occurs.

7.14 Dynamic aggregate extension

- Grow an aggregates dynamically without commands
  - VSAM LDS must have a secondary allocation + space
- Ways to specify aggregate extension
  - **zfsadm config** command
    - aggrgrow on | off
  - An option in the IOEFSPRM file
    - **aggrgrow=on|off**
  - mount command - parm('aggrgrow')
    - mount filesystem('omvs.test.zfs') mountpoint('/tmp/test') type(zfs) mode(rdwr) parm('aggrgrow')
- Aggregate is extended using a secondary allocation
  - Secondary allocation is formatted becomes available

Figure 7-14 Growing the aggregate size dynamically

Growing an aggregate dynamically

If aggregates become full they can be grown by using the **zfsadm grow** command. You need to specify a larger size or specify zero for the size to get a secondary allocation size extension.

You have the possibility of dynamically growing an aggregate if it becomes full. The aggregate is extended automatically when an operation cannot complete because the aggregate is full.

Important: To dynamically grow an aggregate when it becomes full, the VSAM LDS must have a secondary allocation and have space on the volumes.

Specifying the dynamic aggregate grow option

An administrator can specify that an aggregate should be dynamically grown if it becomes full. This is specified by the **-aggrgrow** option on the **zfsadm attach** command, or the `aggrgrow` suboption of the `define_aggr` option of the IOEFSPRM file, or globally by the `aggrgrow` option of the IOEFSPRM file. The aggregate (that is, the VSAM Linear Data Set) must have secondary allocation specified when it is defined, and space must be available on the volume(s). The aggregate will be extended when an operation cannot complete because the aggregate is full. If the extension is successful, the operation is redriven transparent to the application.

Ways to specify the dynamic option

Dynamic aggregate extension can be enabled in the following ways:

- In the IOEFSPRM configuration file, you can dynamically extend an aggregate when it becomes full by specifying one of the following options:
  - Specify a new option, `aggrgrow=on | off`. The default value is off.

    Note: The option specified here is the default if none of the following ways of specifying the `aggrgrow | noaggrgrow` options are used.

- Using the **mount** command, in the PARM keyword you can specify either `aggrgrow` or `noaggrgrow`, as shown in the following example:

```
mount filesystem('omvs.test.zfs') mountpoint('/tmp/test') type(zfs) mode
(rdwr) parm('aggrgrow')
```

Note: This `aggrgrow | noaggrgrow` option can only be used with compatibility mode aggregates.

Example of the dynamic grow option

When an aggregate fills and dynamic aggregate extension has been specified using one of the options, the aggregate is extended using secondary allocation extensions, and the extensions taken are formatted and become available transparently to the application. The messages issued indicating the process are as follows:

```
IOEZ00312I Dynamic growth of aggregate OMVS.TEST.ZFS in progress, (by user
AYVIVAR).
IOEZ00329I Attempting to extend OMVS.TEST.ZFS by a secondary extent.
IOEZ00324I Formatting to 8K block number 360 for secondary extents of
OMVS.TEST.ZFS
IOEZ00309I Aggregate OMVS.TEST.ZFS successfully dynamically grown (by user
AYVIVAR).
```

7.15 The -grow option

- When an aggregate is initially formatted using IOEAGFMT or the zfsadm format command
  - Size, -size specified must be able to be allocated in the primary allocation and one extension
- If you wanted to format a 3-volume aggregate with IOEAGFMT - not possible
  - Requires a primary and at least two extensions
- Need to use the **zfsadm format** command to specify:
  - zfsadm format -size 900720 -grow 300240

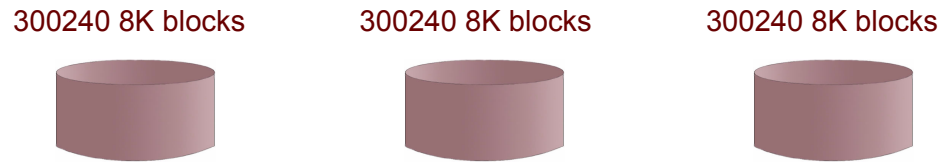


Figure 7-15 Specifying the -grow option

A -grow example

Since the **-size** parameter specified can only be allocated in the primary allocation and one extension, and you wanted to format a 3-volume aggregate with the IOEAGFMT utility—this was not possible because it requires a primary and at least two extensions. However, a new **-grow** option is provided with z/OS V1R4 for the IOEAGFMT utility and the **zfsadm format** command to allow specification of the increment that can be used for extension of the aggregate when **-size** is larger than the primary allocation. This allows the extension by the **-grow** amount until **-size** is satisfied.

-grow Specifies the number of 8K blocks that zFS uses as the increment for an extension when the **-size** option specifies a size greater than the primary allocation.

Other examples of -grow

To illustrate the before and after using the **-grow** option, the following example has a VSAM LDS defined with 2 cylinders of primary space and 1 cylinder of secondary space.

```
//AYVIVAR2 JOB CLASS=J,MSGCLASS=A,NOTIFY=AYVIVAR
/*JOBPARM S=SC65
//P010 EXEC PGM=IDCAMS
//SYSPRINT  DD SYSOUT=*
//SYSIN DD *
  DEFINE CLUSTER(NAME(OMVS.TESTA.ZFS) VOLUMES(SBOX43) -
  LINEAR CYL(2,1) SHAREOPTIONS(2))
//
```

The file created has two extents: the first one corresponds to the primary space specified in the define process (30 tracks), and a second one with 30 tracks.

With the new **-grow** parameter, you are allowed to specify the increment that will be used for an extension size larger than the primary allocation. That is, after the primary space is allocated, multiple extensions of the amount specified by the **-grow** parameter rounded up to a multiple of the secondary space defined will be attempted until the total number of blocks specified by the **-size** parameter is satisfied.

Replacing the example shown previously, use the **-grow** parameter on the format process, as follows:

```
//AYVIVAR2 JOB CLASS=J,MSGCLASS=V,NOTIFY=AYVIVAR
/*JOBPARM S=SC65
//FORMAT EXEC PGM=IOEAGFMT,REGION=0M,
//       PARM=('-aggregate OMVS.TESTA.ZFS -size 276 -grow 90  -compat')
//SYSPRINT DD SYSOUT=*
//STDOUT   DD SYSOUT=*
//STDERR   DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//CEEDUMP  DD SYSOUT=*
```

Now the VSAM LDS has three extensions, the first corresponding to the primary space specified and the next two by the **-grow** amount.

Note: The default for the size of the aggregate is the number of 8K blocks that fit in the primary allocation. You can specify a **-size** option giving the number of 8K blocks for the aggregate. If you specify a number that is less than (or equal to) the number of blocks that fit into the primary allocation, the primary allocation size is used. If you specify a number that is larger than the number of 8K blocks that fit into the primary allocation, the VSAM LDS is extended to the size specified if the total size will fit in the primary allocation and a single extension. The single extension must be no larger than a single volume. This occurs during its initial formatting.

7.16 -grow option for formatting

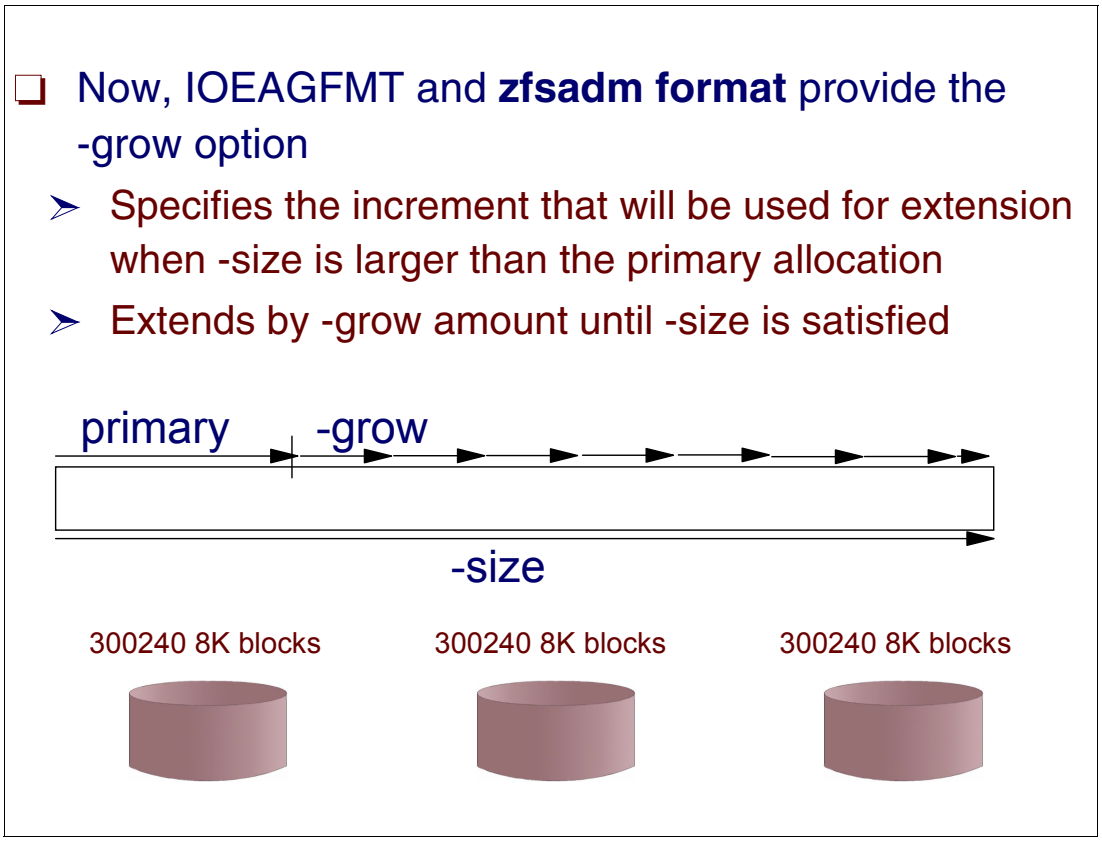


Figure 7-16 The -grow option for formatting

The -grow option for format

There are many allocations of VSAM linear data sets where you may want to format the entire extent during the initial format. The **-grow** option allows you to specify that you want to do this during the format of the aggregate.

Use the **-grow** option to specify the size of a secondary allocation. This causes the format routine to continue formatting the specified allocation until the entire allocation is completely formatted.

The -size option

To format the entire allocation, use the **-size** option, together with the **-grow** option, as follows:

```
zfsadm format -size 900720 -grow 300240
```

The **-size** option specifies the number of 8K blocks that should be formatted to form the zFS aggregate. The default is the number of blocks that will fit in the primary allocation of the VSAM LDS. If a number less than the default is specified, it is rounded up to the default. If a number greater than the default is specified, a single extend of the VSAM LDS is attempted after the primary allocation is formatted unless the **-grow** option is specified. In that case, multiple extensions of the amount specified in the **-grow** option will be attempted until the **-size** is satisfied. The size may be rounded up to a control area (CA) boundary by DFSMS. It is not necessary to specify a secondary allocation size on DEFINE of the VSAM LDS for this extension to occur. Space must be available on the volumes.

7.17 Mounting the file system

- Once a compatibility aggregate is formatted
  - The file system is created
  - The aggregate is not attached
  - When mounting the file system
    - The aggregate is attached

Note: Do not manually attach a compatibility mode aggregate

Figure 7-17 Mounting a file system

Mounting a file system

The zFS file system can now be mounted into the z/OS UNIX hierarchy. This can be accomplished in the following ways:

- With the TSO/E MOUNT command
- Using an OMVS shell **`mount`** command
- Using the ISHELL
- A mount statement in the BPXPRMxx PARMLIB member
- An automount policy

This assumes that the directory mountpoint exists. The TYPE parameter of the MOUNT command specifies ZFS. This is required for any zFS file system. All other forms of the mount function are also supported (for example, the **`/usr/sbin/mount`** command, the automount facility, etc.). Once the zFS file system is mounted, applications and commands can be executed and files and directories can be accessed in zFS just as in HFS.

Note: When the file system is mounted, it causes the aggregate to be attached.

7.18 ISHELL support for zFS

```
ISHELL provides ability to manage  zFS aggregates

  File  Directory  Special_file  Tools  File_systems  Options  Setup  Help
                                                                 ________
                        UNIX System Serv  5  1. Mount table...
Command ===> ________________________________   2. New HFS...        _______
                                              3. Mount(O)...
Enter a pathname and do one of these:         4. New ZFS...
                                              5. ZFS aggregates...
    - Press Enter.
    - Select an action bar choice.
    - Specify an action code or command on the command line.

Return to this panel to work with a different pathname.
                                                                More:     +
   /u
   ______________________________________________________________________
   ______________________________________________________________________
   ______________________________________________________________________

EUID=0
```

Figure 7-18 Managing zFS aggregates from the ISHELL

Manage zFS aggregates

With the ISPF shell (ISHELL), a user or system programmer can use ISPF dialogs instead of shell commands to perform many tasks, especially those related to file systems and files. An ordinary user can use the ISPF shell to work with zFS file system aggregates.

Use the ISHELL command in the ISPF Option 6 Menu to invoke the ISPF shell, which is a panel interface that helps you to set up and manage z/OS UNIX System Services functions.

In the pull-down under File_systems is the new option 5. To make use of it, select **Option 5** in the File_systems menu, and press Enter; the panel shown in Figure 7-19 on page 340 is displayed. This panel enables you to work with all the attached aggregates from the ISHELL.

Similar information about the aggregates can also be obtained with the **`zfsadm`** command from the OMVS shell session.

7.19 Panel of attached zFS aggregates

```
                         Attached zFS Aggregates           Row 1 to 20 of 10:

Select an aggregate with a line command or select an option.
A=Attributes  L=List file systems  D=Detach  E=Extend

Option:    _  1. Attach aggregate          2. Create aggregate

S   Aggregate Name                                  Free Space  Total Space
_   BBRR7064.SBBOHFS                                    297303      2376000
_   BB16164.SBBOHFS                                     159201      1699200
_   BB26163.SBBOHFS                                     159201      1699200
_   BB36163.SBBOHFS                                     310246      2347200
_   HFS.ZOSR1A.Z1ARA1.JAVA31V5                           23670       292320
_   HFS.ZOSR1A.Z1ARA1.JAVA64V5                           22518       276480
_   HFS.ZOSR1A.Z1ARA1.JV390                              20195       220320
_   HFS.ZOSR1A.Z1ARA1.ROOT                               27174      2517840
_   HFS.ZOSR1A.Z1ARA1.SBBN7ZFS                           92824      1728000
_   HFS.ZOSR1A.Z1ARA1.SIZUZFS                            13025        57600
_   HFS.ZOSR1A.Z1ARA1.XML                               102641      1314720
_   HFS.ZOSR1B.Z1BRA1.JAVA31V5                           11729       283680
_   HFS.ZOSR1B.Z1BRA1.JAVA31V6                           18804       408240
_   HFS.ZOSR1B.Z1BRA1.JAVA64V5                           11377       268560
_   HFS.ZOSR1B.Z1BRA1.JAVA64V6                           20097       455760
_   HFS.ZOSR1B.Z1BRA1.ROOT                              343945      2863440
```

Figure 7-19 ISHELL panel showing attached zFS aggregates

Attached zFS aggregate ISHELL panel

This panel shows the aggregate list of all zFS aggregates that are currently attached explicitly or implicitly as compatibility aggregates. The amount of free space and total space is also shown, in units of kilobytes. From this panel you can select an option of one or more aggregates with an action code.

The valid action codes are:

A Show the attributes for the aggregate.
L List the file systems in the aggregate. The file system list will allow you to perform actions on file systems.
D Detach an aggregate that is not in use.
E Extend the size of the aggregate.

The available options are:

1. **Attach aggregate**: Select this option to specify an aggregate to attach.
2. **Create aggregate**: Select this option to create an HFS-compatible aggregate. Note that an HFS-compatible aggregate also contains a file system with the same name as the aggregate. Additional file systems can be created in that aggregate if it has been explicitly attached, and the original file system can also be deleted.

7.20 Display aggregate attributes

```
_                        Attached zFS Aggregates            Row 1 to 20 of 103
 S                           Aggregate Attributes
 A
    Aggregate name  . . . . : BBRR7064.SBBOHFS
 O  Attach mode . . . . . . : Read/write
    Monitored for full  . . : Disabled
 S  New block security  . . : Enabled
 A  HFS compatibility . . . : Enabled
 _  Auto-extend . . . . . . : Enabled
 _  Number of file systems  :          1
 _  Threshold . . . . . . . :          0
 _  Increment . . . . . . . :          0
 _  Number of fragments . . :    2376000
 _  Fragment size . . . . . :       1024
 _  Block size  . . . . . . :       8192
 _  Blocks available  . . . :    2376000
 _  Maximum fragments . . . :     297303
 _  Minimum fragments . . . :          0
 _
 _
 _   F1=Help       F3=Exit       F6=Keyshelp  F12=Cancel
 _
 _  HFS.ZOSR1B.Z1BRA1.ROOT                          343945      2863440
 _  HFS.ZOSR1B.Z1BRA1.XML                           176954      1791360
```

Figure 7-20 Display of an aggregate's attributes

Display attributes for an aggregate

Figure 7-20 shows the screen returned when you specify **A** - Attributes for an aggregate from the previous screen and press Enter.

For detailed information about zFS aggregates, file systems, and their attributes, see *z/OS Distributed File Service zSeries File System Administration*, SC24-5989.

The zfsadm command

If you use the **`zfsadm aggrinfo`** command to display the aggregate information to list the aggregate attributes, the result is as follows:

```
ROGERS @ SC70:/u/rogers>zfsadm aggrinfo BBRR7064.SBBOHFS -long
BBRR7064.SBBOHFS (R/W COMP): 297303 K free out of total 2376000
version 1.4
auditfid E2C2D6E7 F1D8009F 0000

         31627 free 8k blocks;      44287 free 1K fragments
         23776 K log file;             64 K filesystem table
           336 K bitmap file
```

7.21 Display attached aggregates

```
ROGERS @ SC70:/u/rogers>zfsadm lsaggr
IOEZ00106I A total of 103 aggregates are attached
OMVS.DB2V9.SDAHHFS1.D070806                    SC63      R/W
OMVS.DB2V9.SDAHHFS1.D090309                    SC63      R/W
OMVS.RADTOOL7.HFS                              SC63      R/W
ZFSFR.ZFSA.ZFS                                 SC65      R/W
ZFSFR.ZFSB.ZFS                                 SC65      R/W
OMVS.DB2V9.SDSNWORF.D071020                    SC63      R/W
ZFSFR.ZFSC.ZFS                                 SC65      R/W
ZFSFR.ZFSD.ZFS                                 SC65      R/W
ZFSFR.ZFSE.ZFS                                 SC65      R/W
ZFSFR.ZFSF.ZFS                                 SC65      R/W
ZFSFR.ZFSG.ZFS                                 SC65      R/W
OMVS.TWS820.TWSCE2EW.HFS                       SC63      R/W
OMVS.TWS820.TWSCTPRK.HFS                       SC63      R/W
OMVS.DB2V9.SDSNWORF.D081006                    SC63      R/W
OMVS.DB2V9.SDSNWORF.D080701                    SC63      R/W
OMVS.DB2V9.SDSNMQLS.D071020                    SC63      R/W
 ===>
```

Figure 7-21 Displaying attached aggregates

Displaying attached aggregates

The **`zfsadm lsaggr`** command lists all currently attached aggregates for zFS.

This command displays a separate line for each aggregate. Each line displays the following information:

- The aggregate name
- The system name where the aggregate is attached
- The access mode - R/W is Read/Write

You can use the **`zfsadm aggrinfo`** command to display information about the amount of disk space available on a specific aggregate or on all aggregates on a system.

7.22 List file systems

```
ROGERS @ SC65:/u/rogers> zfsadm lsfs -a ROGERS.HARRY.ZFS
IOEZ00129I Total of 1 file systems found for aggregate ROGERS.HARRY.ZFS
ROGERS.HARRY.ZFS         RW (Mounted R/W)       9 K alloc      9 K quota On-line
Total file systems on-line 1; total off-line 0; total busy 0; total mounted 1
ROGERS @ SC65:/u/rogers>
 ===>
```

The -fast option just lists the aggregate names -
For compat mode aggregates - same as file system name

```
ROGERS @ SC65:/u/rogers> zfsadm lsfs -fast
IOEZ00369I A total of 5 aggregates are attached to the sysplex.
ROGERS.HARRY.ZFS
TWS.TWSAWRK.RVA.ZFS
OMVS.HERING.TEST.ZFS
ROGERS.TEST.ZFS
ROGERS.AAA.ZFS
ROGERS @ SC65:/u/rogers>
 ===>
```

Figure 7-22 Displaying zFS file systems

Displaying file systems

The **zfsadm lsfs** command lists all the file systems on a given aggregate or all attached aggregates.

Using the **-a** option and specifying the file system name is shown in Figure 7-22. If you specify an aggregate name that is used to retrieve file system information, the aggregate name is not case sensitive and is always translated to uppercase. If this option is not specified, the command displays information for all attached aggregates.

The **-fast** option causes the output of the command to be shortened to display only the aggregate name if it contains one or more file systems, or a message indicating that there are no file systems contained in the aggregate.

7.23 zFS aggregate space commands

- AGGRINFO - Displays information about an aggregate
- LSQUOTA - Shows quota information about file systems and aggregates
- Space usage on all aggregates
  - zfsadm aggrinfo
- zfsadm aggrinfo shows aggregate disk space usage
  - Based on the number of 8K blocks
  - Subtracts the space reserved for the 3 objects

```
ROGERS @ SC65:/u/rogers>zfsadm aggrinfo ZFSFR.ZFSG.ZFS
ZFSFR.ZFSG.ZFS (R/W COMP): 6374 K free out of total 7200
ROGERS @ SC65:/u/rogers>zfsadm lsquota ZFSFR.ZFSG.ZFS
Filesys Name          Quota    Used  Percent Used  Aggregate
ZFSFR.ZFSG.ZFS         7047     673     9    11 = 826/7200 (zFS)
```

Figure 7-23 Aggregate space commands

zFS commands to determine aggregate space

Each file system has a quota represented in 1 KB fragments. The quota of a file system is a logical number and can be smaller or larger than the size of the disk (if the size of the disk were expressed in 1 KB fragments).

The **`zfsadm lsquota`** command shows the quota in 1 KB units and also shows the aggregate size and usage in 1 KB units (it shows the amount of space used for the three special objects in Figure 7-23 also).

A zFS aggregate is an array of 8 K blocks. Three special objects are present in all zFS aggregates. These objects take up space in an aggregate, which means that space cannot be used for user files:

- Log file - Records metadata changes. By default, its size is 1% of the disk size. However, it will never be smaller than 14 blocks and it will never be larger than 16,384 blocks (128 MB).
- Bitmap - Lists the blocks that are free on disk. The file size depends on the size of the aggregate.
- Aggregate File System List - Describes the file systems contained in the aggregate. For compatibility mode aggregates it is usually only one 8 KB block. For multifile system aggregates, its size depends on how many file systems are in the aggregate.

For compatibility mode aggregates the file system quota is set to be the following size:

Total disk size (in 1 KB units) - size of the above three special objects (in 1 KB units)

zFS aggregate disk usage

zFS stores files on disk in one of three ways:

- Inline - If the file is 52 bytes or less, it is stored in the same data structure on disk that holds the file status (things such as owner, size, and permissions). A file of 52 bytes or less takes no extra disk space.
- Fragmented - If the file is 7 KB or less and has never been larger than 7 KB, it is stored in 1 KB fragments (hence it is stored in part of an 8 KB block). Multiple small files can share the same 8 KB block on disk.
- Blocked - If the file is over 7 KB, it is stored as an array of 8 KB blocks.

Aggregate File System List - This describes the file systems contained in the aggregate. For compatibility mode aggregates it is usually only one 8 KB block. For multifile system aggregates, its size depends on how many file systems there are.

The aggrinfo subcommand

The **zfsadm aggrinfo** command displays information about an aggregate, or all attached aggregates, if there is no specific aggregate specified. This is based on the number of 8 KB blocks. The command shows output in units of 1 KB blocks.

The **zfsadm aggrinfo** command shows aggregate disk space usage. This is based on the number of 8 KB blocks. It subtracts the space reserved for the above three objects in its calculations (and tells you this in the output). The command shows output in units of 1 KB blocks.

```
ROGERS @ SC65:/u/rogers>zfsadm aggrinfo
IOEZ00368I A total of 1 aggregates are attached.
ZFSFR.ROOT.ZFS (R/W COMP): 292908 K free out of total 1440000
```

The -long option

If you use the **-long** option of the **zfsadm aggrinfo** command, it shows the number of free 8 K blocks, the number of free 1 K fragments and the size (in K) taken up by the log file, the file system table, and the bitmap, as follows:

```
PAUL @ SC65:/>zfsadm aggrinfo ZFSFR.ZFSG.ZFS -long
ZFSFR.ZFSG.ZFS (R/W COMP): 6374 K free out of total 7200
version 1.4
749 free 8k blocks;        382 free 1K fragments
           112 K log file;             24 K filesystem table
```

The lsquota subcommand

The **zfsadm lsquota** command displays quota and usage information about a file system. The command also provides usage information about the aggregate in which the file system resides. The file system does not need to be mounted to use this command. The aggregate containing the file system must be attached.

The **zfsadm lsquota** command displays the name of the file system, the quota and the quota used (in kilobytes) of the file system, and the percentage of the quota in use. It also displays the information about the percentage of the aggregate in use, the number of kilobytes in use on the aggregate and the number of available kilobytes on the aggregate in which the file system resides. It also reports that the file system is zFS.

7.24 Command for aggregate display

```
F ZFS,AGGRINFO,ROGERS.LARGE.ZFS
 IOEZ00438I Starting Query Command AGGRINFO.
 Total 8K blocks on aggregate: 70470 (563760K bytes)
 Number of 8K blocks used for log file: 705 (indirect blocks 1)
 Number of 8K blocks reserved for cross-system serialization: 1
 Number of 8K blocks used for filesystem table: 6
 Number of 1K fragments used for bad-block file: 1
 Number of 8K blocks used for bitmap file: 11
 Number of 8K blocks free for use on aggregate: 5711
 Number of free 1K fragments available for use on aggregate: 14

 Filesystem inode table 8K quota limit 557975K used 512273K
 name=ROGERS.LARGE.ZFS

 IOEZ00025I zFS kernel: MODIFY command - AGGRINFO,ROGERS.LARGE.ZFS
 completed successfully
```

Figure 7-24 Display aggregate usage for the specified file system

zFS command to display aggregate usage

The **F ZFS** command enables you to query internal zFS counters and values. They are displayed on the system log. It also allows you to initiate or gather debugging information. The zFS PFS must be running to use this command.

The F ZFS,AGGRINFO command

Using this command, you can display information about specific aggregates.

The IOEZ00438I message just indicates that AGGRINFO is the keyword for running a query command.

This command provides a detailed breakdown of the space utilization in the zFS aggregate that is specified containing a file system.

7.25 zFS threshold monitoring space usage

- zFS threshold monitoring function aggrfull reports:
  - Space usage based on total aggregate disk size
  - Has the space for the three special objects - total disk space and amount used on disk in its messages
  - The aggrfull message shows units in 8K blocks

```
ROGERS @ SC65:/u/rogers>zfsadm config -aggrfull "(80,5)"
IOEZ00300I Successfully set -aggrfull to (80,5)
ROGERS @ SC65:/u/rogers>zfsadm configquery -aggrfull
IOEZ00317I The value for configuration option -aggrfull is
(80,5).
```

Number of 8K blocks in the total aggregate

Number of 8K blocks used in the aggregate

```
IOEZ00078E zFS aggregate Name exceeds Threshold% full (blocks1/blocks2)(WARNING)
```

Figure 7-25 Monitoring aggregate space usage

zFS monitoring aggregate space usage

The zFS threshold monitoring function **aggrfull** reports space usage based on total aggregate disk size. It incorporates the space for the above three special objects when showing total disk space and the amount used on disk in its messages. The **aggrfull** message shows units in 8 K blocks.

The aggrfull parameter

The **aggrfull** parameter specifies the threshold and increment for reporting aggregate full error messages to the operator. The default is the **aggrfull** specification in the IOEFSPRM file.This parameter only applies to compatibility mode aggregates or file systems.

The following message is issued to the operator when issuing the following command:

```
zfsadm configquery -aggrfull
IOEZ00078E zFS aggregate Name exceeds Threshold% full (blocks1/blocks2)
(WARNING)
```

The message indicates that a zFS aggregate used space has exceeded the administrator-defined threshold specified on the **aggrfull** option. The numbers in parentheses are the number of 8 K blocks used in the aggregate and the number of 8 K blocks in the total aggregate, respectively.

7.26 Dynamic configuration parameters

- Commands to display and modify parms
  - The zfsadm configquery command queries the current value of zFS configuration options
  - The zfsadm config command changes the value of zFS configuration (IOEFSPRM) options in memory
- The commands require the following access authority and zfsadm requires superuser authority

Command	Command description	IOEFSPRM	SU mode
zfsadm config	Modify current configuration options	READ	YES
zfsadm configquery	Display current configuration options	READ	NO

Figure 7-26 Changing the zFS IOEFSPRM member parameters dynamically

zfsadm commands

The **`zfsadm`** commands, with subcommands **`config`** and **`configquery`**, are used to change configuration values dynamically without a shutdown and restart of the zFS PFS, and to display the current value of the zFS configuration options, as follows:

- The **`zfsadm config`** command changes the value of zFS configuration options in memory that were specified in the `IOEFSPRM` file (or defaulted) or the IOEPRMxx PARMLIB member.
- The **`zfsadm configquery`** command displays the current value of zFS configuration options retrieved from the zFS address space memory, rather than from the `IOEFSPRM` file or the IOEPRMxx PARMLIB member.

Changing parameters dynamically

The **`zfsadm configquer`**y command displays the current value of zFS configuration options. The value is retrieved from zFS address space memory rather than from the `IOEFSPRM` file. You can specify that the configuration option query request should be sent to another system with the **`-system`** option.

Note: If you are using an `IOEFSPRM` file in your zFS PROC, the issuer must have READ authority to the data set that contains the `IOEFSPRM` file. If you are using PARMLIB (IOEPRMxx), the issuer does not need special superuser authorization.

Using PARMLIB (IOEPRMxx)

The preferred alternative to the IOEZPRM DDNAME specification is specifying the IOEFSPRM member as a true PARMLIB member. In this case, the member has the name IOEPRMxx, where xx is specified in the PARMLIB member list.

When the IOEFSPRM is specified in the IOEZPRM DD statement of the zFS PROC, there can only be one `IOEFSPRM` file for each member of a sysplex. Using PARMLIB, zFS configuration options can be specified in a list of configuration parm files. This allows an installation to specify configuration options that are common among all members of the sysplex (for example, adm_threads) in a shared IOEPRMxx member, and configuration options that are system-specific (for example, define_aggr) in a separate, system-specific IOEPRMxx member. If a configuration option is specified more than once, the first one found is taken.

7.27 zfsadm configquery command options

- Queries the current value of zFS configuration options
  - Use the zfsadm configquery -sysplex_state command to:
    - Indicate 2 or 3 if zFS is running sysplex-aware or 1 if it is not

```
$> zfsadm configquery -sysplex_state
IOEZ00317I The value for configuration option -sysplex_state is 3.
$> zfsadm configquery -all   ---- (Display all of the parms)
[-system system name] [-adm_threads] [-aggrfull] [-aggrgrow]
[-all] [-auto_attach] [-client_cache_size] [-client_reply_storage]
[-cmd_trace] [-convert_auditfid] [-debug_dsn] [-file_threads]
[-fsfull] [-fsgrow] [-group] [-log_cache_size] [-meta_cache_size]
[-metaback_cache_size] [-msg_input_dsn] [-msg_output_dsn] [-nbs]
[-romount_recovery] [-sync_interval] [-syslevel]
[-sysplex_filesys_sharemode] [-sysplex_state] [-token_cache_size]
[-trace_dsn] [-trace_table_size] [-tran_cache_size]
[-user_cache_readahead] [-user_cache_size] [-vnode_cache_size]
[-level] [-help]
```

Figure 7-27 zfsadm configquery command options

The zfsadm configquery command

The **`zfsadm configquery`** command queries the current value of zFS configuration options that you specify in either the IOEFSPRM DD statement in the procedure or the IOEPRMxx PARMLIB member. The command displays the current value of zFS configuration options. The value is retrieved from zFS address space memory rather than from the `IOEFSPRM` file. You can specify that the configuration option query request should be sent to another system by using the **`-system`** option.

Note: Use **`zfsadm configquery -all`** to display all of the parameter definitions.

sysplex_state parameter

Displays the sysplex state of zFS, as follows:

Zero (0 Indicates that zFS is not in a shared file system environment (normal for V1R6 and prior releases and for single system configurations including monoplex and xcflocal).

One (1) Indicates that zFS is in a shared file system environment (normal for V1R7 and above in a shared file system environment). This also indicates that zFS is non-sysplex aware for RW file systems; that is, either sysplex=off is specified or is the default.

Two (2) Indicates that zFS is running in a sysplex-aware environment with sysplex=on.

Three (3) Indicates that zFS is running in a sysplex-aware environment with sysplex=filesys.

7.28 zfsadm config command options

- Change the value of zFS configuration (IOEFSPRM) options in memory
  - zfsadm config -------- option ---------
  - ioezadm -------- option --------

```
[-admin_threads number] [-user_cache_size number[,fixed]]
[-meta_cache_size number[,fixed]] [-log_cache_size number[,fixed]]
[-sync_interval number] [-vnode_cache_size number] [-nbs {on|off}]
[-fsfull threshold,increment] [-aggrfull threshold,increment]
[-trace_dsn PDSE_dataset_name] [-tran_cache_size number]
[-msg_output_dsn Seq_dataset_name] [-user_cache_readahead {on|off}]
[-metaback_cache_size number[,fixed]] [-fsgrow increment,times]
[-aggrgrow {on|off}] [-romount_recovery {on|off}]
[-convert_auditfid {on|off}] [-client_reply_storage storage size]
[-file_threads number] [-client_cache_size cache size[,fixed]]
[-token_cache_size cache size] [-sysplex_filesys_sharemode
{rwshare|norwshare}] [-system system name] [-level] [-help]
```

Figure 7-28 The IOEFSPRM member or IOEPRMxx parameters

zfsadm config command options

The **`zfsadm config`** command changes the configuration options (in memory) that were specified in the `IOEFSPRM` file (or defaulted) or in the IOEPRMxx PARMLIB member. The `IOEFSPRM` file is not changed. If you want the configuration specification to be permanent, you need to modify the `IOEFSPRM` file or the IOEPRMxx PARMLIB member since ZFS reads the one that you use to determine the configuration values the next time ZFS is started.

Note:

```
//ZFS      PROC REGSIZE=0M
//*
//ZFZGO    EXEC PGM=BPXVCLNY,REGION=&REGSIZE,TIME=1440
//* ---------------------------------------------------------------
//*STEPLIB  DD DISP=SHR,DSN=IOE.SIOELMOD
//*
//* Using the parmlib search function instead of explicit IOEZPRM DD
//* by specifying IOEPRMxx members in the filesystype definition of
//* zFS and making the next line just a comment...
//*IOEZPRM  DD DISP=SHR,DSN=SYS1.TEST.IOEFSZFS(IOEFSPRM)
//* ---------------------------------------------------------------
```

Authorization required

The issuer must have READ authority to the data set that contains the `IOEFSPRM` file and must be root or have READ authority to the SUPERUSER.FILESYS.PFSCTL profile in the UNIXPRIV class.

The ioezadm command

The **`zfsadm`** command is executed from the z/OS UNIX shell. It can also be invoked from TSO/E by using the program name IOEZADM or as a batch job by using PGM=IOEZADM. For a batch job, the **`zfsadm`** options are specified in the EXEC PARM as a single subparameter (a single character string enclosed in apostrophes with no commas separating the options). You cannot put the ending apostrophe in column 72. If it needs one, go to the next line, use a continuation character in column 72 (continuing in column 16 with the ending apostrophe on the second line). Remember that a JCL EXEC PARM is limited to 100 characters.

7.29 Defining IOEFSPRM options

- A IOEFSPRM file and parameters in the file are optional
- Parameter file is referenced by the
  - DDNAME=IOEZPRM statement in PROC JCL for zFS

```
[-admin_threads number] [-user_cache_size number[,fixed]]
[-meta_cache_size number[,fixed]] [-log_cache_size
number[,fixed]] [-sync_interval number] [-vnode_cache_size
number] [-nbs {on|off}] [-fsfull threshold,increment]
[-aggrfull threshold,increment] [-trace_dsn
PDSE_dataset_name] [-tran_cache_size number]
[-msg_output_dsn Seq_dataset_name] [-user_cache_readahead
{on|off}] [-metaback_cache_size number[,fixed]] [-fsgrow
increment,times] [-aggrgrow {on|off}] [-romount_recovery
{on|off}] [-convert_auditfid {on|off}] [-client_reply_storage
storage size] [-file_threads number] [-client_cache_size
cache size[,fixed]] [-token_cache_size cache size]
[-sysplex_filesys_sharemode {rwshare|norwshare}]
[-system system name] [-level] [-help]
```

Figure 7-29 IOEFSPRM file options and defaults

IOEFSPRM file options

The best and only reasonable option is to define IOEFSPRM as a member of a PDS data set. This allows updates while the zFS PFS is active.

Figure 7-29 lists the processing options for the zFS PFS. There is no mandatory information in this file, therefore it is not required. The options all have defaults. Aggregates can all be compatibility mode aggregates (which do not need definitions). However, if you need to specify any options (for tuning purposes, for example), you need to have an IOEFSPRM file.

The location of the IOEFSPRM file is specified by the IOEZPRM DD statement in the ZFS PROC. The IOEFSPRM file is normally a PDS member, so the IOEZPRM DD might look like the following:

```
//IOEZPRM  DD  DSN=SYS4.PVT.PARMLIB(IOEFSPRM),DISP=SHR
```

New parameters with z/OS V1R11

The new parameters that are added with z/OS V1R11 are as follows:

- sysplex = {on | off} - Can be set on if SYSPLEX(YES) in BPXPRMxx
- token_cache_size = vnode_cache_size x 2
- file_threads = 40
- client_cache_size=128M
- client_reply_storage = 40M

7.30 Logical PARMLIB support

- A logical parmlib search is used
- Member names are in the form IOEPRMxx
- Multiple members can be specified
- Allows installation to have a common parmlib member that is shared among members of the sysplex but also have an additional member that is unique

The logical parmlib concatination is a set of up to 10 partitioned data sets defined by PARMLIB statements in the LOADxx member of SYSn.IPLPARM or SYS1.PARMLIB

Figure 7-30 Logical PARMLIB support

IOEPRMxx PARMLIB member

As an alternative to the IOEZPRM DDNAME specification, the IOEFSPRM member can be specified as a true PARMLIB member. In this case, the member has the name IOEPRMxx, where xx is specified in the PARMLIB member list.

When the `IOEFSPRM` file is specified in the IOEZPRM DD statement of the ZFS PROC, there can only be one `IOEFSPRM` file for each member of a sysplex.

Using PARMLIB, zFS configuration options can be specified in a list of configuration parm files. This allows an installation to specify configuration options that should be common among all members of the sysplex (for example, adm_threads) in a shared IOEPRMxx member; and configuration options that should be system-specific (for example, define_aggr) in a separate, system-specific IOEPRMxx member. If a configuration option is specified more than once, the first one found is taken.

7.31 Specifying PARMLIB members

- List of member (suffixes) is specified in the FILESYSTYPE statement for ZFS in a BPXPRMxx member

```
FILESYSTYPE TYPE(ZFS) ENTRYPOINT(IOEFSCM)
ASNAME(ZFS,'SUB=MSTR')
PARM('PRM=(01,02,03)')
```

```
FILESYSTYPE TYPE(ZFS) ENTRYPOINT(IOEFSCM)
ASNAME(ZFS,'SUB=MSTR')
PARM('PRM=(AA,&SYSCLONE.)')
```

Figure 7-31 Examples of PARMLIB members

PARMLIB member examples

The IOEPRMxx files are contained in the logical PARMLIB concatenation, which is a set of up to 10 partitioned data sets defined by PARMLIB statements in the LOADxx member of either SYSn.IPLPARM or SYS1.PARMLIB. The logical PARMLIB concatenation contains zFS IOEPRMyy members, which contain zFS configuration statements. Columns 73-80 are ignored in the IOEPRM*yy* member. The *yy* is specified in the PARM option of the FILESYSTYPE statement for ZFS (in the BPXPRMxx). The PARM string is case sensitive. You must enter the string in upper case. For example:

```
FILESYSTYPE TYPE(ZFS) ENTRYPOINT(IOEFSCM)
ASNAME(ZFS,'SUB=MSTR')
PARM('PRM=(01,02,03)')
```

Up to 32 member suffixes can be specified. You can also use any system symbol that resolves to two characters. For example:

```
FILESYSTYPE TYPE(ZFS) ENTRYPOINT(IOEFSCM)
ASNAME(ZFS,'SUB=MSTR')
PARM('PRM=(01,&SYSCLONE.)')
```

If &SYSCLONE.=AB, this specifies that PARMLIB member IOEPRMAB should be searched after PARMLIB member IOEPRM01. IOEPRM01 could contain common configuration options and IOEPRMAB could contain configuration options that are specific to system AB. If a PARMLIB member is not found, the search for the configuration option continues with the next PARMLIB member.

PARMLIB search

If no PRM suffix list is specified (and no IOEZPRM DD is specified in the ZFS PROC), then member IOEPRM00 is read. PARMLIB support is only used when the IOEZPRM DD statement is not specified in the ZFS PROC. When an IOEZPRM DD is specified in the ZFS PROC, the single IOEFSPRM file specified in the DD is used, as previously.

Coexistence

Coexistence between IOEFSPRM in the procedure and IOEPRMxx when using the FILESYSTYPE PARM PRM specification is ignored in previous releases. Therefore, if you specify it in a BPXPRMxx member that is shared between this release and previous releases (and no IOEZPRM DD is specified in the ZFS PROC), the PRM specification will be honored in this release but will be ignored in previous releases. This means that the IOEPRMxx members will be searched for zFS configuration parameters in this release, but defaults will be taken in previous releases. Also, if the ZFS FILESYSTYPE PARM has no PRM specification (and no IOEZPRM DD is specified in the ZFS PROC), then ZFS will attempt to use the IOEPRM00 member in this release, but will take defaults in the previous releases. If IOEPRM00 is not found, defaults are used.

7.32 Add a volume to a zFS aggregate

- Use the IDCAMS utility ALTER command
  - With the ADDVOLUMES parameter
    - Add two volumes to aggregate - **OMVS.ROGERS.TEST**

```
//ROGERSA JOB (ACCTNO),'SYSPROG',CLASS=A,
//     MSGCLASS=H,MSGLEVEL=(1,1),NOTIFY=&SYSUID
//STEP01   EXEC PGM=IDCAMS
//SYSPRINT DD  SYSOUT=*
//SYSIN    DD  *
           ALTER OMVS.ROGERS.TEST.DATA -
           ADDVOLUMES(* *)
/*
```

Figure 7-32 Adding a volume to a zFS aggregate

Add space to a zFS aggregate

To add a candidate volume to a zFS aggregate, use the IDCAMS utility ALTER command with the ADDVOLUMES parameter. The sample job shown in Figure 7-32 adds two volumes to the (SMS-managed) OMVS.ROGERS.TEST zFS aggregate.

In this case, DFSMS is choosing the particular candidate volumes. If you want to specify the volumes, use their volume serials in place of the asterisks. See *z/OS DFSMS: Access Method Services for Catalogs*, SC26-7394 for additional information on IDCAMS ALTER ADDVOLUMES. DFSMS states that if an ALTER ADDVOLUMES is done to a data set already opened and allocated, the data set must be closed, unallocated, reallocated, and reopened before VSAM can extend onto the newly-added candidate volume.

For zFS, this means that if the zFS aggregate is already attached when the ALTER ADDVOLUMES is done, it must be detached and attached again before zFS can extend to the newly-added candidate volume(s). Compatibility mode aggregates must be UNMOUNTed and MOUNTed again (since that is when they are detached and attached). If a backup file system (created by the clone operation) is mounted, it must also be unmounted when the read-write file system is unmounted. Otherwise, the aggregate will not be detached.

7.33 zFS migration considerations

- Migrate from HFS file systems to zFS file systems (recommended) - because zFS is planned to become a requirement in a future release
- zFS is the strategic file system
  - HFS may no longer be supported in future releases and you will have to migrate the remaining HFS file systems to zFS
- HFS and zFS file system types in mount statements and command operands are now generic file system types that can mean either HFS or zFS
  - Based on the data set type, the system will determine which is appropriate

Figure 7-33 zFS migration from HFS considerations

zFS migration considerations

zFS can be used for all levels of the z/OS UNIX System Services hierarchy (including the root file system) when all members are at the z/OS V1R7 level. Because zFS has higher performance characteristics than HFS and is the strategic file system, HFS may no longer be supported in future releases and you will have to migrate the remaining HFS file systems to zFS.

HFS may no longer be supported in future releases. At that time, you will have to migrate the remaining HFS file systems to zFS.

Beginning with z/OS V1R7, you can use the file system type HFS as a generic file system type that can mean either HFS or zFS. Mount processing will direct the mount to the correct PFS.

7.34 HFS/zFS as generic file system type

- They can be used for either HFS or ZFS
- Mount processing first searches for a data set matching the file system name
  - If the data set is not an HFS data set and zFS has been started, the filesystem type is changed to ZFS and the mount proceeds to zFS
  - If the data set was not found, the mount proceeds to zFS (assuming zFS is started)
    - If zFS is not started, the type is not changed and the mount proceeds to HFS

Figure 7-34 Using HFS/zFS as a generic file system type

Generic file system type

HFS is now a generic file system type that can be HFS or zFS.

Mount processing will recognize a substitutable token in the file system name if the file system type is HFS and substitute zFS for HFS as appropriate, based on data set existence.

Mount processing will first search for a data set matching the file system name. If the data set is not an HFS data set and zFS has been started, the file system type is changed to ZFS and the mount proceeds to zFS. If the data set was not found, the mount proceeds to zFS (assuming zFS is started). If zFS is not started, the type is not changed and the mount proceeds to HFS.

Note: If the zFS data set names are to be the same as the HFS data set names, it is likely that scripts or policies that are used to mount file systems would not need to be changed.

7.35 Migration considerations

- Often the file system type is used in a portion of the file system name
  - With the change to a file type of zFS, PARMLIB members, mount scripts and policies are affected as well as the file system name
    - You may prefer to keep the current file system name and then no changes to mount scripts
    - Migration tool will help with handling these changes
- zFS file systems that span volumes must be preallocated prior to invoking the tool
- zFS file systems that are greater than 4 GB must be defined with a data class that includes extended addressability
  - zFS file systems cannot be striped

Figure 7-35 Migration considerations for zFS file systems

Migration considerations

Several considerations are in order regarding file system naming conventions and very large VSAM linear data sets.

File system names

Given that during migration there are now two data sets, it is likely that some installations may want to have a different naming standard for their zFS data sets than their HFS data sets if the file type is part of the file name. It is also likely that they prefer keeping the names just as they are. If the installation chooses to have a new naming convention, then any mount policy or mount scripts (including from PARMLIB) have to be altered to mount the correct file system name.

If you decide to have a new naming convention, then you must change mount policies and mount scripts (including from the BPXPRMxx PARMLIB) to mount the correct file system name. You will have to look at your automount policies very carefully. You may need to modify the generic entry, migrate all file systems at one time, or add specific entries for each migrated file system.

zFS file systems

zFS file systems that will span volumes must be preallocated prior to invoking the tool.

zFS file systems that are greater than 4 GB (about 4825 cylinders of a 3390) must be defined with a data class that includes extended addressability. zFS file systems cannot be striped.

7.36 Migration tool

- Use a migration tool to handle many details needed for the migration of HFS file systems to zFS, including the actual copy operation - z/OS V1R7
  - Migration tool - BPXWH2Z
    - An ISPF based tool for migration from an HFS file system to zFS
    - It has a panel interface that enables you to alter the space allocation, placement, SMS classes, and data set names
    - A HELP panel is provided
    - Uses a new pax utility for copy

Figure 7-36 Migration tool with z/OS V1R7

Migration tool from HFS to zFS

To assist with the migration, a new tool, BPXWH2Z, is provided in the form of an ISPF dialog. It is expected that this tool will have a limited life both within the product stream and especially for any specific customer since it is only there for HFS to zFS migration. For this reason, for service purposes, it should be handled similar to the way samples are handled.

A panel interface allows you to alter the space allocation, placement, SMS classes, and data set names. The actual migration could run either in the TSO foreground or UNIX background.

To understand the panels, there is help information with sample information filled in. The panel syntax is explained in the help information.

This tool uses **pax** to perform the actual copy operation. New functions are in **pax** copy mode to ensure the migration tool performs the conversion.

7.37 Migration checks file system type

- For migration scenario, mount processing first checks for ZFS; if that data set exists, migration has been done
  - To revert back to using HFS, that data set must be renamed or removed
- A new place holder, ///, is created to represent the string HFS or ZFS
  - If type is HFS, as shown in the following mount statement or mount command
  - Mount processing first substitutes ZFS to see if data set exists - If no, HFS is used to see if the data set exists
  - Mount is directed to zFS or HFS

```
MOUNT   FILESYSTEM('ZOSR17.MAN.///')
        TYPE(HFS)
        MODE(READ)
        MOUNTPOINT('/usr/man')
```

Figure 7-37 Migration placeholder for file system names

Migration placeholder for file system names

Support is added in z/OS V1R7 to provide the capability to specify a file system name with substitution place holders. The place holder is /// and represents the string HFS or ZFS as appropriate, if the file system type is specified as HFS.

/// is selected because it is the same number of characters as ZFS and HFS, / is not a character that is processed by the shell, and / is not used in a data set name. Where /// is used, mount processing will first substitute ZFS and check to see if the data set exists and is not an HFS data set. If this is the case it proceeds with that name and directs the mount to ZFS. Otherwise, mount processing substitutes HFS in the name and checks the status of that data set name. If that data set exists and is not an HFS, it directs the mount to ZFS; otherwise, it directs the mount to HFS.

Since this is a migration scenario, mount processing first checks for the ZFS file system type under the assumption that if that data set exists, the migration has been done. In order to revert back to using the HFS, that data set must be renamed or removed.

7.38 Migration tool enhancements with APAR OA18196

- New Function - BPXWH2Z HFS to ZFS Migration Utility
- Enhancements are as follows:
  - 1) Multi-volume conversion support
    - Pre-allocation of a multi-volume zFS no longer required
  - 2) Minimum zFS allocation verification
    - An HFS file system that is too small for zFS file system and fails on zFS allocation - minimum is 6 tracks
  - 3) Panel usability enhancements
    - New zFS name can be changed on change allocation panel
  - 4) New option for exact HFS name match
    - New -e option on BPXWH2Z call

Figure 7-38 Migration tool enhancements

Migration tool implementation

Before this APAR was introduced, the migration tool BPXWH2Z had the following requirements that are fixed by this APAR:

- Support needed to migrate multi-volume HFS file systems to multi-volume zFS file systems. The tool currently requires the zFS file systems to be preallocated for multi-volume HFS file system migration.
- The tool currently tries to convert an HFS file system that is too small for a zFS file system and fails on zFS allocation. The minimum supported size for zFS file systems is 6 tracks. The tool does not validate the minimum zFS allocation.
- When the BPXWH2Z tool is invoked via the ISPF batch job, all the HFS file system names that start with the input HFS file system name are included for migration. There is no way to include or exclude the HFS file systems from the migration list.

Note: APAR OA18196 applies to z/OS V1R7 and z/OS V1R8.

Migration tool enhancements

The BPXWH2Z tool is enhanced to provide the following new functions:

- Migration of SMS-managed multi-volume HFS file systems to SMS-managed multi-volume zFS file systems. If your HFS file system is multi-volume, then make sure that it is SMS-managed before you use this tool to do the migration. Your zFS will be allocated

based upon the allocation attributes of the HFS and you will have an opportunity to modify these attributes via the Change Allocation Attributes panel.

- The tool ensures that the zFS file system allocation is no less than the minimum allocation supported by the zFS file system.
- The Data Set List panel in the BPXWH2Z tool is enhanced to display the “final zFS” field. If you select an HFS data set with character A to change the attributes, you will be able to modify the final zFS name. The Change Allocation Attributes panel is also enhanced to display and modify the Final zFS name field. When a substitution string is specified, the Data Set List panel will be primed such that the HFS will be saved as it is and the final zFS name will be changed according to the specified substitution string. You have the option to change the saved HFS name, the final zFS name as well as the allocation attributes by specifying the character A in front of the HFS name in the Data Set List panel and modifying it in the Change Allocation Attributes panel.

3. A new option, -e, is added to the BPXWH2Z tool to exactly match the HFS file system name. When the new option is used during the invocation of the BPXWH2Z tool, it will match the input HFS file system name as exact name and will not consider it as a pattern string (default behavior).

   The new zFS data set is now being allocated under a user's authority. Therefore, the user must have authority to create, rename, and delete the new zFS data sets. Otherwise, the BPXWH2Z tool will fail.

   When a zFS data set is preallocated, the Initial zFS name field and the Final zFS name field in the Change Allocation Attributes panel must be the same.

7.39 REXX exec - BPXWH2Z

- The REXX is located in library SYS1.SBPXEXEC which is concatenated to SYSPROC
- ISPF Option 6 command line, Enter **bpxwh2z**
- Command line flag options:
  - -v Additional messages are issued at various points in processing
  - -c Summary information goes to a file when background (bg) is selected once you are in the application. If this is not specified, summary information goes to the console
    - bpxwh2z -c - bpxwh2z -v - bpxwh2z -e
    - bpxwh2z -cve

Figure 7-39 Using the BPXWH2Z migration tool

Using the migration tool

From the ISPF Option 6 command line, enter `bpxwh2z` to begin to use the migration tool. The tool is a REXX exec that is located in library SYS1.SBPXEXEC, which is concatenated to SYSPROC.

Command line options for bpxwh2z

The first argument can specify option flags. The option flags are preceded by a dash (-), followed immediately by the flags. The flags are as follows:

-v Additional messages are issued at various points in processing.

-c Summary information goes to a file when background selected; if this is not specified, summary information goes to the console.

Specify the line options as follows:

```
bpxwh2z -v or bpxwh2z -c or bpxwh2z -e or bpxwh2z -vce
```

You can specify any combination of -v, -c, and -e.

7.40 BPXWH2Z panels

- Panels allow for customization of space allocation, placement SMS class and data set name
- Tool makes the zFS a little larger if it sees the HFS is near capacity by:
  - **10%** if it is over 75% full and under 90% and **20%** if over
  - This is a precaution since zFS is a logging file system and includes the log in the aggregate
- If unmounted it creates a mountpoint in /tmp and mounts it read only for the migration, then deletes the mountpoint
- If there are file systems mounted below the migrating file system, they are unmounted for migration
  - The mount points are created in the new file system and when complete, the file systems are remounted

Figure 7-40 Using BPXWH2Z panels

Using BPXWH2Z panels

If the HFS file system is mounted, either unmount it or switch it to read-only mode. The migration switches it to R/O mode if not done manually.

If the file system is not mounted, the tool makes a directory for it and mounts it in R/O mode.

The tool creates a temporary directory for the zFS file system in /tmp for the mountpoint and deletes it when migration is complete).

Size of HFS being migrated

The tool checks the allocation attributes of the HFS file system. It then defines a new zFS file system with appropriate allocation attributes, as follows:

- The tool defaults these to the same attributes as the HFS file system if its utilization is below about 75% full, otherwise it adds about 10% to the new zFS if it's below 90%.
- The tool adds 20% to the zFS if the HFS is above 90% full.

It is difficult to determine the exact size necessary for a zFS file system to contain all of the data within an HFS file system. Depending on the number of files, directories, ACLs, symlinks, and file sizes, zFS can consume either more or less space than HFS. zFS also has a log in the aggregate. For general purpose file systems, it appears that they consume about the same amount of space.

7.41 Space allocations - HFS versus zFS

- It is difficult to determine the exact size necessary for a zFS file system to contain all of the data within an HFS file system
  - Depending on number of files, directories, ACLs, symlinks, and file sizes, zFS can consume either more or less space than HFS
    - zFS also has a log in the aggregate
  - For general purpose file systems, it appears that they consume about the same amount of space
- Two data sets will exist during the migration, so plan for space for two file systems about same size

Figure 7-41 Space allocation between HFS and zFS

Space allocations - HFS versus zFS

Define zFS aggregates by default to be approximately the same size as the HFS. The new allocation size can be increased or decreased.

Depending on the number of files, directories, ACLs, symlinks, and file sizes, zFS can consume either more or less space than HFS. zFS also has a log in the aggregate. For general purpose file systems, it appears that they consume about the same amount of space.

Assistance with space and other data set management issues is a consideration during the migration. This includes having the DASD space available to perform the migrations. You may need to change automatic class selection (ACS) routines to direct allocations to desired storage classes with the right attributes, and security changes that may be necessary for new data set profiles.

7.42 Using the migration tool

- From ISPF Option 6, Enter: bpxwh2z
- When the first panel is displayed
  - Enter HFS data set name to migrate - Then hit Enter

```
---------------------------  DATA SET SPECIFICATION  --------------------------
Command ===>

Use the HELP command for usage information on this tool.
Enter the name of the HFS file system to migrate to a zFS file system.
Note that a data set pattern can be used.
If the file system is not currently mounted it will be mounted on a
temporary directory during the migration.


File system name: rogers.test10.hfs_
```

Figure 7-42 Using the migration tool first panel

Migration tool first panel

The utility can be run with no arguments; it will prompt for an HFS file system name. You can also specify one or more file system names as an argument. The names can be simple names or patterns that you might use for the ISPF data set list panel. However, an asterisk (*) must be specified at the end to find data sets with additional characters at the end of the name.

Using the tool

To use the migration tool, enter `bpxhw2x` on the ISPF Option 6 command line; the panel shown in Figure 7-42 is displayed.

To migrate a file system, specify the file system name on the panel as shown:

```
ROGERS.TEST10.HFS
```

Then press Enter to display the next panel.

7.43 BPXWH2Z panels

- A substitution string can be specified as the first argument on the command line as follows:
  - /fromstring /tostring/ datasetname
- For example, if your data set is OMVS.DB2.HFS and you want your zFS data set to be named OMVS.DB2.ZFS, you can specify the following:
  - bpxwh2z -cv /hfs/zfs/ omvs.db2.hfs

Figure 7-43 Using BPXWH2Z panels

Using BPXWH2Z panels

By default, the panels are initialized such that your HFS data set will be renamed with a .SAV suffix and the zFS data set will be renamed to the original HFS name.

You can preallocate the zFS file system or specify a character substitution string. If you specify a substitution string, the panel will be primed such that the data sets will not be renamed. The substitution string is specified as the first argument on the command line as:

```
/fromstring/tostring/
```

For example, if your data set is OMVS.HFS.WJS and you want your zFS data set to be named OMVS.DB2.HFS you can specify a command argument, as follows:

```
/hfs/zfs/ omvs.db2.hfs
```

The resulting new zFS data set name will be OMVS.DB2.ZFS.

Attention: Be careful—all strings matching the `fromstring` in the data set name will be replaced.

7.44 Using SMS if required

- Alter SMS classes and volume if required
  - Then hit the Enter key to see the allocations

```
---------------------------  CLASS AND VOLUME DEFAULTS --------------------
Command ===>

The volume or SMS classes for zFS allocations will default to the
same names as the current HFS allocation.  You can change these
individually for each new allocation.  If you want the default for
each new allocation to be set to specific values other than that of
the current HFS allocation, enter those values here and Press Enter.

Default volume  . . . . .: _______
Default data class  . . .: ________
Default storage class . .: ________
Default management class : ________
```

Figure 7-44 Specify SMS classes and volume if required

Using SMS classes

There are panel interfaces that allow the user to alter the space allocation, placement, SMS classes, and data set names.

With this panel, you can specify either a default volume or SMS classes to determine where the new zFS data set will be placed.

Otherwise, the new zFS will be placed to the same place as the HFS data set.

When you press Enter, the panel shown in Figure 7-45 is displayed.

7.45 Migrate in the foreground

```
                                        A - Alter allocation
--------------------------  DATA SET LIST  ----------------- Row 1 to 1 of 1
Command ===> fg

  Use the HELP command for full usage information on this tool
  Select items with D to delete items from this migration list
  Select items with A to alter allocation parameters for the items
  Enter command FG or BG to begin migration in foreground or UNIX background
_--------------------------- ROGERS.TEST10.HFS ----------------------------
  Save HFS as . .: ROGERS.TEST10.HFS.SAV                      Utilized : 2%
  Initial zFS . .: ROGERS.TEST10.HFS.TMP                      Allocated: N
  Final zFS . . .: ROGERS.TEST10.HFS
  HFS space  Primary  : 1          Secondary: 1          Units ..: CYL
  zFS space  Primary  : 1          Secondary: 1          Units ..: CYL
             Dataclas :            Mgmtclas :            Storclas:
             Volume   : SBOX1T     Vol count: 1
****************************** Bottom of data ******************************

        Migrating ROGERS.TEST10.HFS   14:14:40
       creating zFS ROGERS.TEST10.HFS.TMP
       copying ROGERS.TEST10.HFS to ROGERS.TEST10.HFS.TMP Blocks to copy: 5
       IDC0531I ENTRY ROGERS.TEST10.HFS.TMP ALTERED
       IDC0531I ENTRY ROGERS.TEST10.HFS.TMP.DATA ALTERED
       Migration complete for ROGERS.TEST10.HFS    14:14:49
       ***
```

Figure 7-45 Migrating the HFS data set in the foreground

Migrate in the foreground

This panel displays the HFS and zFS allocations. You may alter these allocations for the new zFS data set. To begin the migration, enter the command FG or BG to run the migration in the foreground or background. If it is run in the background, the tool will keep a standard output log and also a summary log. The prefix for the pathnames is displayed. The commands D or A may be specified for each HFS data set to delete items from the migration list or to alter allocation parameters for the items.

Each table entry shows the data set names that will be used, current HFS space utilization and space allocation, and allocation parameters that will be used to create the zFS file system. Select a row to alter the allocation attributes or data set names. Rows can also be deleted. Additional rows cannot be added.

The commands that can be used on this panel are as follows:

D To delete items from this migration list.

A To alter allocation parameters for the items, enter an `A` on the action character line to alter the space allocation. A new panel is displayed and is shown in Figure 7-49 on page 375.

FG To begin migration in the TSO/E foreground, enter FG on the command line.

BG To begin migration in the UNIX background, enter BG on the command line.

When you press Enter on this panel using the current specifications, the messages shown are issued on a blank panel.

7.46 Migration steps

- If the HFS file system is mounted, either unmount it or switch it to read-only mode
  - The migration tool switches it to R/O
- If the file system is not mounted make a directory for it and mount it in R/O mode
  - The migration tool creates a temporary directory in /tmp for the mountpoint and deletes it when the migration is complete
- Create a temporary directory for the zFS file system
  - The migration tool creates a temporary directory in /tmp for the mountpoint and deletes it when the migration is complete
- Check the allocation attributes of the HFS file system

Figure 7-46 Migration steps

Migration steps

Following are the steps taken by the migration tool to migrate HFS data sets to zFS data sets:

1. If the HFS file system is mounted, either unmount it or switch it to read-only mode. The tool will switch it to R/O mode.
2. If the file system is not mounted, make a directory for it and mount it in R/O mode. The tool will create a temporary directory in /tmp for the mountpoint and delete it when migration is complete.
3. Create a temporary directory for the zFS file system. The tool will create a temporary directory in /tmp for the mountpoint and delete it when migration is complete.
4. Check the allocation attributes for the HFS file system. See “Size of HFS being migrated” on page 366.

7.47 Migration steps

- Define a new zFS file system with the appropriate allocation attributes
- Mount the zFS file system on its temporary mountpoint (a pre-existing zFS cannot already be mounted)
- Use the pax utility to copy the HFS contents to the zFS by entering the shell, change directory to the HFS mountpoint, and run the pax utility - The pax command you can use is:
  - pax -rw -X -E . /tmp/zfsmountpoint
- If the HFS contains active mountpoints, do an mkdir for each of these in the zFS file system and set directory attributes as required

Figure 7-47 Migration steps continued

Migration steps continued

5. Define a new zFS file system with appropriate allocation attributes.
6. Mount the zFS file system on its temporary mountpoint.
7. Use the **pax** utility to copy the HFS contents to the zFS by entering the shell and then change the directory to the HFS mountpoint and run the new **pax** utility. The **pax** command is as follows:

   ```
   pax -rw -X -E . /tmp/zfsmountpoint
   ```

   The previous **pax -X** option caused **pax** to write only those files that are on the same device as the parent directory. Invoking this option causes active mountpoints to be ignored when **pax** is in copy mode. However, in z/OS V1R7, the new **pax** option causes an empty directory to be created on the target for these mountpoints.

   pax reads the specified path name and copies it to the target directory. The target directory must already exist and you must have write access to it. If a path name is a directory, **pax** copies all the files and subdirectories in that directory, as well as the directory itself, to the target directory.
8. If the HFS contains active mountpoints, do an **mkdir** for each of these in the zFS file system and set directory attributes as required. Set attributes for the zFS based on the HFS that was copied. Note that this must include all extended attributes, ACLs, owner, group, code page, and mode bits.

7.48 Migration steps continued

- ❑ Unmount the zFS file system
- ❑ Unmount the HFS file system
- ❑ Remove any temporary directories used as mountpoints
- ❑ Rename the data sets as appropriate and ensure all mount scripts and policies have been updated as needed
- ❑ The migration tool does not modify or search for any type of mount scripts or policies
- ❑ If the HFS file system was originally mounted, mount the zFS file system in that same location along with any hierarchy that also had to be unmounted to unmount the HFS

Figure 7-48 Migration steps continued

Migration steps continued

9. Unmount the zFS file system.
10. Unmount the HFS file system. Note that if it contains active mountpoints, those file systems and any hierarchy below them must be unmounted first.
11. Remove any temporary directories used as mountpoints.
12. Rename the data sets as appropriate and ensure that all mount scripts and policies have been updated as needed. The migration tool will not modify or search for any type of mount scripts or policies.
13. If the HFS file system was originally mounted, mount the zFS file system in that same location along with any hierarchy that also had to be unmounted to unmount that HFS.

7.49 Alter allocation parameters panel

- Alter allocations of the zFS data set with APAR OA18196
  - This panel is required if the zFS is going to be multi-volume

```
----------------------  CHANGE ALLOCATION ATTRIBUTES  -------------------------
Command ===> _

Use the HELP command for usage information on this tool.
Change the allocation attributes for this new zFS file system.
If the file system is already allocated enter Y for preallocated and the
name of the data set as the Initial zFS name. The attributes will not be used.

File system name . . . ROGERS.TEST10.HFS
Save HFS as  . . . . . ROGERS.TEST10.HFS.SAV
Initial zFS name . . . ROGERS.TEST10.HFS.TMP
Final zFS name . . . . ROGERS.TEST10.HFS
Preallocated zFS . . . N
Primary allocation . . 1
Secondary allocation   1
Allocation units . . . CYL      (CYL or TRK)
Data class . . . . . .
Management class . . .
Storage class  . . . .
Volume . . . . . . . . SBOX1T
Volume count . . . . . 1
```

change zFS name

multi-volume

Figure 7-49 Panel to alter allocations of new zFS

Alter allocation parameters panel

This panel allows you to change the allocation options that were initially determined by the migration tool. The values shown in the figure were filled in by the migration tool. You may change these parameters by overtyping the shown specifications.

Migrating to a multivolume zFS

If your HFS is multivolume, you no longer must preallocate your zFS in order to have similar attributes and multivolume with APAR OA18196. This panel is shown when you specify A for alter allocation on the panel shown in Figure 7-45 on page 371.

7.50 APAR OA18196 - Exact data set match

```
The new option (-e) is available with APAR OA18196 to specify the HFS data set
name exactly. In the example,two HFSes are involved and just that one exactly
matching is selected:
READY
ISPSTART CMD(BPXWH2Z -eVC OMVS.HERING.HFS.TEST)
Not running UID=0 - attempting to set UID=0
-getting ds list for OMVS.HERING.HFS.TEST
Skipping matching patterns of input data set name OMVS.HERING.HFS.TEST.FS2
-getting ds info for OMVS.HERING.HFS.TEST
-getting ds info for OMVS.HERING.HFS.TEST.SAV
-getting ds info for OMVS.HERING.HFS.TEST.TMP
Migrating OMVS.HERING.HFS.TEST   15:50:18
creating zFS OMVS.HERING.HFS.TEST.TMP
copying OMVS.HERING.HFS.TEST to OMVS.HERING.HFS.TEST.TMP Blocks to copy: 220
- cd /tmp/bpxwh2z.HERING.15:50:18.hfs; /bin/pax -rw -peW -XCM .
/tmp/bpxwh2z.HERING.15:50:18.zfs
IDC0531I ENTRY OMVS.HERING.HFS.TEST.TMP ALTERED
IDC0531I ENTRY OMVS.HERING.HFS.TEST.TMP.DATA ALTERED
Migration complete for OMVS.HERING.HFS.TEST    15:50:27
 HERING.HERIJOB.JOB04075.D0000103.? was preallocated (no free was done).
READY
END
```

Figure 7-50 Using the -e option to select an exact data set match

Exact data set match (-e) option

Notice that the migration tool, BPXWH2Z, can be used from TSO ready, as shown in Figure 7-50. Using the -e option indicates that when selecting the HFS file system to migrate, the data set name must be an exact match.

The name specified to migrate is OMVS.HERING.HFS.TEST. As shown in the example, data set name OMVS.HERING.HFS.TEST is migrated and data set name OMVS.HERING.HFS.TEST.FS2, which is almost an exact match, is not migrated.

Note: In the example, note that the HLQ, OMVS, indicates that the data set is SMS-managed.

7.51 Migrating a list of data sets

```
List of data sets to migrate:
     OMVS.ROGERS.TEST1
     OMVS.ROGERS.TEST2
     OMVS.ROGERS.TEST3
     OMVS.ROGERS.TEST4
     OMVS.ROGERS.TEST5
     OMVS.ROGERS.TEST6
     OMVS.ROGERS.TEST7

---------------------------  DATA SET SPECIFICATION  --------------------------
Command ===>

Use the HELP command for usage information on this tool.
Enter the name of the HFS file system to migrate to a zFS file system.
Note that a data set pattern can be used.
If the file system is not currently mounted it will be mounted on a
temporary directory during the migration.

File system name: omvs.rogers.test*_
```

Figure 7-51 A list of data sets to migrate

Migrating a list of data sets

When you have a list of data sets to migrate that have very similar names, you can use a wildcard, * , to specify the list, as shown in the figure. After getting the list of data sets to migrate, the migration tool obtains data set information on each data set. The tool does not migrate the following types of data sets:

- That do not exist
- Are HSM migrated
- Are not HFS data sets

The resulting data set list is presented in a table, shown in Figure 7-52 on page 378.

7.52 Data set list displayed

```
-------------------------- DATA SET LIST ------------------------------------------------------- Row 1 to 3 of 8
Command ===> fg

Use the HELP command for full usage information on this tool
   Select items with D to delete items from this migration list
   Select items with A to alter allocation parameters for the items
   Enter command FG or BG to begin migration in foreground or UNIX background
 ---------------------------------------------------------------------------
 _ HFS data set ..: OMVS.ROGERS.TEST                              Utilized: 62%
   Save HFS as  ..: OMVS.ROGERS.TEST.SAV
   Initial zFS  ..: OMVS.ROGERS.TEST.TMP                          Allocated: N
   HFS space  Primary  : 25         Secondary: 5          Units ..: CYL
   zFS space  Primary  : 25         Secondary: 5          Units ..: CYL
              Dataclas : HFS        Mgmtclas : HFS        Storclas: OPENMVS
   MOUNTED    Volume   : SBOX1F     Vol count: 1
 ---------------------------------------------------------------------------
 _ HFS data set ..: OMVS.ROGERS.TEST1                             Utilized: 62%
   Save HFS as  ..: OMVS.ROGERS.TEST1.SAV
   Initial zFS  ..: OMVS.ROGERS.TEST1.TMP                         Allocated: N
   HFS space  Primary  : 25         Secondary: 5          Units ..: CYL
   zFS space  Primary  : 25         Secondary: 5          Units ..: CYL
              Dataclas : HFS        Mgmtclas : HFS        Storclas: OPENMVS
              Volume   : SBOX1E     Vol count: 1
   ............................ more data sets .............
```

Figure 7-52 List of data sets displayed using a data set name with a wildcard

List of data sets

Use your normal up/down to scroll through the list. Each table entry shows the following:

- Data set names that will be used
- Current HFS space utilization
- Space allocation and allocation parameters

These are used to create the zFS file system. Select a row to alter the allocation attributes or data set names. Rows can also be deleted. Additional rows cannot be added.

To begin the migrations, enter the command FG or BG to run the migration in the foreground or background. Note the copy process may take a long time. If this is run in the background, the tool keeps a standard output log and also a summary log. The prefix for the pathnames is displayed. **pax** is used to do the copy.

7.53 Health Checker USS_HFS_DETECTED

- Beginning with z/OS V1R5, HFS was no longer considered the strategic file system, and zFS became the strategic file system
  - This new check verifies all file systems mounted and issues an exception message if any HFS file systems are found
- This check runs any time an HFS file system is successfully mounted
  - Unless overridden by the RUN_ON_MOUNT=NO parameter
- The check will also run any time a F BPXOINIT,FILESYS=REINIT command is issued

Figure 7-53 Health Checker USS_HFS_DETECTED

Health Checker USS_HFS_DETECTED

Beginning with z/OS V1R5, HFS was no longer considered the strategic file system, and zFS became the strategic file system. This check is used to highlight any HFS file systems that are still being used so that they can be migrated to zFS. This new check verifies that all file systems are mounted and issues an exception message if any HFS file systems are found. The check only looks for HFS file systems that are owned on the system running the health check.

This check runs any time an HFS file system is successfully mounted, unless overridden by the RUN_ON_MOUNT=NO parameter. The check will also run any time a **`F BPXOINIT,FILESYS=REINIT`** command is issued. The check does not run after the **`F OMVS,NEWROOT=xxx`** command is issued.

Check reason: HFS file systems are no longer the strategic file systems. All HFS file systems should be migrated to zFS. This check is a quick way to identify which HFS file systems are mounted on the system and a reminder to migrate to zFS.

This Health Checker routine is run with setting Low Severity once a day. The check will be scheduled to run every 24 hours.

The check will run after a **`F BPXOINIT,FILESYS=REINIT`** command is issued. This can only be on the system where the command is issued.

7.54 Health Checker USS_HFS_DETECTED

- This check is a quick way to identify which HFS file systems are mounted on the system and a reminder to migrate to zFS
  - This check is run with setting Low Severity once a day
  - The check is scheduled to run every 24 hours
  - The check has two parameters
- **RUN_ON_MOUNT** - This indicates wether the check should run after the successful mount of an HFS file system - You can specify **YES** or **NO**
- **HFS_LIST** - You can specify a list of HFS file systems that you wish to ignore for this check - A file system specified for this parameter will not cause the check to issue an exception message.

Figure 7-54 Health Checker USS_HFS_DETECTED

Health Checker USS_HFS_DETECTED

Two parameters can be specified with this health check, as follows:

RUN_ON_MOUNT This indicates wether the check should run after the successful mount of an HFS file system. You can specify **YES** or **NO**.

DEFAULT: **RUN_ON_MOUNT=YES**

HFS_LIST You can specify a list of HFS file systems that you wish to ignore for this check. A file system specified for this parameter will not cause the check to issue an exception message. This parameter is limited to 23 file systems and by the health check limit of 256 characters for a check's parameters. Following is an example for specifying this parameter:

HFS_LIST=(OMVS.TEST1.HFS,OMVS.TEST2.HFS)

Defining the health check

The following shows keywords you can use to override check values on either a POLICY statement in the HZSPRMxx parmlib member or on a **MODIFY** command. This statement may be copied and modified to override the check defaults:

```
UPDATE
CHECK(IBMUSS,USS_HFS_DETECTED)
SEVERITY(LOW)
INTERVAL(24:00)
```

```
PARM('RUN_ON_MOUNT=YES')
DATE(20090427)
REASON('your reason for making the update ')
```

USS_HFS_DETECTED messages

Three specific messages can be displayed with this health check. The messages are used as follows:

- BPXH067I is displayed in the health checker report when everything is okay. This means that no mounted HFS file systems (except those in the HFS_LIST) have been found.

  ```
  BPXH067I No HFS file systems are mounted.
  ```

- BPXH069I is displayed in the health checker report if at least one HFS file system has been found. The message is followed by the list of the mounted HFS file systems.

  ```
  BPXH069I The following HFS file systems were found
  ```

- BPXH068E is an exception message that is shown when errors are found. This means that message BPXH069I is displayed as well. This exception message is listed in the health checker report and is also displayed in the SYSLOG or OPERLOG.

  ```
  BPXH068E One or more HFS file systems mounted.
  ```

7.55 Check with RUN_ON_MOUNT=YES

```
#> /usr/sbin/mount -f hering.testmnt.hfs test
#> df -v test | head -l3 | tail -1
HFS, Read/Write, Device:197, ACLS=Y
```

- Response in SYSLOG to the mount

```
HZS0001I CHECK(IBMUSS,USS_HFS_DETECTED): 235
BPXH068E One or more HFS file systems mounted.
```

- Part of the output in the SDSF health check display

```
BPXH069I The following HFS file systems were found:
-----------------------------------------------------
HERING.TESTMNT.HFS
OMVS.SC74.VAR
OMVS.SC74.ETC
OMVS.DB2V8.UK05586.HFS
```

Figure 7-55 Check with RUN_ON_MOUNT=YES

Check with RUN_ON_MOUNT=YES

You can force running the check if **RUN_ON_MOUNT=YES** is specified, as shown in Figure 7-56.

```
#> /usr/sbin/mount -f hering.testmnt.hfs test
#> df -v test | head -l3 | tail -1
HFS, Read/Write, Device:197, ACLS=Y
#>
```

Figure 7-56 Mounting an HFS file system

This results in the messages shown in Figure 7-57 going to the SYSLOG.

```
HZS0001I CHECK(IBMUSS,USS_HFS_DETECTED): 235
BPXH068E One or more HFS file systems mounted.
```

Figure 7-57 USS_HFS_DETECTED messages in the syslog

Excerpts of the corresponding Health Checker report output are displayed in Figure 7-58 on page 383.

```
CHECK(IBMUSS,USS_HFS_DETECTED)
START TIME: 05/14/2010 15:41:10.052735
CHECK DATE: 20090427  CHECK SEVERITY: LOW
CHECK PARM: RUN_ON_MOUNT=YES

BPXH003I z/OS UNIX System Services was initialized using OMVS=(2A),
where each 2-character item is a BPXPRMxx suffix.

BPXH069I The following HFS file systems were found:
------------------------------------------------------------
HERING.TESTMNT.HFS
OMVS.SC74.VAR
OMVS.SC74.ETC
OMVS.DB2V8.UK05586.HFS

* Low Severity Exception *

BPXH068E One or more HFS file systems mounted.

  Explanation:  The USS_HFS_DETECTED check found one or more active HFS
    file systems on the current system.

  System Action:  The system continues processing.
... (17 lines have been removed)
Check Reason:  HFS file systems are no longer the strategic file
    system.  All HFS file systems should be migrated to zFS.

END TIME: 05/14/2010 15:41:10.053512  STATUS: EXCEPTION-LOW
```

Figure 7-58 USS_HFS_DETECTED Health Checker output

You can also explicitly run this health check, as shown in Figure 7-59.

```
F HZSPROC,RUN,CHECK=(IBMUSS,USS_HFS_DETECTED)
HZS0400I CHECK(IBMUSS,USS_HFS_DETECTED): 261
RUN PROCESSING HAS BEEN COMPLETED
HZS0001I CHECK(IBMUSS,USS_HFS_DETECTED): 262
BPXH068E One or more HFS file systems mounted.
```

Figure 7-59 Running health check USS_HFS_DETECTED using F HZSPROC

7.56 Special characters in zFS aggregates

- Previously, you could not have a zFS aggregate or file system name that contained special characters
- Support in z/OS V1R7 for special characters:
  - (@ # $)

```
# zfsadm define -aggr OMVS.HERING.@TEST.ZFS -volumes CFC000 -cyl 10
IOEZ00248E VSAM linear datase PLEX.JMS.AGGR#06.LDS0006 successfully
created.
# zfsadm format -aggr OMVS.HERING.@TEST.ZFS -compat
IOEZ00077I HFS-compatibility aggregate OMVS.HERING.@TEST.ZFS has been
successfully created
```

Figure 7-60 Special characters in file system names

Special characters

When specifying a new zFS file system name for the read-write file system, the name must be unique within the sysplex (or system, if not in a sysplex).

With z/OS v1R7, the following characters can now be included in the name of a file system:

- The at sign (@)
- The number sign (#)
- The dollar sign ($)

7.57 BPXMTEXT shell command

- Previously, the bpxmtext command could not display the meaning of a zFS reason codes
- With z/OS V1R8, support is added to allow the bpxmtext command to display the meaning of zFS reason codes

```
ROGERS @ SC75:/u/rogers>bpxmtext EF096800
zFS Fri Jul 28 10:36:15 EDT 2006
Description: Mount for file system contain in multi-file system
aggregate is not allowed

Action: Using a release of z/OS prior to z/OS V1R8, attach the
aggregate, mount the file system and copy the file system data to a
compatibility mode aggregate.
```

Figure 7-61 BPXMTEXT command change with z/OS V1R8

The BPXMTEXT command

BPXMTEXT displays the description and action text for a reason code returned from the kernel. It supports reason codes from z/OS UNIX System Services, TCPIP, and now with z/OS V1R8 for zFS. This command is intended as an aid for problem determination.

zFS reason codes

The reason_code is specified as 8 hexadecimal characters. Leading zeros may be omitted. If no text is available for the reason code, a blank line is displayed. An argument that is not 1-8 hex digits will result in a usage message. This message will not be translated.

Distributed File Service reason code qualifiers are found in the range X'EF01' to X'EFFF' for the zSeries File System (zFS).

Note: In addition to displaying z/OS UNIX System Services reason codes, the z/OS UNIX System Services shell command, **`bpxmtext`**, also displays the text and action of zFS reason codes (EFxxnnnn) returned from the kernel. zFS does not use the xx part of the reason code to display a module name. It always displays zFS. If you only know the nnnn part of the zFS reason code, you can use EF00nnnn as the reason code. The date and time returned with the zFS reason code matches the date and time returned from the zFS kernel (displayed with operator command F ZFS,QUERY,LEVEL).

8

File sharing in a sysplex

This chapter describes the hierarchical file system and how to customize it to your requirements. zFS supports a shared file system capability in a multisystem sysplex environment. The term shared file system environment refers to a sysplex that has a BPXPRMxx specification of SYSPLEX(YES).

That is, users in a sysplex can access zFS data that is owned by another system in the sysplex. For full sysplex support, zFS must be running on all systems in the sysplex in a shared file system environment and all zFS file systems must be compatibility mode file systems (that is, they cannot be file systems in multifile system aggregates). zFS multifile system aggregates are not supported by automount.

© Copyright IBM Corp. 2011. All rights reserved.

8.1 Shared file systems in a sysplex

- Filesystem data sets that exist in a Sysplex
  - **(1). Sysplex root** - only 1 for all sharing systems
    - Contains directories and symlinks
    - Redirects addressing to other directories
  - **(2). System-specific**
    - Contains data specific to each system
    - Directories for /dev, /tmp, /var, /etc (mount points)
  - **(3). Version**
    - Contains system code and binaries(/bin, /usr, /lib, /opt, and /samples)
    - Same function as the root in a non-sysplex system

Figure 8-1 Shared sysplex environment file systems

File systems in a sysplex sharing mode

The three file systems in Figure 8-1 are as follows:

1. This is the sysplex root HFS data set, which was created by running the BPXISYSR job. AUTOMOVE is the default and therefore is not specified, allowing another system to take ownership of this file system when the owning system goes down.
2. This is the system-specific HFS data set, which was created by running the BPXISYSS job. It must be mounted read-write. NOAUTOMOVE is specified because this file system is system-specific and ownership of the file system should not move to another system should the owning system go down. The MOUNTPOINT statement `/&SYSNAME.` will resolve to /SC64 during PARMLIB processing. This mount point is created dynamically at system initialization.
3. This is the old root HFS (version HFS). The recommendation is that it should be mounted read-only. Its mount point is created dynamically and the name of the HFS is the value specified on the VERSION statement in the BPXPRMxx member. AUTOMOVE is the default and therefore is not specified, allowing another system to take ownership of this file system when the owning system goes down.

8.2 Sysplex environment setup

- ❑ Create the Sysplex Root zFS
- ❑ Create the system-specific zFS
- ❑ Version HFS supplied by ServerPac
- ❑ Create OMVS couple data set (CDS)
- ❑ Define OMVS CDS to XCF
- ❑ Additional recommendations
  - ➢ Use standard naming convention for zFS data sets
  - ➢ Do not use &SYSNAME as a Root zFS qualifier
- ❑ Update BPXPRMxx member

Figure 8-2 Steps required to set up the shared sysplex environment

Defining the shared sysplex environment

Figure 8-2 shows the steps involved in defining the HFS data sets for sharing in read/write mode in a sysplex environment. Some details are included in the following sections.

Sysplex root

The sysplex root is an HFS data set that is used as the sysplex-wide root. This HFS data set must be mounted read-write and designated AUTOMOVE. Only one sysplex root is allowed for all systems participating in a shared file system environment. The sysplex root is created by invoking the BPXISYSR sample job in SYS1.SAMPLIB.

Note: No files or code reside in the sysplex root data set. It consists of directories and symbolic links only, and it is a small data set.

The sysplex root provides access to all directories. Each system in a sysplex can access directories through the symbolic links that are provided. Essentially, the sysplex root provides redirection to the appropriate directories, and it should be kept very stable; updates and changes to the sysplex root should be made as infrequently as possible.

Version HFS

The version HFS is the IBM-supplied root HFS data set. To avoid confusion with the sysplex root HFS data set, “root HFS” has been renamed to “version HFS.”

System-specific HFS

Directories in the system-specific HFS data set are used as mount points, specifically for /etc, /var, /tmp, and /dev. To create the system-specific HFS, run the BPXISYSS sample job in SYS1.SAMPLIB on each participating system (in other words, you must run the sample job separately for each system that will participate in a shared file system environment). IBM recommends that the name of the system-specific data set contain the system name as one of the qualifiers. This allows you to use the &SYSNAME. symbolic (defined in IEASYMxx) in BPXPRMxx.

OMVS couple data set

The couple data set (CDS) contains the sysplex-wide mount table and information about all participating systems, and all mounted file systems in the sysplex. To allocate and format a CDS, customize and invoke the BPXISCDS sample job in SYS1.SAMPLIB. The job will create two CDSs: one is the primary and the other is a backup that is referred to as the alternate. In BPXISCDS, you also specify the number of mount records that are supported by the CDS.

Define CDS to XCF

Following is the sample JCL with comments to define the CDS to XCF.

```
//STEP10    EXEC PGM=IXCL1DSU
//STEPLIB   DD   DSN=SYS1.MIGLIB,DISP=SHR
//SYSPRINT  DD   SYSOUT=A
//SYSIN     DD   *
 /* Begin definition for OMVS couple data set(1)            */
    DEFINEDS SYSPLEX(PLEX1)
 /* Name of the sysplex in which the OMVS couple data set is to be used
     DSN(SYS1.OMVS.CDS01) VOLSER(3390x1)
 /* The name and volume for the OMVS couple data set.
 The utility will allocate a new data set by the name specified on the
 volume specified.*/
 MAXSYSTEMS(8)
 /* Specifies the number of systems to be supported by the OMVS CDS.
    Default =     8          */
    NOCATALOG
 /* Default is not to CATALOG */
 DATA TYPE(BPXMCDS)
  /* The type of data in the data set being created for OMVS.
BPXMCDS is the TYPE for OMVS. */
 ITEM NAME(MOUNTS) NUMBER(500)
 /* Specifies the number of MOUNTS that can be supported by OMVS.*/
    Default =   100
    Suggested minimum =    10
    Suggested maximum = 35000  */
 ITEM NAME(AMTRULES) NUMBER(50)
  /* Specifies the number of automount rules that can be supported by OMV
     Default =    50
     Minimum =    50
     Maximum =  1000           */
```

BPXPRMxx member

You should also be aware that when SYSPLEX(YES) is specified in BPXPRMxx, each FILESYSTYPE in use within the participating group must be defined for all systems participating in a shared file system environment. The easiest way to accomplish this is to create a single BPXPRMxx member that contains file system information for each system participating in a shared file system environment.

8.3 File systems in a shared sysplex

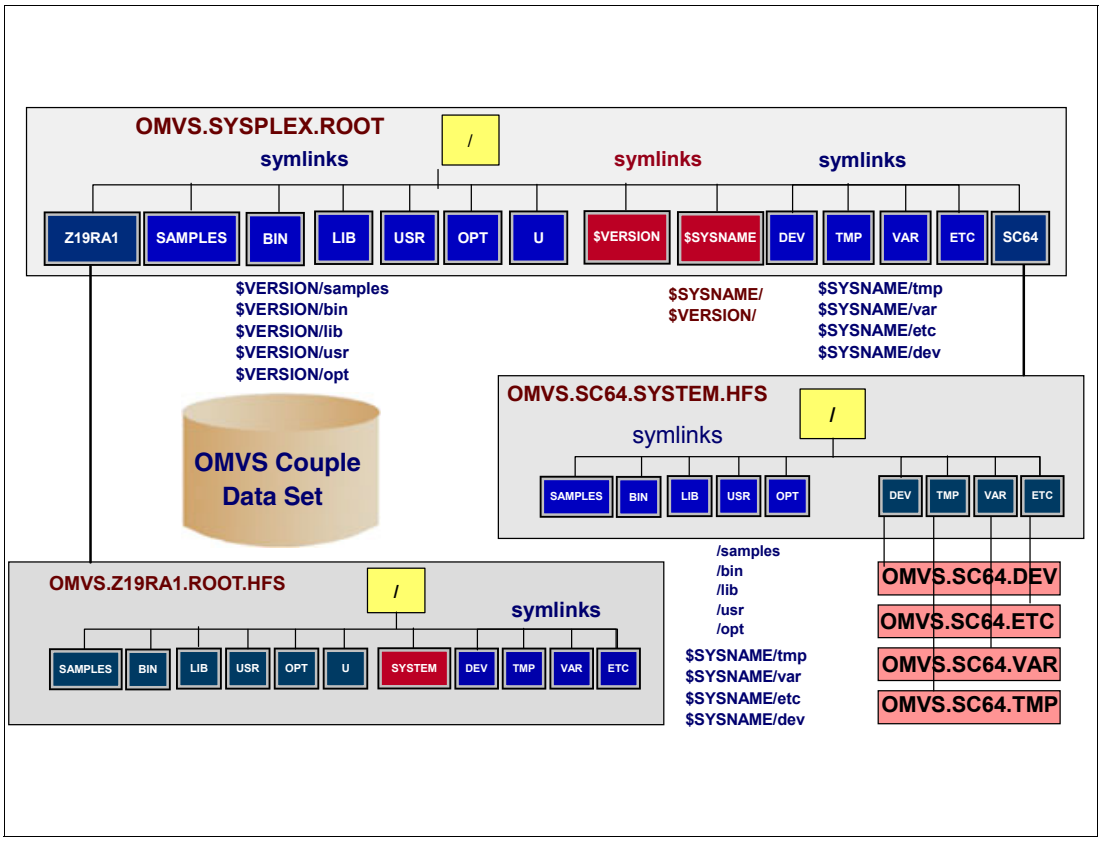


Figure 8-3 Shared sysplex HFS data sets

Sysplex root

No files or code reside in the sysplex root data set. It consists of directories and symbolic links only, and it is a small data set. The sysplex root provides access to all directories. Each system in a sysplex can access directories through the symbolic links that are provided. Essentially, the sysplex root provides redirection to the appropriate directories, and it should be kept very stable; updates and changes to the sysplex root should be made as infrequently as possible. The presence of symbolic links is transparent to the user.

System-specific zFS

The system-specific zFS data set should be mounted read-write, and should be designated NOAUTOMOVE. /etc, /var, /tmp, and /dev should also be mounted NOAUTOMOVE. In addition, IBM recommends that the name of the system-specific data set contains the system name as one of the qualifiers. This allows you to use the &SYSNAME. symbolic (defined in IEASYMxx) in BPXPRMxx.

Version root zFS

The version HFS is the IBM-supplied root zFS data set. To avoid confusion with the sysplex root zFS data set, “root HFS” has been renamed to “version zFS.” IBM supplies the version HFS in ServerPac. CBPDO users obtain the version HFS by following directions in the Program Directory. There is one “version HFS” for each set of systems participating in a shared file system environment and that are at the same release level (that is, using the same SYSRES volume).

8.4 Multiple systems: Different versions

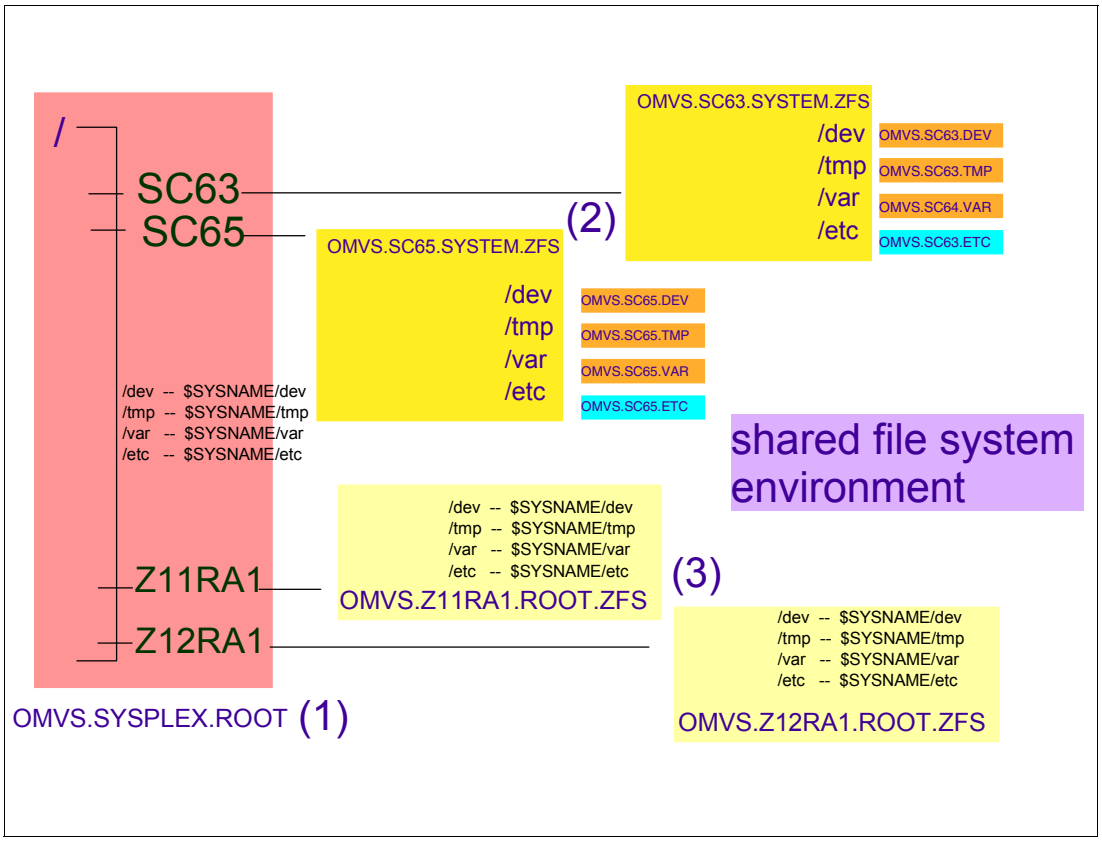


Figure 8-4 Shared sysplex with different z/OS levels

Using multiple roots in a shared environment

The version zFS is the IBM-supplied root HFS data set. To avoid confusion with the sysplex root zFS data set, “root zFS” has been renamed to “version zFS.”

IBM supplies the version HFS in ServerPac. CBPDO users obtain the version HFS by following directions in the Program Directory. There is one version zFS for each set of systems participating in a shared file system environment and that are at the same release level (that is, using the same SYSRES volume).

In Figure 8-4, there are two systems in the sysplex, SC64 and SC65. There are also two different z/OS releases, one on each system for z/OS V1R10 and z/OS V1R11. Therefore, there are two version roots mounted off of the sysplex root. The directories in the sysplex root are defined in the BPXPRMxx member for each system with the VERSION= statement, as follows:

```
VERSION('Z10RA1')          VERSION('Z11RA1')
SYSPLEX(YES)               SYSPLEX(YES)
```

8.5 Update BPXPRMxx for sysplex

```
VERSION('&SYSR1')
SYSPLEX(YES)

ROOT
FILESYSTEM('OMVS.SYSPLEX.ROOT')
TYPE  (ZFS)  MODE(RDWR)

MOUNT
FILESYSTEM('OMVS.&SYSNAME..SYSTEM.ZFS')
TYPE(ZFS)  MODE(RDWR)  UNMOUNT
MOUNTPOINT('/&SYSNAME.')

MOUNT
FILESYSTEM('OMVS.&SYSR1..ROOT.ZFS')
TYPE(ZFS)  MODE(READ)
MOUNTPOINT('/$VERSION')

MOUNT
FILESYSTEM('OMVS.&SYSNAME..ETC')
TYPE(ZFS)  MODE(RDWR)  UNMOUNT
MOUNTPOINT('/&SYSNAME./etc')
```

Created by BPXISYSR job

Created by BPXISYSS job

Version Root

System-specific /etc files

Figure 8-5 BPXPRMxx definitions for sysplex sharing

BPXPRMxx member for sysplex sharing

Sysplex sharing enables you to use one BPXPRMxx member to define all the file systems in the sysplex. This means that each participating system has its own BPXPRMxx member to define system limits, but shares a common BPXPRMxx member to define the file systems for the sysplex. This is done through the use of system symbolics, as shown in Figure 8-5. You can also have multiple BPXPRMxx members defining the file systems for individual systems in the sysplex. The following parameters set up HFS sharing in a sysplex:

- `SYSPLEX(YES)` sets up sysplex sharing for those who are participating in HFS or zFS data sharing. To participate in HFS data sharing, the systems must be at the OS/390 V2R9 level or later. Those systems that specify `SYSPLEX(YES)` make up the participating group for the sysplex.
- `VERSION('nnnn')` allows multiple releases and service levels of the binaries to coexist and participate in HFS sharing. `nnnn` is a qualifier to represent a level of the version HFS. The most appropriate values for `nnnn` are the name of the target zone, &SYSR1., or another qualifier meaningful to the system programmer. A directory with the value nnnn specified on VERSION will be dynamically created at system initialization under the sysplex root and will be used as a mount point for the version HFS.

BPXISYSR and BPXISYSS are provided as sample jobs to build root and system-specific HFS in SYS1.SAMPLIB.

Jobs BPXISYS1 and BPXISYS2 are also provided to build root and system-specific HFS directories and symlinks.

AUTOMOVE options

The AUTOMOVE|NOAUTOMOVE|UNMOUNT parameters on ROOT and MOUNT indicate what happens to the file system if the system that owns that file system goes down, as follows:

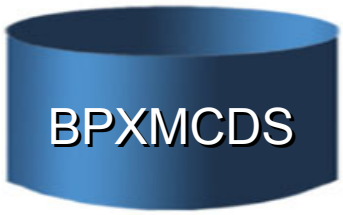
- AUTOMOVE specifies that ownership of the file system is automatically moved to another system. It is the default.
- NOAUTOMOVE specifies that the file system will not be moved if the owning system goes down and the file system is not accessible.
- UNMOUNT specifies that the file system will be unmounted when the system leaves the sysplex. This option is not available for automounted file systems.

You should define your version and sysplex root HFS data as AUTOMOVE, and define your system-specific file systems as UNMOUNT. Do not define a file system as NOAUTOMOVE or UNMOUNT and a file system underneath it as AUTOMOVE. If you do, the file system defined as AUTOMOVE will not be recovered after a system failure until that failing system has been restarted.

Attention: The system-specific file system should be mounted read-write, and should be designated AUTOMOVE(UNMOUNT). Also, /etc, /var, /tmp, and /dev should be mounted AUTOMOVE(UNMOUNT).

8.6 OMVS couple data set

```
ITEM NAME(MOUNTS) NUMBER(500)
/* Specifies the number of MOUNTS that can be supported by OMVS.*/
   Default =   100
   Suggested minimum =    10
   Suggested maximum = 35000  */
ITEM NAME(AMTRULES) NUMBER(50)
/* Specifies the number of automount rules that can be supported by OMVS */
    Default =    50
    Minimum =    50
    Maximum =  1000           */
```



OMVS couple data set

Display status: f bpxoinit,filesys=display,global

Figure 8-6 Defining the OMVS couple data set

Defining an OMVS couple data set

Shared sysplex support uses a type BPXMCDS couple data set (CDS) to maintain data about mounted file systems in the sysplex configuration. The primary and alternate CDSs are formatted, using the IXCL1DSU utility, with a maximum number of mount entries as specified in the NUMBER value that specifies the number of mounts, as shown in Figure 8-6.

In previous releases, users needed the capability to determine when the number of file system mounts in a shared sysplex was approaching the configured limit. Before z/OS V1R3 there was no way to easily determine when the mount limit specified in the BPXMCDS CDS was being approached. z/OS V1R3 introduced the possibility of monitoring the shared file system environment mount limits, specified in the CDS, by issuing a console message when the limit has almost been reached.

Once the mount limit is reached, no more file systems can be mounted in the sysplex until a larger type BPXMCDS CDS is enabled. Mount table limit monitoring allows an installation to detect when a primary CDS is reaching its mount table limit in order to begin corrective actions before denial of service.

Displaying the OMVS couple data set information

You can display the number of mount entries and the number in use by issuing the F BPXOINIT,FILESYS=DISPLAY,GLOBAL command.

```
f bpxoinit,filesys=display,global
BPXM027I COMMAND ACCEPTED.
BPXF242I 2007/10/13 19.18.53 MODIFY BPXOINIT,FILESYS=DISPLAY,GLOBAL
SYSTEM   LFS VERSION ---STATUS--------------------- RECOMMENDED ACTION
SC64       1.  9.  0 VERIFIED                       NONE
SC65       1.  9.  0 VERIFIED                       NONE
SC70       1.  9.  0 VERIFIED                       NONE
SC63       1.  9.  0 VERIFIED                       NONE
CDS VERSION=  2        MIN LFS VERSION=  1.  9.  0
DEVICE NUMBER OF LAST MOUNT=     558
MAXIMUM MOUNT ENTRIES=    1000   MOUNT ENTRIES IN USE=     222
MAXIMUM AMTRULES=           50   AMTRULES IN USE=            2
MAXSYSTEM=                   8
BPXF040I MODIFY BPXOINIT,FILESYS PROCESSING IS COMPLETE.
```

8.7 OMVS couple data set commands

- Output of D XCF,COUPLE,TYPE=BPXMCDS command has been enhanced to include the USS CDS information
  - Provide USS CDS information to help you manage the shared file system configuration to make the functions easier to understand and use and improve system programmer and operator productivity
- Output of SETXCF COUPLE,ACOUPLE=(AlternateCDS),TYPE=BPXMCDS command has been enhanced to better describe the specific error

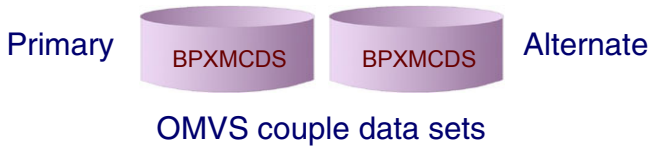


Figure 8-7 OMVS couple data set command changes in z/OS V1R9

OMVS CDS display command

In a sysplex environment, the OMVS couple data sets are defined when using shared file system support. To avoid a single point of failure in the sysplex, there is a primary CDS and an alternate CDS. These CDSs are formatted by a utility provided by z/OS. The alternate CDS must be compatible with the primary CDS (for example, formatted with equal or greater capacity). If the primary CDS fails, the sysplex automatically makes the alternate CDS the primary. The SETXCF COUPLE,PSWITCH command is issued to dynamically make the alternate CDS the primary in case they need to expand the size of the primary CDS, or change the CDS to a different device to circumvent device errors, or take advantage of a different device, in a non-disruptive way.

With z/OS V1R9, the D XCF,COUPLE,TYPE=BPXMCDS command now makes it possible, on a query of the OMVS CDS, to display the VERSION (system software) and the value keywords (MOUNTS and AMTRULES) used to format the CDS.

Previously, when using commands to bring a new CDS into a sysplex that is somehow not compatible with the one that is currently used, the response only described the mismatch in terms of internal information which does not help to determine any incompatibilities with the alternate CDS.

The enhancement displays the output to include the version and current value of MOUNTS and AMTRULES for both the primary and alternate CDS.

8.8 D XCF,COUPLE,TYPE=BPXMCDS

```
IXC358I  18.21.35  DISPLAY XCF 182
BPXMCDS COUPLE DATA SETS
PRIMARY    DSN: SYS1.XCF.OMVS02
           VOLSER: SBOX63     DEVN: 836F
           FORMAT TOD         MAXSYSTEM
           12/18/2003 10:29:20        8
           ADDITIONAL INFORMATION:
             FORMAT DATA
              VERSION(2)
              MOUNTS(1000)   AMTRULES(50)
ALTERNATE  DSN: SYS1.XCF.OMVS03
           VOLSER: SBOX64     DEVN: 8068
           FORMAT TOD         MAXSYSTEM
           12/18/2003 10:29:21        8
           ADDITIONAL INFORMATION:
             FORMAT DATA
              VERSION(2)
              MOUNTS(1000)   AMTRULES(50)
BPXMCDS IN USE BY ALL SYSTEMS
```

Figure 8-8 D XCF,COUPLE,TYPE=BPXMCDS command

The D XCF,COUPLE,TYPE=BPXMCDS command

Figure 8-8 shows an example of an invocation of the D XCF,COUPLE,TYPE=BPXMCDS command. The following message text is the enhanced output:

```
FORMAT DATA
VERSION(2)
MOUNTS(500)   AMTRULES(50)
```

z/OS UNIX provides the version, and the value of MOUNTS and AMTRULES, which are keywords for the z/OS UNIX format utility. z/OS UNIX provides a sample JCL job to format the CDS in SYS1.SAMPLIB(BPXISCDS)). The old respond text used to be `NOT PROVIDED`.

The output shown is for the primary and alternate CDS data sets.

8.9 Change to the SETXCF command

- In this example, the primary CDS was formatted with MOUNTS value of 500 and AMTRULES value of 50
- The alternate CDS was formatted with MOUNTS value of 400 and AMTRULES value of 50
- The alternate CDS is not compatible with the primary because it has a smaller value for MOUNTS
  - SETXCF COUPLE,ACOUPLE… fails because the alternate CDS must be formatted with an equal or greater value for MOUNTS and AMTRULES with the following message:

ALLOWABLE SIZE OF BPXFSMPT RECORDS IS LESS THAN CURRENT PRIMARY RELEVANT BPXMCDS COUPLE DATA SET FORMAT INFORMATION

SETXCF COUPLE,ACOUPLE=(POSIX.OMVS.CDS03,AUXVOL),TYPE=BPXMCDS

Figure 8-9 Changes to the SETXCF command

The SETXCF command

Figure 8-9 shows an example of a SETXCF COUPLE,ACOUPLE=(Alternate_CDS),TYPE=BPXMCDS command.

In this example, the primary CDS was formatted with a MOUNTS value of 500 and an AMTRULES value of 50, but the alternate CDS was formatted with a MOUNTS value of 400 and an AMTRULES value of 50.

Incompatibility in this example

The alternate CDS is not compatible with the primary because it has a smaller value for MOUNTS. So the command fails, because the alternate CDS must be formatted with equal or greater values for MOUNTS and AMTRULES in order for this command to work. Before this enhancement in z/OS V1R9, the information provided for the reason of the failure was the following:

```
ALLOWABLE SIZE OF BPXFSMPT RECORDS IS LESS THAN CURRENT PRIMARY RELEVANT
BPXMCDS COUPLE DATA SET FORMAT INFORMATION
```

BPXFSMPT RECORDS is the internal record name of USS. With this message, it was not easy to determine why the alternate CDS was incompatible.

8.10 New message for command failure

```
SETXCF COUPLE,ACOUPLE=(POSIX.OMVS.CDS03,AUXVOL),TYPE=BPXMCDS
IXC309I SETXCF COUPLE,ACOUPLE REQUEST FOR BPXMCDS WAS ACCEPTED
IXC260I ALTERNATE COUPLE DATA SET REQUEST FROM SYSTEM 405
SY1 FOR BPXMCDS IS NOW BEING PROCESSED.
DATA SET:  POSIX.OMVS.CDS03
IEF196I IEF237I 0483 ALLOCATED TO SYS00038
IXC255I UNABLE TO USE DATA SET 407
POSIX.OMVS.CDS03
AS THE ALTERNATE FOR BPXMCDS:
ALLOWABLE SIZE OF BPXFSMPT RECORDS IS LESS THAN CURRENT PRIMARY
RELEVANT BPXMCDS COUPLE DATA SET FORMAT INFORMATION
PRIMARY
  FORMAT LEVEL:    VERSION(2)
  FORMAT KEYWORDS: MOUNTS(500)   AMTRULES(50)
ALTERNATE
  FORMAT LEVEL:    VERSION(2)
  FORMAT KEYWORDS: MOUNTS(400)   AMTRULES(50)
IXC250I ALTERNATE COUPLE DATA SET REQUEST FAILED FOR DATA SET 408
POSIX.OMVS.CDS03 FOR BPXMCDS:
CONSISTENCY CHECKING FAILED FOR THE NEW ALTERNATE DATA SET
IEF196I IEF285I   POSIX.OMVS.CDS03                              KEPT
IEF196I IEF285I   VOL SER NOS= AUXVOL.
```

Figure 8-10 New message when command fails

New failure message with z/OS V1R9

In this example, the primary CDS was formatted with a MOUNTS value of 500 and an AMTRULES value of 50, but the alternate CDS was formatted with a MOUNTS value of 400 and an AMTRULES value of 50.

The alternate CDS is not compatible with the primary because it has a smaller value for MOUNTS, so the command SETXCF COUPLE,ACOUPLE… failed, because the alternate CDS must be formatted with equal or greater values for MOUNTS and AMTRULES in order for this command to work.

The new message shown in the command output for the failure is:

```
IXC250I ALTERNATE COUPLE DATA SET REQUEST FAILED FOR DATA SET 408
POSIX.OMVS.CDS03 FOR BPXMCDS:
CONSISTENCY CHECKING FAILED FOR THE NEW ALTERNATE DATA SET
```

8.11 File sharing in a sysplex

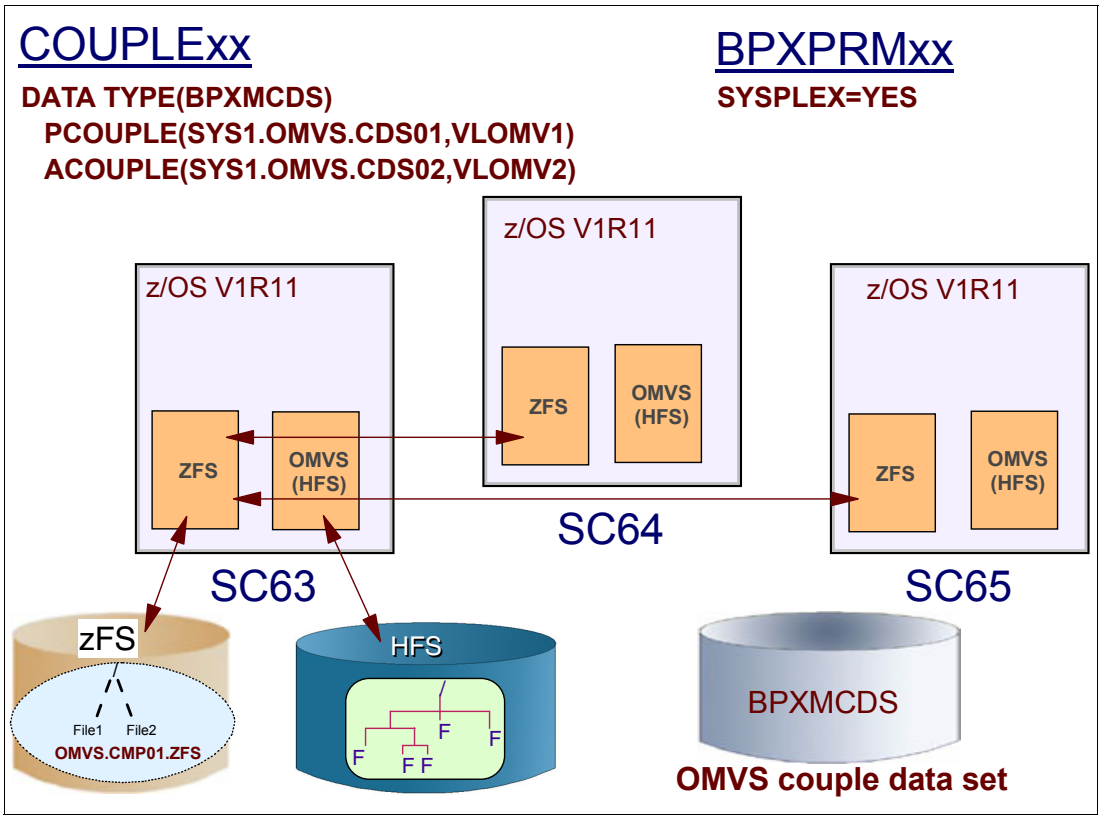


Figure 8-11 File sharing in a sysplex environment

File sharing in a sysplex

Before OS/390 UNIX V2R9, users could have read/write access only to data in file systems mounted on their own system. With a shared file system environment, users have greater access to the file systems in a sysplex. They have read/write access to file systems that are mounted on other systems.

With a shared file system environment, all file systems that are mounted by a system participating in a shared file system environment are available to all participating systems. In other words, once a file system is mounted by a participating system, that file system is accessible by any other participating system. It is not possible to mount a file system so that it is restricted to just one of those systems. Consider a sysplex that consists of three systems, SC63, SC64, and SC65:

- A user logged onto any system can make changes to file systems mounted on /u, and those changes are visible to all systems.
- The system programmer who manages maintenance for the sysplex can change entries in both /etc file systems from either system.

The couple data set (CDS) contains the sysplex-wide mount table and information about all participating systems, and all mounted file systems in the sysplex. To allocate and format a CDS, customize and invoke the BPXISCDS sample job in SYS1.SAMPLIB. The job will create two CDSs: one is the *primary* and the other is a backup that is referred to as the *alternate*. In BPXISCDS, you also specify the number of mount records that are supported by the CDS. The CDS must be defined in the COUPLExx PARMLIB member. A shared file system environment must be defined in the BPXPRMxx PARMLIB member.

8.12 Mounting shared sysplex file systems

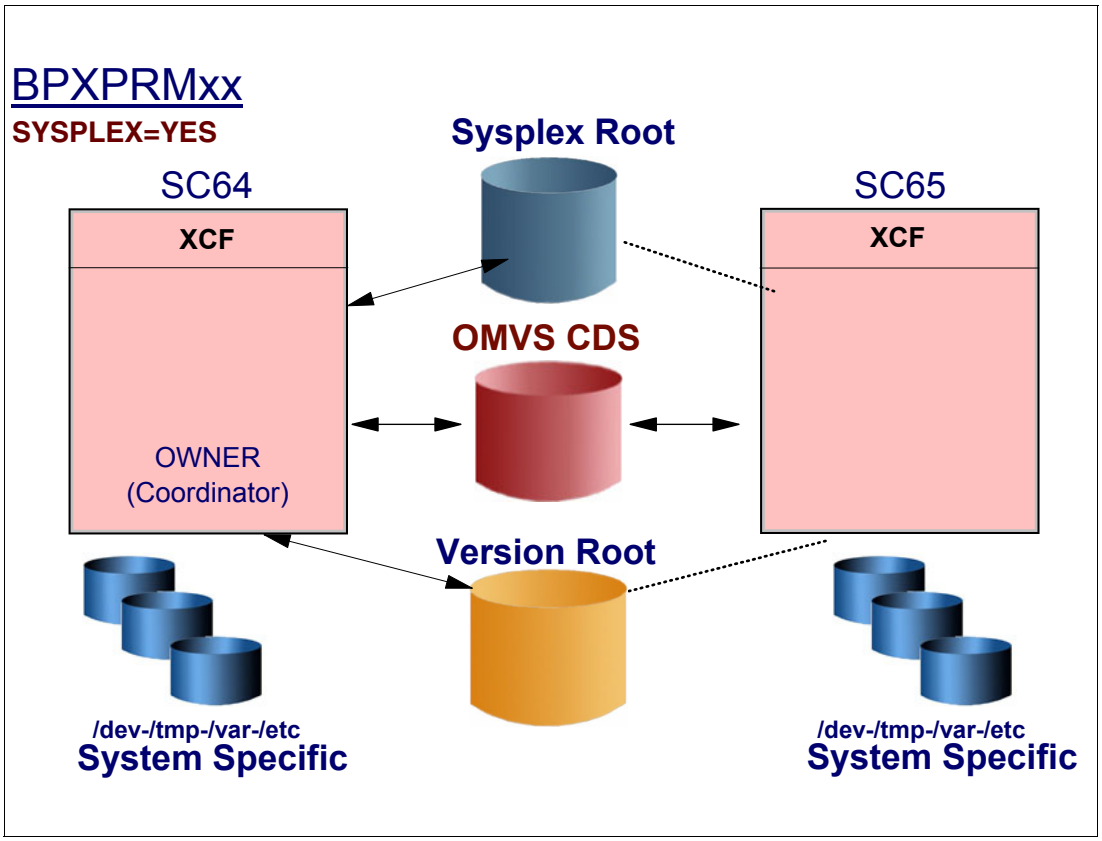


Figure 8-12 Shared sysplex mounted file systems

Shared sysplex environment

With a shared sysplex environment, sharing HFS and zFS file systems, each system must have specific HFS data sets for each of these file systems, meaning the two systems each have HFS data sets for /etc, /tmp, /var, and /dev. The file systems are then mounted under the system-specific HFS, as shown in Figure 8-12. With shared file system environment support, one system can access system-specific file systems on another system. For example, while logged onto SC65, you can gain read-write access to SC64's /tmp by specifying /SC64/tmp/.

BPXPRMxx PARMLIB specification

You should also be aware that when SYSPLEX(YES) is specified, each FILESYSTYPE in use within the participating group must be defined for all systems participating in a shared file system environment. The easiest way to accomplish this is to create a single BPXPRMxx member that contains file system information for each system participating in a shared file system environment. If you decide to define a BPXPRMxx member for each system, the FILESYSTYPE statements must be identical on each system.

The sysplex root provides access to all directories. Each system in a sysplex can access directories through the symbolic links that are provided. Essentially, the sysplex root provides redirection to the appropriate directories, and it should be kept very stable; updates and changes to the sysplex root should be made as infrequently as possible.

Note: The first system that IPLs in the sysplex becomes the owner of the sysplex root.

8.13 Accessing shared sysplex file systems

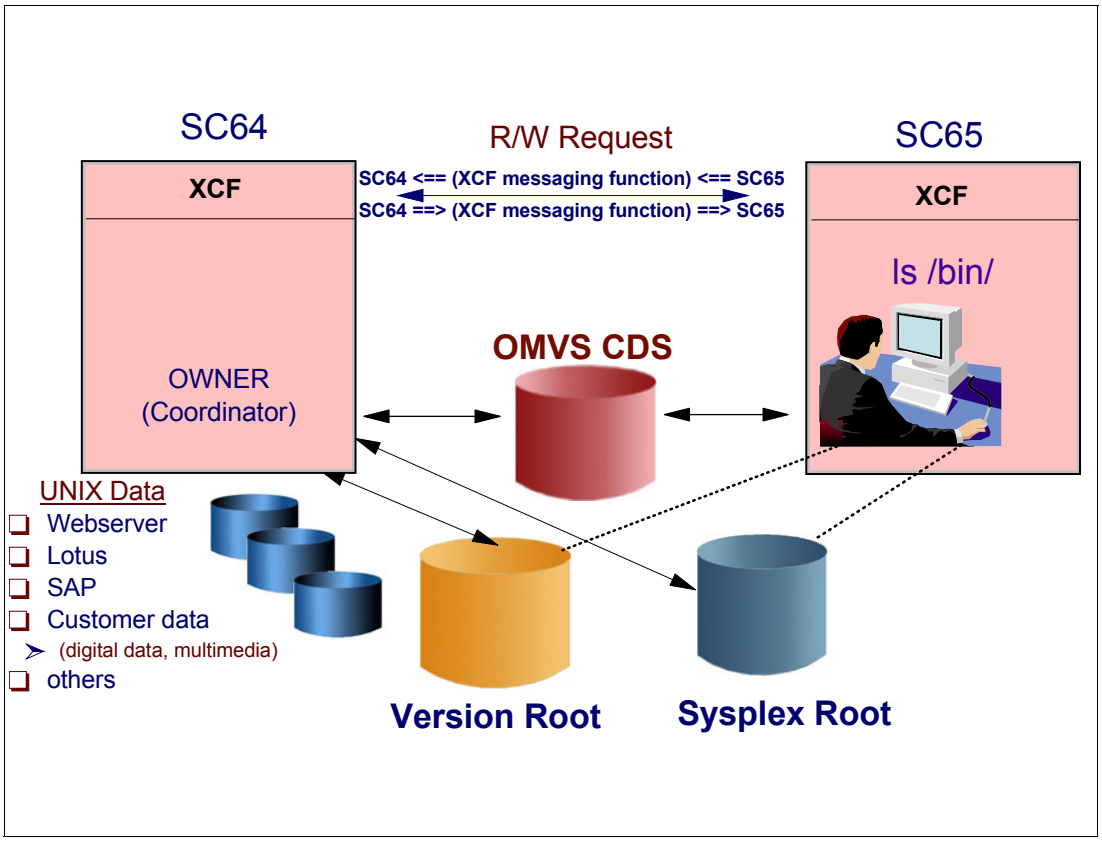


Figure 8-13 Accessing shared sysplex mounted file systems

Accessing file systems

The intersystem communication required to provide the additional availability and recoverability associated with z/OS UNIX in a shared file system environment support affects response time and throughput on R/W file systems being shared in a sysplex.

For example, assume that a user on SC64 requests a read on a file system mounted R/W and owned by SC65. Using shared file system environment support, SC64 sends a message requesting this read to SC65 via an XCF messaging function:

SC64 ===> (XCF messaging function) ===> SC65

After SC65 gets this message, it issues the read on behalf of SC64, and gathers the data from the file. It then returns the data via the same route the request message took:

SC65 ===> (XCF messaging function) ===> SC64

8.14 Shared file system AUTOMOVE takeover

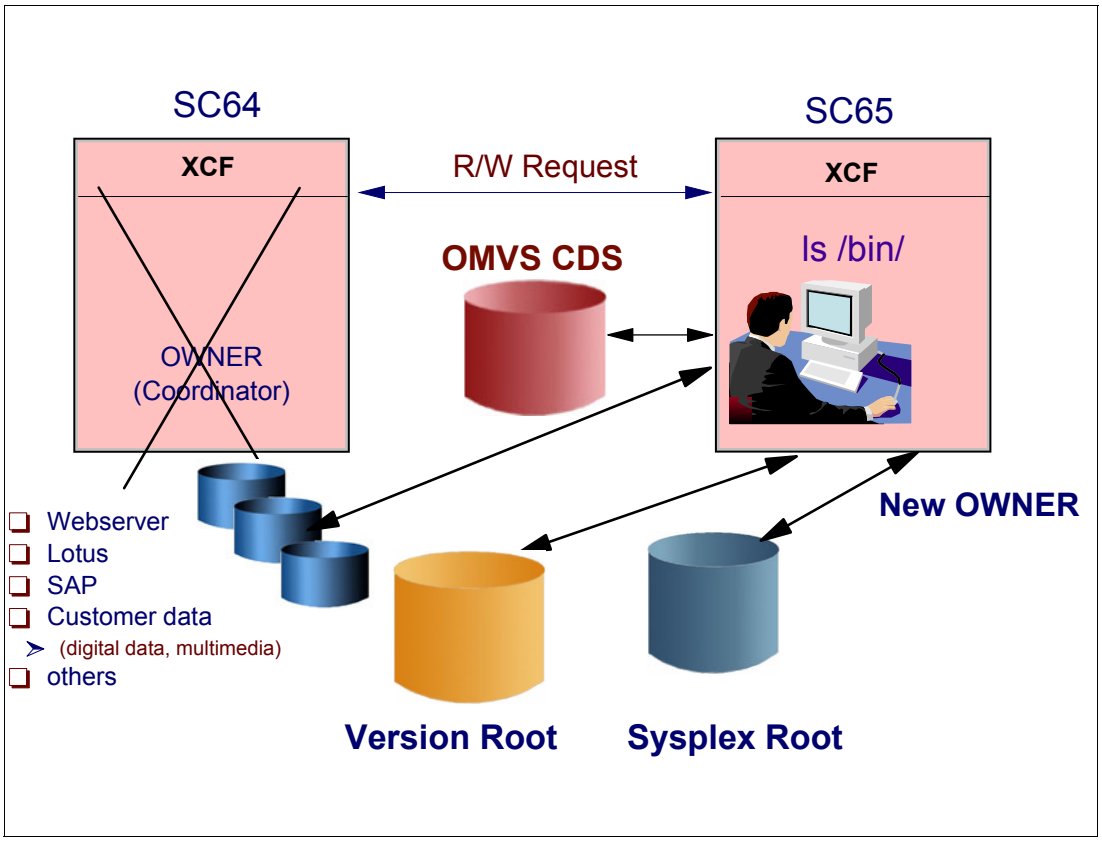


Figure 8-14 AUTOMOVE options in a shared sysplex

AUTOMOVE takeover of file systems

File system recovery in a shared file system environment takes into consideration file system specifications such as AUTOMOVE, NOAUTOMOVE, UNMOUNT, and whether or not the file system is mounted read-only or read-write.

Generally, when an owning system fails, ownership over its automove-mounted file system is moved to another system and the file is usable. However, if a file system is mounted read-write and the owning system fails, then all file system operations for files in that file system will fail. This happens because data integrity is lost when the file system owner fails. All files should be closed (BPX1CLO) and reopened (BPX1OPN) when the file system is recovered.

Read-only file systems

For file systems that are mounted read-only, specific I/O operations that were in progress at the time the file system owner failed may need to be started again.

Moving file systems

In some situations, even though a file system is mounted AUTOMOVE, ownership of the file system may not be immediately moved to another system. This may occur, for example, when a physical I/O path from another system to the volume where the file system resides is not available. As a result, the file system becomes unowned; if this happens, you will see message BPXF213E. This is true if the file system is mounted either read-write or read-only. The file system still exists in the file system hierarchy so that any dependent file systems that

are owned by another system are still usable. However, all file operations for the unowned file system will fail until a new owner is established. The shared file system environment support will continue to attempt recovery of AUTOMOVE file systems on all systems in the sysplex that are enabled for a shared file system environment. Should a subsequent recovery attempt succeed, the file system transitions from the unowned to the active state.

NOAUTOMOVE file systems

File systems that are mounted NOAUTOMOVE or UNMOUNT will become unowned when the file system owner exits the sysplex. The file system will remain unowned until the original owning system restarts or until the unowned file system is unmounted. Because the file system still exists in the file system hierarchy, the file system mount point is still in use.

An unowned file system is a mounted file system that does not have an owner. The file system still exists in the file system hierarchy. As such, you can recover or unmount an unowned file system.

8.15 Moving file systems in a sysplex

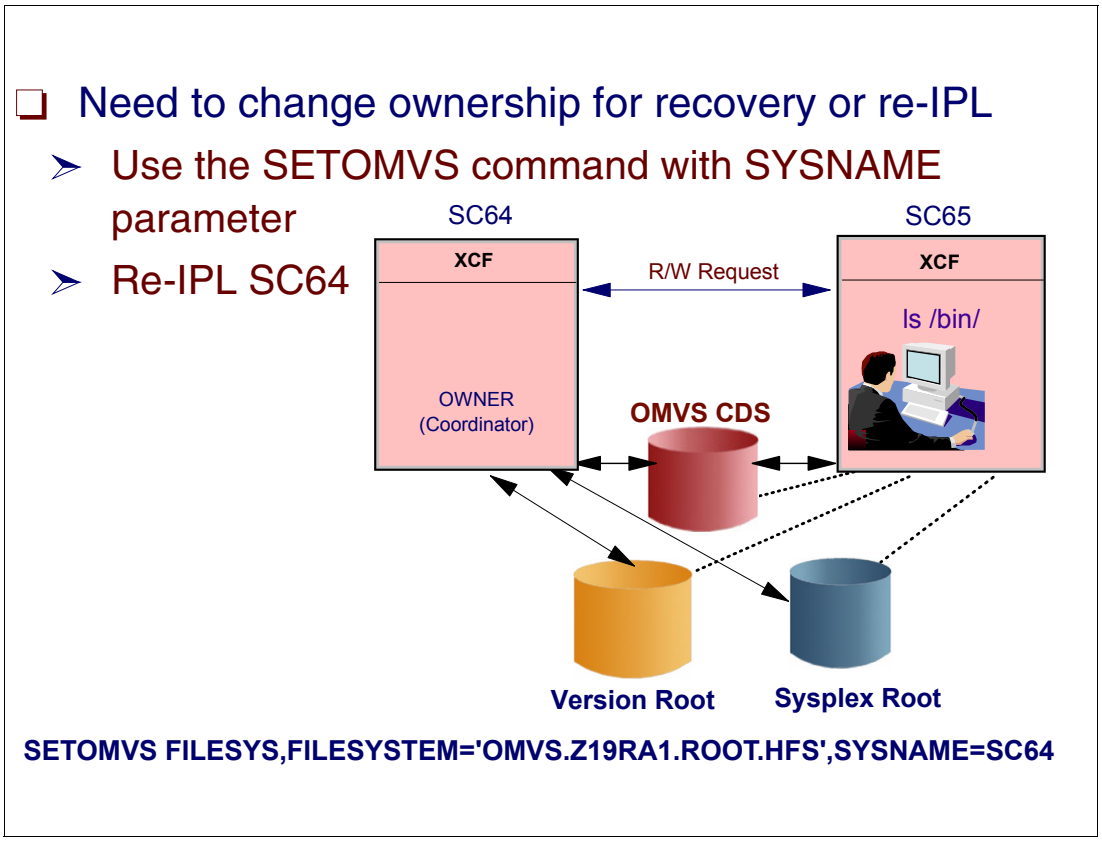


Figure 8-15 Command to move a file system in a sysplex environment

Commands to move file systems

You may need to change ownership of the file system for recovery or re-IPLing. To check for file systems that have already been mounted, use the **df** command from the shell. The **setomvs** command used with the `FILESYS`, `FILESYSTEM`, `mount point` and `SYSNAME` parameters can be used to move a file system in a sysplex, or you can use the **chmount** command from the shell. However, do not move these two types of file systems:

- System-specific file systems.
- File systems that are being exported by DFS. You have to unexport them from DFS first and then move them.

Examples of moving file systems are:

- To move ownership of the file system that contains /u/user1 to SC64:
  `chmount -d SC64 /u/user1`
- To move ownership of the payroll file system from the current owner to SC64 using `SETOMVS`, issue:

  ```
  SETOMVS FILESYS,FILESYSTEM='POSIX.PAYROLL.HFS',SYSNAME=SC64
  ```
- Assuming the mount point is over directory /PAYROLL:

  ```
  SETOMVS FILESYS,mountpoint='/PAYROLL',SYSNAME=SC64
  ```
- Move the root file system:

  ```
  SETOMVS FILESYS,FILESYSTEM='OMVS.Z19RA1.ROOT.HFS',SYSNAME=SC64
  ```

8.16 Requests to LFS to PFS to files

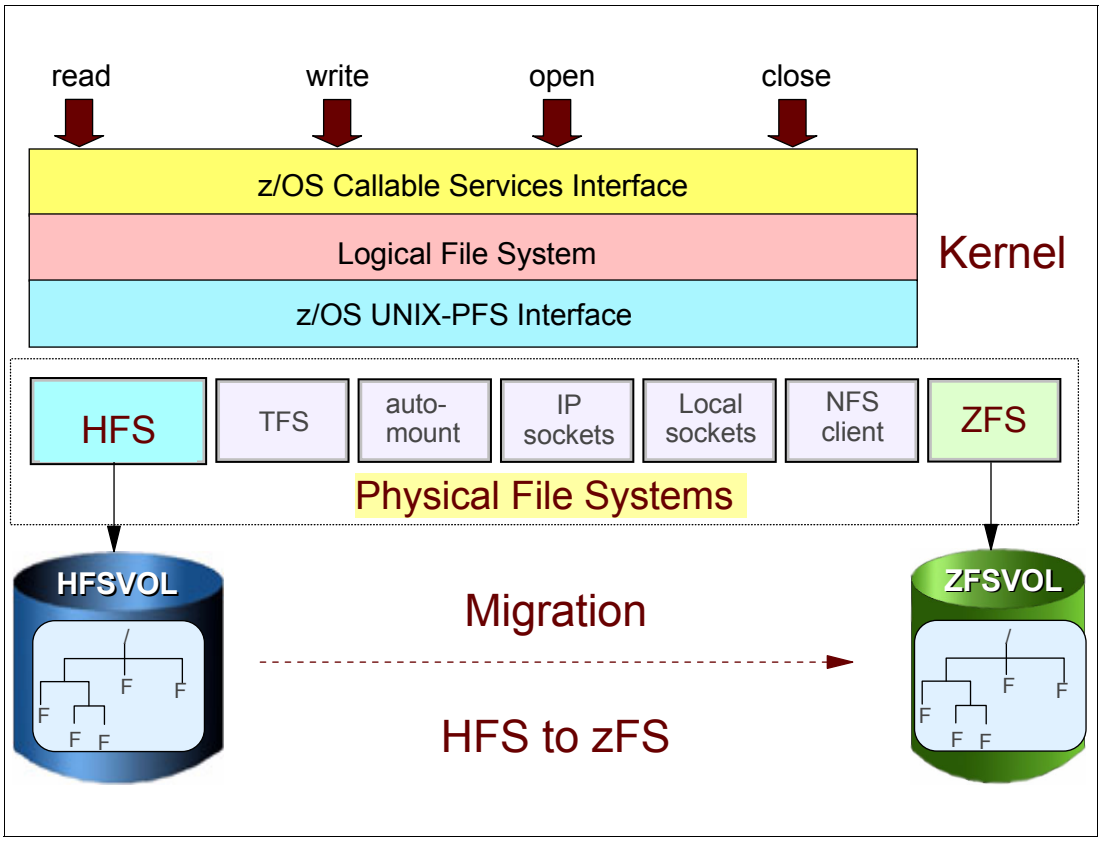


Figure 8-16 The logical file system

Logical file system (LFS)

The PFS interface is a set of protocols and calling interfaces between the logical file system (LFS) and the PFSs that are installed on z/OS UNIX. PFSs mount and unmount file systems and perform other file operations.

The LFS is called by POSIX programs, non-POSIX z/OS UNIX programs, and VFS servers.

LFS implementation

Changes were made to LFS termination of a PFS, such as zFS, in order to improve the availability of file systems on the system where a PFS is terminating.

The old design of PFS termination is that file systems for the terminating PFS, and subtrees of those file systems, get moved to another system (if locally owned), and then get locally unmounted and become unavailable on the system where the PFS is terminating. If they could not be moved, then they became globally unmounted.

This new design is that the ownership of these file systems can be moved to another system in the sysplex, and then allow for function-shipping requests on the system where a PFS is terminating and avoid the local unmounts. This provides improved availability of file systems.

8.17 Systems accessing file systems

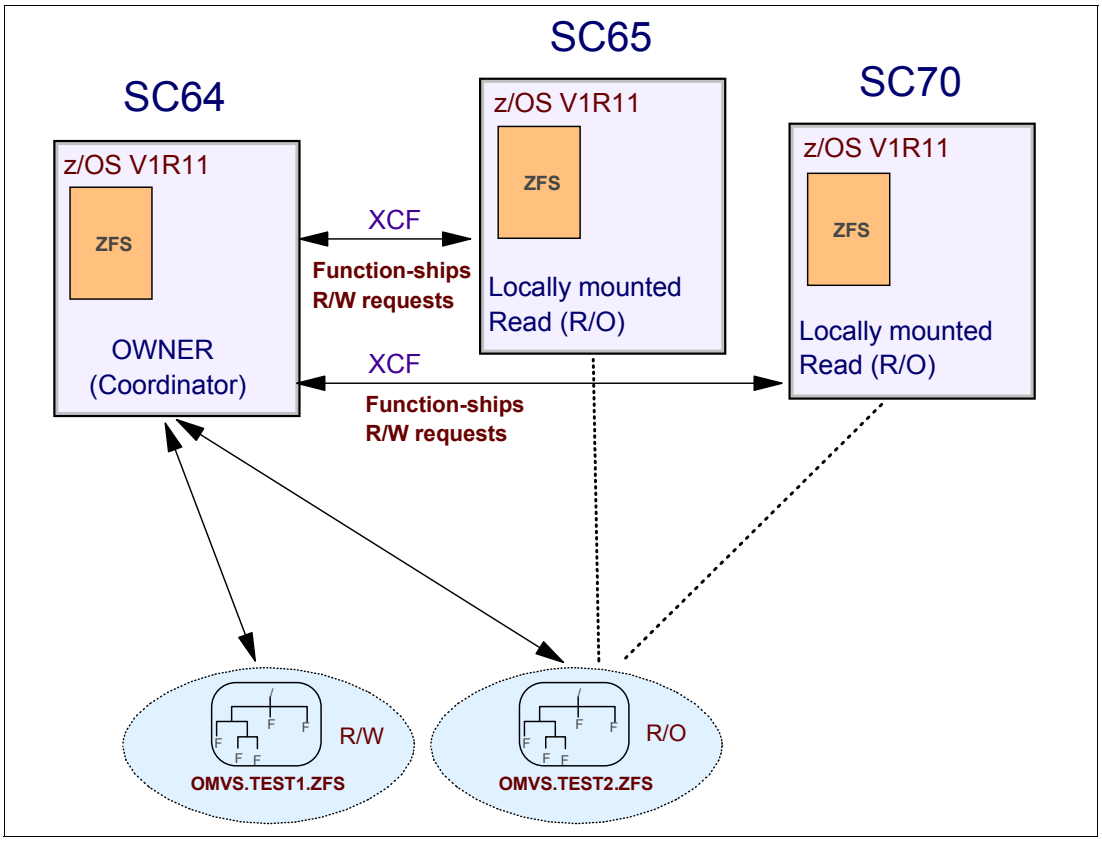


Figure 8-17 zFS environment with three systems accessing two file systems

Example of file access

Function shipping is the process of requesting function from the owning file system and returning the response to the requester through XCF communications.

For read/write mounted file systems, file requests are function shipped to the owning system. The owning system is the only system where the file system is locally mounted and the only system have a longer path length through XCF communications. Read/write mounted file systems are referred to as being non-sysplex aware.

In Figure 8-17 you see a USS sysplex sharing environment with three systems that share two zFS file systems. In this example, the following is taking place:

- System SY1 owns the zFS file systems OMVS.TEST1.ZFS (R/W) and OMVS.TEST2.ZFS (R/O).
- The other two systems, SY2 and SY3, have the R/O zFS file system locally mounted as read (R/O).
- Any R/W requests from SY2 and SY3 to the R/W file system owned by SY1 must be passed through the XCF messaging function, which are referred to as function-shipping requests.

8.18 zFS sharing mode terminology

- Sysplex-unaware
  - A file system is sysplex-unaware if the PFS supporting that file system requires it to be accessed through the remote owning system from all other systems in a sysplex (allowing only one connection for update at a time) for a particular mode (read-only or read-write).
  - Access from a system not being the owner (sometimes referred to as being a client) is provided through XCF communication to the owning system (sometimes called “function shipping”) - This is controlled by z/OS UNIX and the interface is also known as a PFS named XPFS (“Cross System PFS”)

Figure 8-18 Sysplex-unaware mounted file system

Sysplex-unaware file systems

A file system is non-sysplex aware (or sysplex-unaware) if the PFS (Physical File System) supporting that file system requires it to be accessed through the remote owning system from all other systems in a sysplex (allowing only one connection for update at a time) for a particular mode (read-only or read-write).

The system that connects to the file system is called the file system owner. Other systems’ access is provided through XCF communication with the file system owner. In non-sysplex aware zFS, file requests for read-write mounted file systems are function shipped to the owning system by z/OS UNIX. The owning system is the only system where the file system is locally mounted and the only system that does I/O to the file system.

8.19 Defining zFS as sysplex-unaware

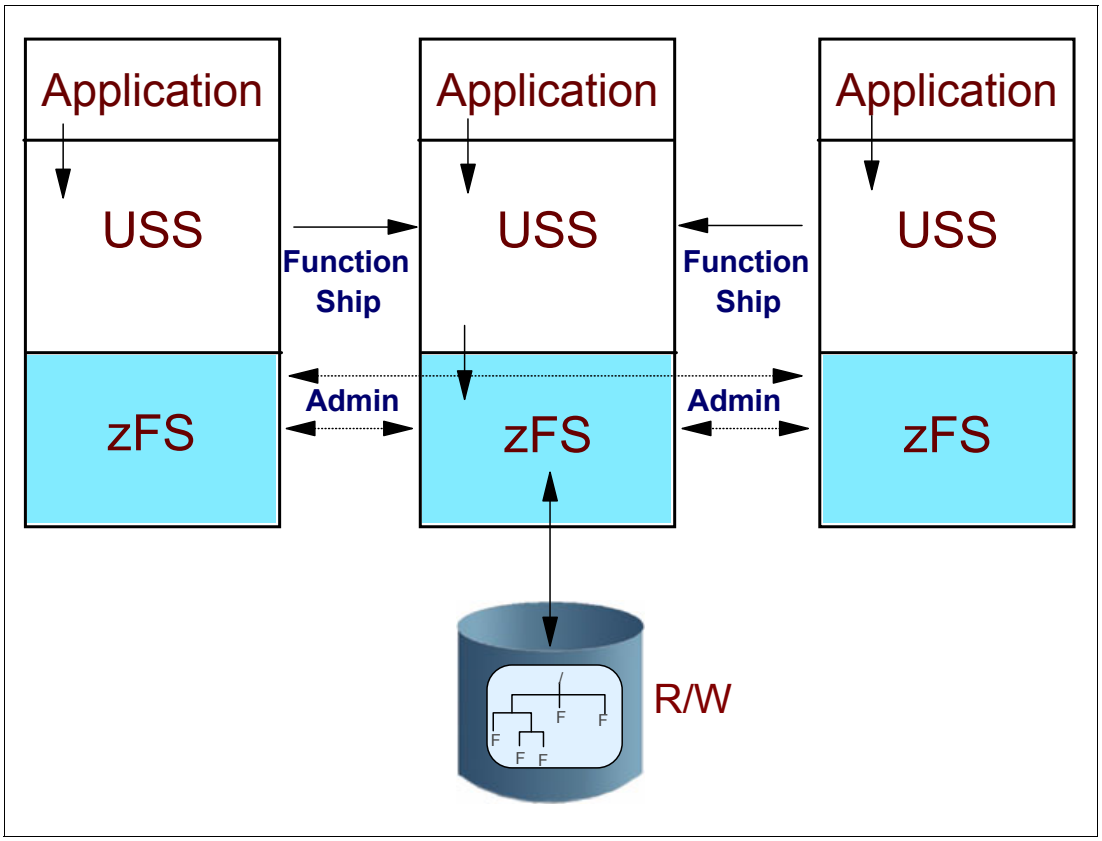


Figure 8-19 Shared file systems in a sysplex

Function shipping in a sysplex

For read/write mounted file systems, file requests are function shipped to the owning system. The owning system is the only system where the file system is locally mounted and the only system that does I/O to the file system. File requests from systems other than the owning system have a longer path length through XCF communications. Read/write mounted file systems are referred to as being non-sysplex aware.

When a file system is mounted read/write (as shown in Figure 8-19), the mount request is sent to the local physical file system (in this case, zFS) and zFS opens the file system data set (for read/write). If the mount is successful on that system, z/OS UNIX sends a signal to the other sysplex member systems to issue a catch-up mount on each system. Each z/OS UNIX on each other system then reads the CDS and determines that it needs to record that the file system is mounted read/write and that system is the owning system. The local zFS on the other systems does not receive a local mount request. The mount on that system is “logically” mounted on each of the other systems.

Catch-up mount

A catch-up mount is when a file system mount is successful on a system in a shared file system environment. z/OS UNIX automatically issues a corresponding local mount, the catch-up mount, to every other system's PFS that is running sysplex-aware for that mode (read-write or read-only). If the corresponding local mount is successful, z/OS UNIX does not function ship from that system to the z/OS UNIX owning system when that file system is accessed.

8.20 zFS File systems with APAR OA29712

- Beginning with z/OS V1R11, zFS introduced the ability zFS read-write filesystems to be sysplex-aware
  - This means that USS sends all file requests directly down to the local zFS physical file system (PFS)
  - It does not “function ship” the requests
- For z/OS V1R11 - APAR OA29712 updates the support that was shipped with V1R11
  - USS SPE OA29712 is a pre-requisite zFS SPE OA29619
  - Conditioning APAR OA29786 must be installed and active on all your z/OS V1R9 and V1R10 systems
- z/OS V1R12 contains OA29712 making V1R11 and V1R12 at equivalent levels for zFS

Figure 8-20 Sysplex-aware mounted file systems

Sysplex-aware file systems

A PFS (Physical File System) that is sysplex-aware for a particular mode (read-only or read-write) allows file requests for file systems that are mounted in that mode to be handled by that local PFS (that is, they are locally mounted on that system). z/OS UNIX does not function ship those file requests to the z/OS UNIX owning system. Note that if the file system volume is not online to a system, then the system becomes a client to the file system through function shipping with the owner because the local mount is unsuccessful. Beginning with z/OS V1R11, you can enable zFS to be sysplex-aware for read-write mounted file systems.

Note: With z/OS V1R11, if you enable zFS sysplex-aware for read/write zFS file systems, you see a performance improvement in most shared file system environments.

If you decide to run zFS sysplex-aware for read/write, roll this support through all your sysplex members (this assumes all your members are using z/OS V1R11). You cannot change from zFS non-sysplex aware to zFS sysplex-aware dynamically.

File systems mounted as R/O have been sysplex-aware from the beginning, and HFS and zFS file systems mounted in R/W mode were sysplex-unaware in all releases prior to z/OS V1R11.

APAR 29712

APAR OA29712 updates the support that was shipped with V1R11 for zFS. Before you can run zFS sysplex-aware on a file system basis, you must have z/OS V1R11 zFS APAR OA29619 and z/OS V1R11 UNIX APAR OA29712 installed and active on all of your z/OS V1R11 systems. In addition, conditioning APAR OA29786 must be installed and active on all your z/OS V1R9 and V1R10 systems for subsequent zFS V1R11 APAR OA29619. Finally, if you use the SMB server, APAR OA31112 must also be installed.

APAR 29786

APAR OA29786 will verify that all zFS systems in the sysplex are capable of using the new zOS V1R11 zFS IOEFSPRM option, SYSPLEX=FILESYS, which will allow sysplex support on a file system basis. When this new function is shipped on zFS V1R11, OA29786 will allow zFS V1R9 and zFS V1R10 to be able to also specify a new zFS parameter that indicates a specific file system will use the sysplex file support.

APAR 29619

zFS APAR OA29619 provides the capability of setting read-write sysplex-awareness on a file system (mount) basis, which requires the USS support provided with this APAR. USS is providing support for allowing a PFS to set the read-write sysplex-awareness of a file system during a mount. Prior to this support, the sysplex-awareness of a PFS that was set during PFS initialization would apply to all file systems mounted on that system for that PFS. This affects all flows where USS sends a vfs_mount to the PFS, which includes mount, catchup mount, remount, dead system recovery, unowned file system recovery or catchup, newroot/altroot, move, and PFS termination/restart.

Note: z/OS V1R12 contains OA29712, setting V1R11 and V1R12 at equivalent levels for zFS, thus setting z/OS V1R11 and z/OS V1R12 at equivalent levels for zFS.

8.21 Defining zFS as sysplex-aware

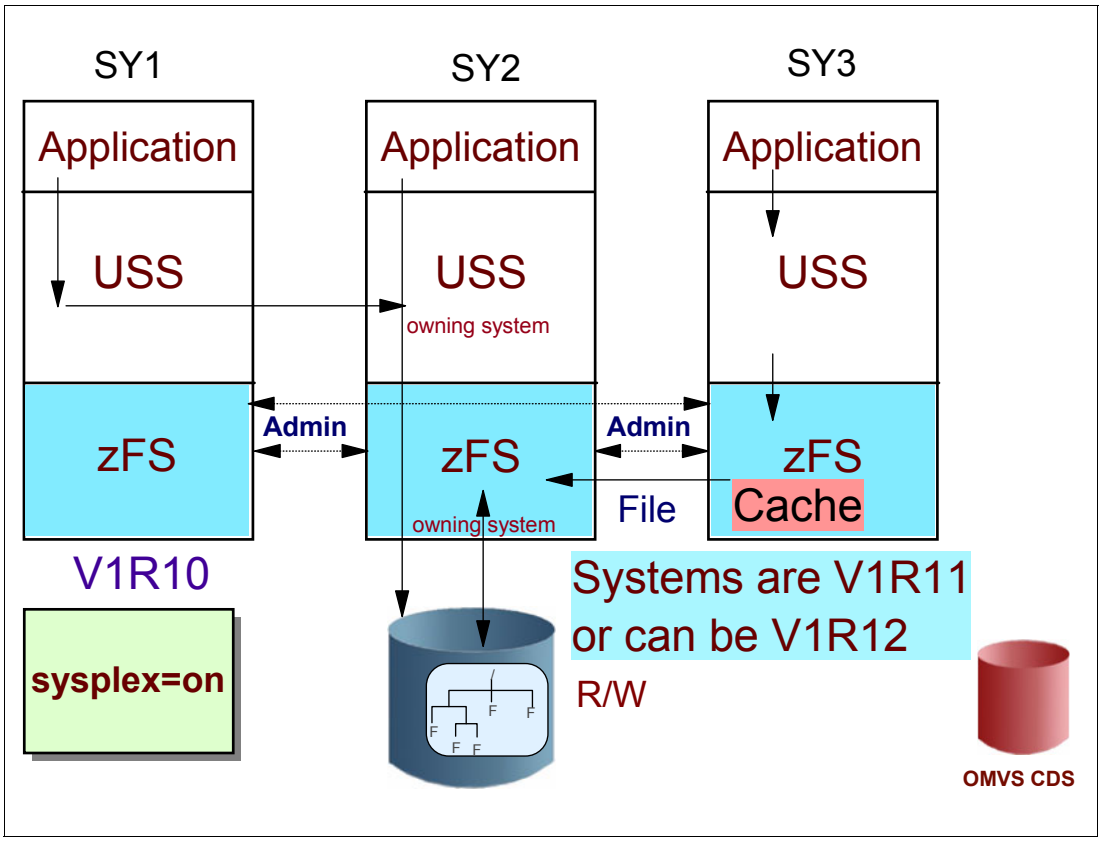


Figure 8-21 Defining zFS as sysplex-aware

zFS as sysplex-aware

For read-write mounted file systems, file requests are sent directly to zFS, which in turn function ships the request to the zFS owning system. In many cases, zFS does not need to function ship the request to the owning zFS system because the data is in the local cache. zFS caches data to avoid frequent forwarding to an owner, and this typically improves performance for most workloads. The zFS owning system is the only system that does I/O to the file system. In this case, zFS is running sysplex-aware for read-write mounted file systems.

Beginning with z/OS V1R11, you can enable zFS to run sysplex-aware for zFS read-write file systems. When a file system is mounted read-write sysplex-aware (for example, on SY2), the mount request is sent to the local zFS and zFS opens the file system data set (for read-write). If the mount is successful on that system, z/OS UNIX sends a signal to the other sysplex member systems telling them to issue a catch-up mount on each system. Each z/OS UNIX on each other system then reads the CDS and determines that it needs to send a mount request to the local zFS for that file system. Each local mount causes zFS to open the data set (for read-write). In this way, the mount on SY2 causes the file system to be mounted on every member of the sysplex.

sysplex=on

zFS can be sysplex-aware for read-write mounted file systems. zFS can be configured (zFS IOEFSPRM option sysplex=on) to treat all zFS read-write mounted file systems owned on that system as sysplex-aware file systems.

8.22 APAR OA29619

- APAR OA29619 provides a new function that allows users to specify which zFS read-write file systems are to be made sysplex-aware
  - You can choose individual file systems to be sysplex-aware (requires zFS APAR OA29619)
  - You can choose to have all zFS file systems be sysplex-aware
- To enable zFS sysplex-aware on a file system basis, specify sysplex=filesys in the IOEFSPRM
- You can specify it in a shared IOEFSPRM and each system picks up the specification in a rolling IPL - The sysplex option is ignored by previous releases

Figure 8-22 APAR OA29619

APAR OA29619

To enable zFS sysplex-aware on a file system basis, specify sysplex=filesys in the IOEFSPRM configuration file. Leave the sysplex_filesys_sharemode with its default of norwshare. You can specify it in a shared `IOEFSPRM` configuration file and each system picks up the specification in a rolling IPL. The sysplex option is ignored by previous releases. Any z/OS V1R9 or V1R10 systems in the shared file system environment must have zFS APAR OA29786 active before you can have zFS sysplex=filesys active on any z/OS V1R11 system.

When you run sysplex=filesys, the zFS PFS runs sysplex-aware, but each zFS file system is mounted non-sysplex-aware (by default). zFS is enabled to allow zFS read-write file systems to be sysplex-aware but it must be explicitly requested on a file system basis. Note that although it is possible to change the sysplex_filesys_sharemode from its default of norwshare to rwshare at IPL using the IOEFSPRM option or dynamically using the zfsadm config command, this is not recommended while migrating from sysplex=off to sysplex=filesys because it can cause performance problems.

After you have sysplex=filesys active on all your systems, you can consider which zFS read-write file systems you want to be sysplex-aware. Good candidates are zFS read-write file systems that are accessed from multiple systems or are mounted with AUTOMOVE and might be moved by z/OS UNIX (as a result of a shutdown or IPL) to systems that do not necessarily do the most accesses.

8.23 New zFS configuration options (OA29619)

- Options changed with V1R11 and **OA29619**
  - sysplex = {on | off | **filesys**} - Default is **OFF**
  - **sysplex_filesys_sharemode = norwshare | rwshare**
  - sysplex_admin_level = 0 | 1 | 2 | 3
  - token_cache_size = vnode_cache_size x 2
  - file_threads = 40
  - client_cache_size =128M --------> Cache
  - client_reply_storage = **10M ----> (changed from 40M)**
  - recovery_max_storage = 256M

Underlined options are with OA29619

Figure 8-23 New configuration with APAR OA29619

APAR OA29619 configuration options

Following are new configuration options that can be defined either in the IOEPRMxx parmlib member or in the IOEFSPRM DD member in the ZFS procedure. These parameters are changed as indicated by z/OS V1R11 and with APAR OA29619.

Sysplex options, filesys and on

Sysplex=filesys specifies whether zFS should run sysplex-aware and if so, whether zFS is sysplex-aware on a file system basis (sysplex=filesys) or sysplex-aware on a system basis (sysplex=on). When sysplex=off, zFS does not automatically move zFS ownership of the aggregate.

Note: If you decide to run zFS non-sysplex aware, shared file system support works as in prior releases. Ensure that you do not specify sysplex=on or sysplex=filesys in your zFS `IOEFSPRM` configuration options file.

The sysplex_filesys_sharemode option

This option specifies the default for the mount PARM for a zFS read-write file system mounted on a sysplex=filesys system. You can override this setting at mount time by specifying an alternate value in the actual mount PARM.

The NORWSHARE file system is a non-sysplex-aware file system; it is only locally mounted on the z/OS UNIX owner and requests from z/OS UNIX clients are function shipped to the z/OS UNIX owner by z/OS UNIX.

Note: When you run zFS sysplex-aware on a file system basis (sysplex=filesys) on all your members, the zFS PFS initializes as sysplex-aware. However, it can determine which individual file system is sysplex-aware and which is not based on the RWSHARE or NORWSHARE mount parm or the sysplex_filesys_sharemode option if that MOUNT PARM is not specified.

The following example shows the mount of a zFS read-write file system with a mount PARM of RWSHARE:

```
MOUNT FILESYSTEM('OMVS.PRV.COMPAT.AGGR001') TYPE(ZFS) MODE(RDWR)
MOUNTPOINT('/usr/mountpt1') PARM('RWSHARE')
```

sysplex_admin_level

Specifies the zFS XCF communication interface level that zFS is running as sysplex-aware. This is only valid for systems running zFS V1R9 or V1R10 with the proper APARs applied.

One (1) indicates that zFS running on z/OS V1R9 or V1R10 is in preconditioning state and is using interface level 1. zFS uses the existing XCF protocol but also supports enough of the new XCF protocol for zFS interface level 2 to run properly. Two (2) indicates that zFS running on z/OS V1R9 or V1R10 is in toleration mode and is using interface level 2. zFS uses the new XCF protocol and is able to communicate with other zFS instances running at interface level 1 to display aggregate information for aggregates owned by zFS interface level 1 systems. At level 2, zFS is able to communicate with other zFS instances running at z/OS V1R11 (level 3). The value must be 2 on all members of the sysplex in order to bring z/OS V1R11 zFS into the shared file system environment.

client_cache_size

This parameter specifies the amount of storage used to cache sysplex client user data. You can also specify a fixed option that indicates that the pages are permanently fixed for performance. When zFS is running sysplex-aware for read-write file systems, a new cache is added to zFS controlled by the client_cache_size IOEFSPRM configuration option. The client_cache_size option is used to cache user data when zFS files are accessed from a system that is not the zFS owner of the file system. The default size is 128 MB.

8.24 Admin levels in a mixed sysplex

- zFS in V1R9 or V1R10 with sysplex_admin_level=1
  - Uses and follows the XCF Admin protocol among systems with level 0 and 1 and does not initialze if a system with XCF Admin level 3 is active in the sysplex
- zFS in V1R9 or V1R10 with sysplex_admin_level=2
  - Uses and follows the new XCF Admin protocol among systems with levels 2 or 3 and does not initialze if a system with XCF Admin level 0 is active in the sysplex
- zFS in R11 and R12 running with sysplex_admin_level=3
  - By default follows the new XCF Admin protocol with systems at levels 2 or 3 and does not initialze if a system with XCF Admin level 0 or 1 is active in the sysplex

Figure 8-24 Admin levels in a sysplex

Admin levels in a sysplex

In z/OS V1R11, you must apply the zFS sysplex administration function using the following two-step APAR procedure:

1. Install APAR OA25026 on all z/OS V1R9 and z/OS V1R10 systems. This is a conditioning function for zFS on z/OS V1R11. Make APAR OA25026 active on all systems through a rolling IPL. You are now running with zFS sysplex_admin_level=1.
2. After APAR OA25026 is active on each z/OS V1R9 and V1R10 system, specify the sysplex_admin_level=2 configuration option in the `IOEFSPRM` file. Make this level active on all z/OS V1R9 and V1R10 systems through another rolling IPL. This is the toleration function for zFS on z/OS V1R11 (the default for sysplex_admin_level is sysplex_admin_level=1).

Note: You cannot skip step 1 and do only step 2 if you are running any z/OS V1R9 or V1R10 systems that do not have APAR OA25026 installed and active.

- Your systems must be running zFS V1R9 and above to support a zFS V1R11 environment. In other words, zFS V1R11 is not compatible with V1R8 and below.
- The sysplex_admin_level configuration option cannot be specified dynamically through the zfsadm config command.

sysplex_admin_level

The sysplex_admin_level parameter specifies the zFS XCF communication interface level that zFS is running as sysplex-aware. This is only valid for systems running zFS V1R9 or V1R10 with the proper APARs applied. The sysplex_admin_level option is ignored in z/OS V1R11 because zFS runs at sysplex_admin_level 3 on z/OS V1R11.

The sysplex_admin_level is specified as follows:

1. Indicates that zFS running on z/OS V1R9 or V1R10 is in preconditioning state and is using interface level 1. zFS uses the existing XCF protocol but also supports enough of the new XCF protocol for zFS interface level 2 to run properly. Two (2) indicates that zFS running on z/OS V1R9 or V1R10 is in toleration mode and is using interface level 2. zFS uses the new XCF protocol and is able to communicate with other zFS instances running at interface level 1 to display aggregate information for aggregates owned by zFS interface level 1 systems.
2. At level 2, zFS is able to communicate with other zFS instances running at z/OS V1R11 (level 3). The value must be 2 on all members of the sysplex in order to bring z/OS V1R11 zFS into the shared file system environment.
3. The default value is 3 in z/OS V1R11 and z/OS V1R12. The default level is 1 in z/OS V1R9 and V1R10.

Note: The expected value in z/OS V1R11and z/OS V1R12 is 3 and this option is ignored. In z/OS V1R9 and V1R10, the level is 1 or 2.

8.25 Defing zFS as syplex-aware

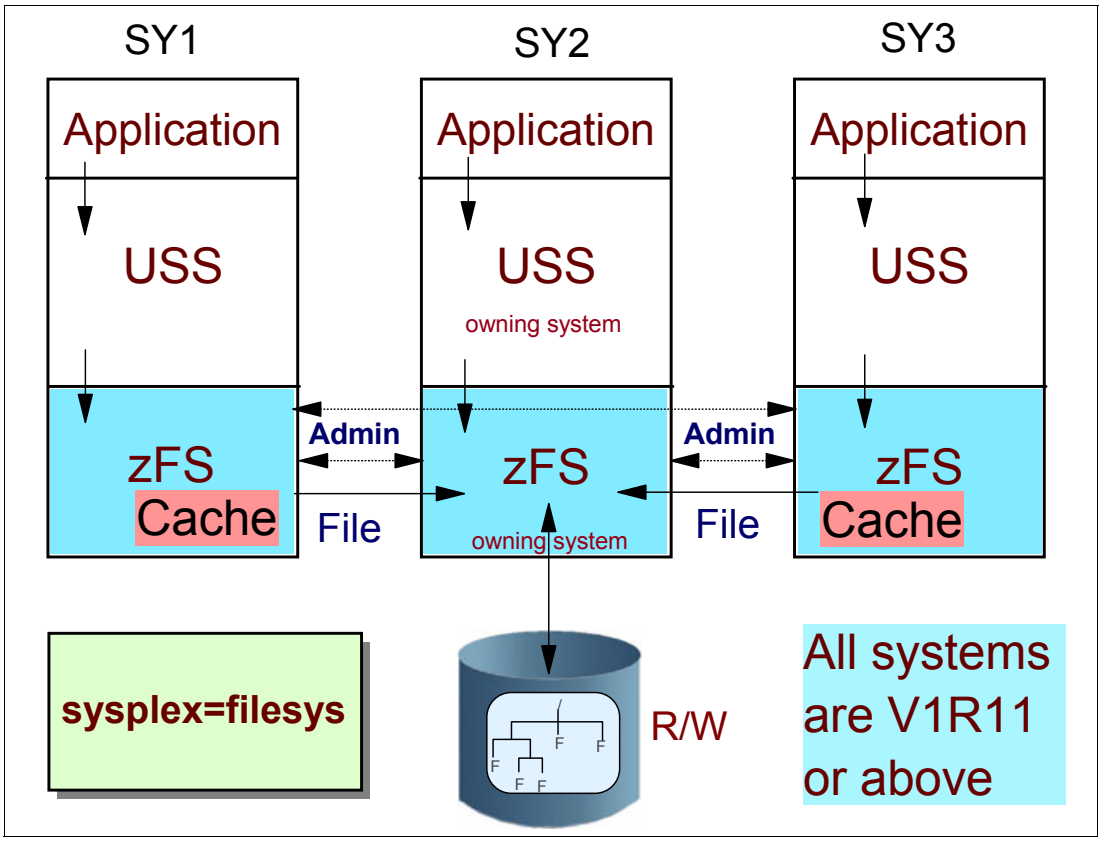


Figure 8-25 zFS as sysplex-aware with sysplex=filesys

zFS as sysplex-aware with sysplex=filesys

If you are running your sysplex in a shared file system environment, that is BPXPRMxx specifies SYSPLEX(YES), you can set zFS V1R11 or any system above to run in one of the following modes:

- Non-sysplex aware for read-write zFS file systems as in previous releases (the default).
- Sysplex-aware on a file system basis—that is, enabling zFS to run sysplex-aware, but individually choosing which file systems are sysplex-aware for read-write and which ones are not (enable by specifying sysplex=filesys in your IOEFSPRM and then perform a rolling IPL).
- Sysplex-aware for all read-write zFS file systems—that is, enabling zFS to run sysplex-aware, and having all zFS read-write file systems mounted sysplex-aware (enable by specifying sysplex=on in your IOEFSPRM and then perform a rolling IPL).

Recommendation: Run sysplex-aware on a file system basis because that gives you the most control and flexibility in deciding which zFS file systems should be sysplex-aware. It also avoids the restriction noted below.

Typically, if you make a zFS read-write file system sysplex-aware, you see a performance improvement in most shared file system environments when accessing the data from a system that is not the zFS owner. There are, however, some servers that are not transparent to zFS sysplex-aware for read-write support.

Sysplex-aware PFS

A physical file system (PFS), for example zFS, is sysplex-aware or non-sysplex aware for a particular mount mode (read-only or read-write) in a shared file system environment. When it is sysplex-aware, it means that the PFS is capable of handling a local mount on the system that is not the z/OS UNIX owning system. The PFS that is sysplex-aware can avoid z/OS UNIX function shipping for that mode. All physical file systems are always sysplex-aware for read-only mounts. HFS is always non-sysplex aware for read-write mounts and always results in z/OS UNIX function shipping from systems that are not the z/OS UNIX owning system.

Sysplex-aware file system

A file system can be mounted sysplex-aware or non-sysplex aware. When a file system is mounted sysplex-aware, it means that the file system is locally mounted on every system (when the PFS is capable of handling a local mount on every system—that is, the PFS is running sysplex-aware) and therefore, file requests are handled by the local PFS. All read-only mounted file systems are always mounted sysplex-aware. HFS read-write mounted file systems are always mounted non-sysplex aware. This means that file requests from non-z/OS UNIX owning systems are always function shipped by z/OS UNIX to the z/OS UNIX owning system where the file system is locally mounted and the I/O is actually done. Beginning with z/OS V1R11, zFS read-write mounted file systems can be mounted sysplex-aware.

zFS can run sysplex-aware for read-write mounted file systems in the following ways:

- Sysplex-aware on a file system basis (sysplex=filesys)

  Selected zFS read-write file systems are sysplex-aware.
- Non-sysplex aware by default)

  zFS read-write file systems that specify the RWSHARE mount PARM are sysplex-aware.
- Sysplex-aware on a system basis (sysplex=on)

  Every zFS read-write file system owned on the system is sysplex-aware.

When running in a shared file system environment, configure all your systems with the same zFS sysplex option. That is, run in one of the following ways:

- All your systems with sysplex=filesys (recommended)
- All your systems with sysplex=off
- All your systems with sysplex=on

8.26 Using the sysplex=filesys parameter

- To enable zFS sysplex-aware on a file system basis
  - Specify sysplex=filesys in the IOEFSPRM configuration file
  - Leave the sysplex_filesys_sharemode with its default of norwshare
  - You can specify it in a shared IOEFSPRM configuration file and each system picks up the specification in a rolling IPL
    - The sysplex option is ignored by previous releases
    - Any z/OS V1R9 or V1R10 systems in the shared file system environment must have zFS APAR OA29786 active before you can have zFS sysplex=filesys active on any z/OS V1R11 system

Figure 8-26 Using the sysplex=filesys parameter

Using the sysplex=filesys parameter

Beginning with z/OS V1R11, zFS can be sysplex-aware for read-write mounted file systems. zFS can be configured (zFS IOEFSPRM option sysplex=on) to treat all zFS read-write mounted file systems owned on that system as sysplex-aware file systems or it can be configured (zFS IOEFSPRM option sysplex=filesys) to allow some zFS read-write mounted file systems owned on that system to be sysplex-aware file systems and some to be non-sysplex aware file systems. zFS must be running sysplex-aware for read-write in order to allow a zFS read-write file system to be mounted as sysplex-aware.

To enable zFS sysplex-aware on a file system basis, specify sysplex=filesys in the IOEFSPRM configuration file. Leave the sysplex_filesys_sharemode with its default of norwshare. You can specify it in a shared IOEFSPRM configuration file and each system picks up the specification in a rolling IPL. The sysplex option is ignored by previous releases. Any z/OS V1R9 or V1R10 systems in the shared file system environment must have zFS APAR OA29786 active before you can have zFS sysplex=filesys active on any z/OS V1R11 or later system.

If you are migrating from a shared file system environment with all systems running zFS sysplex=on to a shared file system environment with all systems running zFS sysplex=filesys, you should specify sysplex_filesys_sharemode=rwshare on each sysplex=filesys system before you migrate with a rolling IPL. This way, each zFS read-write file system will be (remain) sysplex-aware as you roll each system to sysplex=filesys even if you must unmount and mount the file system.

8.27 Using the sysplex=filesys parameter

- **sysplex=filesys** - zFS can be configured sysplex=filesys to allow some zFS read-write mounted file systems owned on that system to be sysplex-aware file systems and some to be non-sysplex aware file systems
  - When you run sysplex=filesys, the zFS PFS runs sysplex-aware, but each zFS file system is mounted non-sysplex aware (by default)
  - zFS is enabled to allow zFS read-write file systems to be sysplex-aware but it must be explicitly requested on a file system basis

Figure 8-27 Using the sysplex=filesys parameter

Using sysplex=filesys

If you have (at least) APAR OA29619 and you intend to run zFS sysplex-aware on a file system basis (sysplex=filesys), you must ensure that you have (at least) APAR OA29786 on your prior releases.

When the BPXPRMxx parmlib member specifies SYSPLEX(YES), you can specify sysplex=on or sysplex=filesys in the IOEFSPRM configuration file and perform an IPL. After changing IOEFSPRM, you must perform an IPL or a restart of zFS. IPL is the recommended method because in many ways it is less disruptive than a restart of zFS, which can cause zFS file systems to become unmounted. This allows you to have zFS read-write sysplex-aware file systems and z/OS UNIX does not forward file requests to the z/OS UNIX owning system. Rather, z/OS UNIX typically sends requests to the local zFS and lets zFS take the responsibility to forward the request, if necessary.

Sysplex-aware on a file system basis

zFS V1R11 to run in one of the following modes:

- Sysplex-aware on a file system basis—that is, enabling zFS to run sysplex-aware, but individually choosing which file systems are sysplex-aware for read-write and which ones are not (enable by specifying sysplex=filesys in your IOEFSPRM and then perform a rolling IPL).

- Sysplex-aware for all read-write zFS file systems—that is, enabling zFS to run sysplex-aware, and having all zFS read-write file systems mounted sysplex-aware (enable by specifying sysplex=on in your `IOEFSPRM` and then perform a rolling IPL).

Note: If you decide to run zFS non-sysplex aware, shared file system support works as in prior releases. Ensure that you do not specify sysplex=on or sysplex=filesys in your zFS IOEFSPRM configuration options file.

Using sysplex-aware on a file system basis

If you decide that you want some zFS read-write file systems to be sysplex-aware, you can run zFS sysplex-aware on a file system basis. Roll this support through all your sysplex members (this assumes all your members are at z/OS V1R11 with the V1R11 APAR OA29619 applied). You cannot change from zFS non-sysplex aware to zFS sysplex-aware on a file system basis dynamically. After changing `IOEFSPRM`, you must perform an IPL or a restart of zFS. IPL is the recommended method because in many ways it is less disruptive than a restart of zFS, which can cause zFS file systems to become unmounted.

To enable zFS sysplex-aware on a file system basis, specify sysplex=filesys in the `IOEFSPRM` configuration file. Leave the sysplex_filesys_sharemode with its default of norwshare. You can specify it in a shared `IOEFSPRM` configuration file and each system picks up the specification in a rolling IPL. The sysplex option is ignored by previous releases. Any z/OS V1R9 or V1R10 systems in the shared file system environment must have zFS APAR OA29786 active before you can have zFS sysplex=filesys active on any z/OS V1R11 system.

8.28 The sysplex_filesys_sharemode parameter

- RWSHARE | NORWSHARE
  - Specifies whether a zFS read-write mounted file system will be mounted sysplex-aware or non-sysplex aware
  - zFS must be running sysplex-aware on a file system basis (IOEFSPRM specifies sysplex=filesys) for this parameter to take effect
  - The default for sysplex_filesys_sharemode is:
    - norwshare when running zFS as sysplex=filesys or <no value>
  - Otherwise specify in the IOEFSPRM file, or subsequently using the zfsadm config command

Figure 8-28 The sysplex_filesys_sharemode parameter

The sysplex_filesys_sharemode parameter

The sysplex_filesys_sharemode option on a system specifies whether a zFS read-write file system will be mounted sysplex-aware when a MOUNT is issued on that system without specifying either NORWSHARE or RWSHARE in the MOUNT PARM.

The default value for sysplex_filesys_sharemode is norwshare. This means that a MOUNT for a zFS read-write file system that does not have NORWSHARE or RWSHARE specified in the MOUNT PARM will result in the file system being non-sysplex aware. If you want all zFS read-write mounts to be sysplex-aware, then specify sysplex_filesys_sharemode=rwshare.

This option can be specified in the IOEFSPRM configuration options file and takes effect on the next IPL (or restart of zFS). It can also be specified dynamically with the **`zfsadm config`** -sysplex_filesys_sharemode command. Typically, you should have the same sysplex_filesys_sharemode value on all your systems. Otherwise, z/OS UNIX file system ownership movement might change the sysplex-awareness of a file system that does not have NORWSHARE or RWSHARE specified in the MOUNT PARM).

If you decide that you want some zFS read-write file systems to be sysplex-aware, you can run zFS sysplex-aware on a file system basis. Roll this support through all your sysplex members (this assumes all your members are at z/OS V1R11 with the V1R11 APAR OA29619 applied). You cannot change from zFS non-sysplex aware to zFS sysplex-aware on a file system basis dynamically. After changing IOEFSPRM, you must perform an IPL or a restart of zFS. IPL is the recommended method because in many ways it is less disruptive than a restart of zFS, which can cause zFS file systems to become unmounted.

8.29 sysplex_filesys_sharemode considerations

- Do not make any zFS (R/W) file systems sysplex-aware until you have all systems in the shared file system environment at z/OS V1R11+ with sysplex=filesys active
- To make a zFS read-write file system sysplex-aware when running sysplex=filesys on all systems
  - You must unmount the file system
  - Specify RWSHARE as a mount PARM
  - Then mount the file system
- The following TSO/E example shows a zFS mount PARM of RWSHARE:

```
MOUNT FILESYSTEM('OMVS.PRV.COMPAT.AGGR001') TYPE(ZFS)
MOUNTPOINT('/etc/mountpt') PARM('RWSHARE')
```

Figure 8-29 sysplex_filesys_sharemode considerations

sysplex_filesys_sharemode considerations

The sysplex_filesys_sharemode option on a system specifies whether a zFS read-write file system will be mounted sysplex-aware when a MOUNT is issued on that system without specifying either NORWSHARE or RWSHARE in the MOUNT PARM. The default value for sysplex_filesys_sharemode is norwshare. This means that a MOUNT for a zFS read-write file system that does not have NORWSHARE or RWSHARE specified in the MOUNT PARM will result in the file system being non-sysplex aware.

Sysplex-aware with rwshare

If you want all zFS read-write mounts to be sysplex-aware, then specify sysplex_filesys_sharemode=rwshare. This option can be specified in the IOEFSPRM configuration options file and takes effect on the next IPL (or restart of zFS). It can also be specified dynamically with the **`zfsadm config`** -sysplex_filesys_sharemode command.

Typically, you should have the same sysplex_filesys_sharemode value on all your systems. Otherwise, z/OS UNIX file system ownership movement might change the sysplex-awareness of a file system that does not have NORWSHARE or RWSHARE specified in the MOUNT PARM).

Note: Although it is possible to change the sysplex_filesys_sharemode from its default of norwshare to rwshare at IPL using the IOEFSPRM option or dynamically using the **zfsadm config** command, this is not recommended while migrating from sysplex=off to sysplex=filesys because it can cause performance problems.

Using sysplex-aware on all file systems

Do not make any zFS read-write file systems sysplex-aware until you have all systems in the shared file system environment at z/OS V1R11 with sysplex=filesys active. To make a zFS read-write file system sysplex-aware when running sysplex=filesys on all systems, you must unmount the file system, specify RWSHARE as a mount PARM and then mount the file system. The following TSO/E example shows a zFS mount PARM of RWSHARE:

```
MOUNT FILESYSTEM('OMVS.PRV.COMPAT.AGGR001') TYPE(ZFS)
MOUNTPOINT('/etc/mountpt') PARM('RWSHARE')
```

8.30 sysplex_filesys_sharemode considerations

- If you are migrating from a shared file system environment with all systems running zFS sysplex=on to a shared file system environment with all systems running zFS sysplex=filesys
  - You should specify sysplex_filesys_sharemode=rwshare on each sysplex=filesys system before you migrate with a rolling IPL
  - This way, each zFS read-write file system (remains) sysplex-aware as you roll each system to sysplex=filesys even if you must unmount and mount the file system
- Sysplex root considerations

Figure 8-30 sysplex_filesys_sharemode considerations

sysplex_filesys_sharemode considerations

If you are migrating from a shared file system environment with all systems running zFS sysplex=on to a shared file system environment with all systems running zFS sysplex=filesys, you should specify sysplex_filesys_sharemode=rwshare on each sysplex=filesys system before you migrate with a rolling IPL. This way, each zFS read-write file system will be (remain) sysplex-aware as you roll each system to sysplex=filesys even if you must unmount and mount the file system.

Mounts without specifying rwshare or norwshare

The sysplex_filesys_sharemode option on a system specifies whether zFS read-write file system will be mounted sysplex-aware when a MOUNT is issued on that system without specifying either NORWSHARE or RWSHARE in the MOUNT PARM.

The default value for sysplex_filesys_sharemode is norwshare. This means that a MOUNT for a zFS read-write file system that does not have NORWSHARE or RWSHARE specified in the MOUNT PARM will result in the file system being non-sysplex aware. If you want all zFS read-write mounts to be sysplex-aware, then specify sysplex_filesys_sharemode=rwshare. This option can be specified in the `IOEFSPRM` configuration options file and takes effect on the next IPL (or restart of zFS). It can also be specified dynamically with the zfsadm config -sysplex_filesys_sharemode command. Typically, you should have the same sysplex_filesys_sharemode value on all your systems. Otherwise, z/OS UNIX file system ownership movement might change the sysplex-awareness of a file system that does not have NORWSHARE or RWSHARE specified in the MOUNT PARM).

Rolling IPL

If you decide that you want all your zFS read-write file systems to be sysplex-aware, roll this support as quickly as you can through all your sysplex members (this assumes all your members are using z/OS V1R11 or later). You cannot change from zFS non-sysplex aware to zFS sysplex-aware dynamically. After changing `IOEFSPRM`, you must perform an IPL or a restart of zFS. IPL is the recommended method because in many ways it is less disruptive than a restart of zFS, which can cause zFS file systems to become unmounted. To make all your zFS read-write file systems sysplex-aware, specify sysplex=on in the `IOEFSPRM` configuration file. You can specify it in a shared `IOEFSPRM` configuration file and each system picks up the specification in a rolling IPL. The sysplex option is ignored by previous releases.

Sysplex root considerations

Your sysplex root file system should be read-only. However, if your sysplex root file system is normally read-write, you should make it sysplex-aware. You cannot unmount the sysplex root file system, so you need an alternative method. One method is to replace your sysplex root using the **`MODIFY OMVS,NEWROOT`** operator command, which is described in *z/OS UNIX System Services Planning*, GA22-7800 and *z/OS MVS System Commands*, SA22-7627.

First move z/OS UNIX ownership of the file system, if necessary, to a system running zFS sysplex=filesys that has sysplex_filesys_sharemode=rwshare. Then, replace your sysplex root with a zFS sysplex root file system. Another method is to remount your sysplex root to read-only, move z/OS UNIX ownership of the file system, if necessary, to a system running zFS sysplex=filesys that has sysplex_filesys_sharemode=rwshare, and then remount the sysplex root back to read-write.

8.31 zFS mixed environment

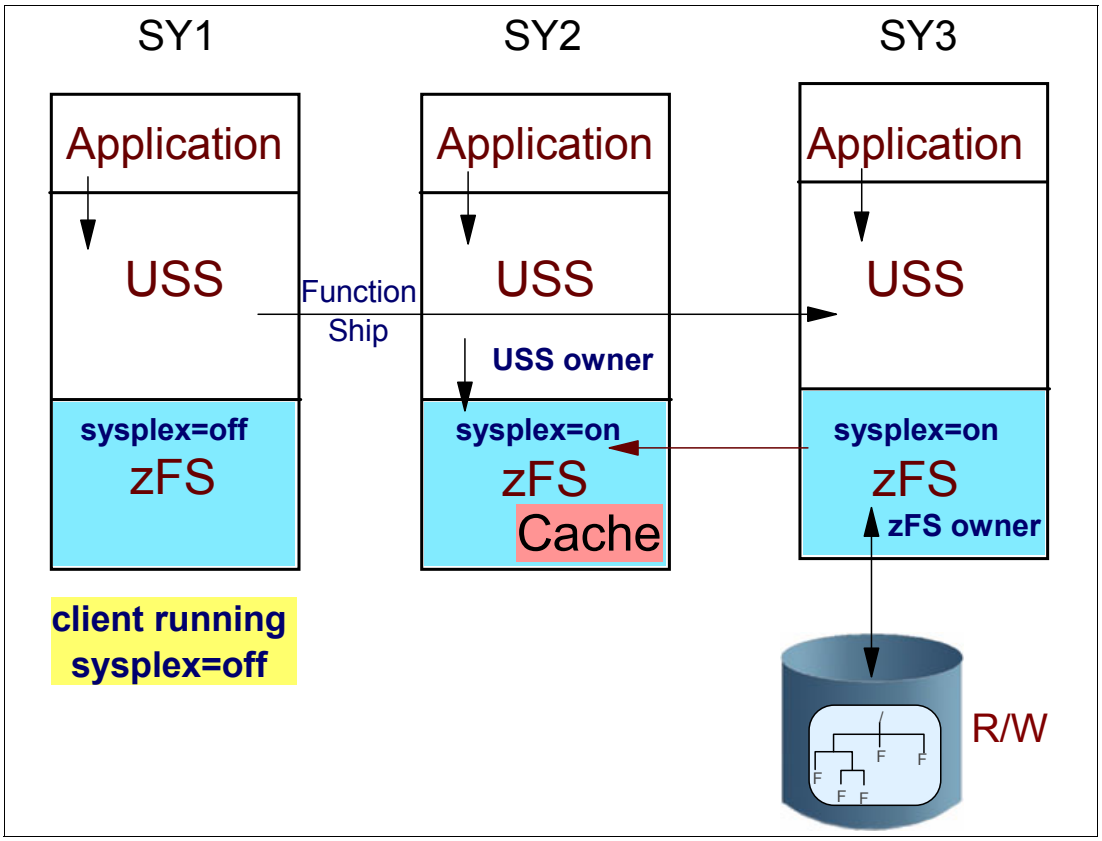


Figure 8-31 Running some zFS sysplex-aware and some zFS non-sysplex aware

Running some zFS sysplex-aware and some zFS non-sysplex aware

You can decide to run in a mixed environment with some systems running zFS sysplex-aware (sysplex=on) and some systems running zFS non-sysplex aware (sysplex=off). In this scenario, some zFS read-write file systems would be sysplex-aware and some zFS read-write file systems would be non-sysplex aware.

However, a mixed environment is not recommended and is not necessary. That is, when you have all systems at z/OS V1R11 with zFS APAR OA29619 applied, you can run all systems zFS sysplex-aware on a file system basis (sysplex=filesys). A mixed environment can cause a reduction in performance. If you want to continue to run a prior release in your sysplex, you should keep running zFS non-sysplex aware. If you want to run the SMB server or the IBM HTTP Server with Fast Response Cache Accelerator you should keep running zFS non-sysplex aware or run all systems with zFS sysplex-aware on a file system basis.

If you still want to run with some systems running zFS sysplex-aware and some systems running zFS non-sysplex aware, you will see a reduction in performance and you will have a more complex environment. You will need to be more concerned with z/OS UNIX ownership of zFS file systems. There are additional restrictions on file system ownership movement when some systems are running zFS sysplex-aware and some systems are running zFS non-sysplex aware. Typically, after a zFS file system is owned by a system running zFS sysplex-aware, the file system cannot be moved to a system running zFS non-sysplex aware without unmounting it, and then mounting the file system on the system running zFS non-sysplex aware. This causes the failure of explicit moves (for example, the z/OS UNIX **`chmount`** command or the SETOMVS FILESYS operator command) and could cause implicit

moves during shutdown, recovery, or PFS termination (for example, for a file system mounted AUTOMOVE with a system list) to fail.

zFS mixed configuration considerations

There are performance implications with running in a mixed environment. In some cases, the performance might be poorer than running all zFS file systems non-sysplex aware. One reason for this is that z/OS UNIX only performs caching of directory information on z/OS UNIX clients when the read-write file system is non-sysplex aware. If the read-write file system is sysplex-aware, z/OS UNIX does not perform directory caching. Any z/OS UNIX client lookup request goes to the z/OS UNIX owning system through XCF communications rather than potentially being satisfied in the local z/OS UNIX directory cache. This means that explicitly choosing which system should be the z/OS UNIX owner of a read-write file system in a mixed environment is a trade-off between the following two options:

- Getting the benefit of zFS sysplex-aware among the zFS sysplex-aware systems by making the zFS file system sysplex-aware.
- Getting the benefit of z/OS UNIX directory caching for all systems and no benefit of zFS sysplex-aware by making the zFS file system non-sysplex aware.

This is further complicated by the fact that after a zFS file system is sysplex-aware, you cannot explicitly move it to a system running zFS non-sysplex aware, unless every other system is doing z/OS UNIX function shipping to this system for that file system; that is, every other system shows Client=Y for that file system. In general, if there are more systems running zFS sysplex-aware than not, it is most likely better to have the zFS read-write file system be sysplex-aware for overall sysplex z/OS UNIX application performance. If you let z/OS UNIX choose the system to own the zFS read-write file system (for example, during shutdown or dead system recovery), z/OS UNIX chooses the z/OS UNIX owner randomly (unless, of course, you have specified a system list on MOUNT). Normally, after a read-write file system is sysplex-aware, it is only moved to systems running zFS sysplex-aware.

8.32 zFS cache management

- When the zFS PFS is sysplex-aware
  - z/OS UNIX sends requests to the local zFS and zFS takes on the responsibility to forward the request, if necessary
    - For many requests, it is not necessary because zFS caches data locally and satisfies the request from its cache
  - Cache consistency is maintained through a token management mechanism and may provide improved performance in a shared file system environment
  - If it is a read or lookup or readdir and the data is contained in the cache, the request can be satisfied without communication with the zFS owning system
  - Write requests are written forwarded to the owner system
  - Caching improves performance

Figure 8-32 zFS cache management

zFS cache management

z/OS UNIX typically sends requests to the local zFS and lets zFS take the responsibility to forward the request, if necessary. Cache consistency is maintained through a token management mechanism. This typically provides improved performance in a shared file system environment.

When zFS is running sysplex-aware for read-write file systems, a new cache is added to zFS controlled by the client_cache_size IOEFSPRM configuration option. The client_cache_size option is used to cache user data when zFS files are accessed from a system that is not the zFS owner of the file system. The default size is 128 MB. This is in addition to the existing user_cache_size option, which is used when zFS files are accessed from a system that is the zFS owner of the file system.

For read-write mounted file systems, file requests are sent directly to zFS, which in turn function ships the request to the zFS owning system. In many cases, zFS does not need to function ship the request to the owning zFS system because the data is in the local cache. zFS caches data to avoid frequent forwarding to an owner, and this typically improves performance for most workloads. The zFS owning system is the only system that does I/O to the file system. In this case, zFS is running sysplex-aware for read-write mounted file systems.

Cache consistency is maintained through a token management mechanism. This typically provides improved performance in a shared file system environment.

8.33 zFS sysplex-aware on a file system basis

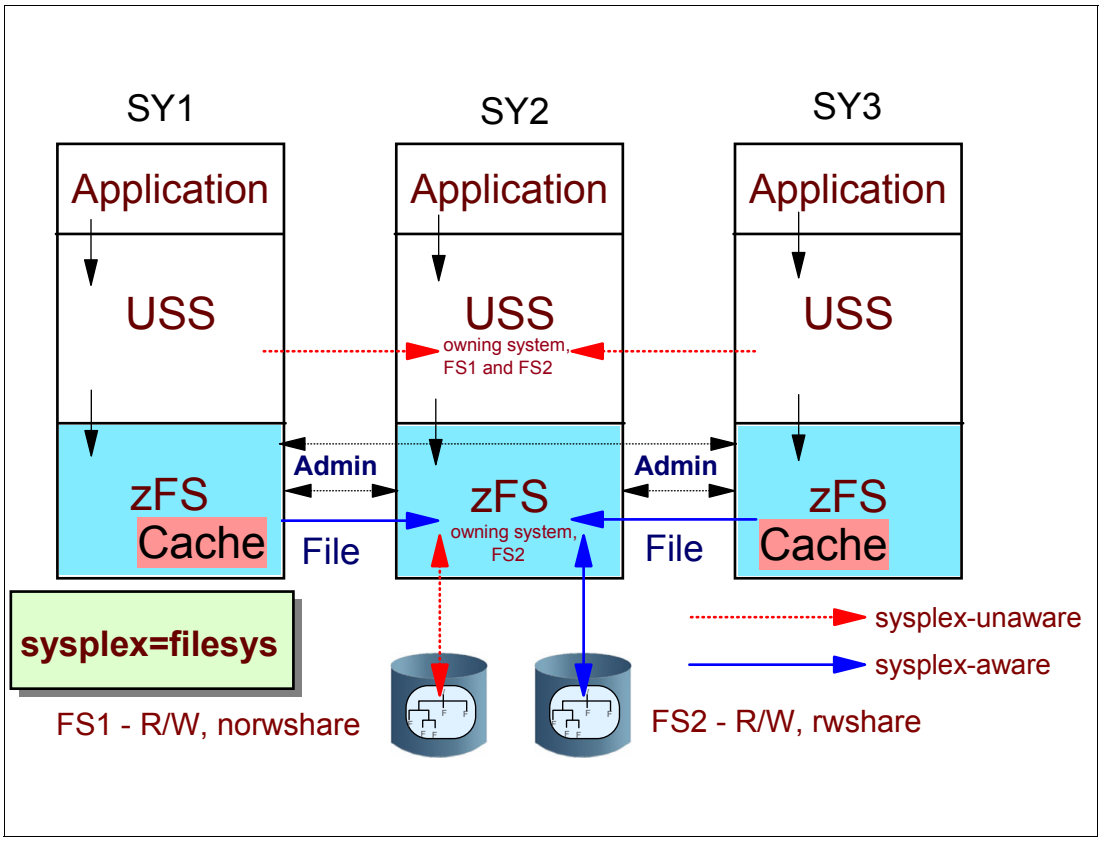


Figure 8-33 zFS sysplex-aware on a file system basis

zFS sysplex-aware on a file system basis

With z/OS V1R12 and V1R11, to enable zFS sysplex-aware on a file system basis, specify sysplex=filesys in the `IOEFSPRM` configuration file. Leave the sysplex_filesys_sharemode with its default of norwshare. You can specify it in a shared `IOEFSPRM` configuration file and each system picks up the specification in a rolling IPL. The sysplex option is ignored by previous releases. Any V1R10 systems in the shared file system environment must have zFS APAR OA29786 active before you can have zFS sysplex=filesys active on any z/OS V1R11 and z/OS V1R12 systems.

When you run zFS sysplex-aware on a file system basis on all your members, the zFS Physical File System initializes as sysplex-aware but it can individually determine which file system is sysplex-aware and which is not based on the mount parameters rwshare and norwshare.

Note: MOUNT commands for file systems in Figure 8-33.

- For file system FS2

```
MOUNT FILESYSTEM('OMVS.PRV2.COMPAT.AGGR001') TYPE(ZFS)
MODE(RDWR) MOUNTPOINT('/usr/mountpt2') PARM('RWSHARE')
```

- For file system FS1

```
MOUNT FILESYSTEM('OMVS.PRV1.COMPAT.AGGR001') TYPE(ZFS)
MODE(RDWR) MOUNTPOINT('/usr/mountpt1') PARM('NORWSHARE')
```

Mixed sysplex example - sysplex-aware on a file system basis

Figure 8-33 shows three systems with zFS sysplex-aware on a file system basis (`sysplex=filesys`). There are two zFS read-write file systems on a sysplex running zFS sysplex-aware on a file system basis on all members. One file system is mounted `NORWSHARE` and the other is mounted `RWSHARE`. They are both z/OS UNIX owned by system SY2.

The norwshare file system is a non-sysplex-aware file system. It is only locally mounted on the z/OS UNIX owner and requests from z/OS UNIX clients are function shipped to the z/OS UNIX owner by z/OS UNIX.

The other file system shown in Figure 8-33 on page 432 is mounted rwshare. It is a sysplex-aware file system and locally mounted on all systems; and z/OS UNIX never function ships requests to the z/OS UNIX owner.

Note: To enable zFS sysplex-aware on a file system basis, specify sysplex=filesys in the `IOEFSPRM` configuration file. Leave the sysplex_filesys_sharemode with its default of norwshare. You can specify it in a shared `IOEFSPRM` configuration file and each system picks up the specification in a rolling IPL. The sysplex option is ignored by previous releases. Any z/OS V1R9 or V1R10 systems in the shared file system environment must have zFS APAR OA29786 active before you can have zFS sysplex=filesys active on any z/OS V1R11 system.

After you have sysplex=filesys active on all your systems, you can consider which zFS read-write file systems you want to be sysplex-aware. Good candidates are zFS read-write file systems that are accessed from multiple systems or are mounted with AUTOMOVE and might be moved by z/OS UNIX (as a result of a shutdown or IPL) to systems that do not necessarily do the most accesses.

If you decide that you want some zFS read-write file systems to be sysplex-aware, you can run zFS sysplex-aware on a file system basis. Roll this support through all your sysplex members (this assumes all your members are at z/OS V1R11 with the V1R11 APAR OA29619 applied). You cannot change from zFS non-sysplex aware to zFS sysplex-aware on a file system basis dynamically. After changing `IOEFSPRM`, you must perform an IPL or a restart of zFS. IPL is the recommended method because in many ways it is less disruptive than a restart of zFS, which can cause zFS file systems to become unmounted.

Attention: When you run sysplex=filesys, the zFS PFS runs sysplex-aware, but each zFS file system is mounted non-sysplex aware (by default). zFS is enabled to allow zFS read-write file systems to be sysplex-aware but it must be explicitly requested on a file system basis. Note that although it is possible to change the sysplex_filesys_sharemode from its default of norwshare to rwshare at IPL using the IOEFSPRM option or dynamically using th z**`fsadm config`** command, this is not recommended while migrating from sysplex=off to sysplex=filesys because it can cause performance problems.

8.34 Automatic movement of file systems

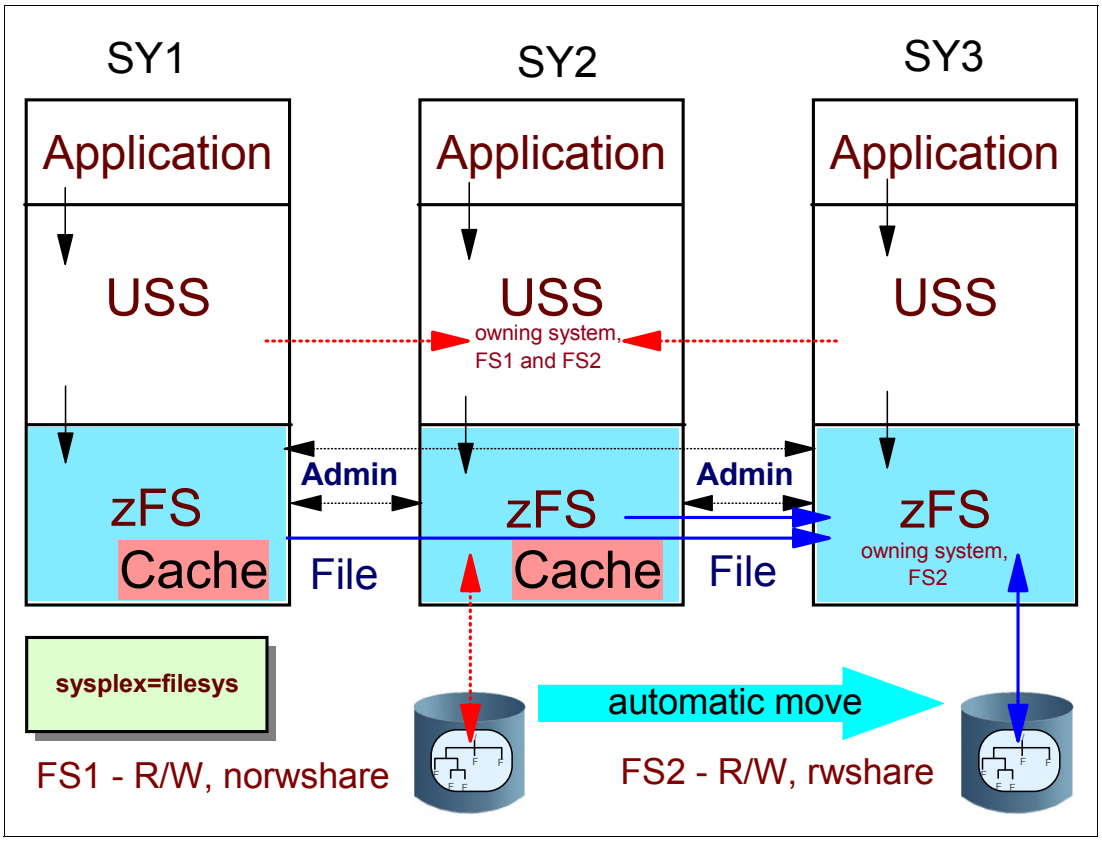


Figure 8-34 Automatic movement of file systems

Automatic movement of file systems

Figure 8-34 shows again the three systems with zFS sysplex-aware on a file system basis (`sysplex=filesys`). However, this time the sysplex-aware zFS has two different owners. Beginning with z/OS V1R11, as a part of supporting read-write mounted file systems that are accessed as sysplex-aware, zFS automatically moves zFS ownership of a zFS file system to the system that has more read-write activity coming from it compared to the total amount from all systems.

This is because zFS can move its ownership of zFS read-write sysplex-aware file systems dynamically based on system usage among zFS systems where the file systems are locally mounted. If one sysplex member is a better fit for ownership, because more requests to the file system are being done there than on any other sysplex member, then zFS dynamically moves the zFS ownership to that system.

9

Managing file systems

This chapter describes the hierarchical file system and how to customize it to your requirements. In addition to providing an introduction to HFS concepts, it provides details on how to:

- Manage and create a hierarchical file system
- Use commands to display useful information
- Perform space management for a hierarchical file system

© Copyright IBM Corp. 2011. All rights reserved.

9.1 Hierarchical file system (HFS)

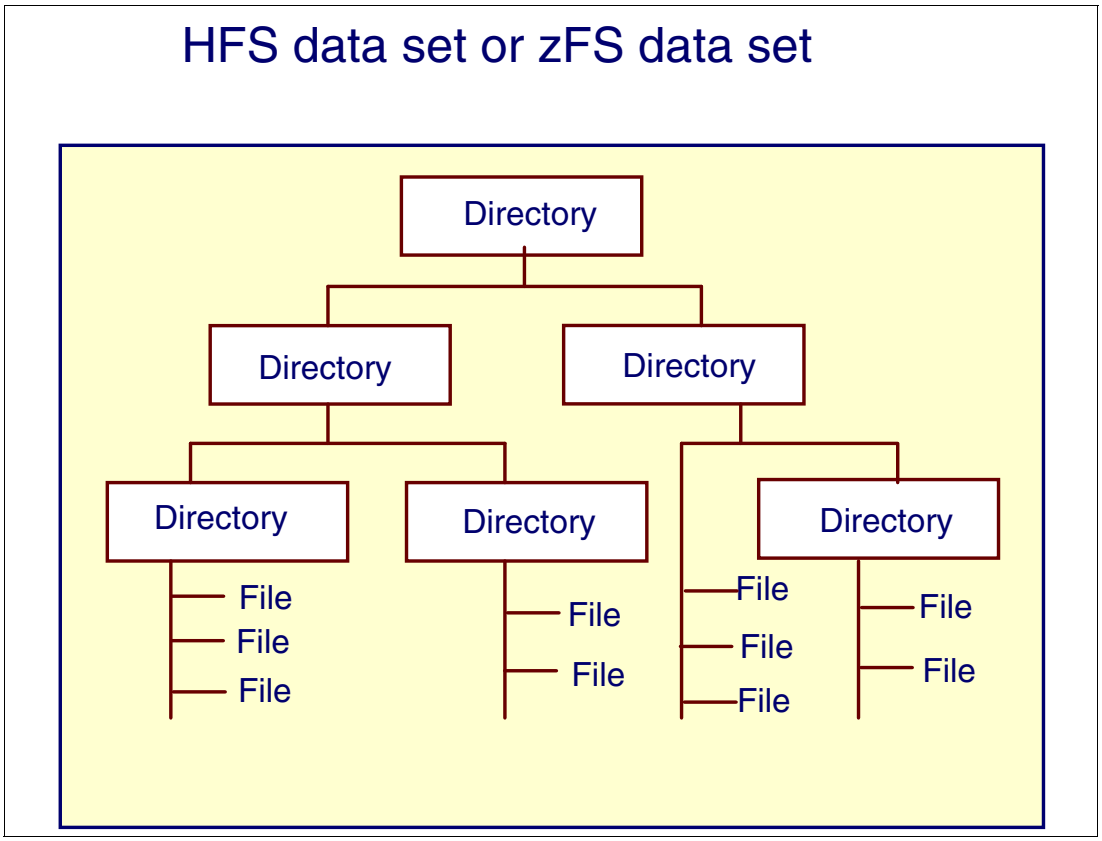


Figure 9-1 Hierarchical file system structure

Hierarchical file system structure

The z/OS UNIX file system is hierarchical and byte-oriented. Finding a file in the file system is done by searching a directory or a series of directories. There is no concept of a z/OS catalog that points directly to a file.

z/FS and HFS are both UNIX file systems and both can participate in shared sysplexes. However, while HFS always has a single file system per data set, zFS may have multiple file systems in a single data set. These data sets are called *aggregates* and are a collection of data sets.

A path name identifies a file and consists of directory names and a file name. A fully qualified file name, which consists of the name of each directory in the path to a file plus the file name itself, can be up to 1023 bytes long, for example: /u/joe/projects/p1/prog1.

The hierarchical file system allows for file names in mixed case. The files in the hierarchical file system are sequential files, and they are accessed as byte streams. A record concept does not exist with these files, other than the structure defined by an application.

The files can be text files or binary files. In a text file, each line is separated by a *newline delimiter*. A binary file consists of sequences of binary words, and the application reading or writing to the file is responsible for the format of the data.

A file name can be up to 255 characters long. HFS data sets, zFS data sets, and z/OS data sets can reside on the same SMS-managed DASD volume.

9.2 File linking

- Hard link
  - ln oldfile newfile
  - newfile becomes a new pathname for the existing file oldfile - If oldfile names a symbolic link, link creates a hard link to the file
- Symbolic link
  - A symbolic link is another file that contains the pathname for the original file
- External link
  - An external link is a special type of symbolic link, a file that contains the name of an object outside of the hierarchical file system

Figure 9-2 File linking

File linking

A *link* is a new pathname, or directory entry, for an existing file. The new directory entry can be in the same directory that holds the file, or in a different directory. You can access the file under the old pathname or the new one. After you have a link to a file, any changes you make to the file are evident when it is accessed under any other name. Links are useful when:

- A file is moved and you want users to be able to access the file under the old name.
- You want to create a link with a short pathname (an alias) for a file that has a long pathname.

You can use the **ln** command to create a hard link or a symbolic link. A file can have an unlimited number of links to it.

Hard link

A *hard link* is a new name for an existing file. You cannot create a hard link to a directory, and you cannot create a hard link to a file on a different mounted file system.

Symbolic link

A *symbolic link* is another file that contains the pathname for the original file—in essence, a reference to the file. A symbolic link can refer to a pathname for a file that does not exist.

External link

A file can be an *external link* to a sequential data set, a PDS, or a PDS member.

9.3 Hard links

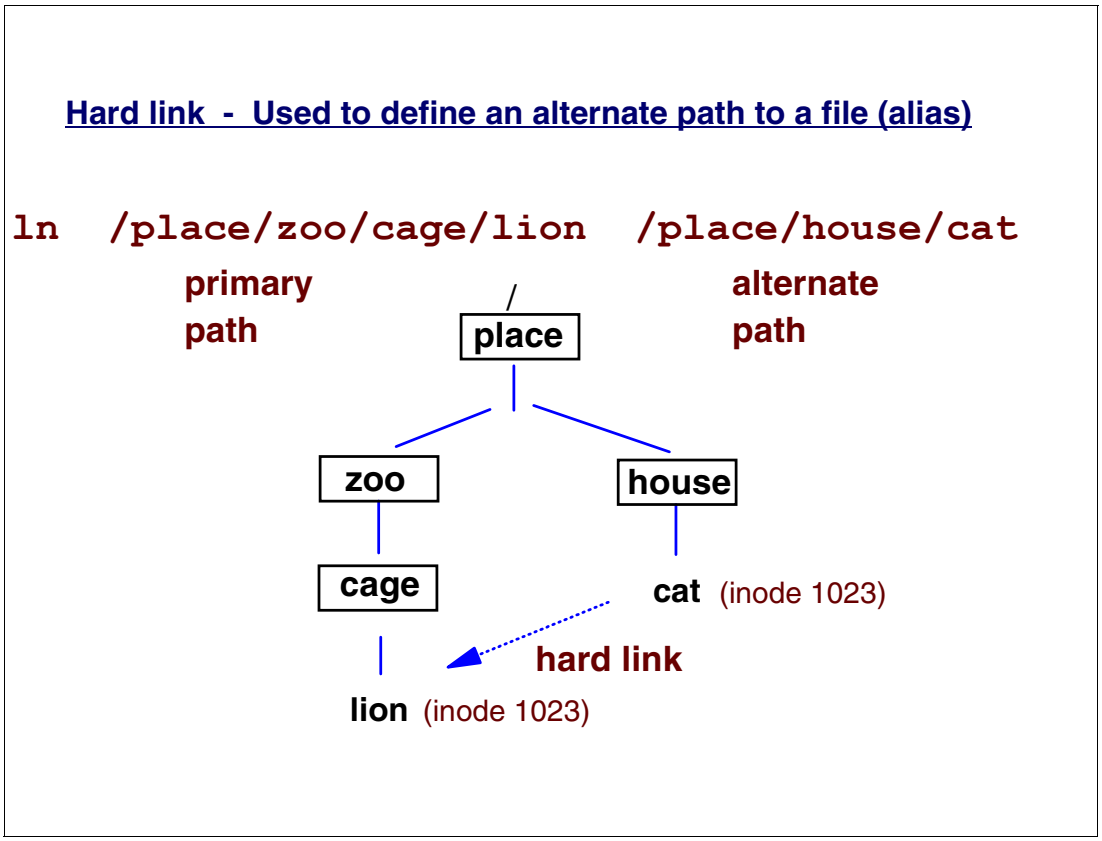


Figure 9-3 Hard link example

Hard links

A hard link is a new name for an existing file. You cannot create a hard link to a directory, and you cannot create a hard link to a file on a different mounted file system.

All the hard link names for a file are as important as its original name. They are all real names for the one original file. To create a hard link to a file, use this command format:

```
ln old new
```

When you create a hard link to a file, the new filename shares the inode number of the original physical file. Because an inode number represents a physical file in a specific file system, you cannot make hard links to other mounted file systems.

INODE

The *inode* is the internal structure that describes the individual files in the operating system; there is one inode for each file. An inode contains the node, type, owner, access times, number of links, and location of a file. A table of inodes is stored near the beginning of a file system. The file system is used to store data and organize it in a hierarchical way by using file system entries such as directories and files. These file system entries have certain attributes, such as ownership, permission bits, and access time stamps. The data and the attributes of a file are stored with the file in the file system. All file attributes are stored in a control block that is sometimes called the inode.

The inode number specifies a particular inode file in the file system.

9.4 Symbolic links

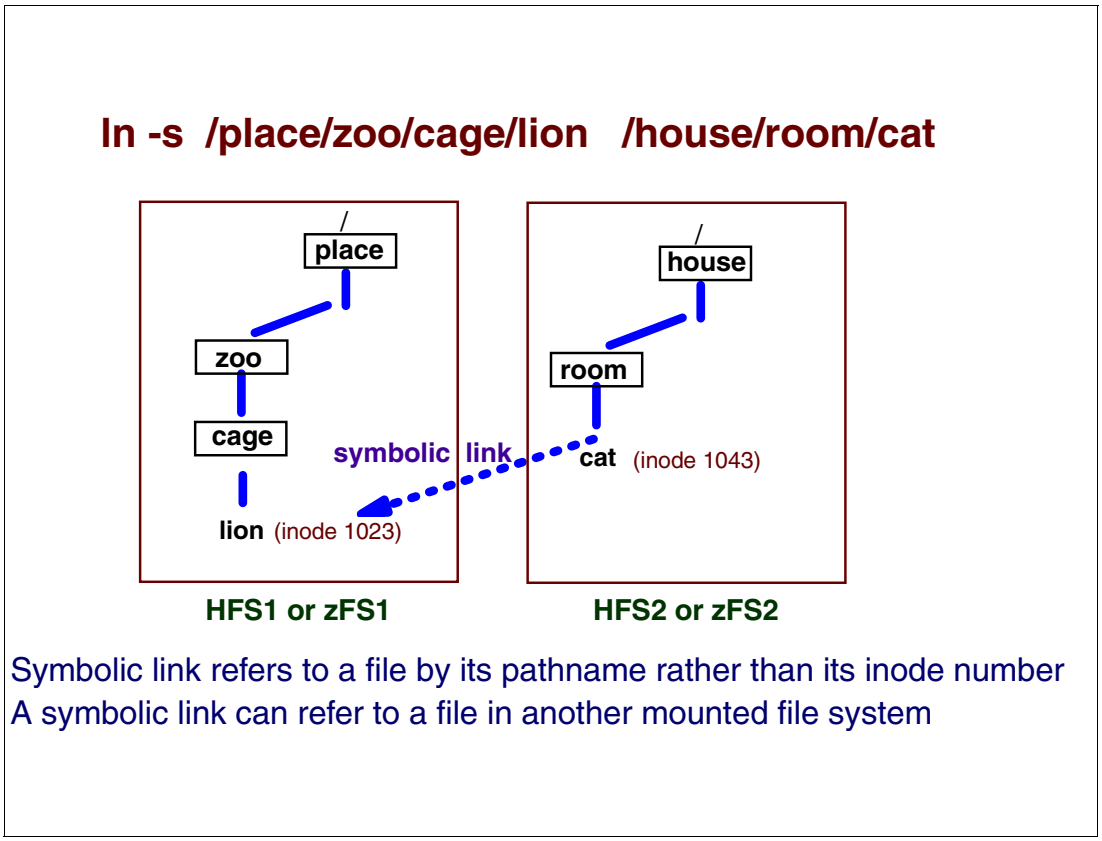


Figure 9-4 Symbolic link examples

Symbolic links

A symbolic link is a type of file system entry that contains the pathname of, and acts as a pointer to another file or directory.

You can create a symbolic link to a file or a directory. Additionally, you can create a symbolic link across mounted file systems, which you cannot do with a hard link. A symbolic link is another file that contains the pathname for the original file—in essence, a reference to the file. A symbolic link can refer to a pathname for a file that does not exist. To create a symbolic link to a file, use this command format:

```
ln -s old new
```

where *new* is the name of the new file containing the reference to the file named *old*. In Figure 9-4, /house/room/cat is the name of the new file that contains the reference to /place/zoo/cage/lion.

When you create a symbolic link, you create a new physical file with its own inode number, as shown in the figure. Because a symbolic link refers to a file by its pathname rather than by its inode number, a symbolic link can refer to files in other mounted file systems.

9.5 External links

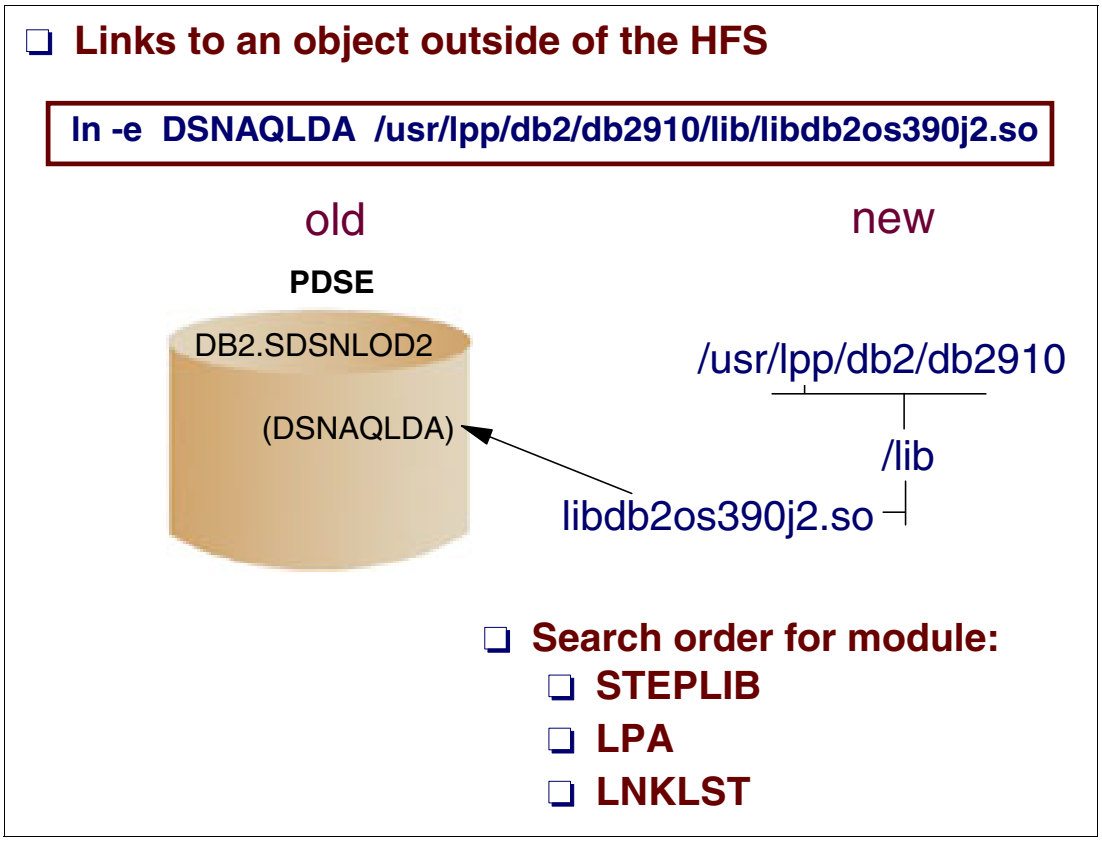


Figure 9-5 Defining external links

External links

An external link is a special type of symbolic link. It is a file that contains the name of an object that is outside of the hierarchical file system.

DFSMS/NFS uses this link to access z/OS data sets.

An example of defining an external link is to make a link between an HFS file and an MVS program, as follows:

- Define an HFS file as:

  `/usr/lpp/db2/db2910`

- Where the MVS program name IMWYWWS is an external link, the command to define the external link is:

  `ln -e DSNAQLDA /usr/lpp/db2/db2910/lib/libdb2os390j2.so`

When you are done, you have created an external link that can be used to access an MVS load library.

Using an external link, you associate that object with a pathname. For example, **`setlocale()`** searches for locale object files in the HFS, but if you want to keep your locale object files in a partitioned data set, you can create an external link in the HFS that points to the PDS. This will improve performance by shortening the search that **`setlocale()`** makes.

A file can be an external link to a sequential data set, a PDS, PDSE, or a PDS member.

External link examples

Consider the following examples:

```
ln -e SYS1.PRIVATE.PGMA   /u/ggi/plib/pgma
ln -e PGMPDS              /u/ggi/plib/pgm
```

The second example assumes that the PGMPDS member will be found in a PDS library, using the traditional z/OS search order. Because of performance considerations, put the library into the LPA or LNKLST.

9.6 File system structure

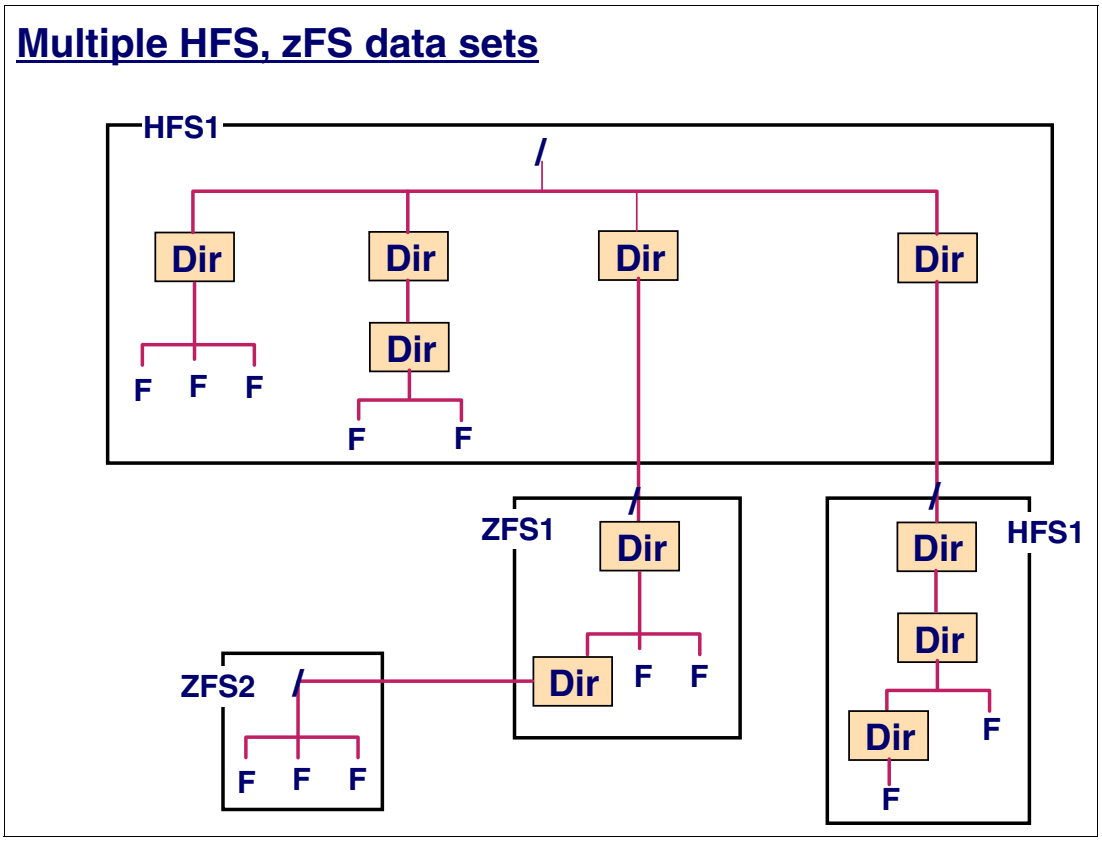


Figure 9-6 File system structure

File system structure

The z/OS UNIX file system can consist of multiple file systems joined together to form a hierarchical file system. Connecting one file system to another is called *mounting*.

The root file system is the first file system that is mounted, and subsequent file systems can be mounted on a directory in a file system that is already mounted. The file systems will form a hierarchy of file systems. A file system must be mounted before anybody can access it.

An HFS or zFS data set is a mountable file system. A mountable file system can be logically mounted to a mount point, which is a directory in another file system, with a TSO/E MOUNT command. A mountable file system can be unmounted from a mount point with a TSO/E UNMOUNT command. ISPF users can also perform these tasks using the ISPF shell.

Build a directory in the root file system. A directory can be used as a mount point for a mountable file system. The mount point should be an empty directory; otherwise, its contents will be hidden for the duration of any subsequent mounts.

Note: If you want to unmount ZFS1, you first have to unmount ZFS2 or the system will tell you that the resource is busy. However, you can unmount by using the immediate option or the force option without first unmounting ZFS2. If users are working on those directories, they will get errors trying to list the current directories.

9.7 Temporary directory space

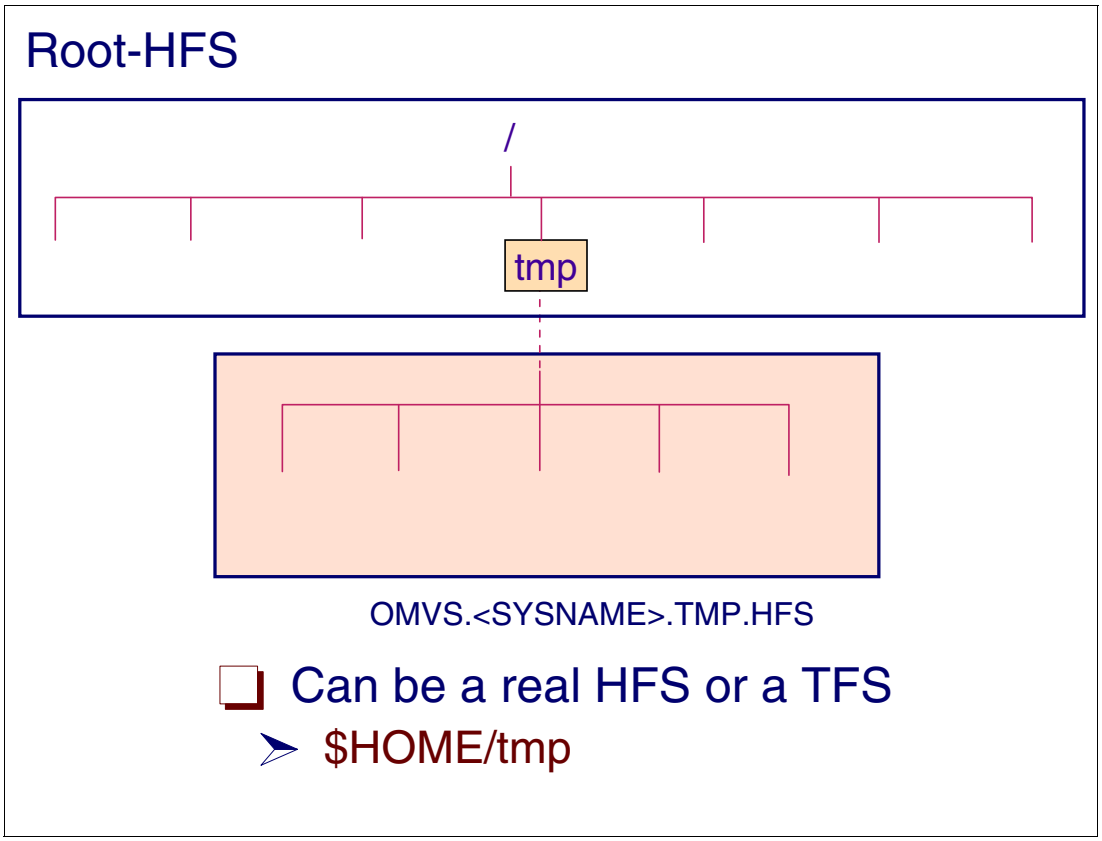


Figure 9-7 Temporary file system and temporary directory space

Temporary directory space

When compiling C programs, the compiler will use the /tmp directory for temporary files. When compiling large applications, the /tmp directory can use a lot of space. Use a separate file system for temporary data. In this way, a maximum of one volume will be used for temporary space.

Separate file system

Placing the /tmp directory in a separate file system can also improve performance in a system with a large number of interactive z/OS UNIX users, where the I/O activity can be very high on the /tmp directory. Do this with the following procedure:

- Allocate an HFS data set for the temporary data file system, for example: OMVS.<SYSNAME>.TMP.HFS
- Mount this data set on the /tmp directory in the root file system. When data is written to files in the /tmp directory, the data will be physically located in the OMVS.<SYSNAME>.TMP.HFS file system.

Temporary file system (TFS)

When you have a large number of interactive users, the /tmp directory can sustain large amounts of I/O activity. There are several approaches you can take:

- Mount a temporary file system (TFS) over /tmp in the HFS, so that you have a high-speed file system for temporary files. The temporary file system is an in-memory file system that is not written to DASD.
- Place the /tmp directory in its own mountable file system and put the file system on its own pack.
- Reduce /tmp activity by setting the TMPDIR environment variable to $HOME/tmp in each user's .profile. This causes various utilities to put temporary files in the user's $HOME/tmp directory rather than in the common /tmp directory.

Recommendations

It is suggested that you place the temporary data in a separate file system. In this way, it will be easier to manage the space used by temporary files. How much temporary space is needed will depend on how z/OS UNIX is used. When the C compiler is invoked from the z/OS UNIX shell, the /tmp directory will be used for temporary data. Miscellaneous shell commands use the /tmp directory as well. If the z/OS UNIX shell is not installed, there will probably not be that much activity in the /tmp directory.

Another reason for placing the /tmp in a separate file system is to improve file system performance. If there is a lot of I/O activity on the /tmp directory, it would improve the performance if the OMVS.<SYSNAME>.TMP.HFS file system is placed on a different volume than the root file system.

9.8 Temporary file system (TFS)

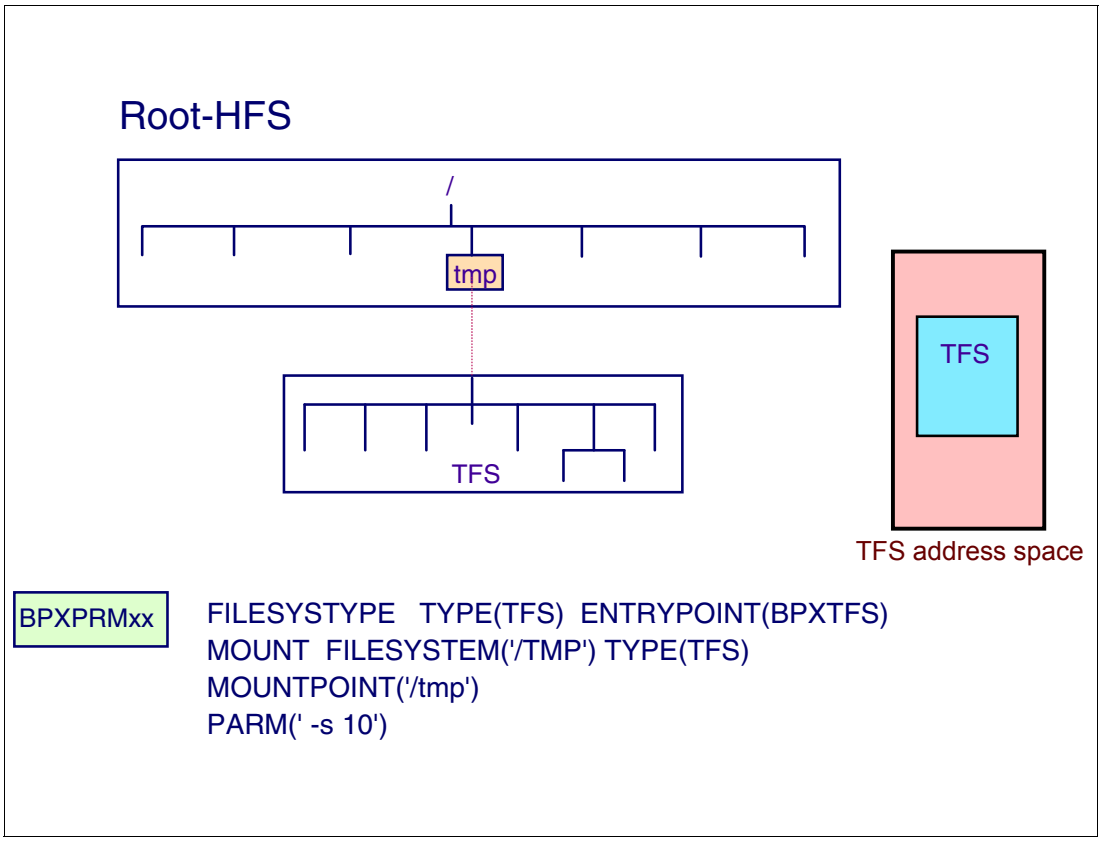


Figure 9-8 Temporary file system and temporary file system address space

Temporary file system (TFS)

The /tmp (temporary) directory is where many programs running on the system keep temporary copies of files or data. Users often use the tmp directory for working space knowing that they should not put anything out there that they want to keep. Currently the /tmp directory is part of the root HFS data set. The data in /tmp is not automatically deleted unless you have set up some mechanism (like the cron automation daemon) to delete the data at certain intervals. If the data is not deleted, you could run into space problems over time.

The TFS is an in-memory physical file system that supports in-storage mountable file systems. Because it is an in-memory file system, all data that is written to a TFS is lost over an IPL. This makes it a good candidate for the /tmp directory. Not only do you get a new clean /tmp file system at each IPL, you also get very high I/O access because it resides in a data space that is part of the kernel address space.

Creating a TFS

The ServerPac installation jobs already defined the following FILESYSTYPE definition in SYS1.PARMLIB(BPXPRMFS):

```
FILESYSTYPE TYPE(TFS) ENTRYPOINT(BPXTFS)
```

Edit SYS1.PARMLIB(BPXPRMFS), that is, add the mount statement to mount an in-memory file system at the /tmp mount point. You can add the following mount statement right under the FILESYSTYPE TYPE(TFS) definition:

```
MOUNT FILESYSTEM('/TMP')
       MOUNTPOINT('/tmp')
       TYPE(TFS)
       PARM('-s 10')
```

(-s 10) allocates 10MB of storage.

Figure 9-8 shows the mount command for a TFS. Normally this would be included in BPXPRMxx. Key points to remember are the following:

- The file system name should be unique. If you use the pathname of the mount point, it will simplify the understanding of the output of the **df** command.
- Type must be a `TFS`.
- Mountpoint is /tmp.
- `Parm` specifies how much virtual storage the TFS should use. The default is 1 MB; `PARM (-s 10)` specifies 10 MB.
- You cannot mount a TFS using a DDname.
- If unmounted, all data stored in a TFS is lost; when remounted, the file system has only dot(.) and dot-dot(..) entries.

9.9 Colony address space

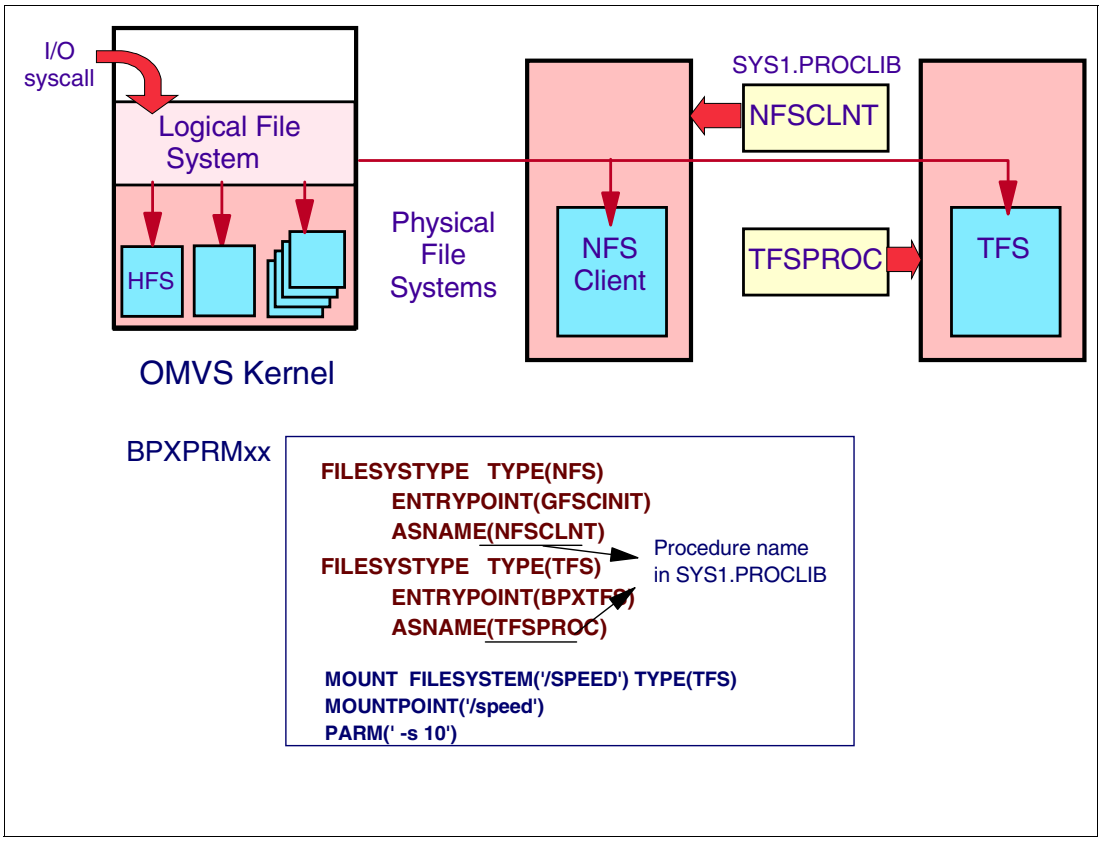


Figure 9-9 Colony address spaces

z/OS UNIX colony address spaces

The standard z/OS UNIX Physical File Systems (PFS) run as tasks in the kernel address space. However, it is possible to package a physical file system component in a separate address space, called a *colony address space*.

The Logical File System in the kernel provides the linkage to the new PFS in the colony address space. To a z/OS UNIX user application, there is no difference in interaction between accessing the standard PFS and one in a colony address space.

Running a PFS in a colony address space has some advantages, as follows:

- It is a simpler way for other components to add a new PFS to z/OS UNIX environment.
- A PFS can use a different security ID to kernel, and allocate its own z/OS data sets.
- A PFS can run as a standard process with normal C/C++ SYSCALL linkages.

When a PFS runs in a colony address space, an extra address space is created, and each PFS operation has a slightly longer path length.

Note: The cataloged procedure (named with ASNAME) must contain the statement:

```
EXEC PGM=BPXVCLNY
```

Creating colony address spaces

The following tasks must be performed to run the colony address space:

- Define FILESYSTYPE TYPE(TFS) in BPXPRMxx.
- Use a procedure name for the TFS, in the ASNAME in the BPXPRMxx.
- Define a profile in the started class for the procedure name.
- Add the MOUNT FILESYSTEM('/xxxx') to the BPXPRMxx.
- IPL the system.

The RACF profile for started class could look like:

```
RDEFINE STARTED TFSPROC.**   -
    STDATA(USER(OMVSKERN) GROUP(OMVSGRP) TRUSTED(NO))
```

For physical file systems that do not run in the kernel address space, ASNAME specifies the name of the procedure in SYS1.PROCLIB used to start the address space in which this PFS will be initialized. The procedure name is also used as the name of the address space. For example, for the NFS client, you must create a procedure to run a PFS in a colony address space.

9.10 Mounting file systems

- Using the usr/sbin/mount REXX exec from the shell
- Using the TSO MOUNT command
- Using the mount shell command (/usr/sbin)
- Using the ISHELL File_Systems pull-down
- Adding a mount statement to the BPXPRMxx member in SYS1.PARMLIB so that it will be mounted when the system re-IPLs
- Direct mount
- Automount facility

Figure 9-10 Ways to mount zFS file systems

Ways to mount file system

A file system is the entire set of directories and files, consisting of all HFS files shipped with the product and all those created by the system programmer and users. The system programmer (superuser) defines the root file system; subsequently, a superuser can mount other mountable file systems on directories within the file hierarchy. Altogether, the root file system and mountable file systems comprise the file hierarchy used by shell users and applications.

Figure 9-10 shows the different methods of mounting file systems.

UNIX shell mounts

The REXX exec **/usr/sbin/unmount** performs essentially the same functions that the TSO/E **mount** command performs. You can run it from the shell.

For hierarchical file systems, you can use the **chmount** command to logically mount, or add, a mountable file system to the file system hierarchy. This is the same command used by the REXX exec **/usr/sbin/mount**.

TSO mounts

Have an authorized user enter a TSO/E **mount** command to logically mount the file system.

Using the File Systems pull-down from the ISHELL panel and Option 3, you can perform the mount for a file system.

PARMLIB mounts

The MOUNT statement defines the hierarchical file systems to be mounted at initialization and where in the file hierarchy they are to be mounted. It is up to the installation to ensure that all HFS data sets specified on MOUNT statements in the BPXPRMxx PARMLIB member are available at IPL time.

z/OS UNIX mounting

For a direct mount, you need to allocate an intermediate HFS data set to be mounted between the root file system and all user file systems. Create a mount point using the **mkdir** command and issue the **mount** command. (To make the mount permanent you will also need to add the HFS data set name and its mount point to the BPXPRMxx member of PARMLIB.)

Using the automount facility simplifies management of file systems. You do not need to mount most file systems at initialization and you do not need to request that operators perform mounts for other file systems. In addition, it is easier to add new users because you can keep your PARMLIB specification stable. You can establish a simple automount policy to manage user home directories.

9.11 Mount and unmount

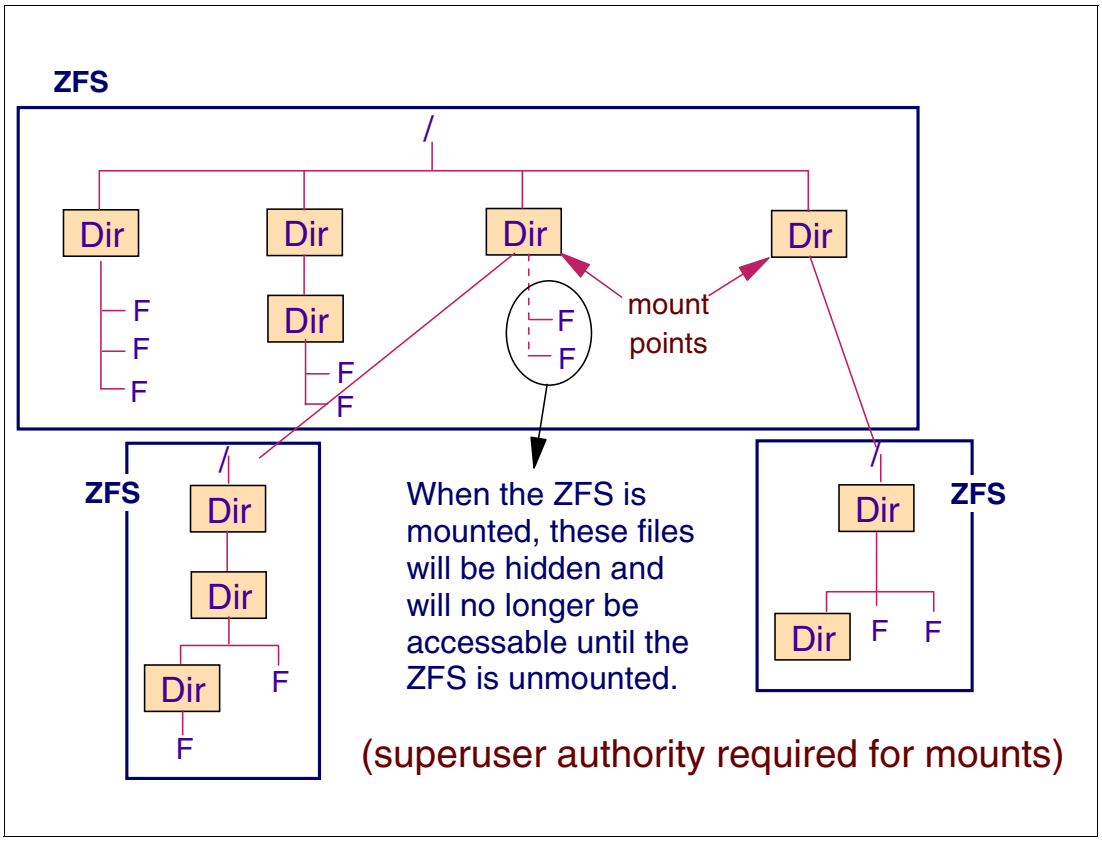


Figure 9-11 Mounting of file system considerations

Mounting and unmounting file systems

A file system must be mounted before it can be accessed. When z/OS UNIX is started, the root file system is mounted as the first file system. All other file systems included on the MOUNT statements in the BPXPRMxx members or mounted with the TSO command MOUNT will be mounted on the root file system or on other file systems.

A file system can be mounted in read/write mode or in read-only mode. If it is mounted in read-only mode, nobody can update any files or directories in the file system. To change the mount mode, the file system must be unmounted and then remounted with a mode of read/write.

A file system must be mounted on a directory; this is called a *mount point*. Use empty directories as mount points. If a directory is not empty, the existing files will be overlapped by the file system which is mounted on the directory. When this file system is unmounted, the existing files will be accessible and visible again.

Superuser authority is required to mount or unmount a file system. That means the user must have a OMVS UID(0) or have a RACF profile that defines superuser authority for the user.

If the file system does not need to be updated, it could be mounted in a read-only mode. This will improve the performance. However, if a file system mounted as read-only needs updates, it must be unmounted and remounted again.

9.12 Managing user file systems

- Mount user file systems at the /u mount point
  - Two ways to mount:
    - (1). Direct mount (2) Automount facility

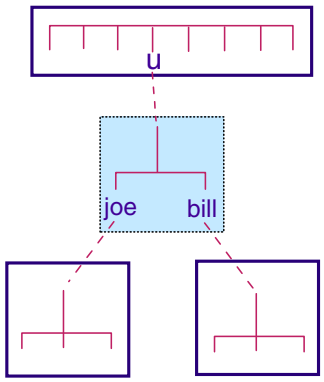


Figure 9-12 Managing the mounting of file systems of z/OS UNIX users

Ways to mount

When a user requires an HFS or zFS file system to be accessed, you need to get it mounted at a mount point off of the root directory to make it available. The preferred place to mount all user HFS or zFS file systems is the **/u** mount point. In z/OS UNIX, there are two ways to accomplish this:

Direct mount Allocate an intermediate HFS data set to be mounted between the root file system and all user file systems.

Create a mount point using the **`mkdir`** command and issue the **`mount`** command. To make the mount permanent you will also need to add the HFS data set name and its mount point to the BPXPRMxx member of PARMLIB.

Automount You need to customize the automount facility to control all user file systems to automatically mount them when they are needed. This is the preferred method to manage user HFS data sets because it saves administration time.

The automount facility lets you designate directories as containing only mount points. This is the preferred method of managing user HFS file systems. As each of these mount points is accessed, an appropriate file system is mounted. The mount point directories are internally created as they are required. Later, when the file system is no longer in use, the mount point directories are deleted.

9.13 User file systems: Direct mount

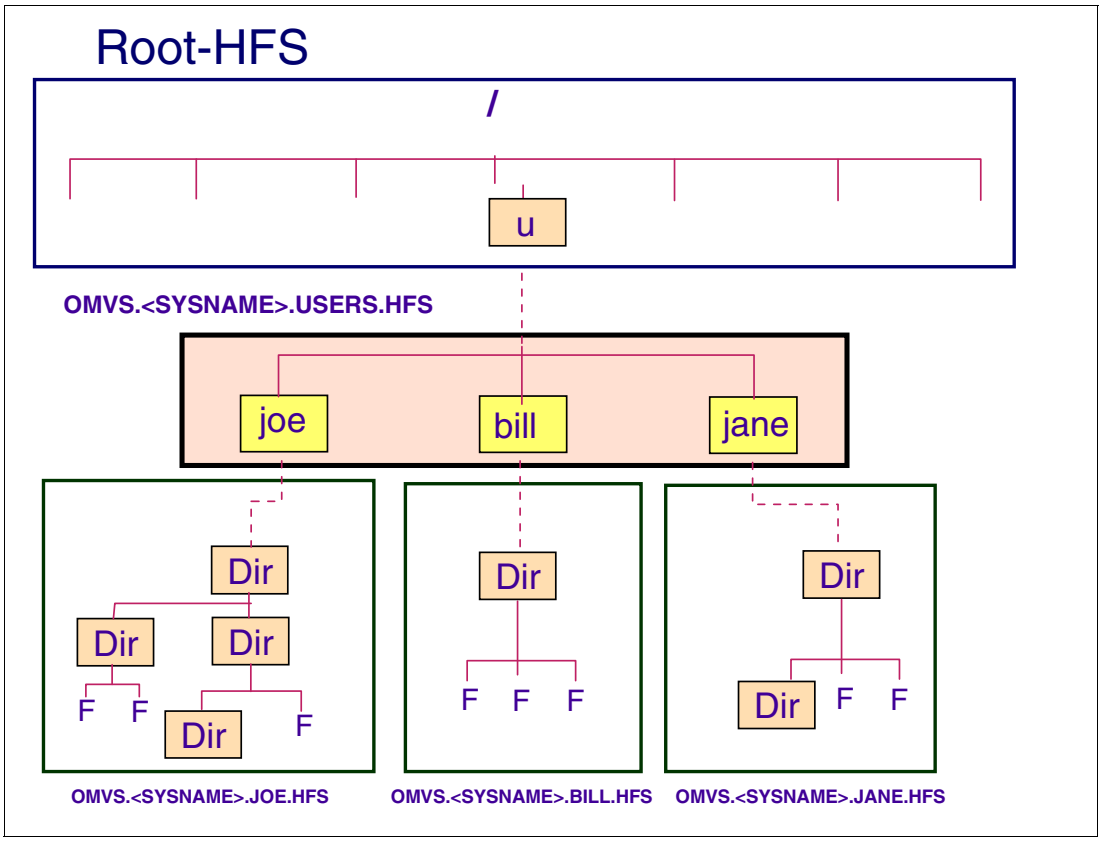


Figure 9-13 Mounting user file systems with direct mount

Direct mount

You should set up the root file system so that it does not require frequent changes. This can be accomplished by putting user data in user file systems. The **/u** directory contains the user home directories; if these directories are placed in separate file systems, most of the user data will be kept out of the Root file system.

The UNIX System Service programmer can choose to use only one HFS for all users, or use one HFS per user. If the user HFS is too small, you can mount some of the users to another user's HFS, or increase the user space for the HFS. This will make it easier to manage the HFS.

User home directory

A recommendation is to name the user home directories **/u/userid** with the user ID in lowercase, for example **/u/joe** for user JOE as shown in the visual.

Following is a suggested method for creating user file systems:

- Leave the /u directory in the root file system empty.
- Create an HFS file called OMVS.<SYSNAME>.USERS.HFS.

This file system will contain the home directories for all users, and will be mount points for the user file systems. The reason for keeping these home directories in a separate file system is to avoid updating the root file system each time a new home directory is created or deleted. The second qualifier of the DSNAME identifies the z/OS system image it relates to.

- Create a file system for each user or for a department, depending on what is best for the installation. The user file systems can be called OMVS.<SYSNAME>.<userid>.HFS.

  The user file systems will be mounted on the home directories in the OMVS.<SYSNAME>.USERS.HFS file system.

Define a file system for user directories

You should have a separate file system just to contain the user's home directories. These home directories will be empty, and they will act as mount points for the user file systems. Depending on how the installation is organized and on the need for user space, each user can have their own file system, or users in a department can share a system. It is easier to control space usage when each user has their own file system. It can be compared to users having their own z/OS data set for user data, or their own minidisk on VM. How large it should be depends entirely on who will use it and what the user will do. If multiple users share an HFS data set, it is not possible to guarantee space for each user. One user can dominate the space in a file system shared between multiple users.

When making plans for a file structure, it is important to think about a naming convention for the HFS data sets. If the automount facility will be used, one of the data set qualifiers should be equal to the directory name where the file system will be mounted.

9.14 Mounting file systems

```
  File  Directory  Special_file  Tools  File_systems  Options  Setup  Help
                                                                 ________
                      UNIX System Serv  | 3  1. Mount table...  |
Command ===> ________________________   |    2. New HFS...     |  _______
                                        |    3. Mount(O)...    |
Enter a pathname and do one of these:   |    4. New ZFS...     |
                                        |    5. ZFS aggregates...|
    - Press Enter.
    - Select an action bar choice.
    - Specify an action code or command on the command line.

Return to this panel to work with a different pathname.
                                                            More:     +
   /u
   ____________________________________________________________________
   ____________________________________________________________________
   ____________________________________________________________________

EUID=0
```

TSO/E Commands

- **MOUNT FILESYSTEM('OMVS.JANE.HFS') MOUNTPOINT('/u/jane') TYPE(HFS) MODE(RDWR)**
- **UNMOUNT FILESYSTEM('OMVS.JOE.HFS') NORMAL**

Figure 9-14 How to mount file systems

Mounting from the ISHELL

The ISHELL is a powerful application based on ISPF. You can do many things with the menu shown at the top of Figure 9-14. The eight menu options have the following meanings:

File	Edit, browse, delete, copy, rename, print, run, and so on
Directory	List, create, rename, print, find string, and so on
Special_file	New FIFO, link, Attribute, delete, rename, and so on
Tools	Processes, Shell commands, run programs, and so on
File_systems	Mount, unmount, change attribute, allocate HFS, and so on
Options	Directory list, edit, browse, settings, and so on
Setup	User and Group Admin., create TTY, RACF permit Field, and so on
Help	Action code, Help, Keys help, About, and so on

Mounting from TSO/E

A file system can be mounted by using the TSO/E MOUNT command or the ISHELL. Superuser authority is required for mounting or unmounting a file system.

The options for the MOUNT command are the same as for the MOUNT statement in BPXPRMxx, except for an additional option called `WAIT` or `NOWAIT`. This option specifies whether to wait for an asynchronous mount to complete before returning.

The syntax for the UNMOUNT command is as follows:

```
UNMOUNT FILESYSTEM(file_system_name)
DRAIN | FORCE | IMMEDIATE | NORMAL | REMOUNT(RDWR | READ) | RESET
```

DRAIN Specifies that an unmount drain request is to be made. The system will wait for all use of the file system to be ended normally before the unmount request is processed or until another UNMOUNT command is issued.

FORCE Specifies that the system is to unmount the file system immediately. Any users accessing files in the specified file system receive failing return codes. All data changes to files in the specified file system are saved, if possible. If the data changes to the files cannot be saved, the unmount request continues and data is lost.

IMMEDIATE Specifies that the system is to unmount the file system immediately. Any users accessing files in the specified file system receive failing return codes. All data changes to files in the specified file system are saved. If the data changes to files cannot be saved, the unmount request fails.

NORMAL Specifies that if no user is accessing any of the files in the specified file system, the system processes the unmount request. Otherwise, the system rejects the unmount request. This is the default.

REMOUNT (RDWR|READ) Specifies that the specified file system be remounted, changing its mount mode. REMOUNT takes an optional argument of RDWR or READ. If you specify either argument, the file system is remounted in that mode if it is not already in that mode. If you specify REMOUNT without any arguments, the mount mode is changed from RDWR to READ or READ to RDWR.

RESET A reset request stops a previous UNMOUNT DRAIN request.

WAIT Specifies that MOUNT is to wait for the mount to complete before returning. WAIT is the default.

NOWAIT Specifies that if the file system cannot be mounted immediately (for example, a network mount must be done), then the command will return with a return code indicating that an asynchronous mount is in progress.

9.15 Option 3: Mount

```
File  Directory  Special_file  Tools  File_systems  Options  Setup  Help
                          Mount a File System

Mount point:
                                                              More:     +
   /web/hod
   ____________________________________________________________________

File system name   omvs.sc64.web.hod
File system type   ZFS_____  New owner  . . . . . ________
Owning system  . . sc64____  Character Set ID . . _____

Select additional mount options:
_  Read-only file system          _  Set automove attribute...
_  Ignore SETUID and SETGID       _  Text conversion enabled
_  Bypass security

Mount parameter:
_____________________________________________________________

 F1=Help        F3=Exit        F4=Name        F6=Keyshelp   F12=Cancel
```

Figure 9-15 Mounting file systems from the ISHELL with Option 3

Mounting from the ISHELL

The ISHELL command invokes the z/OS ISPF shell, a panel interface that helps you to set up and manage z/OS UNIX System Services functions such as mounting of file systems.

To mount a file system, you must be a superuser.

If you are a user with an MVS background, you may prefer to use the ISPF shell panel interface instead of shell commands or TSO/E commands to work with the file system. The ISPF shell also provides the administrator with a panel interface for setting up users for z/OS UNIX access, for setting up the root file system, and for mounting and unmounting a file system.

Select the options you require on the panel. The mount point must be a directory. If it is not an empty directory, files in that directory are not accessible while the file system is mounted.

Only one file system can be mounted at a directory (mount point) at any one time.

9.16 Automount facility

- Used to simplify management of your filesystems
  - Eliminates mounting files at initialization
  - Eliminates operators performing mounts
  - Easier to add a new user's file system
  - Files not mounted until required
- Use BPXPRMxx and two definition files
  - /etc/auto.master
  - /etc/auto.map

Figure 9-16 Using the automount facility

Automount facility

Use the automount facility to simplify management of your file system. With this facility, you do not need to mount most file systems at initialization and you do not need to request that operators perform mounts for other file systems. In addition, the facility simplifies the addition of new users because you can keep your PARMLIB specification stable. You can establish a simple automount policy to manage user home directories.

The automount facility also helps you to avoid consuming resources until they are requested. A file system that is managed by the automount facility remains unmounted until its mount point is accessed.

In addition, the automount facility helps you to reclaim system resources used by a mount if that file system has not been used for some period of time. You can specify how long the file system should remain mounted after its last use.

The automount facility lets you designate directories as containing only mount points. This is the preferred method of managing user HFS data sets. As each of these mount points is accessed, an appropriate file system is mounted. The mount point directories are internally created as they are required. Later, when the file system is no longer in use, the mount point directories are deleted.

Try to think of automount as an administrator that has total control over a directory. When a name is accessed in this directory, it looks up in its policy what file system is supposed to be associated with that name. If it finds one, it (logically) performs a **`mkdir`** followed by a **`mount`**, then quietly moves out of the way. Once out of the way, the root directory of that newly mounted file system is now accessed as that name.

automount definition files

You need to customize the configuration files before you can start using the automount facility, as follows:

/etc/auto.master

This file contains the directory or directories that the automount facility will monitor. It also contains an associated MapName file that contains the mount parameters.

It specifies that the automount facility is to manage the `/u` directory. If someone using kernel services tries to access a directory in the `/u` directory, the automount facility automatically mounts the data set based on the MapName policy in:

/u /etc/u.map

The name of the map file can be specified as a data set name, which must be specified as a fully qualified name and can be uppercase or lowercase. Single quotation marks are not needed.

Example: /u //sys1.parmlib(amtmapu)

9.17 Automount setup

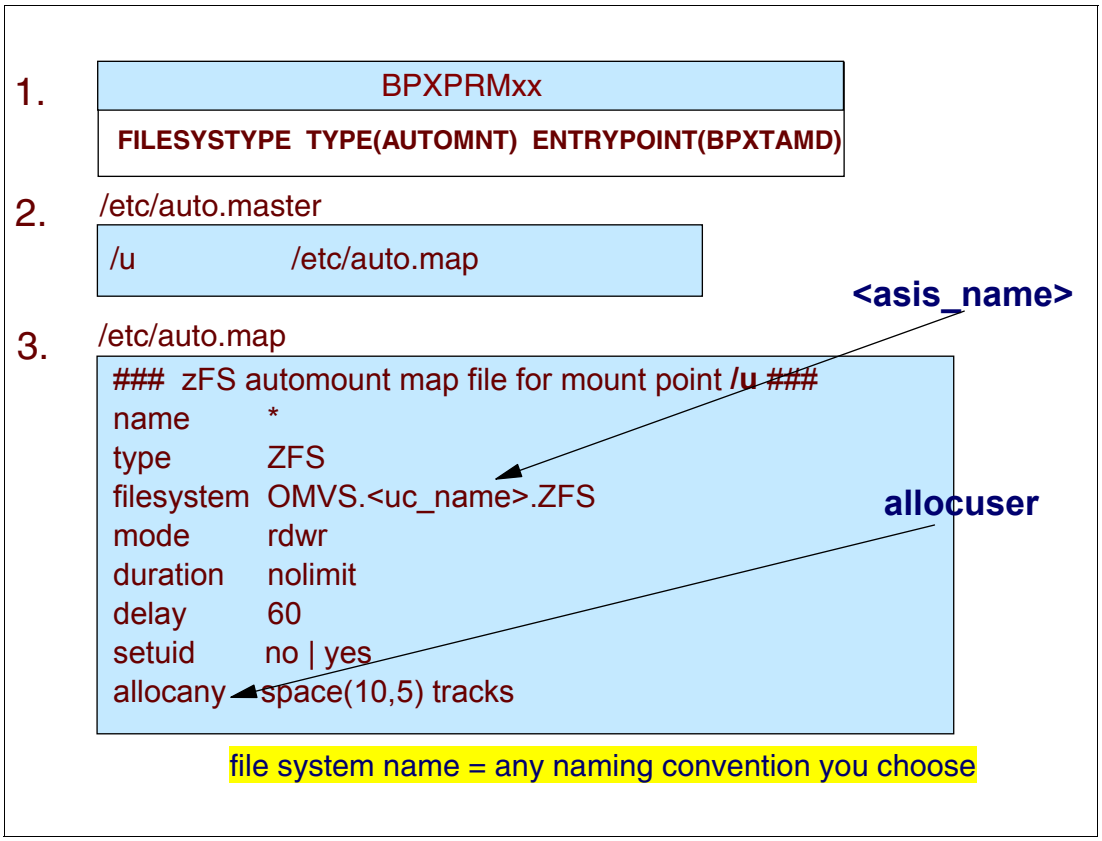


Figure 9-17 Automount facility policy setup

Setting up the automount policy

To use the automount facility, the following statement must be added to the BPXPRMxx member:

```
FILESYSTYPE TYPE(AUTOMNT) ENTRYPOINT(BPXTAMD)
```

Automount policy files

Automount uses two HFS files for specifying the policy for which file systems should be automatically mounted when referenced:

/etc/auto.master Contains a list of directories to be managed, along with their MapName files.

/etc/u.map Is the MapName file for a directory.

The MapName file contains the mapping between a subdirectory of a directory managed by the automount facility and the mount parameters. It can contain both specific entries and a generic entry. When the automount facility tries to resolve a lookup request, it attempts to find a specific entry. If a specific entry does not exist for the name being looked up, it will then attempt to use the generic entry, as follows:

ZFS automount map file for mount point /u ###. The use of the # symbol indicates a comment statement in the map file.

name The name of the mount point directory. An asterisk (*) specifies a generic entry for the automount-managed directory.

type File system type. The default is HFS.

filesystem Data set name of the file system to mount. Two special symbols are supported to provide name substitution:

<asis_name> Used to represent the name exactly, as is.

<uc_name> Used to represent the name in uppercase characters.

These can be used when specifying a file system name or file system parameter that has a specific form, with the name inserted as a qualifier.

mode Mount mode.

duration The minimum amount of time in minutes to leave the file system mounted. The default is nolimit.

delay The minimum amount of time in minutes to leave the file system mounted after the duration has expired and the file system is no longer in use. The default is 0.

setuid Can be specified as yes or no. This will support or ignore the SETUID and SETGID mode bits on executable files loaded from the file system.

&SYSNAME Indicates that system symbols are supported in the map file.

allocany This will cause an allocation if the zFS data set does not exist for any name looked up in the automount managed directory.

allocuser This will cause an allocation to occur only if the name looked up matches the user ID of the current user.

allocuser allocation-spec - Specifies the allocation parameters when using automount to allocate HFS or zFS file systems, keeping in mind that zFS is the preferred file system. allocuser will cause an allocation to occur only if the name looked up matches the user ID of the current user.

Where:

allocation-spec is a string that specifies allocation keywords. Keywords can be specified in the string, as follows:

```
space(primary-alloc[,secondary alloc])
cyl | tracks |  block(block size)
vol(volser[,volser]...)
maxvol(num-volumes)
unit(unit-name)
storclas(storage-class)
mgmtclas(management-class)
dataclas(data-class)
```

The next four keywords are automatically added:

```
dsn(filesystem)
dsntype(hfs)
dir(1)
new
```

9.18 Generic match on lowercase names

- <asis_name>
  - This represents the exact name of the subdirectory to be “automounted”. If name is in uppercase, the substitution name in the file system name is in uppercase. If the name is in lowercase, the substitution name results in lowercase.
- <uc_name>
  - This represents the name of the subdirectory to be “automounted” in uppercase characters. In this case, /u/user1 and /u/USER1 mount point directories map the same file system
- lowercase[YES]
  - This indicates that only names in lowercase (special characters are also allowed) match the * specification
- lowercase[NO]
  - This is the default and indicates that any names will match the * specification.

Figure 9-18 Specification of the automount map file keywords

Keywords in automount policy

Four special symbols are supported to provide name substitution:

<asis_name> Used to represent the name exactly, as is.

<uc_name> Used to represent the name in uppercase characters.

<sysname> Used to substitute the system name.

&SYSNAME. Used to substitute the system name. This is new with z/OS V1R3.

Attention: IBM recommends that you use &SYSNAME. <sysname> is only temporarily supported for compatibility.

You can use these when specifying a file system name or file system parameter that has a specific form with the name inserted as a qualifier.

lowercase [Yes|No] - It indicates the case for names that can match the * specification. This keyword is valid on any specification, but is only meaningful on the generic entry.

Yes Only names composed of lowercase characters can match the * specification (numbers and special characters may also be used). When this is specified, uppercase characters are not allowed.

No Any names can match the * specification. This is the default.

9.19 Automount facility overview

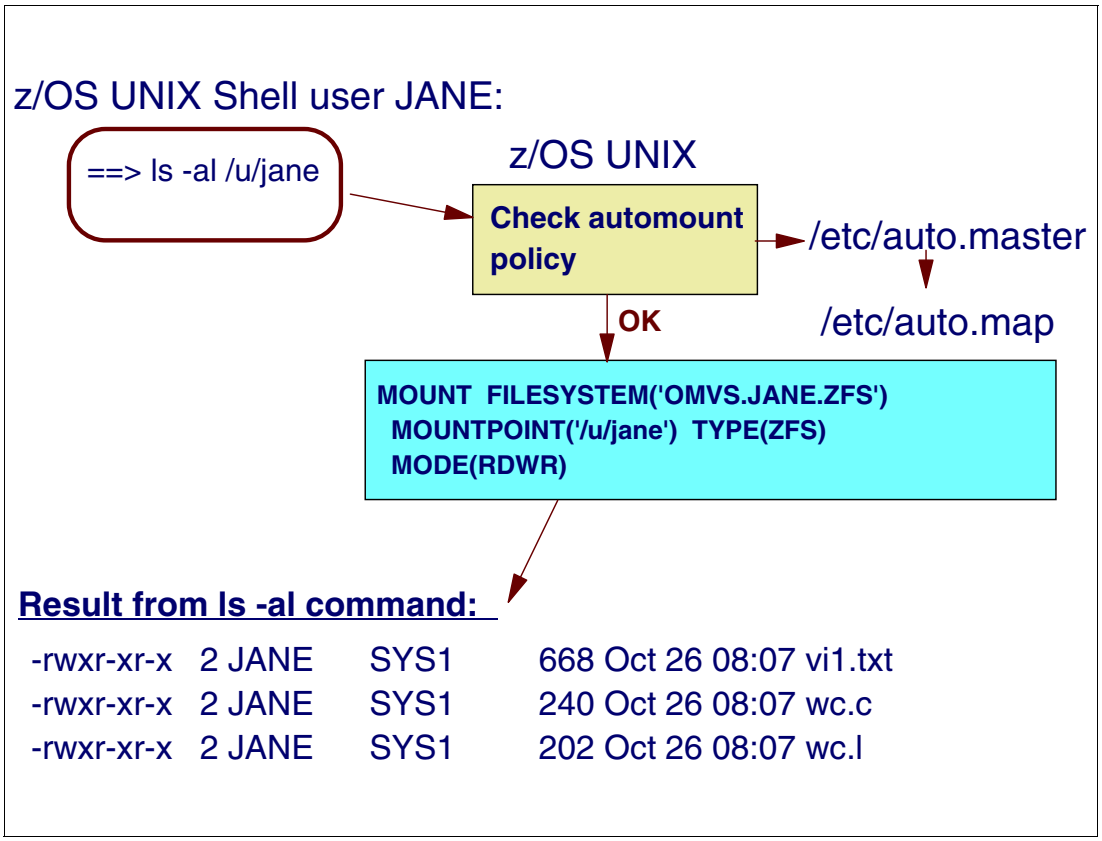


Figure 9-19 An overview of automount facility processing

Overview of automount facility processing

The automount facility simplifies the management of mountable file systems.

With automount, an installation defines the directories that should be managed in a policy file. These directories must be the parent directories of the mount point directories. When a mount point directory (which is a subdirectory of a managed directory) is referenced in a pathname, the system creates the mount point directory and mounts the proper file system.

The root directory cannot be managed by automount.

With automount, it is also possible to specify how long a file system should remain unreferenced before it is automatically unmounted. When a mounted file system is unmounted, the mount point directory is deleted.

9.20 Activating automount

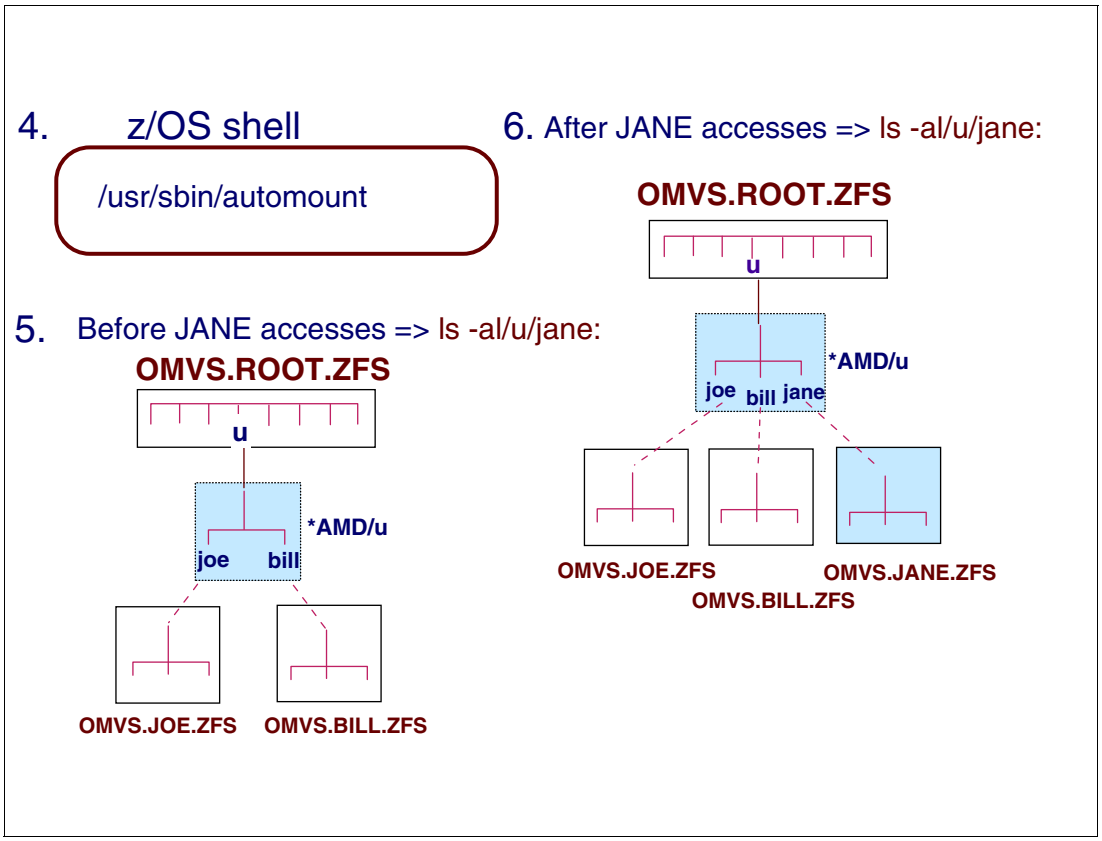


Figure 9-20 Activating the automount facility when a user accesses a file system

Activating the automount policy

The **`automount`** command is used to configure the z/OS UNIX automount facility. When run with no arguments, automount reads the /etc/auto.master file to determine all directories that are to be configured for automounting and the filenames that contain their configuration specifications.

`automount -s` can be specified to check the syntax of the configuration file. No automount is performed when using the -s option.

The **`automount`** command requires superuser authority.The automount command is located in the /usr/sbin directory. The superuser should have this directory in their PATH environment variable.

When the HFS is first allocated for a new user and **`automount`** is used to dynamically allocate a mount point, the new mountpoint directory has permission bits of 700 and the owner is the superuser. A **`chown`** command will have to be issued to change the ownership.

After the **`automount`** command is done, the system will automatically mount a file system if the mount point directory is managed by the automount facility.

9.21 SETOMVS RESET=xx implementation

- Designed to be used with a subset of the BPXPRMxx parmlib statements to add Physical File Systems to a configuration
  - **FILESYSTYPE** - Basic PFS Definition Statement
  - **SUBFILESYSTYPE** - Stack Definitions under Cinet
  - **NETWORK** - Socket Stack Domain Parameters
  - System limits - MAXPROCSYS, etc...
- Logical extension of SET OMVS=(xx,yy,...)
- Only one keyword and member allowed:
  SETOMVS RESET=(xx)

Figure 9-21 Option to start a PFS with an operator command

Defining automount in BPXPRMxx member

The **setomvs** command enables you to modify BPXPRMxx PARMLIB settings without re-IPLing. So if you need to add the automount physical file system to your environment, the **setomvs** command can be used with the reset= keyword. For example:

```
setomvs reset=xx
```

You can use the **setomvs reset** command to dynamically add the FILESYSTYPE, NETWORK, and SUBFILESYSTYPE statements without having to re-IPL. You place those statements in a BPXPRMxx member, say BPXPRMtt, and issues the command, as follows:

```
setomvs reset=tt
```

However, if you change the values in the PARMLIB member, a re-IPL will be necessary.

To make the dynamic changes when you use the **setomvs** command permanent, you will have to edit the BPXPRMxx member that is used for IPLs.

9.22 Issue the SETOMVS command

- Add an automnt physical file system to a new
  - BPXPRMtt member
- Issue system command:
  - SETOMVS RESET=(tt)

FILESYSTYPE TYPE(AUTOMNT)
ENTRYPOINT(BPXTAMD)

- With this command, you can mount the following:
  - SET OMVS=tt
    - ROOT, MOUNT, FILESYSTYPE, SUBFILESYSTYPE, and NETWORK statements

Figure 9-22 Example of issuing the SETOMVS command

Activate the automount physical file system

These steps show how to set up the automount facility to mount user file systems:

- Add the FILESYSTYPE statement to the BPXPRMxx member as follows:
  ```
  FILESYSTYPE TYPE(AUTOMNT)
  ENTRYPOINT(BPXTAMD)
  ```
- Use the **setomvs** command to dynamically specify the new FILESYSTYPE statements; this changes the current system settings. (You cannot use the **setomvs** or **set omvs** command to specify the new statements.)
- Issue the **setomvs reset=** command to dynamically create to AUTOMNT physical file system.

To make a permanent change, edit the BPXPRMxx member used for IPLs by placing the FILESYSTYPE statement into it.

The SET OMVS command

Another way to dynamically reconfigure parameters is to use the **SET OMVS** command to specify a BPXPRMxx PARMLIB member to make changes in your system dynamically. With the **SET OMVS** command, you can have multiple BPXPRMxx definitions and use them to easily reconfigure a set of the z/OS UNIX system characteristics. You can keep the reconfiguration settings in a permanent location for later reference or reuse.

SET OMVS=xx can be used to execute the ROOT, MOUNT, FILESYSTYPE, SUBFILESYSTYPE, and NETWORK statements in the BPXPRMxx PARMLIB member.

9.23 Updating an existing automount policy

- An option (-a) added to the command line:
  - "-a" - option indicates that the policy being loaded is to be appended
  - Example: /usr/sbin/automount -a
  - Option “-a” is mutually exclusive with query option “-q”
- New option (-q) added to command line
  - "-q" - option displays current active policy

Figure 9-23 New options to update an existing automount policy

Update automount policy in storage

The **automount** command is used to configure the automount facility. This facility can automatically mount file systems at the time they are accessed, and also unmount them later. You can use a single automount policy to manage both HFS and zFS file systems.

```
/usr/sbin/automount
```

When run with no arguments, **automount** reads the `/etc/auto.master` file to determine all directories that are to be configured for automounting and the file names that contain their configuration specifications.

Operands to update a policy

Two new operands can be used to update an automount policy, as follows:

-a Indicates that the policy being loaded is to be appended to the existing policy rather than replace the existing policy. For example:

```
/usr/sbin/automount -a
```

-q Displays the current automount policy.

The operands -q and -a are mutually exclusive.

9.24 Example of new options

```
ROGERS @ SC65:/u/rogers>/usr/sbin/automount -q
/u
name                    *
filesystem              OMVS.<uc_name>.ZFS
type                    ZFS
allocuser               space(1,1) storclas(OPENMVS)
mode                    rdwr
duration                1440
delay                   360

ROGERS @ SC65:/u/rogers>oedit /etc/u.map      <---- Changed duration to 600
ROGERS @ SC65:/u/rogers>/usr/sbin/automount -a
FOMF0107I Processing file /etc/u.map
FOMF0108I Managing directory /u
ROGERS @ SC65:/u/rogers>/usr/sbin/automount -q
/u
name                    *
filesystem              OMVS.<uc_name>.ZFS
type                    ZFS
allocuser               space(1,1) storclas(OPENMVS)
mode                    rdwr
duration                600
delay                   360
```

Figure 9-24 Example showing changing the auto.map file while policy is active

Changing the auto.map file

The first command in Figure 9-24 displays the current active automount auto.map file, as follows:

```
ROGERS @ SC65:/u/rogers>/usr/sbin/automount -q
```

The next command entered from the OMVS shell is to edit the current auto.map file. During the edit of the file, the duration is changed from 1440 to 600.

The next command, with the -a option, causes the changed auto.map file to replace the current active auto.map file in storage with the changed one.

The last command in the figure displays the current active auto.map file which shows the changed duration.

9.25 One auto.master for a sysplex

- The syntax is enhanced to indicate a master file name as a data set name:
  - Master file name is specified on the command line
  - Data set name can be sequential or member of PDS
  - Convention of // preceding the data set name is used
  - Data set name must be a fully qualified name and can be in upper or lower case
- /usr/sbin/automount “//auto.mapfile(automap)”
- auto.master contains - /u //auto.mapfile(automap)

Figure 9-25 One auto.master for a sysplex

auto.master for a sysplex

The default file name of the master file is /etc/auto.master. It contains the directory or directories that the automount facility will monitor. It also contains an associated MapName file that contains the mount parameters.

Option to use an MVS data set

Figure 9-25 shows an example of a /etc/auto.master file that is placed in an MVS data set. It specifies that the automount facility is to manage the /u directory. If someone using kernel services tries to access a directory in the /u directory, the automount facility automatically mounts the data set based on the MapName policy shown in the visual.

This change makes it possible in a sysplex to share the `auto.master` file and allow protection against simultaneous updates from each system.

The name of the map file can be specified as a data set name. The data set name must be specified as a fully qualified name and can be uppercase or lowercase. Single quotes are not needed.

9.26 HFS to zFS automount

- Change automount to set the file system type to either HFS or zFS when either of those is specified based on the type of data set that is being mounted
  - Specify and manage both HFS and zFS file systems in one automount policy
  - Facilitates migration from HFS to zFS over time rather than all file systems at once
- Determine whether the data set is a HFS type or not:
  - If it is, then set the file system type to HFS
  - If it is not, then set the file system type to zFS

Figure 9-26 Using a single auto.map file for automounting HFS and zFS file systems

Single auto.map file for HFS and zFS

You can use a single automount policy to manage both HFS and zFS file systems in the same managed directory. Automount is changed such that when HFS or zFS is specified as the file system TYPE, the data set will be checked to determine what type of data set it is, and direct the mount to the appropriate physical file system (HFS or zFS).

This allows automount managed file systems to be changed from HFS to zFS without changing the file system name and without changing the automount policy. If the file system name must be changed, it will be necessary to add a specific entry in the automount policy for this file system or manage it on another managed directory.

9.27 HFS to zFS automount

- New support is used when HFS or zFS is specified as the file system type in the automount policy
  - If "allocany" or "allocuser" is specified, a new file system is allocated using the file system type specified in the automount policy
- If the data set already exists, a check is done to see whether it is an HFS file system or a zFS aggregate and then the mount is directed to the appropriate PFS

```
/u
name                    *
filesystem              OMVS.<uc_name>.HFS
type                    ZFS
allocuser               space(1,1)
storclas(OPENMVS)
mode                    rdwr
duration                1440
delay                   360
```

Figure 9-27 HFS to zFS automount policy

HFS to zFS automount policy

Automount recognizes the `type` specification in the automount map files of HFS and zFS as potentially interchangeable file system types.

At the time automount applies the specification for the mount it will determine if the file system is the name of either an HFS or zFS data set and alter the type as appropriate.

If the data set does not exist and `allocany` or `allocuser` is not specified, the type will generally be assumed to be zFS. If either `allocany` or `allocuser` is specified, a new file system is allocated as before, of the file system type specified in `type`. If your preference is to have new file systems allocated as zFS file systems, the automount policy should be changed to specify type zFS.

allocuser

The automount facility recognizes the type specification in the automount map files of HFS and zFS as potentially interchangeable file system types. At the time automount applies the specification for the mount, it will determine whether the file system is the name of either a zFS or HFS file system and alter the type as appropriate. If the data set does not exist and if **`allocany`** or **`allocuser`** is not specified, a new file system is allocated as the file system type as specified in type. Allocation is only done if **`allocuser`** or **`allocany`** is specified. If it is preferred to have new file systems allocated as zFS file systems, the automount policy should be changed to specify type zFS. The following keywords can be specified in the string:

```
space(primary-alloc[,secondary alloc])
```

```
cyl | tracks |  block(block size)
vol(volser[,volser]...)
maxvol(num-volumes )
unit(unit-name)
storclas(storage-class)
mgmtclas(management-class)
dataclas(data-class)
```

The next four keywords are automatically added:

```
dsn(filesystem)
dsntype(hfs)
dir(1)
new
```

allocany

allocany specifies the allocation parameters when using automount to allocate HFS or zFS file systems, keeping in mind that zFS is the preferred file system. **allocany** will cause an allocation if the data set does not exist for any name looked up in the automount-managed directory. The following keywords can be specified in the string:

```
space(primary-alloc[,secondary alloc])
cyl | tracks |  block(block size)
vol(volser[,volser]...)
maxvol(num-volumes)
unit(unit-name)
storclas(storage-class)
mgmtclas(management-class)
dataclas(data-class)
```

The next four keywords are added as appropriate:

```
dsn(filesystem)
dsntype(hfs)
dir(1)
new
```

9.28 Automount migration considerations

- File system name contains substitution strings
  - Data set status is determined after substitution is performed substituting ZFS or HFS for any /// in the name

```
name          *
type          ZFS
filesystem    OMVS.<uc_name>.///
mode          rdwr
duration      1440
delay         360
```

Figure 9-28 Automount policy migration considerations

Automount policy for migration

Automount does have support for recalling migrated data sets and will recall data sets as necessary. The automount processing for HFS file system types is as follows:

- If the file system name does not contain any substitution strings, the data set status is determined to see if the data set is migrated. If migrated, a recall is done.
  - If the data set is an HFS data set, the mount is directed to the HFS PFS.
  - If the data set is not an HFS data set, the mount is directed to the ZFS PFS.
- If the file system name contains substitution strings, the data set status is determined after substitution is performed substituting ZFS for any /// in the name.
  - If the data set exists (migrated or not), the mount is directed to ZFS without automount performing the recall (recall is done by zFS).
- If the file system name contains substitution strings and is for type HFS, the data set status is determined for data set names after substitution is performed substituting HFS for any /// in the name. If the data set is migrated, a recall is done.
  - If the data set is an HFS data set, the mount is directed to HFS.
  - If the data set is not an HFS data set, the mount is directed to ZFS.

9.29 How to mount zFS file systems

- z/OS UNIX mounting for zFS
  - (1). Direct mount
  - (2). Automount facility
    - For all zFS file systems

Figure 9-29 Methods to use when mounting zFS file systems

Mounting file systems

After you have created your zFS file systems, you need to get them mounted at a mount point off the root directory to make them available. The first consideration for mounting zFS file systems is to decide where in the root file system to create the starting mount point.

Compatibility mode aggregates

A zFS file system in a compatibility mode aggregate can be mounted, using the **`mount`** command, at an installation-created mount point. A zFS file system in a compatibility mode aggregate can also be AUTOMOVEed or automounted using the automount facility.

A preferred place to mount all user flie systems is a user directory under the `/u` user directory. Therefore, we describe both methods for the mounting of zFS file systems, since some installations may already be using direct mount while others may be using the automount facility. We suggest that you use one of the following methods:

- Direct mount
- Automount facility using zFS

9.30 Using direct mount commands

- Using the usr/sbin/mount REXX exec from the shell
  - /usr/sbin/mount /u omvs.users.zfs
- Using the mount shell command
- Using the ISHELL File_Systems pull-down
- Adding an entry to the BPXPRMxx member in SYS1.PARMLIB so that it will be mounted when the system re-IPLs
  - Makes the mount permanent at IPL time

Requires an ID that has superuser authority

Figure 9-30 Direct mount commands

Direct mount commands

After a user's file system is allocated, you need to get it mounted at a mount point off the root directory to make it available. Mount points are created by using the **mkdir** command. The preferred place to mount all individual user file systems is a user directory under the /u user directory, such as /u/john, for example. The root file system should be set up so that it does not require frequent changes or updates (outside of SMP/E maintenance).

Mount command example

An example of a direct mount from a shell session for file system OMVS.JOHN follows:

```
# /usr/sbin/mount /u/john omvs.user1
OMVS.JOHN is now mounted at
/u/john
# df -P
Filesystem     512-blocks        Used  Available  Capacity Mounted
OMVS.JOHN           12960          40      12920        1% /u/john
OMVS.ROOT           82800       79608       3192       97% /
# chown user1:grpoe /u/john
# ls -l /u/john
total 16
drwx------   2 JOHN  GRPOE       0 Nov  7 09:09 john
#
```

Note: Since only superusers can mount file systems, in order for JOHN to use this new file system, the superuser must issue the **chown** command to change the ownership and to change the group to the user's default group. Issue this command to set the owner and group fields of this mount point directory for the JOHN user ID. The need to issue the **chown** command is only once because the values will be saved in the new file system and will be reused even when the file system is remounted later.

Permanent mounts

To make a mount permanent you will also need to add the file system name and its mount point to the BPXPRMxx parmlib member, as follows:

```
MOUNT    FILESYSTEM('OMVS.JOHN')
         TYPE(HFS)
         MOUNTPOINT('/u/john')
         MODE(RDWR)
```

9.31 Mounting zFS file systems

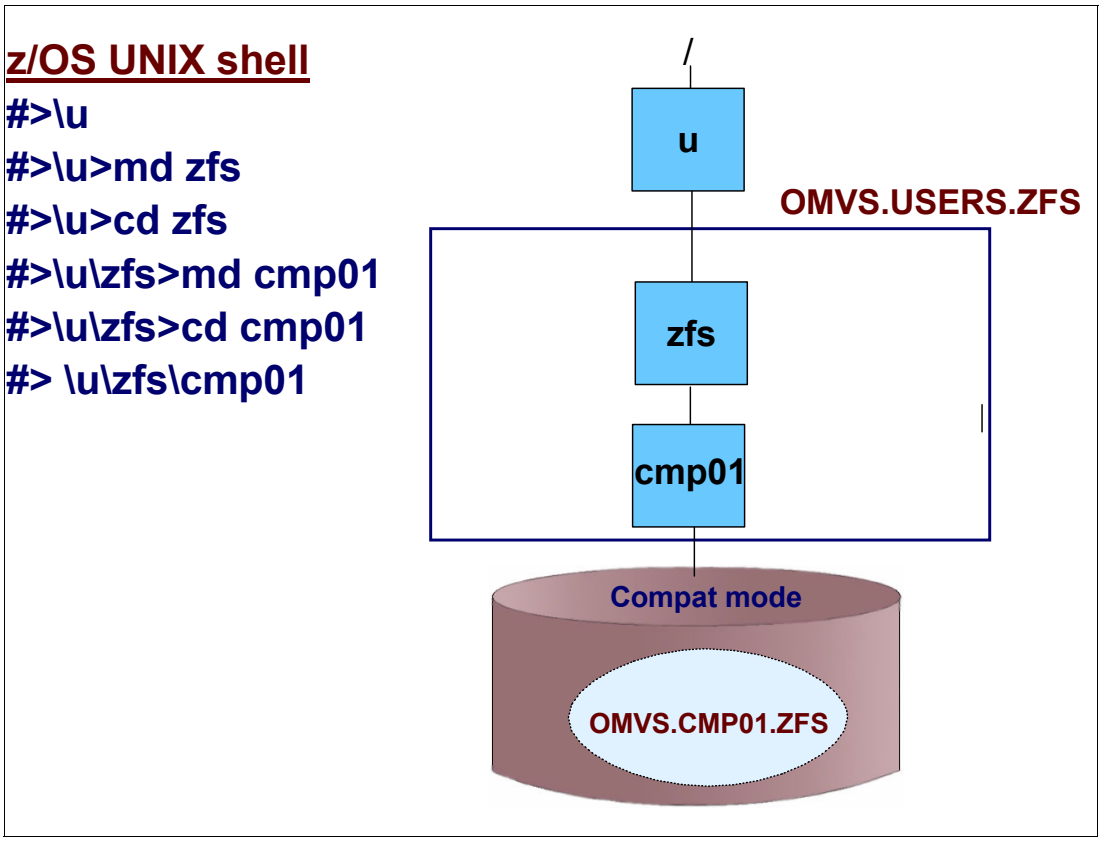


Figure 9-31 Mounting zFS file system with direct mount

Mounting zFS file systems

In Figure 9-31 we show multiple zFS file systems mounted at specified mount points. For the multi-file system aggregates, we are using single-qualifier file system names.

> **Attention:** In this direct mount example, we are using a different naming convention for the aggregate names and file system names. We have also not shown the creation of the aggregates and file systems.

To create the directories and issue the mounts, the example continues as follows:

```
#>\u
#>\u>md zfs
#>\u>cd zfs
#>\u\zfs>md cmp01
#>\u\zfs>cd cmp01
#> \u\zfs\cmp01Mounted file system
```

As shown in Figure 9-31, the compatibility mode aggregate, of course, has only one file system, which is named the same as the aggregate name.

9.32 MOUNT commands

Mount a compatibility mode file system

MOUNT FILESYSTEM('OMVS.CMP01.ZFS') TYPE(ZFS) MODE(RDWR) MOUNTPOINT('/u/zfs/cmp01')

- Mounts could be done via
  - /etc/rc
  - BPXPRMxx parmlib member

```
MOUNT FILESYSTEM(file_system_name) or DDNAME(ddname)
      TYPE(file_system_type)
      MOUNTPOINT(pathname)
      MODE(READ | RDWR)
      PARM(parameter_string)
      SETUID | NOSETUID
```

Figure 9-32 Issuing a mount command

TSO mount commands

Once your zFS file systems have been defined, you can make the following decisions on how to mount them:

- You can issue the **mount** command from TSO/E for the compatibility mode file system as follows:
- MOUNT FILESYSTEM(‘OMVS.CMP01.ZFS’) TYPE(ZFS) MODE(RDWR) MOUNTPOINT('/u/zfs/cmp01')
- IPL the system and use the startup of z/OS UNIX to have them mounted.
- Use the **mount** command from TSO/E or the **mount** command from the OMVS shell.

Note: There are some further possibilities for mounting file systems. You can use option 3 in the File_systems action menu of the ISHELL, for example.

Mounting file systems

Mountable file systems are subsets of the file hierarchy that are added and deleted by mount and unmount. Each has its own root and hierarchical directory structure. One such file system serves as the root of the whole file hierarchy, and mounts are done upon the directories of other mounted file systems.

Mounts in the BPXPRMxx parmlib member

A mount may be issued from the BPXPRMxx parmlib member that is used with the start of z/OS UNIX, by a user through ISHELL, by the TSO/E MOUNT command, by automount, or by a program using the mount() function. The latter function is restricted to users with appropriate privileges.

Here is the syntax of a MOUNT statement, showing the parameters that are important to this discussion:

```
MOUNT FILESYSTEM(file_system_name) or DDNAME(ddname)
      TYPE(file_system_type)
      MOUNTPOINT(pathname)
      MODE(READ | RDWR)
      PARM(parameter_string)
      SETUID | NOSETUID
```

9.33 zFS file system clone

- zFS file system clone is a "copy" of a zFS file system
  - In the same aggregate
  - The clone file system is R/O
  - Only the metadata is copied
  - The cloned file system is called *filename*.bak

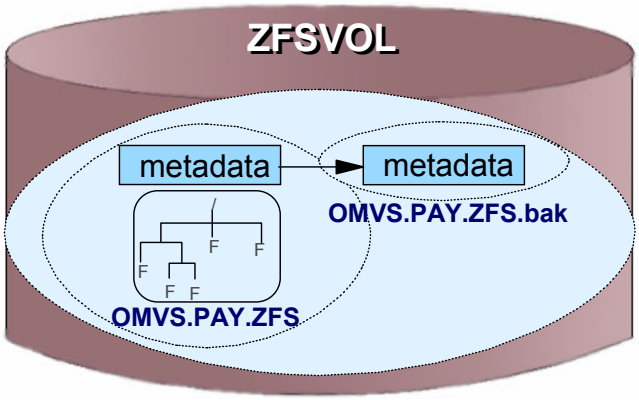


Figure 9-33 zFS file system clone

zFS file system clone

A clone is an optional function. zFS allows the administrator to make a read-only clone of a file system in the same data set. This clone file system can be made available to users to provide a read-only point-in-time copy of a file system. The clone operation happens relatively quickly and does not take up too much additional space because only the metadata is copied.

Cloning a file system

zFS provides the ability to clone a file system with the **`zfsadm clone`** command. When you clone a file system, you create a copy of the file system in the same aggregate. There must be enough physical space available in the aggregate for the clone to be successful. This copy of the file system is read-only and is called a backup file system. The name of the backup file system is the same as the original (read-write) file system with .bak (in lower case) appended to the file system name. This means that you need to limit the length of the file system name to 40 characters if you want to clone it.

The zFS file system, which is the source file system for the clone, is referred to as the read-write file system. The backup file system is a read-only file system and can only be mounted as read-only. You can create a backup file system for every read-write file system that exists.

The aggregate containing the read-write file system to be cloned must be attached. Creating a file system clone is accomplished with the **`zfsadm clone`** command, as follows:

```
zfsadm clone -filesystem OMVS.CMP01.ZFS
```

```
IOEZ00225I File system OMVS.CMP01.ZFS successfully cloned.
```

Clone a file system name

When a file system is cloned, a copy of the file system is created in the same aggregate; space must be available in the aggregate for the clone to be successful. The file system name of the backup file system is the same as the original (read-write) file system with .bak (in lower case) appended to the file system name, as shown in Figure 9-33 on page 480. The clone operation happens relatively quickly and does not take up too much additional space because only the metadata is copied. Metadata consists of things such as owner, permissions, and data block pointers.

The backup file system's data block pointers point to the same data blocks that the read-write file system's data block pointers point to. After a clone operation, if the read-write file system user data is updated, zFS ensures that new physical blocks are allocated to hold the updates, while maintaining the backup file system's data pointers to the original data. The backup file system remains a point-in-time read-only copy in the face of updates to the read-write file system. This backup file system can be mounted (read-only) so that users with the authority to perform mounts of the read-write file system can have an online backup of that file system available. That is, if a user accidently erases a file from the read-write file system, they can simply copy the file from the backup into the read-write file system to restore the file to the time the backup was created.

Clone considerations

The read-write file system can be cloned again (recloned). When a backup file system already exists, it is replaced during the clone operation. One backup file system can exist for each read-write file system. Backup file systems cannot be mounted during the clone operation.

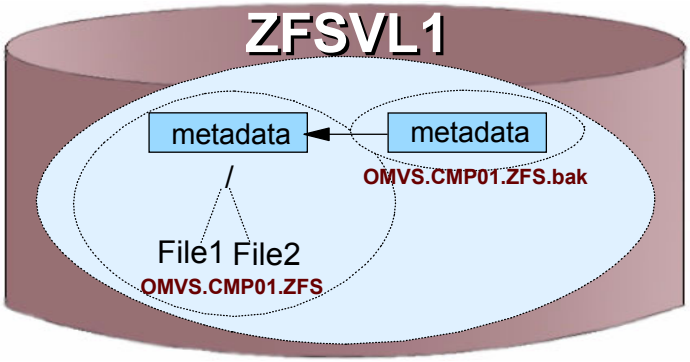
If a failure occurs during a clone of the file system (causing the clone to be only partially created), zFS will delete the partial clone on the next mount. If the file system contains many files, the delete of the partial clone might delay the mount.

You can clone or reclone a set of file systems with the **`zfsadm clonesys`** command. This can be specified in terms of file systems with a file system name prefix or file systems in an aggregate or both.

9.34 Backup file system - zFS clone

zfsadm clone -filesystem OMVS.CMP01.ZFS

IOEZ00225I File system OMVS.CMP01.ZFS successfully cloned.



RDEFINE UNIXPRIV SUPERUSER.FILESYS.PFSCTL UACC(NONE)
PERMIT SUPERUSER.FILESYS.PFSCTL CLASS(UNIXPRIV) ID(ROGERS) ACCESS(READ)

Figure 9-34 zFS clone

zFS clone

This command creates a backup version, or clone, of the indicated read-write zFS file system. It names the new backup version by adding a .bak extension to the name of its read-write source file system. It places the backup version on the same aggregate as the read-write version. The aggregate that the read-write file system is contained in must be attached. The read-write file system may or may not be mounted when the clone operation is issued. The backup file system cannot be mounted when the clone operation is issued.

After the clone operation, the backup file system can be mounted read-only. The **`zfsadm clone`** command cannot clone non-zFS file systems.

Permission to clone

The issuer must have READ authority to the data set that contains the IOEFSPRM file and must be root or must have READ authority to the SUPERUSER.FILESYS.PFSCTL profile in the z/OS UNIXPRIV class. For more information about this authority, see “Authorization for administrators for zFS commands” on page 307.

```
RDEFINE UNIXPRIV SUPERUSER.FILESYS.PFSCTL UACC(NONE)
PERMIT SUPERUSER.FILESYS.PFSCTL CLASS(UNIXPRIV) ID(ROGERS) ACCESS(READ)
```

9.35 zFS clone mounted

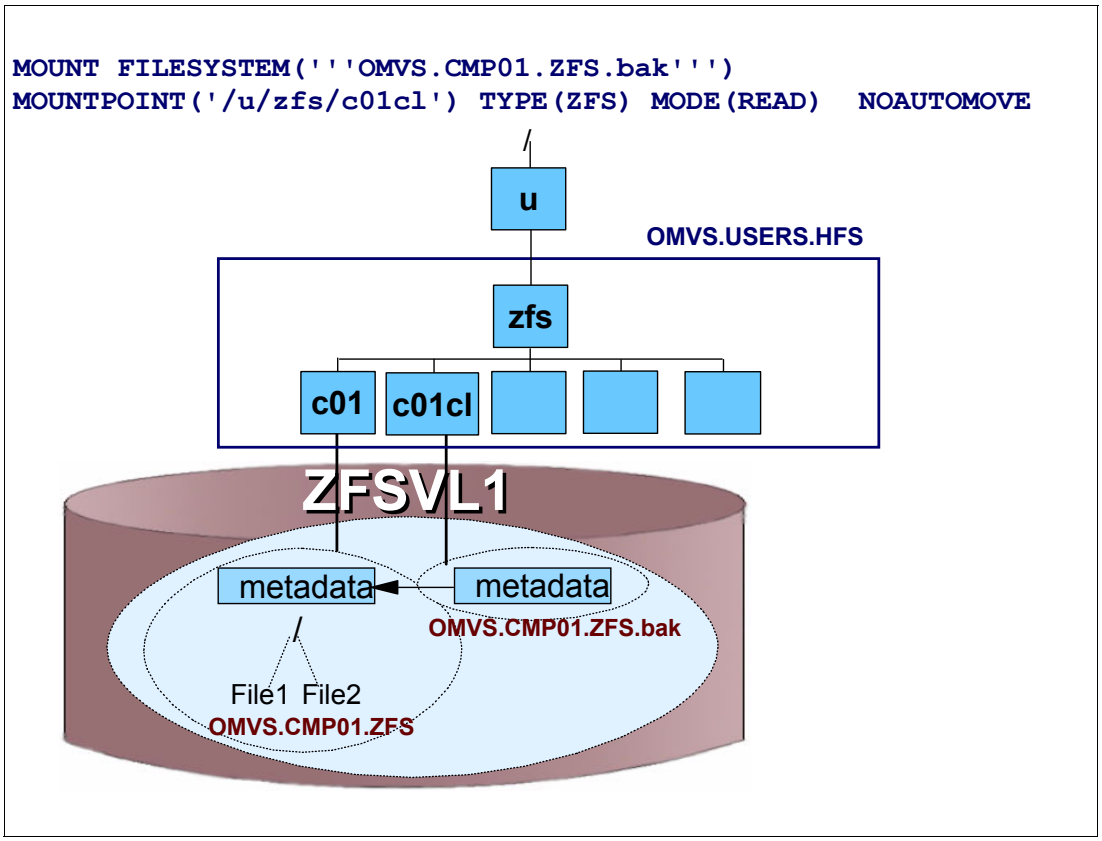


Figure 9-35 A mounted zFS clone

Mounted zFS clones

Once you have created the clone, it must be mounted before anyone can use it. You can use the TSO/E MOUNT command for the backup file system (clone), as follows:

```
MOUNT FILESYSTEM('''OMVS.CMP01.ZFS.bak''') MOUNTPOINT('/u/zfs/c01cl') TYPE(ZFS)
MODE(READ) NOAUTOMOVE
```

The clone must be mounted NOAUTOMOVE.

Note: Using the TSO/E MOUNT command for a clone requires that you use three quotes around the file system name. This is required because the file system name is mixed case with .bak being lower case.

If you prefer to use an OMVS **mount** command for the clone, enter:

```
/usr/sbin/mount -t ZFS -r -a no -f OMVS.CMP01.ZFS.bak /u/zfs/c01cl
```

Note: The backup file system or clone can be mounted (read-only) so that users of the read-write file system can have an online backup of that file system available without administrative intervention.

9.36 Using the clone

- After a clone operation creates the backup file system, if the read-write file system user data is updated, zFS does the following:
  - Makes sure that new physical blocks are allocated to hold the updates
  - Maintains the backup file system's data pointers to the original data
  - Keeps the backup file system as an exact copy at the point-in-time the clone was made when updates to the read-write file system are made

```
zfsadm clonesys -aggregate OMVS.CMP01.ZFS
  ---- Update previous clone ----
```

Figure 9-36 Using the clone

Using a clone

The clone uses a small amount of space since only the metadata is copied, not the user data. The backup file system's data block pointers point to the same data blocks that the read-write file system's data block pointers point to. After a clone operation creates the backup file system, then if the read-write file system user data is updated, zFS does the following:

- Makes sure that new physical blocks are allocated to hold the updates.
- Maintains the backup file system's data pointers to the original data.
- Keeps the backup file system as an exact copy (at the point-in-time the clone was made) when updates to the read-write file system are made.

If a user accidentally erases a file from the read-write file system, they can simply copy the file from the clone into the read-write file system to restore the file to the time the backup was created.

The read-only clone of a file system resides in the same data set as the file system that is cloned. This clone file system can be made available to users to provide a read-only point-in-time copy of a file system.

Updating the clone

The read-write file system can be cloned again (reclone). If the clone file system already exists, it is replaced during this clone operation. Backup file systems cannot be mounted during the clone operation.

9.37 Mounting File Systems - (HFS - zFS)

- Functions added in shared HFS
  - AUTOMOVE - NOAUTOMOVE - SYSNAME
  - UNMOUNT - AUTOMOVE (syslist)

```
MOUNT file system(file_system_name)
                MOUNTPOINT(pathname)
                TYPE(file_system_type)
                MODE(RDWR|READ)
                PARM(parameter_string)
                TAG(NOTEXT|TEXT,ccsid)
                SETUID|NOSETUID
                WAIT|NOWAIT
                SECURITY|NOSECURITY
                SYSNAME (sysname)
AUTOMOVE|AUTOMOVE(indicator,sysname1,sysname2,...,sysnameN
                NOAUTOMOVE|UNMOUNT
```

Figure 9-37 Options when mounting file systems

Changes to the MOUNT command

The parameters that have been added to the MOUNT processing commands apply only in a sysplex where systems are participating in a shared file system environment. They indicate what happens if the system that owns a file system goes down.

AUTOMOVE(indicator,sysname1,sysname2,...,sysnameN)

This new AUTOMOVE feature of z/OS V1R4 allows for a list of systems to be included or excluded (indicated by the indicator field) from being AUTOMOVEd. The `include` indicator and system list provides an ordered list of systems that should be moved to if the file system's owning system should leave the sysplex. The `exclude` indicator and system list provides a list of systems that the file system should not be moved to if the file system's owning system should leave the sysplex.

SYSNAME

For systems participating in a shared file system environment, SYSNAME specifies the particular system on which a mount should be performed. This system will then become the owner of the file system mounted. This system must be IPLed with SYSPLEX(YES). IBM recommends that you specify SYSNAME(&SYSNAME.) or omit the SYSNAME parameter. In this case, the system that processes the mount request mounts the file system and becomes its owner.

9.38 MOUNT command options

- **AUTOMOVE** - Specifies that ownership of the file system is automatically moved to another system. It is the default
- **NOAUTOMOVE** - Specifies that the file system will not be moved if the owning system goes down and the file system is not accessible
- **UNMOUNT** - Specifies that the file system will be unmounted when the system leaves the sysplex
  - Note: This option is not available for automounted file systems

Figure 9-38 MOUNT command options for the shared sysplex environment

AUTOMOVE

When AUTOMOVE is specified for a file system and the file system's owner goes down, AUTOMOVE indicates that ownership of the file system can be automatically moved to another system participating in shared sysplex HFS or zFS. AUTOMOVE is the default.

NOAUTOMOVE

When NOAUTOMOVE is specified for a file system, it indicates that ownership should not be moved to another system participating in a shared file system environment if the file system's owner should crash.

Note: AUTOMOVE is not supported for zFS file systems mounted in a multi-system mode aggregate. They must be mounted NOAUTOMOVE.

UNMOUNT

When UNMOUNT is specified for a file system, this indicates that the file system will not be moved and will be unmounted if the file system's owner should crash. This file system and any file systems mounted within its subtree will be unmounted.

The alternate CDS is not compatible with the primary because it has a smaller value for MOUNTS, so the command SETXCF COUPLE,ACOUPLE failed, because the alternate CDS must be formatted with equal or greater values for MOUNTS and AMTRULES in order for this command to work.

9.39 UNMOUNT option

- **Requirement:** Unmount certain file systems associated with a failed system, rather than recovering and converting to "unowned" status
- **Solution:** Provide UNMOUNT option on MOUNT command and change partition recovery to unmount these file systems
  - Partition recovery will unmount Automount managed file systems if owner fails and no application on active systems is referencing
  - Partition recovery will unmount all file systems associated with a PFS that does not support "move" (e.g. TFS)

Figure 9-39 UNMOUNT option on mount command

Unmount option

When a file system is mounted with the UNMOUNT parameter, attempts will not be made to keep the file system active when the current owner fails. The file system will be unmounted when the owner is no longer active in the participating group, as well as all the file systems mounted within it. It is suggested for use on mounts for system-specific file systems, such as those that would be mounted at /etc, /dev, /tmp, and /var.

Because the file system still exists in the file system hierarchy, the file system mount point is still in use.

An unowned file system is a mounted file system that does not have an owner. The file system still exists in the file system hierarchy. As such, you can recover or unmount an unowned file system.

Note: AUTOMOVE and UNMOUNT are the only options you can use for file systems that are mounted in a mode for which they are capable of being directly mounted to the PFS on all systems (sysplex-aware). If you specify any other option on a MOUNT, it is ignored and you will see message BPXF234I.

9.40 UNMOUNT option support

- BPXPRMxx parmlib MOUNT statement
  - MOUNT FILESYSTEM('OMVS.&SYSNAME..SYSTEM.HFS') TYPE(HFS) MODE(RDWR) **UNMOUNT** MOUNTPOINT('/&SYSNAME.')
- TSO/E MOUNT command
  - MOUNT filesystem(OMVS.VIVAR.HFS) mountpoint('/u/vivar') type(HFS) mode(rdwr) **UNMOUNT**
- mount shell command
  - mount [–t fstype][–rv][–a yes|no|**unmount**][–o fsoptions][–d destsys][–s nosecurity|nosetuid] –f fsname pathname

Figure 9-40 UNMOUNT option support for activation

BPXPRMxx PARMLIB MOUNT statement

The AUTOMOVE | NOAUTOMOVE | UNMOUNT parameters on ROOT and MOUNT statements indicate what happens to the file system if the system that owns that file system goes down. UNMOUNT specifies that the file system will be unmounted when the system leaves the sysplex. This option is not available for automounted file systems. AUTOMOVE is the default.

TSO/E MOUNT command

The options are: AUTOMOVE | NOAUTOMOVE | UNMOUNT. The AUTOMOVE | NOAUTOMOVE parameters apply only in a sysplex where systems are participating in a shared file system environment. They indicate what happens if the system that owns a file system goes down. The default setting is AUTOMOVE.

The `mount` shell command

The **`mount`** shell command, located in /usr/sbin, is used to mount a file system or list all mounts over a file system. A mount user must have UID(0) or at least have READ access to the SUPERUSER.FILESYS.MOUNT resource found in the UNIXPRIV class.

Specify unmount to allow a specified file system (and any file systems mounted on it) to be unmounted when its owning system leaves the sysplex due to a system outage.

The options are: `-a yes|no|unmount`. The default is `yes`.

9.41 UNMOUNT option support

- SETOMVS command
  - SETOMVS FILESYS,FILESYS=filesystem,AUTOMOVE=YES|NO|**UNMOUNT**
- Shell chmount command:
  - chmount [–R [–D |–d destsys][–a yes|no|**unmount**]pathname...
- The ISHELL mount interface in the Mount File System panel is accessed by ISHELL panel -> File System pulldown Menu -> Option 3 - Mount). The new mount option is: Automove unmount file system

Figure 9-41 UNMOUNT options for activation

UNMOUNT options

The UNMOUNT option can be used on the following commands:

- The SETOMVS command used with the FILESYS, FILESYSTEM, mount point and SYSNAME parameters can be used to move a file system in a sysplex.
- You can use the **`chmount`** command from the shell.

  In a shared file system environment, **`-a yes`** allows the system to automatically move logical ownership for a specified file system due to a system outage. **`-a unmount`** specifies that this file system is to be unmounted when the file system's owner leaves the sysplex.

  > **Note:** A **`chmount`** user must have UID(0) or at least have read access to the SUPERUSER.FILESYS.MOUNT resource found in the UNIXPRIV class.

- The ISHELL can be used to set the UNMOUNT option, as shown in Figure 9-42 on page 490.

9.42 Mount file system panel

```
File  Directory  Special_file  Tools  File_systems  Options  Setup  Help
                          Mount a File System

Mount point:
                                                          More:     +
   /u/rogers/largezfs
   ____________________________________________________________________

File system name   omvs.zfs.test
File system type   zfs      New owner  . . . . . ________
Owning system  . . sc64    Character Set ID . . _____

Select additional mount options:
_  Read-only file system         /  Set automove attribute...
_  Ignore SETUID and SETGID      _  Text conversion enabled
_  Bypass security

Mount parameter:
aggrgrow

 F1=Help        F3=Exit        F4=Name        F6=Keyshelp   F12=Cancel
```

Figure 9-42 Mount file system panel in the ISHELL

ISHELL mount file system panel

Figure 9-42 shows the panel used to mount a file system. Beginning with z/OS V1R6, this panel changed with the removal of the option that appeared above the Set automove attribute, as follows:

```
Do not automove file system
```

Set automove attribute

The field Automove indicates whether the system can automatically move the file system to another system owner, remain local and unowned, or be unmounted. Other indications are an include list and exclude list for system selection for automoved file systems.

Automatic move processing is normally done when a system fails and it is the owner of some file systems. The file systems that contain /dev should never be moved. This should always be a local file system. Automove should be unmount for this file system. There are some other file systems where this is also the case.

9.43 Set AUTOMOVE options

```
File  Directory  Special_file  Tools  File_systems  Options  Setup  Help
                            Mount a File System
M                           Set automove attribute

    Select the automove attribute:
    __  1.  Yes
        2.  No
F       3.  Unmount
F       4.  Include systems
O       5.  Exclude systems

S   System names for Include or Exclude
_   ________  ________  ________  ________  ________
_   ________  ________  ________  ________  ________
_   ________  ________  ________  ________  ________
    ________  ________  ________  ________  ________
M   ________  ________  ________  ________  ________
a   ________  ________  ________  ________  ________
    ________  ________

     F1=Help       F3=Exit       F6=Keyshelp  F12=Cancel
```

Figure 9-43 Setting the AUTOMOVE options

Setting the AUTOMOVE options

The field Automove indicates whether the system can automatically move the file system to another system owner, remain local and unowned, or be unmounted. Other indications are an include list and an exclude list for system selection for automoved file systems.

Automatic move processing is normally done when a system fails and it is the owner of some file systems. The file systems that contain /dev should never be moved. This should always be a local file system. Automove should be Unmount for this file system. There are some other file systems where this is also the case.

Automove Unmount means that when a system fails and if this file system is not in use, it will be unmounted. If the file system contains mount points, this file system and all file systems under it will be unmounted.

Automove can also specify a list of systems that may be included when selecting a system for the target of the move. It can also specify a list of systems that are to be excluded as move targets. If the list is long, the display may be truncated. All system names can be viewed and modified by using the Modify file system action code and selecting Automove for change. If the last system name is specified as * for an include list, the system will treat this as a list of all remaining eligible systems.

The field Owning system is the name of the system that owns this file system.

9.44 AUTOMOVE system list (syslist)

- In a shared sysplex environment, AUTOMOVE(YES) on MOUNT moves the ownership of the filesystem to some other system in the sysplex if the current server system for that filesystem is brought down
- syslist provides the capability to specify which system or systems in a sysplex will takeover as server for a filesystem
  - Specify a system list on the AUTOMOVE parameter for MOUNT
  - The list begins with either include or exclude, (abbreviated i or e) followed by a list of system names

Figure 9-44 AUTOMOVE system list specifications

AUTOMOVE system list

When mounting file systems, you can specify an automove system list to indicate where the file system should or should not be moved when a system leaves the sysplex. Previously, on a system failure, with more than one system capable of taking over ownership, it was random which system was the new owner.

The automove system list can be specified in many different ways to automove a file system. The list begins with an indicator to either *include* or *exclude* (abbreviated as **i** or **e**) followed by a list of system names. Specify the indicator as follows:

i Use this indicator with a prioritized list of systems to which the file system may be moved if the owning system goes down. If none of the systems specified in the list can take over as the new owner, the file system will be unmounted.

e Use this indicator with a list of systems to which the file system may *not* be moved.

9.45 AUTOMOVE parameters for mounts

- BPXPRMXX PARMLIB member MOUNT statement or TSO/E MOUNT command
- Shell mount command
- ISHELL panels for mounts
- C program, assembler program, or REXX program
  - AUTOMOVE(indicator, name1, name2,.....)
    - **i** - Provides a prioritized list of systems where the file system may be moved - If no system can takeover as the new owner, the file system is unmounted
    - **e** - This system list provides a list of systems to where the file system may not be moved

Figure 9-45 AUTOMOVE specifications for system lists

Specifying AUTOMOVE on MOUNT commands

When mounting file systems in the sysplex, you can specify a prioritized automove system list to indicate where the file system should or should not to moved to when the owning system leaves the sysplex. There are different ways to specify the automove system list, as follows:

- On the MOUNT statement in BPXPRMxx, specify the AUTOMOVE keyword, including the indicator and system list.
- For the TSO MOUNT command, specify the AUTOMOVE keyword, including the indicator and system list.
- Use the **`mount`** shell command.
- Use the ISHELL MOUNT interface.
- Specify the MNTE_SYSLIST variable for REXX.
- Specify the indicator and system list for the automove option in the **`chmount`** shell command.
- Specify the indicator and system list for the automove option in the SETOMVS operator command.

9.46 AUTOMOVE wildcard support

Wildcard support for system lists:

- Explicitly list the systems and include the remaining systems using wildcard character in the include system list:
  - Parmlib member
  - TSO MOUNT command
  - Shell - chmount
  - ISHELL
  - C program
  - REXX
  - Assembler program
  - SETOMVS FILESYS,FILESYSTEM='X.Y.Z',AUTOMOVE=(I,*)
- Automove syslist wildcard support is invoked by:
  - SETOMVS FILESYS,FILESYSTEM='X.Y.Z',SYSNAME=*
  - F BPXOINIT, SHUTDOWN=FILESYS
  - F OMVS,SHUTDOWN

Figure 9-46 AUTOMOVE wildcard support

AUTOMOVE wildcard support

You can use the wildcard support on all methods of mounts, including the MOUNT statement in BPXPRMxx, the TSO MOUNT command, the **mount** shell command, the ISHELL MOUNT interface, the MNTE_SYSLIST variable for REXX, C program, and assembler program.

Using AUTOMOVE wildcard support, the customer can specify only those systems in the INCLUDE list that should have priority in file system takeover, and use the wildcard '*' to specify all remaining systems:

AUTOMOVE(INCLUDE,SY1,SY3,*)

A wildcard character is permitted as an only item or as a last item of the AUTOMOVE Include Syslist in place of a system name.

Note: The wildcard is only allowed with an Include list, not with an Exclude list.

Advantages of using AUTOMOVE wildcard support

With wildcard support, the remaining systems do not have to be explicitly listed in the Include system list, which can reduce typing errors.

9.47 AUTOMOVE wildcard examples

- Wildcard character permitted as an only item or as a last item of the Automove **Include** Syslist in place of a system name
- The wildcard is only allowed with an Include list, not with an **Exclude** list:
- Examples
  - AUTOMOVE(include,sy2,*)
  - AUTOMOVE(include,*)
  - AUTOMOVE(i,sy1,sy2,sy3,*)
  - AUTOMOVE(i,sy*)

Figure 9-47 AUTOMOVE wildcard examples

Examples of wildcard usage

If you have a large number of systems in your sysplex, you can specify only the systems that should have priority and use a wildcard to indicate the rest of the systems.

In the examples, at first glance, AUTOMOVE INCLUDE (*) appears to work the same way as AUTOMOVE(YES) because all of the systems will try to take over the file system. However, with AUTOM0VE INCLUDE (*), if none of the systems can take over the file system, it will be unmounted. If AUTOMOVE(YES) is used, the file system will become unowned.

Note: All systems must be at the V1R6 level or later. Otherwise, results will be unpredictable.

9.48 Stopping zFS

- Use the F OMVS,STOPPFS=ZFS operator command
- ZFS address space can have an internal failure
- File system status if ZFS address space goes away
  - All zFS file systems on that system will be unmounted
  - Or moved if AUTOMOVE in a sysplex is specified
- Issuing P ZFS

IOEZ00523I zFS no longer supports the P ZFS stop command. Please issue f omvs,stoppfs=zfs

Figure 9-48 Ways zFS address space stops running

Stopping zFS

zFS can be stopped using the F OMVS,STOPPFS=ZFS operator command. zFS file systems should be unmounted or moved to another sysplex member before zFS is actually stopped by z/OS UNIX. When zFS is stopped, you receive the following message:

```
nn BPXF032D FILESYSTYPE ZFS TERMINATED. REPLY 'R' WHEN READY TO RESTART. REPLY
'I' TO IGNORE.
```

In general, you should not stop zFS. Stopping zFS is disruptive to applications that are using zFS file systems. zFS stops automatically when you shut down z/OS UNIX. If you want to shut down an LPAR or re-IPL an LPAR, use the **F OMVS,SHUTDOWN** operator command to shut down z/OS UNIX. This synchronizes data to the file systems and unmounts or moves ownership in a shared file system environment. A planned system shutdown must include the unmount or move of all owned file systems and the shutdown of zFS. The **F OMVS,SHUTDOWN** command unmounts and moves the owned file systems and shuts down zFS. For shutdown procedures using **F OMVS,SHUTDOWN**, see the topic on Planned shutdowns using **F OMVS,SHUTDOWN** in *z/OS UNIX System Services Planning*, GA22-7800.

File system ownership

zFS can be stopped using the F OMVS,STOPPFS=ZFS operator command. Automatic ownership movement can occur for both the z/OS UNIX owner and the zFS owner. See *z/OS UNIX System Services Planning*, GA22-7800 for information about the various AUTOMOVE settings for z/OS UNIX file system ownership. zFS aggregate ownership will move unless the

file system is unmounted by z/OS UNIX. zFS file systems that become unmounted will need to be mounted again after zFS is restarted.

ZFS address space fails

If the ZFS colony address space has an internal failure, it may abnormally terminate and stop. Normally, the ZFS colony address space restarts automatically. Otherwise, message BPXF032D (the same message you receive when the F OMVS,STOPPFS=ZFS operator command is used) is issued and a reply is requested.

File system status

If the F OMVS,STOPPFS=ZFS operator command is issued, all ZFS file systems on that system will be unmounted (or moved if AUTOMOVE in a sysplex is specified). Applications with an open file on these file systems will receive I/O errors until the file is closed. Once zFS is restarted, the operator must remount any file systems that were locally mounted (that is, file systems that were owned by that system and were not moved). This can be done with the F BPXOINIT,FILESYS=REINIT operator command. This causes a remount for each file system that was mounted through a BPXPRMxx parmlib statement.

New stop command

This command is introduced to stop the ZFS address space, as follows:

```
F OMVS,STOPPFS=ZFS
```

This new operator command allows z/OS UNIX to move file system ownership (non-disruptively) to another system before the PFS is terminated. This allows applications that are accessing file systems owned on the system where the zFS PFS is being terminated to continue to proceed without I/O errors in a shared file system environment.

After receiving the **stop** command, the system invokes a sync to all local mounted zFS file systems. Afterward, all zFS file systems are unmounted in a single system. When you are in a shared file system environment, all AUTOMOVE-defined file systems will be moved to another system. Before the file system move starts, a new message, BPXI068D, appears to confirm the file system shutdown.

```
 F OMVS,STOPPFS=ZFS
*010 BPXI078D STOP OF ZFS REQUESTED. REPLY 'Y' TO PROCEED. ANY OTHER REPLY WILL
 CANCEL THIS STOP
 R 10,Y
 IEE600I REPLY TO 010 IS;Y
 IOEZ00048I Detaching aggregate LUTZ.ZFS
 IOEZ00048I Detaching aggregate PELEG.ZFS
 IOEZ00048I Detaching aggregate DAURCES.ZFS
 IOEZ00048I Detaching aggregate TEMEL.ZFS
 IOEZ00048I Detaching aggregate VAINI.ZFS
 IOEZ00050I zFS kernel: Stop command received.
 IOEZ00057I zFS kernel program IOEFSCM is ending
 IEF352I ADDRESS SPACE UNAVAILABLE
 IEA989I SLIP TRAP ID=X33E MATCHED.  JOBNAME=*UNAVAIL, ASID=0057.
*011 BPXF032D FILESYSTYPE ZFS TERMINATED.  REPLY 'R' WHEN READY TO RESTART.
```

9.49 Restarting the PFS

- If the reply to restart the PFS is **"I"**
  - (Do not restart the PFS), then the file systems are locally unmounted as before
- If the reply to restart the PFS is **"R"**
  - Then any sysplex-aware file systems converts back from function-shipping to local mount
  - Sysplex-unaware file systems remain function-shipping to the current owner
- Restart the zFS PFS with if the "I" option terminates it
  - Issue a SET OMVS=SS command
    - BPXPRMSS contains the FILESYSTYPE statement

Figure 9-49 zFS PFS termination message and operator replies

zFS PFS termination message

When the PFS terminates, there is a system prompt message waiting for a reply to be answered in system SC70, for example:

```
*015 BPXF032D FILESYSTYPE ZFS TERMINATED.  REPLY 'R' WHEN READY TO RESTART.
REPLY 'I' TO IGNORE.
```

- If the reply to re-start the PFS is `I` (do not restart the PFS), then the file systems will be locally unmounted as before.
- If the reply to re-start the PFS is `R`, then any sysplex-aware file systems will convert back from function-shipping to local mount. Sysplex-unaware file systems continue function-shipping to the current owner.

The SET OMVS= command

This command, available with z/OS V1R7, can be used to start the ZFS PFS if it has not been started at IPL. It can also be used to redefine it if it has been terminated by replying `i` to the BPXF032D operator message (after stopping the ZFS PFS). The suffix SS is a BPXPRMxx member of PARMLIB that contains the FILESYSTYPE statement for the ZFS PFS.

Note: Beginning with z/OS V1R8, the **`p zfs`** command is only supported in a non-sysplex sharing mode.

When the zFS address space stops either by the operator or abnormally and you are not in sysplex sharing mode, all zFS file systems are unmounted as the aggregates are detached.

9.50 Mounting file systems with SET OMVS

- The SET OMVS command enables use of a BPXPRMxx parmlib member without re-IPLing
  - Allows mounts from a BPXPRMxx member from an operator console

```
FILESYSTYPE TYPE(ZFS)             /* Type of file system to start */
            ENTRYPOINT(IOEFSCM)   /* Entry Point of load module   */
            ASNAME(ZFS)           /* Procedure name */
ROOT     FILESYSTEM('OMVS.&SYSNAME..&SYSR1..ROOT.ZFS')
         TYPE(ZFS)                /* Filesystem type ZFS            */
         MODE(RDWR)               /* Mounted for read/write         */
MOUNT FILESYSTEM('OMVS.&SYSNAME..ETC.ZFS')
      MOUNTPOINT('/etc')
         TYPE(ZFS)                /* Filesystem type ZFS            */
         MODE(RDWR)               /* Mounted for read/write         */
MOUNT FILESYSTEM('OMVS.&SYSNAME..VAR.ZFS')
       MOUNTPOINT('/var')
         TYPE(ZFS)                /* Filesystem type ZFS            */
         MODE(RDWR)               /* Mounted for read/write         */
```

Figure 9-50 Mounting file systems using the SET OMVS command

BPXPRMSS PARMLIB member

The following BPXPRMSS PARMLIB member is used to restart zFS and mount critical file systems. The SET OMVS=SS operator command is used to read the member, execute the mount statements, and start the ZFS PFS following a **p zfs** command.

```
FILESYSTYPE TYPE(ZFS)            /* Type of file system to start */
            ENTRYPOINT(IOEFSCM)  /* Entry Point of load module   */
            ASNAME(ZFS)          /* Procedure name */
ROOT     FILESYSTEM('OMVS.&SYSNAME..&SYSR1..ROOT.ZFS')
                                 /* z/OS root filesystem          */
         TYPE(ZFS)               /* Filesystem type ZFS           */
         MODE(RDWR)              /* Mounted for read/write        */
MOUNT FILESYSTEM('OMVS.&SYSNAME..ETC.ZFS')
                                 /* ZFS for /etc directory        */
      MOUNTPOINT('/etc')
         TYPE(ZFS)               /* Filesystem type ZFS           */
         MODE(RDWR)              /* Mounted for read/write        */
MOUNT FILESYSTEM('OMVS.&SYSNAME..VAR.ZFS')
                                 /* ZFS for /var directory        */
      MOUNTPOINT('/var')
         TYPE(ZFS)               /* Filesystem type ZFS           */
         MODE(RDWR)              /* Mounted for read/write
```

9.51 Messages from shutdown of a ZFS single system

```
 P ZFS
 IOEZ00541I 729
 zFS filesystems owned on this system should be unmounted or moved
 before stopping zFS. If you do not applications may fail.
*005 IOEZ00542D Are you sure you want to stop zFS? Reply Y or N
 IEE600I REPLY TO 005 IS;Y
 IOEZ00050I zFS kernel: Stop command received.
 IOEZ00048I Detaching aggregate OMVS.TC7.VAR.ZFS
 IOEZ00048I Detaching aggregate OMVS.TC7.ETC.ZFS
 IOEZ00048I Detaching aggregate OMVS.TC7.TRNRS1.ROOT.ZFS
 IOEZ00057I zFS kernel program IOEFSCM is ending
*006 BPXF032D FILESYSTYPE ZFS TERMINATED.  REPLY 'R' WHEN READY TO
 RESTART. REPLY 'I' TO IGNORE.
 IEF352I ADDRESS SPACE UNAVAILABLE
 $HASP395 ZFS      ENDED
 $HASP250 ZFS PURGED -- (JOB KEY WAS BE68CA18)
 R 6,I
 IEE600I REPLY TO 006 IS;I
 SET OMVS=SS
 IEE252I MEMBER BPXPRMSS FOUND IN SYS1.PARMLIB
 BPXO032I THE SET OMVS COMMAND WAS SUCCESSFUL.
```

Figure 9-51 Shutdown of the ZFS address space

Messages from the restart of zFS

After stopping zFS and issuing the SET OMVS command the following messages are issued:

```
 P ZFS
 IOEZ00541I 729
 zFS filesystems owned on this system should be unmounted or moved
 before stopping zFS. If you do not applications may fail.
*005 IOEZ00542D Are you sure you want to stop zFS? Reply Y or N
 IEE600I REPLY TO 005 IS;Y
 IOEZ00050I zFS kernel: Stop command received.
 IOEZ00048I Detaching aggregate OMVS.TC7.VAR.ZFS
 IOEZ00048I Detaching aggregate OMVS.TC7.ETC.ZFS
 IOEZ00048I Detaching aggregate OMVS.TC7.TRNRS1.ROOT.ZFS
 IOEZ00057I zFS kernel program IOEFSCM is ending
*006 BPXF032D FILESYSTYPE ZFS TERMINATED.  REPLY 'R' WHEN READY TO
 RESTART. REPLY 'I' TO IGNORE.
 IEF352I ADDRESS SPACE UNAVAILABLE
 $HASP395 ZFS      ENDED
 $HASP250 ZFS PURGED -- (JOB KEY WAS BE68CA18)
 R 6,I
 IEE600I REPLY TO 006 IS;I
 SET OMVS=SS
 IEE252I MEMBER BPXPRMSS FOUND IN SYS1.PARMLIB
 BPXO032I THE SET OMVS COMMAND WAS SUCCESSFUL.
```

9.52 Messages for the restart of ZFS

```
$HASP100 ZFS      ON STCINRDR
$HASP373 ZFS      STARTED
IOEZ00052I zFS kernel: Initializing z/OS    zSeries File System
Version 01.07.00 Service Level OA12872 - HZFS370.
Created on Mon Sep 26 16:19:25 EDT 2005.
IOEZ00178I SYS1.TC7.IOEFSZFS(IOEFSPRM) is the  configuration dataset
currently in use.
IOEZ00055I zFS kernel: initialization complete.
BPXF013I FILE SYSTEM OMVS.TC7.TRNRS1.ROOT.ZFS 768
WAS SUCCESSFULLY MOUNTED.
BPXF013I FILE SYSTEM OMVS.TC7.ETC.ZFS 769
WAS SUCCESSFULLY MOUNTED.
BPXF013I FILE SYSTEM OMVS.TC7.VAR.ZFS 770
WAS SUCCESSFULLY MOUNTED.
```

Figure 9-52 Messages during restart of the ZFS address space

ZFS address space restart messages

Reading the BPXPRMSS PARMLIB member uses the FILESYSTYPE statement to start the ZFS address space. The mount statements in the member are used to start the other file systems that were stopped when the **P ZFS** command was issued.

```
$HASP100 ZFS      ON STCINRDR
$HASP373 ZFS      STARTED
IOEZ00052I zFS kernel: Initializing z/OS    zSeries File System
Version 01.07.00 Service Level OA12872 - HZFS370.
Created on Mon Sep 26 16:19:25 EDT 2005.
IOEZ00178I SYS1.TC7.IOEFSZFS(IOEFSPRM) is the  configuration dataset
currently in use.
IOEZ00055I zFS kernel: initialization complete.
BPXF013I FILE SYSTEM OMVS.TC7.TRNRS1.ROOT.ZFS 768
WAS SUCCESSFULLY MOUNTED.
BPXF013I FILE SYSTEM OMVS.TC7.ETC.ZFS 769
WAS SUCCESSFULLY MOUNTED.
BPXF013I FILE SYSTEM OMVS.TC7.VAR.ZFS 770
WAS SUCCESSFULLY MOUNTED.
```

Figure 9-53 Example of stopping a zFS file system

9.53 Shutdown and recovery scenarios

- "Soft shutdown processing"
  - F BPXOINIT,SHUTDOWN=FILESYS
  - F BPXOINIT,SHUTDOWN=FILEOWNER
  - F OMVS,SHUTDOWN
- Member gone recovery on hard failures
- PFS termination
- Multiple file system moves using:

  SETOMVS FILESYS,FROMSYS=SY1,SYSNAME=SY2

Figure 9-54 Recovery scenarios

File system recovery scenarios

File system recovery in a shared file system environment takes into consideration file system characteristics such as the PFS capabilities and whether the file system was mounted as read-write or read-only.

Recovering file systems in a shared file system takes into consideration file system specifications such as the sysplex awareness capability, the AUTOMOVE value, and whether or not the file system is mounted read-only or read/write.

Generally, when an owning system fails, ownership of a file system that is mounted as AUTOMOVE is moved to another system and the file system remains usable. However, if a file system is mounted as read-write and the owning system fails, then all file system operations for files in that file system will fail. This happens because data integrity is lost when the file system owner fails.

When mounting file systems in the sysplex, you can specify a prioritized system list to indicate where the file system should or should not be moved to when the owning system leaves the sysplex changes due to any of the following:

- A soft shutdown request has been issued.
- Dead system takeover occurs (when a system leaves the sysplex without a prior soft shutdown).
- A PFS terminates on the owning system.

- A request to move ownership of the file system is issued. The following command can be used:

  ```
  SETOMVS FILESYS,FROMSYS=sysname,SYSNAME=sysname|*
  ```

  `FROMSYS=sysname` in a sysplex environment is a parameter that indicates the system where all the file systems will be moved from. The file systems will be moved to the system identified by the sysname keyword.

Soft shutdown

Soft shutdown is done by issuing one of the following MODIFY (F) commands:

```
F BPXOINIT,SHUTDOWN=FILESYS
F BPXOINIT,SHTUDOWN=FILEOWNER
F OMVS,SHUTDOWN
```

Automount-managed file systems are unmounted by a soft shutdown operation if the file system is not referenced by any other system in the sysplex. If it is referenced by another system or systems, ownership of the file system is moved. If the move fails, an unmount is not attempted and ownership does not change.

PFS termination

If a PFS terminates on one system, it can affect file system availability on other systems. If a PFS terminates on one system, any file systems of that type that are owned by other systems are not affected. File systems of that type are moved to new owners whenever possible if they are owned by the system where the PFS is terminating and if they can be automoved. These file systems remain accessible to other systems. If they cannot be moved to new owners, they are unmounted across the sysplex. It may not be possible to move a file system due to a lack of connectivity from other systems, or if the file system containing the mount point for the file system needed to be moved but could not be.

9.54 zFS commands in a sysplex

- Previously, zfsadm commands (and the pfsctl APIs) could only display or change information that is located or owned on the current member of the sysplex
  - Only issued from the owning system
- When DFSMS is used to back up a zFS aggregate, the aggregate is quiesced by DFSMS before the backup
  - In order for this quiesce to be successful, it must be issued on the system that owns the aggregate

Figure 9-55 Command changes in a sysplex

zfsadm commands in a sysplex

Previously, **zfsadm** commands (and the pfsctl APIs) could only display or change information that is located or owned on the current member of the sysplex. The pfsctl (BPX1PCT) application programming interface is used to send physical file system specific requests to a physical file system. zFS pfsctl APIs do not work across sysplex members. zFS pfsctl APIs can query and set information on the current system only. However, if all systems are running z/OS V1R7, zFS pfsctl APIs will work across sysplex members.

DFSMS backups

A particular problem is quiescing of a zFS aggregate. When DFSMS is used to back up a zFS aggregate, the aggregate is quiesced by DFSMS before the backup. Currently, in order for this quiesce to be successful, it must be issued on the system that owns the aggregate. If it is issued on any other system in the sysplex, the quiesce fails (and the backup fails). It is a problem to issue the backup on the owning system because ownership can change at anytime as a result of operator command or system failure. **zfsadm** command forwarding allows the quiesce (or any other zfsadm command) to be issued from any member of the sysplex. This function is used by:

- DFSMS backup (ADRDSSU)
- ISHELL
- zFS Administrators

9.55 zfsadm command changes for sysplex

- If you are in a shared file system environment, zfsadm commands and zFS pfsctl APIs generally work across the sysplex
- There are two main changes:
  - The zfsadm commands act globally and report or modify all zFS objects across the sysplex
  - They support a new –system option that limits output to zFS objects on a single system or sends a request to a single system
    - -system system name

Figure 9-56 Sysplex support for zfsadm commands

zfsadm commands with sysplex sharing

All **zfsadm** commands work across sysplex members. The **zfsadm** commands act globally and report or modify all zFS objects on any system in the sysplex.

However, there are configuration dependencies. Whether **zfsadm** commands act globally across the sysplex depends on the BPXPRMxx SYSPLEX option. If SYSPLEX(YES) is specified, then you are in a shared file system environment and **zfsadm** commands will act globally. If the IOEFSPRM sysplex=off is specified, or if the BPXPRMxx SYSPLEX option specifies SYSPLEX(NO), then **zfsadm** commands will not act globally.

All **zfsadm** commands that apply to zFS aggregates or file systems work against all aggregates and file systems across the sysplex. The following **zfsadm** commands can optionally direct their operation to a particular member of the sysplex: aggrinfo, attach, clonesys, config, configquery, define, detach, format, lsaggr, lsfs, and query.

New -system option

The -system system name specifies the name of the system the report request will be sent to, to retrieve the data requested.

9.56 zfsadm command changes

- zfsadm configquery command options
  - -group
  - -system system name
  - -sysplex_state
- -sysplex_state
  - Displays the sysplex state of ZFS. Zero (0) indicates that ZFS is not in a shared file system environment. One (1) indicates that ZFS is in a shared file system environment

```
$> zfsadm configquery -sysplex_state
IOEZ00317I The value for configuration option -sysplex_state is 1.
```

Figure 9-57 New options with the zfsadm configquery command

zfsadm command options

The **`zfsadm configquery`** command displays the current value of zFS configuration options. The value is retrieved from ZFS address space memory rather than from the IOEFSPRM file or BPXPRMxx PARMLIB member. You can specify that the configuration option query request should be sent to another system by using the -system option.

The **`zfsadm configquery`** command has the following new options:

-system system name Specifies the name of the system the report request will be sent to, to retrieve the data requested.

-group Displays the XCF group used by ZFS for communication between sysplex members.

-sysplex_state Displays the sysplex state of ZFS. Zero (0) indicates that ZFS is not in a shared file system environment. One (1) indicates that ZFS is in a shared file system environment.

9.57 Configuration options

- -system system name
  - Specifies the name of the system the report request will be sent to, to retrieve the data requested
- -group
  - Display XCF group for communication between members
    - Default Value: IOEZFS
    - Expected Value: 1 to 8 characters

```
PAUL @ SC65:/>zfsadm aggrinfo -system sc70
IOEZ00368I A total of 1 aggregates are attached to system SC70.
ROGERS.HARRY.ZFS (R/W COMP): 3438 K free out of total 3600

PAUL @ SC65:/>zfsadm configquery -group
IOEZ00317I The value for configuration option -group is IOEZFS.
```

Figure 9-58 Examples of new command configuration options for sysplex

Command forwarding with z/OS v1R7

Using the -system system name as a parameter with the **zfsadm** command, it specifies which system to route the command to. The following command displays the attached aggregates on system SC70. The command was issued from system SC65.

```
PAUL @ SC65:/>zfsadm aggrinfo -system sc70
IOEZ00368I A total of 1 aggregates are attached to system SC70.
ROGERS.HARRY.ZFS (R/W COMP): 3438 K free out of total 3600
```

Sysplex group

Beginning with z/OS V1 R7, zFS administration commands use XCF communications to exchange zFS aggregate and file system information between members of the sysplex. During zFS initialization, zFS must contact each other zFS system that is active in the sysplex group to announce itself to the other members of the group and to receive information about attached aggregates from the other members of the group. The zFS group name can be defined in the IOEFSPRM file or the IOEPRMxx PARMLIB member. The following command displays the group and IOEZFS is the default if not specified.

```
PAUL @ SC65:/>zfsadm configquery -group
IOEZ00317I The value for configuration option -group is IOEZFS.
```

9.58 Command forwarding support

- zfsadm commands will now display and work with zFS aggregates and file systems across the sysplex
- This enhancement is exploited by:
  - DFSMS backup (ADRDSSU)
    - Backups can now be issued from any member of the sysplex when all members are at z/OS V1R7 regardless of which member owns the file system
  - ISHELL
  - zFS administrators

Figure 9-59 Other functions enhanced by the command forwarding support

Command forwarding

Command forwarding support is an enhancement in z/OS V1R7. This allows zFS commands to be issued from any system in a sysplex without regard to which system owns the file system. DFSMS backup (ADRDSSU) now automatically exploits this function. Backups can now be issued from any member of the sysplex

DFSMS backup with ARDRSSU

If any systems in the sysplex are running a release of z/OS lower than Version 1 Release 7, the job must be run on the sysplex member where the aggregate(s) is attached. If the job is not run on the same member of the sysplex, the quiesce will fail and the job will terminate. However, if all systems in the sysplex are running z/OS Version 1 Release 7, you can run the backup job on any member of the sysplex.

9.59 Indirect volsers with zFS data sets

- Cloning of zFS data sets to different systems
  - Decide which volumes to copy zFS data sets to
  - Ensure you have a system symbol defined for the target volume
    - It is best to use the ADRDSSU COPY utility with the PHYSINDYNAM option to copy your zFS data sets
  - A single catalog entry (for zFS VSAM LDS data sets) can represent different volumes on various systems
  - This support is limited to single volume zFS data sets

Figure 9-60 Indirect volsers with zFS data sets

Indirect volsers with zFS data sets

Starting with z/OS V1R12, you can make a clone, or copy, of a zFS data set and use indirect volume serials to access it. This lets you define a catalog entry for a zFS to specify different volume serials for different systems, which might allow the use of existing processes for cloning systems when zFS is used for the version root file system.

To make cloning of zFS data sets to different systems easier, decide which volumes to copy zFS data sets to and review your copy jobs.

Use system symbols

Ensure you have a system symbol defined for the target volume. It is best to use the ADRDSSU COPY utility with the PHYSINDYNAM option to copy your zFS data sets if you are using this function.

Note: This support is limited to single-volume zFS data sets.

9.60 Using indirect volume serials with cloned zFS data sets

- Clone a zFS by making copies of the existing zFS data sets using COPY with the PHYSINDYNAM (PIDY) parameter for the following reasons:
  - Using PHSYINDYNAM creates an uncataloged copy of the data set
  - You need the uncataloged version so that it can be accessed by the catalog entry with the indirect volser
- Using PHYSINDYNAM (PIDY) lets you use the same name for your original and copied zFS
- Create a system symbolic in IEASYMxx for &VOL01 for each system that has a copy of the zFS data set
  - SYSDEF SYMDEF(&VOL01.='1P0301')

Figure 9-61 Indirect volsers with cloned zFS data sets

Indirect volsers with cloned zFS data sets

Clone a zFS by making copies of the existing zFS data sets using COPY with the PHYSINDYNAM (PIDY) parameter for the following reasons:

- Using PHSYINDYNAM creates an uncataloged copy of the data set. You need the uncataloged version so that it can be accessed by the catalog entry with the indirect volser.
- Using PHYSINDYNAM (PIDY) lets you use the same name for your original and copied zFS.

Specifying symbolic volsers for zFS data sets

To use an indirect VOLSER for zFS data sets, you need to set up a system symbolic in an IEASYMxx member of SYS1.PARLMLIB. The following example shows how to define a symbolic using the SYMDEF statement for IEASYMxx. In this example, the symbolic name is &VOL01 and it is assigned a specific volume serial number of 1P0301 to be used on that system.

```
SYSDEF SYMDEF(&VOL01.='1P0301')
```

Note: The symbol must be set to the whole volume number. The period at the end of the symbolic is optional.

IEASYMxx parmib member

To use the IEASYMxx parmlib member on a given system, append the 2-character xx identifier to the IEASYM parameter in the LOADxx parmlib member of SYS1.IPLPARM. The LOADxx member specifies the IEASYMxx member(s) to search for any symbolic definitions used on that system.

JCL for the COPY command

The following JCL shows a recommended COPY command to clone a zFS.

```
//STEPT07 EXEC PGM=ADRDSSU
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
   COPY DS(INC(ZFS.LDS)) -
   PIDY ( -
   (1P0301) -
   ) -
   OUTDY ( -
   (1P0302) -
   ) -
   REPLACEU -
   ALLDATA(*) -
   ALLEXCP
```

9.61 Define a VSAM LDS and format

- Define a VSAM LDS using the real volume serial

  DEFINE CLUSTER (NAME(ZFS.LDS) CYL(1 1) -
  VOLUMES(1P0301) STORCLAS(S1P03S02) -
  LINEAR)

- Run the USS IOEAGFMT utility against the data set
  - This formats the LDS as a zFS data set by setting the zFS indicator in the catalog record for the data set

```
//CREATE   EXEC   PGM=IOEAGFMT,REGION=0M,
//      PARM=('-aggregate ZFS.LDS -compat')
//SYSPRINT DD     SYSOUT=*
//STDOUT   DD     SYSOUT=*
//STDERR   DD     SYSOUT=*
//SYSUDUMP DD     SYSOUT=*
//CEEDUMP  DD     SYSOUT=*
```

Figure 9-62 Define a VSAM LDS and format it

Define a VSAM LDS and format

The **DEFINE CLUSTER** command builds a cluster entry and a data entry to define the linear data set cluster. The parameters are:

- NAME specifies that the cluster's name is ZFS.LDS.
- VOLUMES specifies that the cluster is to reside on volume IP0301.
- STORCLAS for SMS-managed data sets gives the 1-8 character name of the storage class. When possible, allow STORCLAS to default through the ACS routines established by your storage administrator. Attributes assigned through storage class and the ACS routines replace storage attributes such as UNIT and VOLUME. If SMS is not active, the system checks the syntax and then ignores the STORCLAS parameter.
- LINEAR specifies that the cluster's data organization is to be linear.

IOEAGFMT utility

This is a batch utility that formats a VSAM linear data set to become a compatibility mode aggregate.

- -aggregate name - Specifies the name of the data set to format. This is also the aggregate name.
- -compat - Indicates that a compatibility mode aggregate should be created. This means that in addition to formatting the VSAM LDS as a zFS aggregate, a zFS file system by the same name (the aggregate name) is created.

9.62 Delete the data set and IDCAMS DEFINE

- Run an IDCAMS DELETE NOSCRATCH against the data set
  - DELETE ZFS.LDS NOSCRATCH
  - This deletes the catalog entry of the data set
- Next - run an IDCAMS DEFINE RECATALOG using the system symbolic (indirect volser) against the data set
  - DEFINE ZFS.LDS VOLUMES(&VOL01) RECATALOG
- This defines the data set with the system symbolic

Figure 9-63 Delete the data set and IDCAMS DEFINE

Delete the data set and IDCAMS DEFINE

Use an IDCAMS DELETE NOSCRATCH against the data set, as follows:

```
DELETE ZFS.LDS NOSCRATCH
```

This deletes the catalog entry of the data set.

An existing data set can be cataloged through the access method services DEFINE RECATALOG command. Next, you should then run an IDCAMS DEFINE RECATALOG using the system symbolic (indirect volser) against the data set, as follows:

```
DEFINE  ZFS.LDS  VOLUMES(&VOL01)  RECATALOG
```

This defines the data set with the system symbolic.

Note: On each system set the value of the system symbolic to a different value. You then will have a single catalog entry (for zFS VSAM LDS data sets) that represents different volumes on various systems. You can then use this to clone volumes.

9.63 Centralized BRLM support

- Lock all or part of a file that you are accessing for:
  - Read-write purposes
- As a default, the lock manager is initialized on only one system in the sysplex
  - The first system that enters the sysplex initializes the BRLM and becomes the system that owns the manager - This is called a centralized BRLM
- If the BRLM server crashes, or owning system partitioned out of the sysplex
  - BRLM is reactivated on another system in the group
  - All locks that were held are lost
  - An application that accesses a file previously locked receives an I/O error - must close and reopen the file

Figure 9-64 Centralized BRLM support

Centralized BRLM

The byte-range lock manager (BRLM) is used to lock all or part of a file that you are accessing for read-write purposes.

As a default, the lock manager is initialized on only one system in the sysplex. The first system that enters the sysplex initializes the BRLM and becomes the system that owns the manager. This is called a centralized BRLM.

In a sysplex environment, a single BRLM handles all byte-range locking in the shared file system environment group. If the BRLM server crashes, or if the system that owns the BRLM is partitioned out of the sysplex, the BRLM is reactivated on another system in the group. All locks that were held under the old BRLM server are lost. An application that accesses a file that was previously locked receives an I/O error, and has to close and reopen the file before continuing.

9.64 Distributed BRLM

- Can have distributed BRLM initialized on every system in the sysplex
  - Each BRLM is responsible for handling locking requests for files whose file systems are mounted locally in that system
  - Use distributed BRLM if you have applications that lock files that are mounted and owned locally
- To activate distributed BRLM
  - z/OS UNIX couple data set (BPXMCDS) must be updated
  - Supported code must be installed and running on each system - See APAR OW52293 for more information

Figure 9-65 Distributed BRLM

Distributed BRLM

You can choose to have distributed BRLM initialized on every system in the sysplex. Each BRLM is responsible for handling locking requests for files whose file systems are mounted locally in that system. Use distributed BRLM if you have applications that lock files that are mounted and owned locally.

For distributed BRLM to be activated, the z/OS UNIX couple data set (BPXMCDS) must be updated and the supported code must be installed and running on each system. See APAR OW52293 for more information.

BRLM with z/OS V1R4

z/OS R1V4 implements the first phase of moveable BRLM in a sysplex. Moveable BRLM provides the capability of maintaining the byte range locking history of applications, even when a member of the sysplex dies. This first phase will focus on distributing the locking history across all members of the sysplex. As a result, many applications that lock files that are locally mounted will be unaffected when a remote sysplex member dies. Movement away from a centralized to a distributed BRLM will provide greater flexibility and reliability.

9.65 Define BRLM option in CDS

```
ITEM NAME(DISTBRLM) NUMBER(1)
     /*Enables conversion to a distributed BRLM.
       1, distributed BRLM enabled,
       0, distributed BRLM is not enabled during next sysplex IPL
       Default = 0  */
```

OMVS couple data set

Applications (including cron, inetd, and Lotus Domino) lock local files

See http://www-03.ibm.com/servers/eserver/zseries/zos/unix/bpxa1ty2.html for the rangelks REXX tool to identify currently running lock holders

Figure 9-66 BRLM and the OMVS couple data set

BRLM and OMVS couple data set

This support allows you to change to using distributed BRLM (rather than a single, central BRLM) in the sysplex. With distributed BRLM, each system in the sysplex runs a separate BRLM, which is responsible for locking files in the file systems that are owned and mounted on that system. Because most applications (including cron, inetd, and Lotus® Domino) lock local files, the dependency on having a remote BRLM up and running is removed. Running with distributed BRLM is optional.

9.66 BRLM problems in a sysplex

- A file system cannot be moved in a sysplex when an application holds byte range locks in that file system
  - Moving of locks was not supported with distributed BRLM
- Also, applications holding byte range locks in a remote file system are exposed when that remote sysplex member normally terminates
  - The locks are lost and the application is notified
- **Solution z/OS V1R6:**
  - Distributed BRLM is enhanced to support moving byte range locks

Figure 9-67 Moving byte range locks in z/OS V1R6

Byte-range locks with z/OS V1R6

With V1R6, the lock manager is initialized on every system in the sysplex. This is known as distributed BRLM, and it is the only supported byte-range locking method when all systems are at the V1R6 level. Each BRLM is responsible for handling locking requests for files whose file systems are mounted locally in that system. Distributed BRLM was an option on previous levels of z/OS, and central BRLM was formerly the default.

When a system failure occurs, all byte-range locks are lost for files in file systems owned by that system. To maintain locking integrity for those locked files that are still open on surviving systems, z/OS UNIX prevents further locking or I/O on those files. In addition, the applications are signaled, in case they never issue locking requests or I/O. Running applications that did not issue locking requests and did not have files open are not affected.

If you are already running with a z/OS UNIX CDS indicating that distributed BRLM is enabled, there is no change required to activate distributed BRLM for V1R6. Likewise, if your sysplex only has systems at the V1R6 level, there is no change required because distributed BRLM is the default. V1R6 systems ignore the z/OS UNIX CDS DISTBRLM setting.

However, if you migrate to V1R6 by running mixed levels in a sysplex, you should enable distributed BRLM before IPLing the V1R6 system because a V1R6 system may attempt to activate distributed BRLM when the central BRLM server leaves the sysplex, regardless of the z/OS UNIX CDS setting.

9.67 z/OS V1R8 BRLM recovery of locks

- When an USS application locks a remote file, the lock is lost when the remote system is lost
- Solution: Back up remote locks locally, and recover them when a system fails
  - An application issues a lock for a file owned by another system
  - USS forwards the lock to that systems BRLM
  - USS forwards the lock to the owner systems BRLM
  - …... file owner system fails
  - USS recovers the file system and declares a new owning system
  - USS re-issues the lock to new owner

Figure 9-68 BRLM recovery of locks

BRLM lock recovery

Management of locks in a z/OS UNIX environment began with OS/390 V2R9 using central BRLM in a shared HFS sysplex that included support for one BRLM. This implementation became a single point of failure. z/OS V1R4 offered distributed BRLM, which provided one BRLM per system in a sysplex where lock commands are routed to the file system owner. In z/OS V1R6, distributed BRLM with moveable locks was introduced and the locks moved when the file system moved. Distributed BRLM then became the default in a z/OS V1R6 sysplex.

In z/OS V1R8, the ability to recover file locks when the remote system fails is implemented. The order for a remote file lock occurs in the following way:

1. An application issues a lock for a file owned by another system.
2. USS forwards the lock to that system's BRLM.
3. USS forwards the lock to the owner system's BRLM.
4. …... file owner system fails.
5. USS recovers the file system and declares a new owning system.
6. USS re-issues the lock to the new owner.

Attention: When the file system containing the file is owned remotely, the lock will also be remote.

9.68 File system access

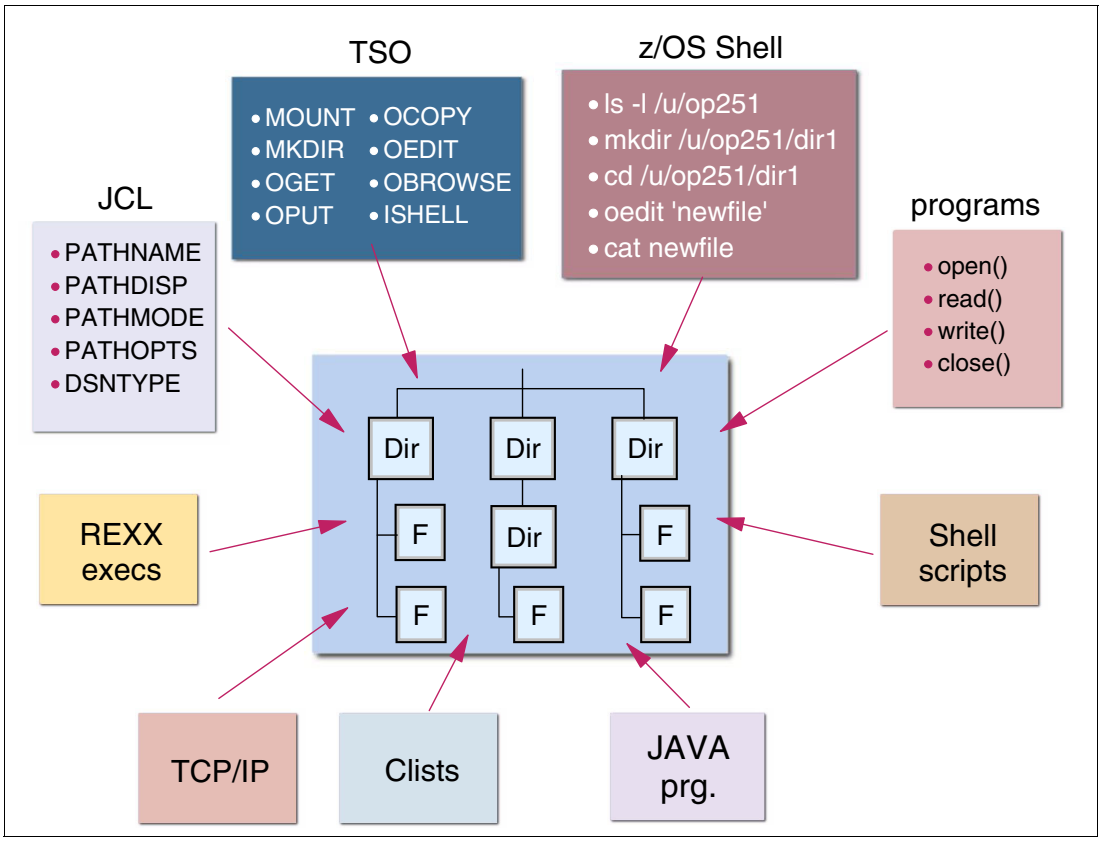


Figure 9-69 Accessing file system data from different methods

Accessing file system data

We have seen how we can customize the HFS for use. But who can use the HFS and the z/OS UNIX? The z/OS UNIX file system can be used or accessed by using any of the following:

TSO/E TSO/E has commands for browsing and editing HFS files, for copying data between HFS files and MVS data sets, to mount file systems, to invoke the z/OS UNIX shell. There is also a command called ISHELL which provides an ISPF menu-driven interface to the file system.

JCL JCL provides keywords which support the specification of HFS path names.

REXX REXX has a set of z/OS UNIX extensions (`syscall` commands) to access z/OS UNIX callable services.

z/OS UNIX shell The shell is the UNIX interface to the file system. It contains commands and utilities to access HFS files.

C programs z/OS UNIX supports many C functions to access HFS files.

Shell scripts Shell scripts are similar to REXX execs. z/OS UNIX System Services shell commands and utilities can be stored in a text file that can be executed.

The HFS data set cannot be OPENed by any MVS utilities. Only DFSMSdss can be used to dump and restore this data set type. The internal structure of an HFS data set is only accessible via z/OS UNIX System Services.

9.69 File access

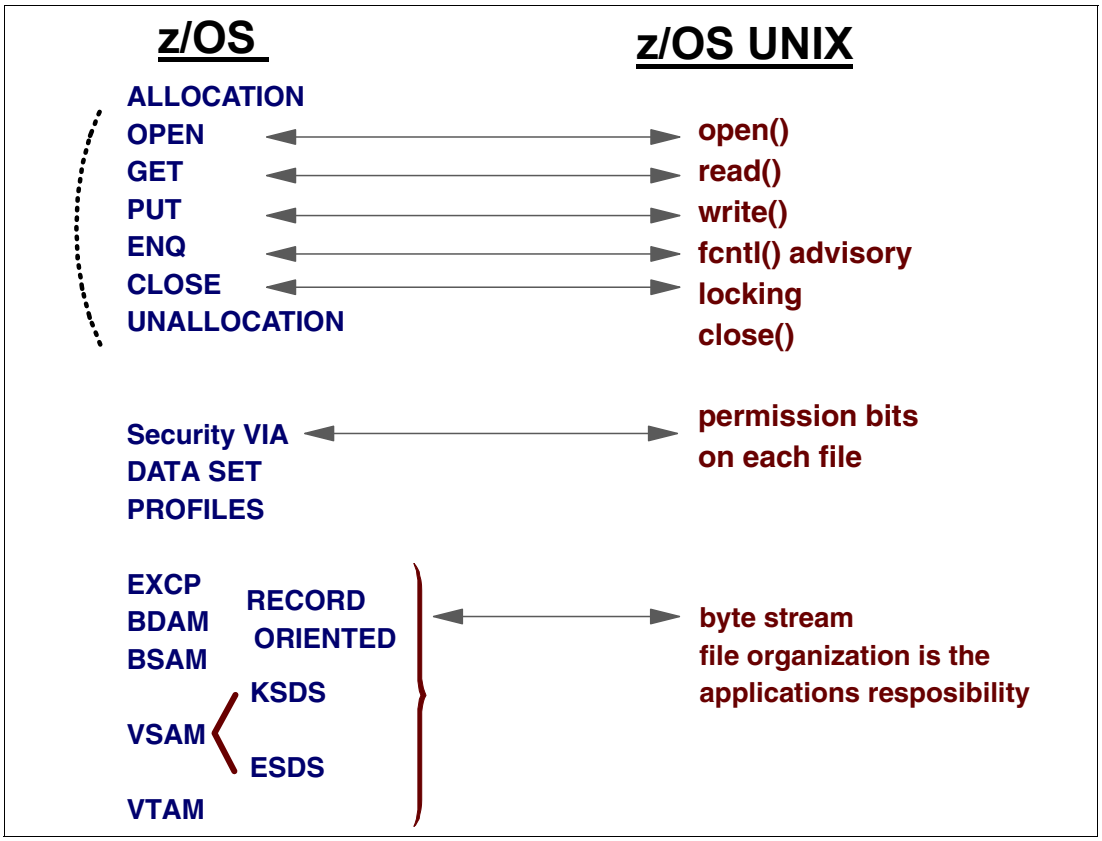


Figure 9-70 File system access versus MVS data set access

File access comparison

Figure 9-70 shows corresponding file access methods in z/OS and the POSIX standard.

The big difference is only the file handling. While OS/390 gives you file handle methods (you do not have to implement this), the POSIX standard does not define the file handling methods. So any application running on the POSIX standard is responsible for storing and retrieving data from and to the file.

9.70 List file and directory information

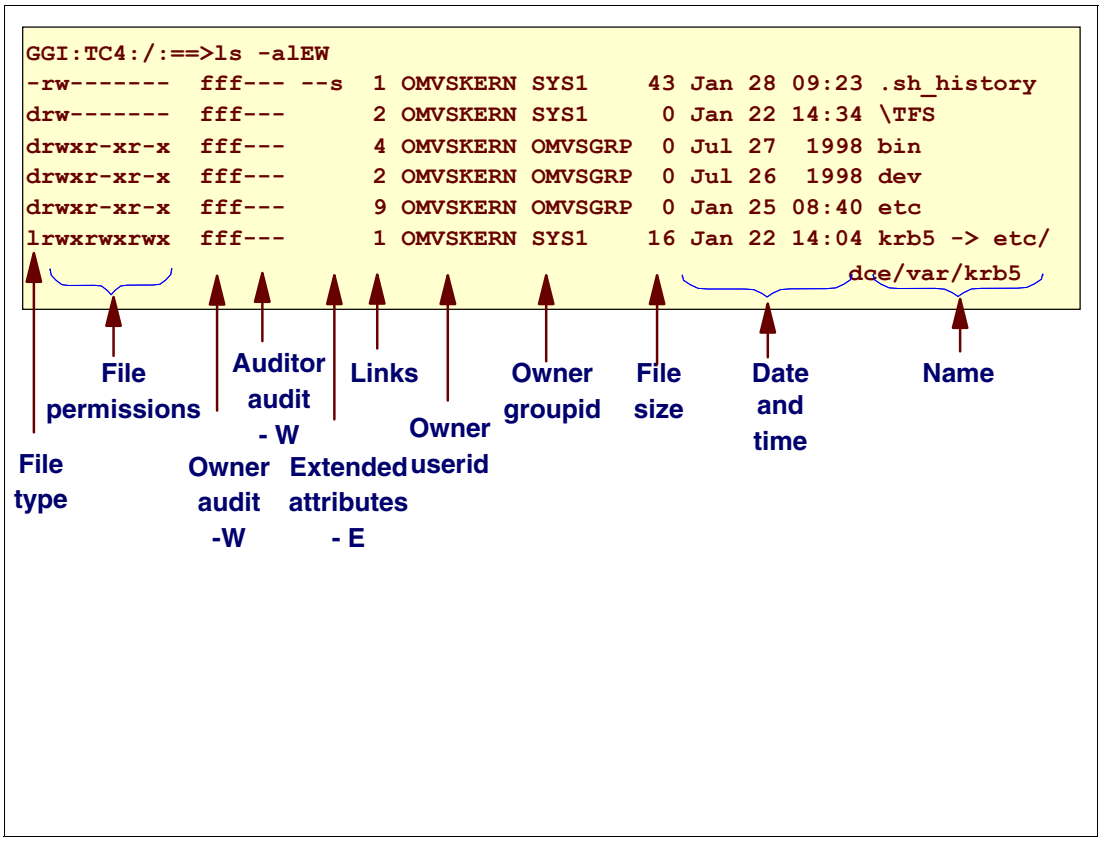


Figure 9-71 Command to access file and directory information

Displaying files and directories

Figure 9-71 shows the output of the **`ls -alEW`** command.

- File type describes the type of file (for example, `d` Directory, `l` Symbol link, `c` Character special file, `f` Regular file, and so on).
- File permissions are Read Write Execute for user group and other. Sticky Bit.
- Owner audit (f for failed, s for successful access if audit attribute is set).
- Auditor audit is the same as for owner audit.
- External attributes.
- Links are the number of links to the file.
- Owner user ID shows which user ID owns the file or directory.
- Owner Group ID shows the name of the group that owns this file or directory.
- File size is the size of the file in bytes.
- Date and Time shows the date and time of change or creation.
- Name specifies a file name or link pointing to the file name.

9.71 File security packet - extattr bits

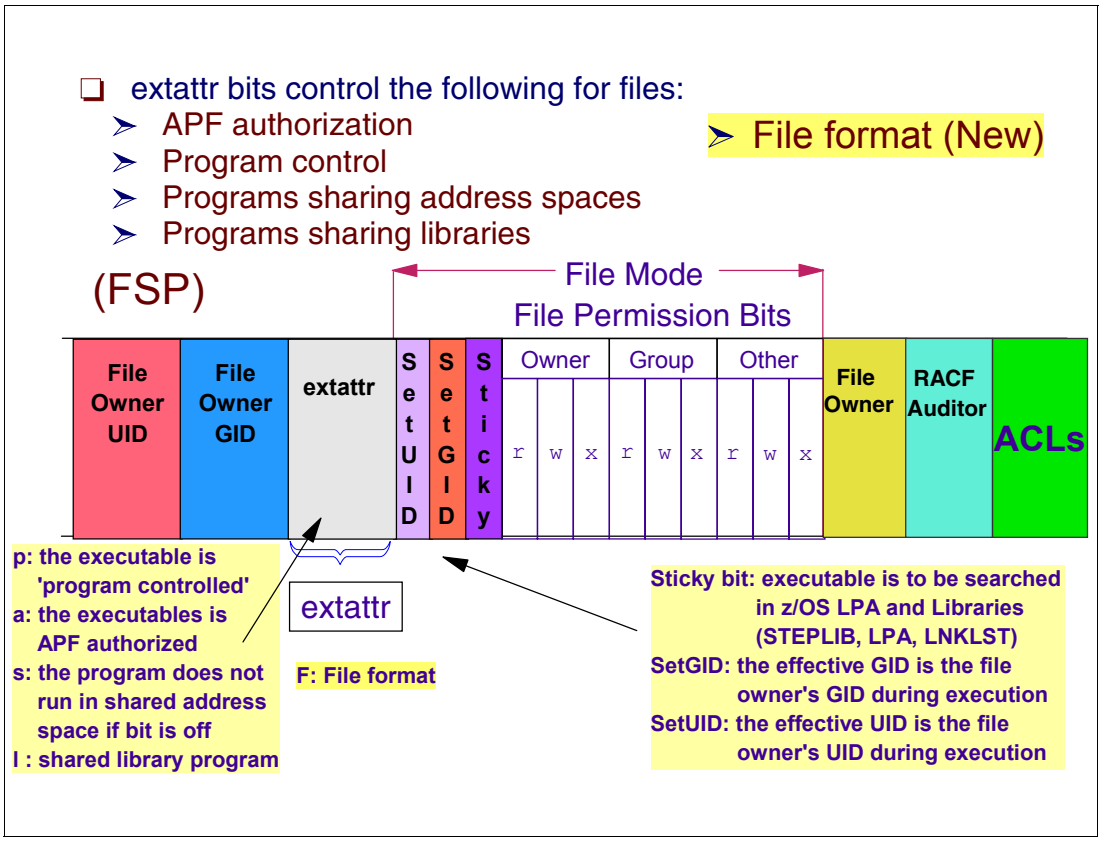


Figure 9-72 Extended attribute bits in the FSP

extattr bits in the FSP

The extended attributes are kept in the file security packet (FSP). The extended attributes give special authorities to the files. The **`extattr`** command is used to set these attributes.

a When this attribute is set (+a) on an executable program file (load module), it behaves as if loaded from an APF-authorized library. For example, if this program is exec()ed at the job step level and the program is linked with the AC=1 attribute, the program will be executed as APF-authorized. To be able to use the **`extattr`** command for the +a option, you must have at least READ access to the BPX.FILEATTR.APF FACILITY class profile.

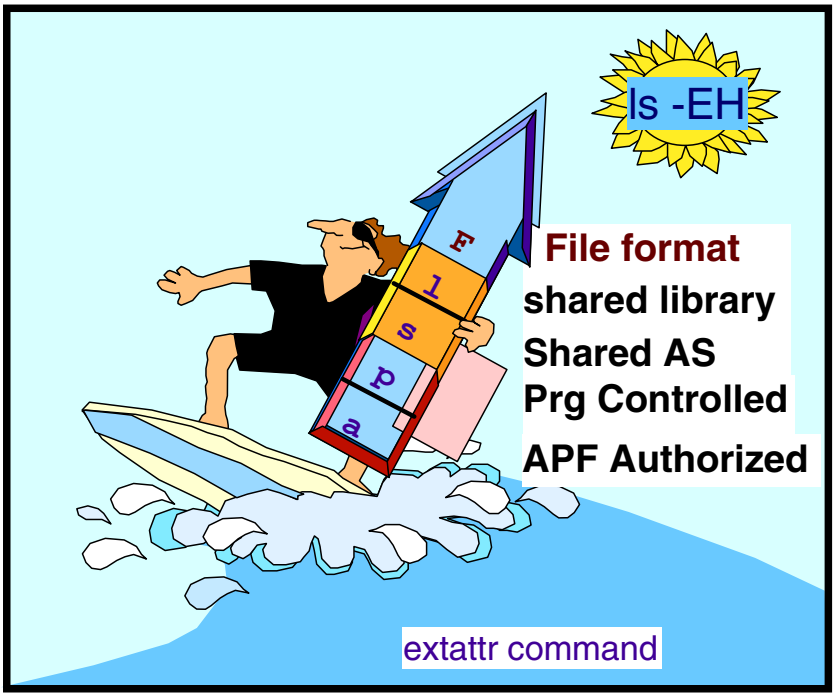
l When this attribute is set (+l) on an executable program file (load module), it will be loaded from the shared library region. To be able to use the extattr command for the +l option, you must have at least READ access to the BPX.FILEATTR.SHARELIB FACILITY class.

p When this attribute is set (+p) on an executable program file (load module), it causes the program to behave as if an RDEFINE had been done for the load module to the PROGRAM class. When this program is brought into storage, it does not cause the environment to be marked dirty. To be able to use the extattr command for the +p option, you must have at least READ access to the BPX.FILEATTR.PROGCTL FACILITY class.

s If the extended attribute for the shared address space is not set, the program will not run in a shared address space, regardless of the setting of _BPX_SHAREAS. The attribute is set by extattr +s and reset by extattr -s.

F The **`extattr`** command is used to set a file format for a file, but it cannot be set for symlinks and directories.

9.72 Extended attributes




```
rwx rwx rwx    fff---  a p s l F filename
```

Figure 9-73 Listing the extended attributes

Extended attributes bits

The extended attributes give special authorities to the files. Four different extended attributes are defined:

APF Authorized programs	The behavior of these programs is the same as other programs that are loaded from APF-authorized libraries.
Program-controlled program	All programs that are loaded into an address space that requires daemon authority need to be marked as “controlled.”
Shared AS	The program shares its address space with other programs.
Shared library	Programs using shared libraries contain references to the library routines that are resolved by the loader at run time.
File format	The file format can now be specified as any of the following formats: binary data, newline, carriage return, line feed, carriage return followed by line feed, line feed followed by carriage return, or carriage return followed by newline.

Extended attribute bits - (ls -E)

The `ls -E` shell command has an option that displays these attributes:

a Program runs APF-authorized if linked AC=1

p Program is considered program-controlled

s Program runs in a shared address space

l Program is loaded from the shared library region

Extended attribute bits - (ls -H)

In Figure 9-74 on page 525 you can see the output of the **ls** command using the **-H** option, which allows the file format to be displayed for a file.

```
ROGERS @ SC75:/u/rogers>extattr -F CRLF maxcore
ROGERS @ SC75:/u/rogers>ls -H
total 160
-rwxr-xr-x  crlf   1 SYSPROG  SYS1       73728 Apr 21 16:08 maxcore
ROGERS @ SC75:/u/rogers>
```

Figure 9-74 Sample output of ls -H

```
ROGERS @ SC64:/u/rogers>ls -EH
total 240
-rw-r--r--  --s- nl     1 HAIMO    SYS1        1570 Jun 14  2006 CEEDUMP.2006061
4.094113.67305623
-rwxr-xr-x  a-s- ----   1 HAIMO    SYS1           0 Aug 17  2005 apf.file
drwxr-xr-x              2 HAIMO    SYS1        8192 Feb 28  2006 cache1
drwxr-xr-x              2 HAIMO    SYS1        8192 Feb 28  2006 cache2
drwxr-xr-x              2 HAIMO    SYS1        8192 Mar  2  2006 cachex
drwxr-xr-x              2 HAIMO    SYS1        8192 Mar  2  2006 cachey
-rwx------  --s- nl     1 HAIMO    SYS1         729 Aug  2 11:32 cbprnt
drwxr-xr-x              2 HAIMO    SYS1        8192 Feb 21  2006 fill
drwxr-xr-x              2 HAIMO    SYS1        8192 Feb 21  2006 fill3
drwxr-xr-x              2 HAIMO    SYS1        8192 Feb 21  2006 fill4
drwxr-xr-x              2 HAIMO    SYS1        8192 Feb 23  2006 fill5
-rwx------  --s- nl     1 HAIMO    SYS1         648 Sep  4 01:31 gener
drwxr-xr-x              2 HAIMO    SYS1        8192 Feb  8  2006 largezfs
```

Figure 9-75 Sample output of ls -EH

extattr command

To set the extended attribute bits, the **extattr** command can be used.

Displaying extended attributes

The status of the extended attributes can be shown with the following commands:

ls Sticky Bit and Set UID/GID Bit

ls -E APF Auth., Program Control, Shared AS, and shared library

ls -W RACF Audit

Setting the extended attributes

With the following commands you can set the different extended attributes:

chmod Sticky Bit, Set UID/GID

mkdir Sticky Bit

extattr Program Control, APF authorized, Shared AS

9.73 APF-authorized attribute

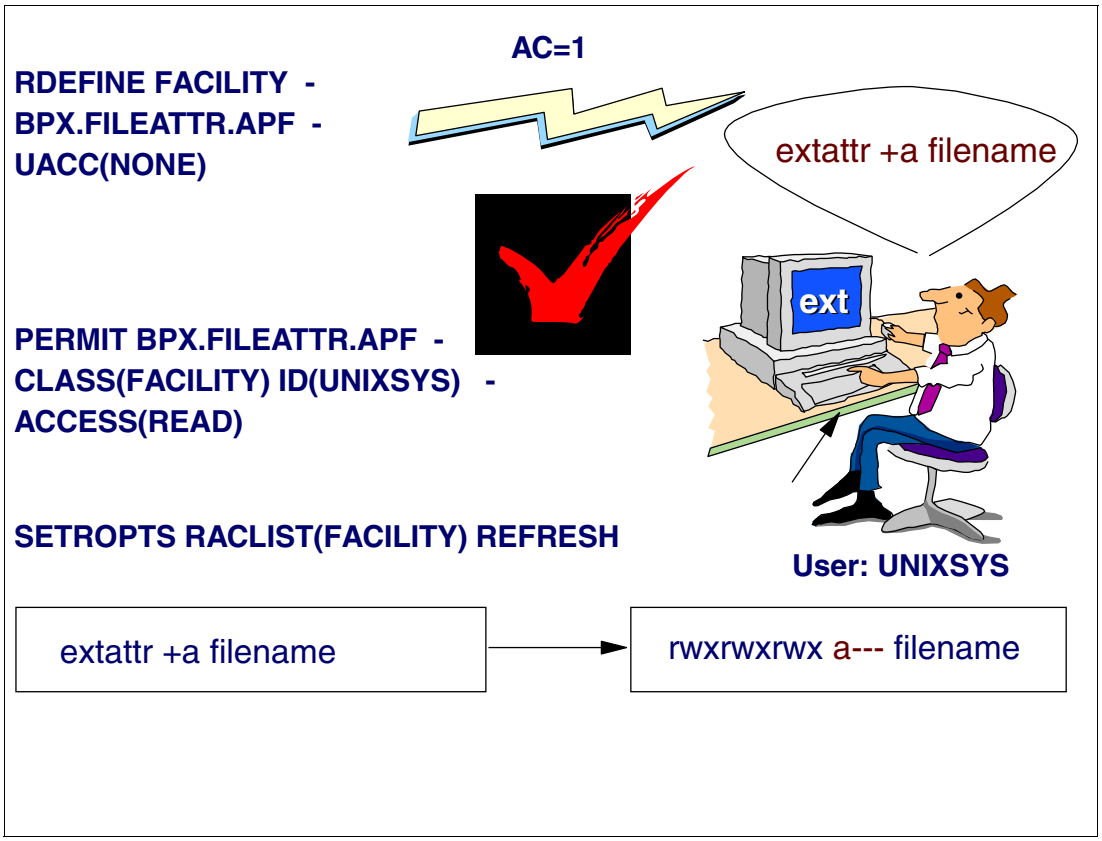


Figure 9-76 APF-authorized attribute

APF-authorized bit

The entire HFS is considered as an unauthorized library. You can authorize individual programs within the HFS as APF-authorized by setting the APF-extended attribute. If a program running in an APF-authorized address space attempts to load a program from the HFS that does not have the APF-extended attribute set, the load is rejected. This applies to non-jobstep exec local spawn, attach_exec, and DLL loads.

In order to activate APF authorization, the RACF profile BPX.FILEATTR.APF must be defined. The z/OS UNIX administrator must have read access to this profile to execute the **extattr** command, as follows:

- To turn on the APF-authorized program bit, issue:
  ```
  extattr +a  filename
  ```
- To remove the APF-authorized program bit, issue:
  ```
  extattr -a  filename
  ```

You can link-edit the program into an APF-authorized library and turn on the sticky bit in the HFS. You can use the **extattr** shell command to set the APF-authorized extended attribute of the file.

If an APF-authorized program is the first program to be executed in an address space, then you also need to set the Authorization Code to 1 (AC=1) when your program is link-edited. If a program is loaded into an APF-authorized address space but is not the first program to be executed, it should not have the AC=1 attribute set.

9.74 Activate program control

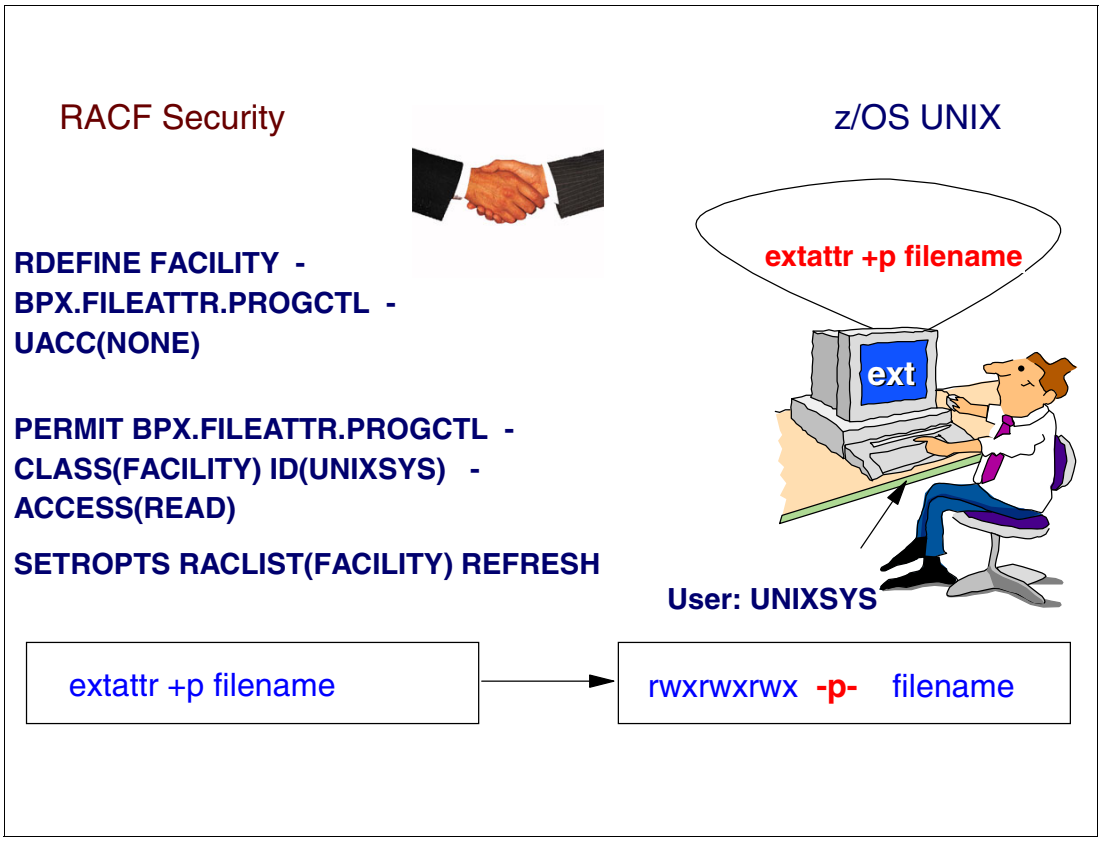


Figure 9-77 Program control extended attribute

Program control attribute bit

All programs loaded into an address space that requires daemon authority need to be marked as controlled. This means that user programs and any run-time library modules that are loaded must be marked as controlled.

You can mark programs in HFS files as controlled by turning on the extended attribute that allows you to run program-controlled. To turn this extended attribute on:

```
extattr +p  filename
extattr +p /user/sbin/proga
```

Only users with the correct permission can turn on the extended attribute. The example above shows the RACF command used to give this permission to z/OS UNIX administrator UNIXSYS. To turn off the extended attribute, use the **extattr** shell command:

```
extattr -p  filename
extattr -p /user/sbin/su
```

After a file is marked program-controlled, any activity that can change its contents results in the extended attribute being turned off. If this occurs, a system programmer with the appropriate privilege will have to verify that the file is still correct and reissue the **extattr** command to mark the file as program-controlled.

All modules loaded from LPA are considered to be controlled.

9.75 Shared address space attribute

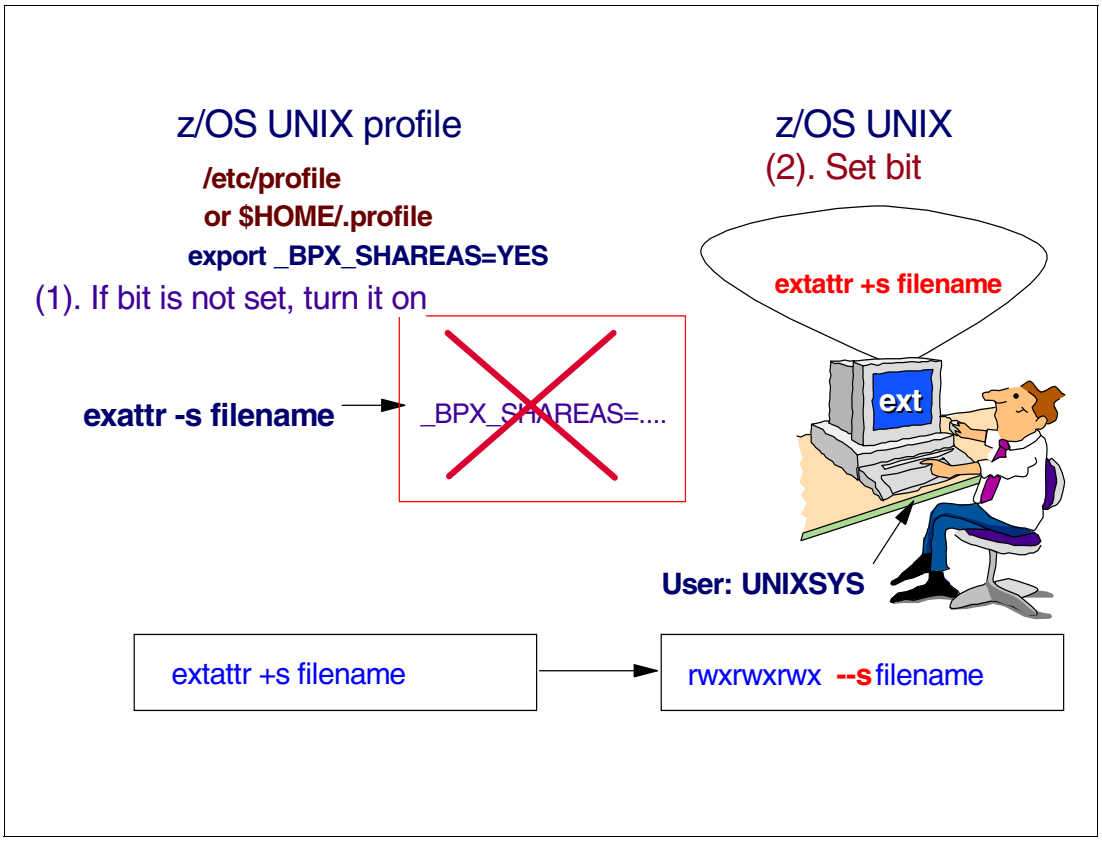


Figure 9-78 Shared AS extended attribute

Shared address space attribute bit

To improve the performance in the shell, the extended attribute for shared AS can be set. The program will share this address space with other programs.

To turn this extended attribute on:

```
extattr +s  filename
```

To turn off the bit:

```
extattr -s  filename
```

When this attribute is not set (-s), the _BPX_SHAREAS environment variable is ignored when the file is spawn()ed. By default, this attribute is set (+s) for all executable files.

To improve performance for all shell users, it is recommended that /etc/profile or $HOME/.profile set the environment variable.

9.76 Shared library attribute

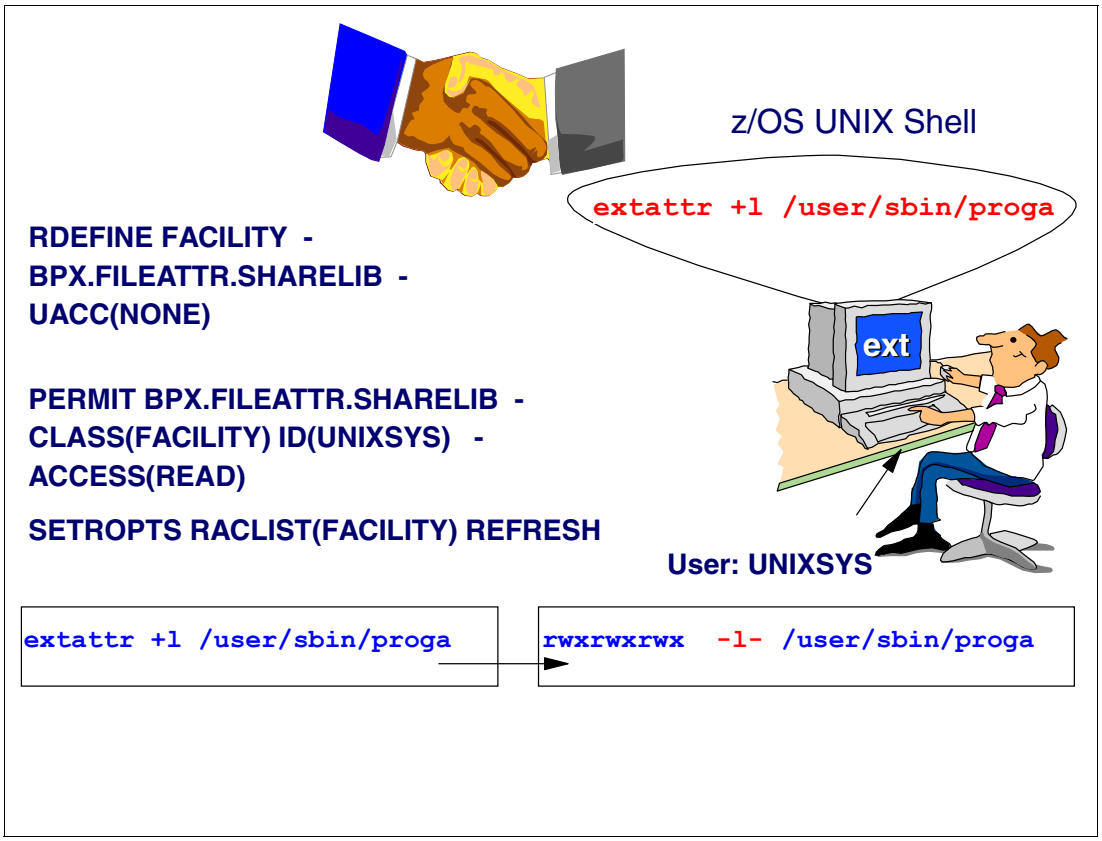


Figure 9-79 Shared library extended attribute

Shared library attribute bit

When this attribute is set (`+l`) on an executable program file (load module), it will be loaded from the shared library region.

To be able to use the **`extattr`** command for the `+l` option, you must have at least READ access to the BPX.FILEATTR.SHARELIB FACILITY class. For more information, see *z/OS UNIX System Services Planning,* GA22-7800.

Note: `l` is a lower case L, not the numeral one or an upper case i.

The BPX.FILEATTR.SHARELIB FACILITY class profile controls who can set the shared library extended attribute. The following example shows the RACF command that was used to give READ access to user Ralph Smorg with user ID SMORG:

```
RDEFINE FACILITY BPX.FILEATTR.SHARELIB UACC(NONE)
PERMIT BPX.FILEATTR.SHARELIB CLASS(FACILITY) ID(SMORG) ACCESS(READ)
SETROPTS RACLIST(FACILITY) REFRESH
```

To set the shared library attribute, issue the **`extattr`** command with the `+l` option. In the following example, `proga` is the name of the file.

```
extattr +l  /user/sbin/proga
```

9.77 File format attribute

- Support the file format from the OMVS shell
- extattr command is enhanced to accept a –F option
  - Values - consistent with the cp command to indicate the format of the file
    - extattr –F command from the shell
  - For format you can specify:
    - NA - Not specified
    - BIN - Binary data
  - Or the following text data delimeters:
    - NL – New Line
    - CR - Carriage Return
    - LF - Line Feed
    - CRLF - Carriage Return followed by Line Feed
    - LFCR - Line Feed followed by Carriage Return
    - CRNL - Carriage Return followed by New Line

Figure 9-80 Specifying the file format attribute

File format attribute

Prior to z/OS V1R8, there was no supported commands to set the file format from the OMVS shell. The **extattr** command is enhanced to accept a –F option with values consistent with the **cp** command to indicate the format of the file.

To copy file f1 to a fully qualified sequential data set 'turbo.gammalib' and treat it as a binary:

```
cp -F bin f1 "//'turbo.gammalib'"
```

The syntax for the **extattr** command is as follows:

```
extattr [+alps] [-alps] [-F] file
```

Specifying the file format option

The format option can be specified in lowercase, uppercase, or in mixed case. The format option can also be specified with a space or no space after the file format flag (-F). For example, by specifying -F with line feed (LF) and carriage return (cr), as follows:

```
extattr -FLFcr filename
```

The file format flag (-F) can be used with other **extattr** flags (+alps/-alps), but it must be separated by a space or tab. For example:

```
extattr +aps -F BIN filename            (is a valid entry)
extattr -apsF NA filename               (is not a valid entry)
```

So the option -F can have the values shown in Table 9-1 on page 531.

Table 9-1 Possible -F options for extattr

Value	Format options
NA	Not specified
BIN	Binary data file
Value	**Text data delimeters**
NL	New line
CR	Carriage return
LF	Line feed
CRLF	Carriage return followed by line feed
LFCR	Line feed followed by carriage return
CRNL	Carriage return followed by new line

The setting of a file format flag has no impact on the contents of the file. You can easily set the file format flag as follows:

```
extattr -F CRLF extattr.test
```

In this example, the file extattr.test is set as a text file that contains carriage returns and file feed—also known as a regular text file.

9.78 Extended attribute command example

- Example shows now both sets of extattr values

extattr -F CRLF maxcore

```
ROGERS @ SC75:/u/rogers>ls -alHE
total 184
drwxr-x---              7 SYSPROG  SYS1         544 Sep 13 14:15 .
dr-xr-xr-x              7 SYSPROG  TTY            0 Sep 13 13:54 ..
-rw-------  --s- ----   1 SYSPROG  SYS1         638 Sep 15 13:35 .sh_history
drwxr-xr-x              2 SYSPROG  SYS1         256 Sep 13 14:15 00000100
drwxr-xr-x              5 SYSPROG  OMVSGRP     1120 Jul 26 21:57 ITSO-link
drwxr-xr-x              2 SYSPROG  SYS1         256 Sep 13 14:06 db2
drwxr-xr-x              2 SYSPROG  SYS1         256 Sep 13 14:05 echo
drwxr-xr-x              2 SYSPROG  SYS1         256 Sep 13 14:15 ims
-rwx------  --s- ----   1 SYSPROG  SYS1           0 Aug 28 09:31 inetd-stderr
-rwx------  --s- ----   1 SYSPROG  SYS1           0 Aug 28 09:31 inetd-stdout
-rwxr-xr-x  --s- crlf   1 SYSPROG  SYS1       73728 Apr 21 16:08 maxcore
```

Figure 9-81 Extended attributes displayed

Display of extended attributes

Setting the file format flag on a file does not modify the data in the file. The file format can be displayed via the **`ls -H`** command.

Figure 9-81 shows a **`ls -alHE`** command that displays all five extended attributes. `-E` displays the original four attributes.

In the output, `--s-` is the shared address space attribute for the files.

The file, maxcore, has the attributes `--s- crlf`, where `crlf` is the file format carriage return and line feed.

9.79 Sticky bit

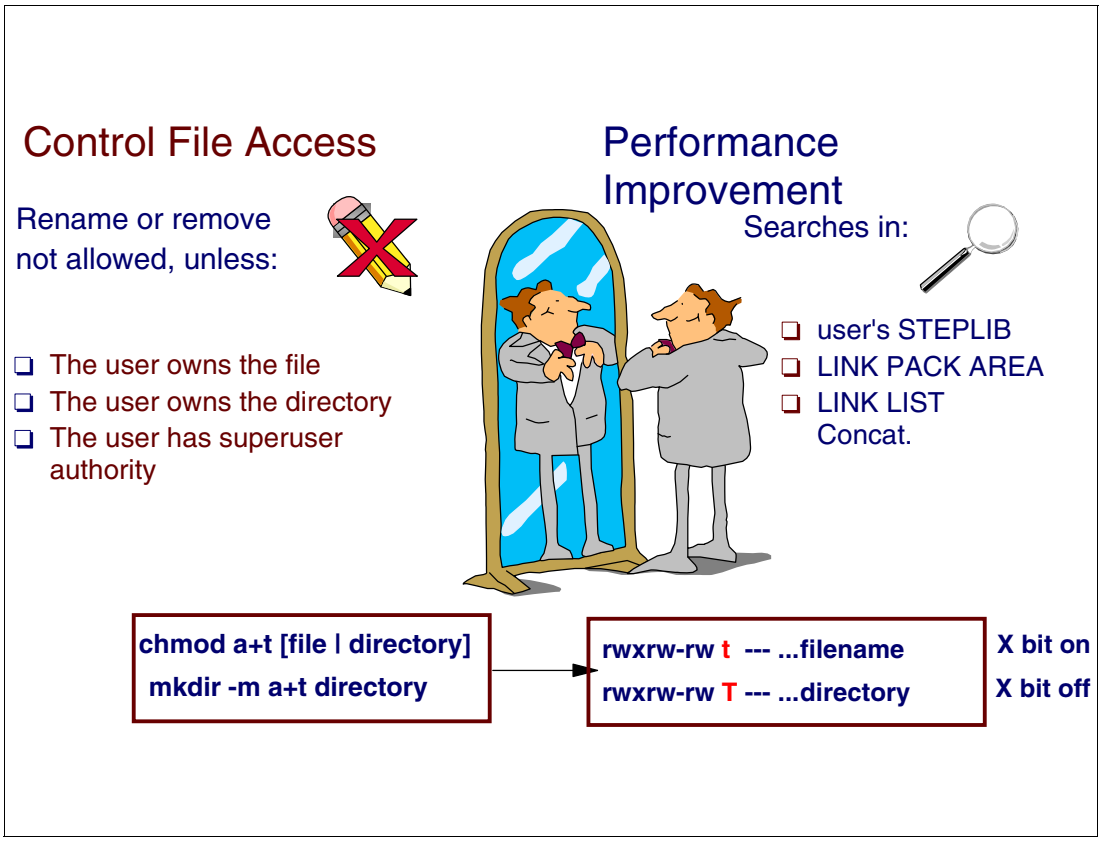


Figure 9-82 Using the sticky bit

Sticky bit for files

The *sticky bit* is a file access permission bit that allows multiple users to share a single copy of an executable file.

For frequently used programs in the HFS, you can use the **chmod** command to set the sticky bit. This reduces I/O and improves performance. When the bit is set on, z/OS UNIX searches for the program in the user's STEPLIB, the link pack area, or the link list concatenation.

Sticky bit for directories

Using the **mkdir**, MKDIR, or **chmod** command, you can set the sticky bit “on” in a directory to control permission to remove or rename files or subdirectories in the directory. When the bit is set, a user can remove or rename a file or remove a subdirectory only if one of these is true:

- The user owns the file or subdirectory.
- The user owns the directory.
- The user has superuser authority.

The **mkdir** command creates a new directory for each named directory argument. The mode for a directory created by **mkdir** is determined by taking the initial mode setting of 777 (a=rwx) or the value of **-m** if specified and applying the umask to it.

The sticky bit is also set for frequently used programs in the file system, to reduce I/O and improve performance. When the bit is set on, z/OS UNIX searches for the program in the user's STEPLIB, the link pack area, or the link list concatenation.

Try the list command: **`ls -al`**.

Using the sticky bit

The following two possibilities show how the sticky bit is used:

T This is the same as **`t`**, except that the execute bit is turned off. That means 750 has not turned on the executable bit for others, so Sticky Bit is: T

t Sticky bit is on. That means 755 has all executable bit on, so Sticky Bit is: t

The z/OS UNIX shell program (sh) is a good candidate, as follows:

- As part of the SMP/E install procedure for the shell, the shell program is placed in the root directory (/bin/sh) and the sticky bit is set.
- The shell program is also placed in SYS1.LPALIB (SH).
- During an IPL with CLPA (create LPA), the shell program will be placed in the LPA.
- Any user that invokes the shell will use the shell module in LPA and save space and I/O, since the module is not loaded into the user's address space.

If you use the **`rmdir`**, **`rename`**, **`rm`**, or **`mv`** utility to work with a file, and you receive a message that you are attempting an `operation not permitted`, check to see if the sticky bit is set for the directory the file resides in.

Note: Only someone with root authority can set the sticky bit.

9.80 Set the UID/GID bit

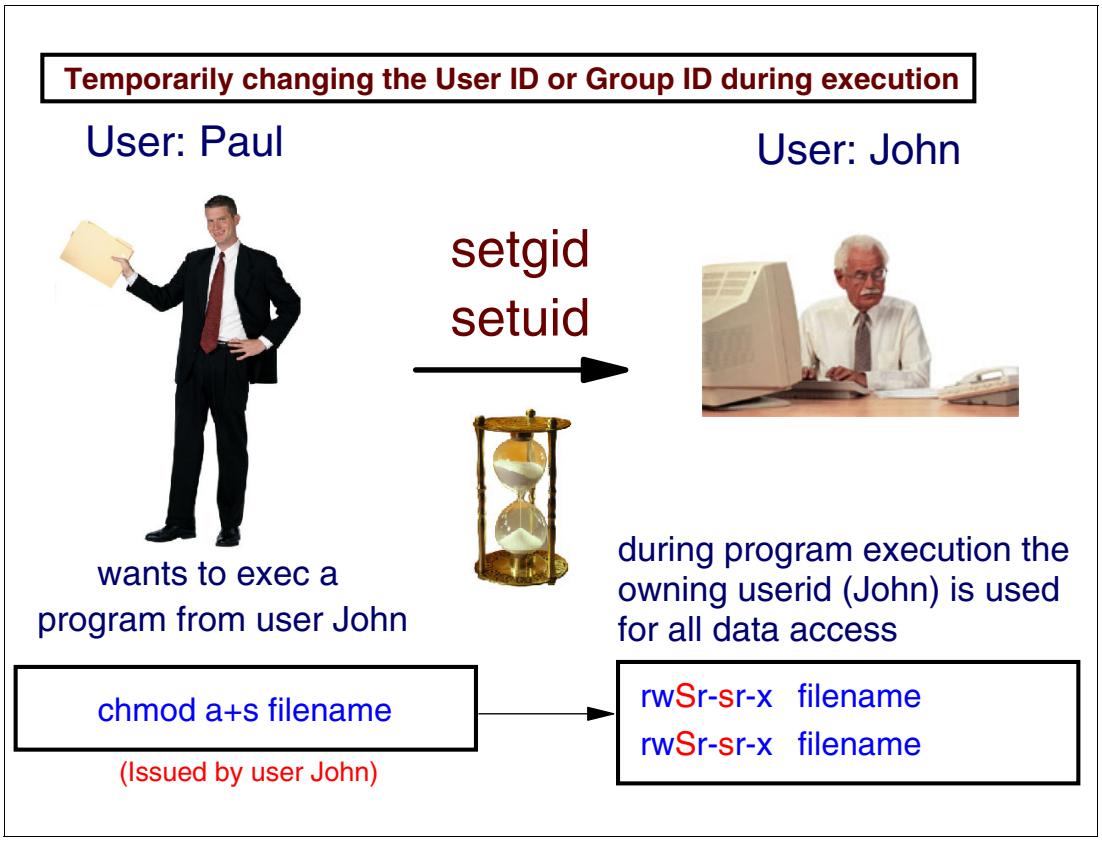


Figure 9-83 Using the setuid and setgid bits

Setuid and setgid bits

If one or both of these bits are on, the effective UID and/or GID plus the saved UID and/or GID for the process running the program are changed to the owning UID, GID, or both, for the file. This change temporarily gives the process running the program access to data the file owner or group can access.

Settings for executable files

An executable file can have an additional attribute, which is displayed in the execute position (x) when you issue `ls -l`. This permission setting is used to allow a program temporary access to files that are not normally accessible to other users. An `s` or `S` can appear in the execute permission position; this permission bit sets the effective user ID or group ID of the user process executing a program to that of the file whenever the file is run. The setuid and setgid bits are only honored for executable files.

Permission bit settings

These bits are not honored for shell script and REXX execs that reside in the file system.

s In the owner permissions section, this indicates that both the set-user-ID (S_ISUID) bit is set and the execute (search) permission is set.

In the group permissions section, this indicates that both the set-group-ID (S_ISGID) bit is set and the execute (search) permission is set.

S In the owner permissions section, this indicates the set-user-ID (S_ISUID) bit is set, but the execute (search) bit is not.

In the group permissions section, this indicates the set-group_ID (S_ISGID) bit is set, but the execute (search) bit is not.

Mailx utility example

A good example of this behavior is the mailx utility. A user sending something to another user on the same system is actually appending the mail to the recipient's mail file. Even though the sender does not have the appropriate permissions to do this, the mail program does.

10

Overview of TCP/IP

The *Internet*, the world's largest network, grew from fewer than 6000 participants at the end of 1986 to millions of networks today. Networks have grown so quickly because they provide an important service. It is the nature of computers to generate and process information, but this information is useless unless it can be shared with the people who need it. The common thread that ties the enormous Internet together is TCP/IP network software. TCP/IP is a set of communication protocols that define how different types of computers talk to each other.

This chapter explains the basic concepts of TCP/IP, including architecture and addressing, and describes how to customize TCP/IP to use it with z/OS UNIX System Services in z/OS.

Note: Many of the figures and examples included in this chapter refer to the lab exercises for the companion class that is taught periodically. See the IBM Redbooks Web site for more information about classes offered in your geographic area.

© Copyright IBM Corp. 2011. All rights reserved.

10.1 Introduction to TCP/IP

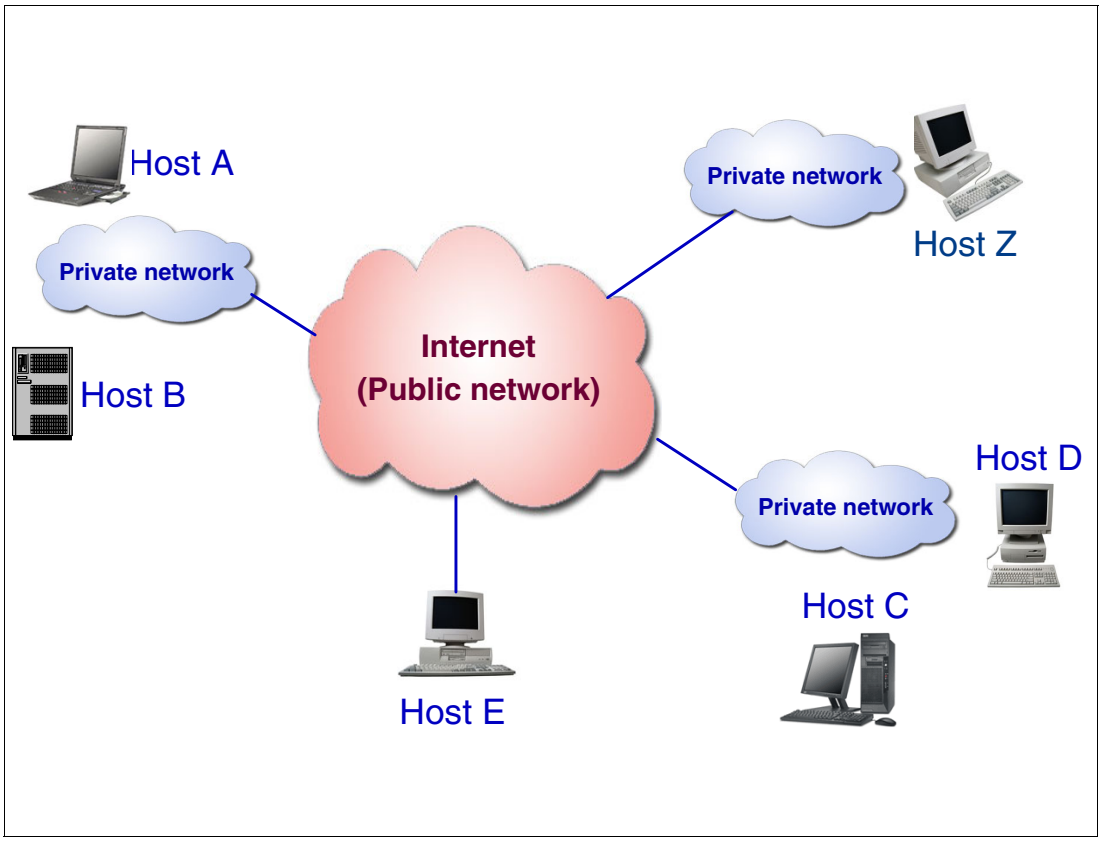


Figure 10-1 TCP/IP network

A TCP/IP network

In Figure 10-1:

- Host A wants to send an e-mail to Host B.
- Host A wants to transfer data to Host D.
- Host C wants to get a Web page from Host E.
- Host Z also wants access to Host E.

These activities are possible because a public network exists. Public networks are established and operated by telecommunication administrations or by Recognized Private Operating Agencies (RPOAs) for the specific purpose of providing circuit-switched, packet-switched, and leased-circuit services to the public.

TCP/IP protocol suite

The TCP/IP protocol suite is named for two of its most important protocols: Transmission Control Protocol (TCP) and Internet Protocol (IP). Another name for it is the Internet Protocol Suite, and this is the phrase used in official Internet standards documents. The more common term TCP/IP is used to refer to the entire protocol suite.

TCP/IP design goals

The first design goal of TCP/IP was to build an interconnection of networks that provided universal communication services. Each physical network has its own technology-dependent communication interface, in the form of a programming interface that provides basic communication functions (*primitives*). Communication services are provided by software that runs between the physical network and the user applications and that provides a common interface for these applications, independent of the underlying physical network. The architecture of the physical networks is hidden from the user.

The second aim is to interconnect different physical networks to form what appears to the user to be one large network. Such a set of interconnected networks is called an *internetwork* or an *Internet*.

To be able to interconnect two networks, we need a computer that is attached to both networks and that can forward packets from one network to the other; such a machine is called a *router*. The term *IP router* is also used because the routing function is part of the IP layer of the TCP/IP protocol suite.

10.2 TCP/IP terminology

- **Host:** Any systems attached to an IP network
- **Gateway:** Another name for a router
- **Port:** An entrance to or exit from a network. The part of a socket address that identifies a port within a host.
- **Socket:** A logical entity used to identify a remote application - **socket = <IP address> + a port number**
- **Router:** Connects networks and routes packets between them

Figure 10-2 TCP/IP terminology

TCP/IP terminology

The following terminology is commonly used to describe a TCP/IP environment:

Host In the Internet suite of protocols, this is an end system. The end system can be any workstation; it does not have to be a mainframe.

Gateway A functional unit that interconnects two computer networks with different network architectures. A *gateway* connects networks or systems of different architectures. A *bridge* interconnects networks or systems with the same or similar architectures. In TCP/IP, this is a synonym for router.

Port Each process that wants to communicate with another process identifies itself to the TCP/IP protocol suite by one or more ports. A port is a 16-bit number, used by the host-to-host protocol to identify to which higher level protocol or application program (process) it must deliver incoming messages. There are two types of ports:

Well-known Well-known ports belong to standard servers, for example Telnet uses port 23. Well-known port numbers range between 1 and 1023 (prior to 1992, the range between 256 and 1023 was used for UNIX-specific servers). Well-known port numbers are typically odd, because early systems using the port concept required an odd/even pair of ports for duplex operations. Most servers require only a single port. The well-known ports are controlled and assigned by the Internet central authority (IANA) and on most systems can only be used by system processes or by programs executed by privileged users. The reason for

well-known ports is to allow clients to be able to find servers without configuration information.

Ephemeral Clients do not need well-known port numbers because they initiate communication with servers and the port number they are using is contained in the UDP datagrams sent to the server. Each client process is allocated a port number as long as it needs it by the host it is running on. Ephemeral port numbers have values greater than 1023, normally in the range 1024 to 65535.

Socket An endpoint for communication between processes or application programs. A synonym for port.

Socket address The address of an application program that uses the socket interface on the network. In Internet format, it consists of the IP address of the socket's host and the port number of the socket. The application program is usually not aware of the structure of the address.

Socket interface A Berkeley Software Distribution (BSD) application programming interface (API) that allows users to easily write their own programs.

Router A router interconnects networks at the internetwork layer level and routes packets between them. The router must understand the addressing structure associated with the networking protocols it supports and make decisions on whether, or how, to forward packets. Routers are able to select the best transmission paths and optimal packet sizes. The basic routing function is implemented in the IP layer of the TCP/IP protocol stack, so any host or workstation running TCP/IP over more than one interface could, in theory and also with most of today's TCP/IP implementations, forward IP datagrams. However, dedicated routers provide much more sophisticated routing than the minimum functions implemented by IP.

10.3 IPv4 addressing

- 32 bit format
  - 0000 0000 . 0000 0000 . 0000 0000 . 0000 0000
  - Example:
    - 9.12.1.43
    - **0000 1001 . 0000 1100 . 0000 0001 . 0010 1011**

Network part of address | Host part of address

- IP address classes

Class	Address Range (First Octet)	Default Subnet Mask	Network Host ID Split
A	0 to 127	225.0.0.0	N.h.h.h
B	128 to 191	255.255.0.0	N.N.h.h
C	192 to 223	255.255.255.0	N.N.N.h

Figure 10-3 IPv4 addressing

IP addresses

An Internet Protocol (IP) address is a numerical label that is assigned to devices participating in a computer network that uses the Internet Protocol for communication between its nodes. An IP address serves two principal functions:

- Host or network interface identification
- Location addressing

Its role has been characterized as follows:

- A name indicates what we seek.
- An address indicates where it is.
- A route indicates how to get there.

IP history

The designers of TCP/IP defined an IP address as a 32-bit number and this system, known as Internet Protocol Version 4 or IPv4, is still in use today. However, due to the enormous growth of the Internet and the resulting depletion of available addresses, a new addressing system (IPv6), using 128 bits for the address, was developed in 1995 and last standardized by RFC 2460 in 1998. Although IP addresses are stored as binary numbers, they are usually displayed in human-readable notations, such as:

```
208.77.188.166 (for IPv4)
2001:db8:0:1234:0:567:1:1 (for IPv6)
```

IPv4 addresses

An IPv4 address enables data to be routed to the chosen destination. Each destination in your network, as well as any other TCP/IP network you have access to, can be uniquely identified by its assigned address TCP/IP protocols rely upon a strict system of addressing in order to reach a host in a network. IPv4 addresses are represented in dotted-decimal format. The 32-bit address is divided along 8-bit boundaries. Each set of 8 bits is converted to its decimal equivalent and separated by periods.

An IPv4 TCP/IP address is a 32-bit number written in dotted decimal notation. This scheme is numeric and consists of four groups separated by a period (.). For example, 9.12.1.43 represents a single host on a single network. 193.5.86.9 represents another host on another network.

Subnet mask

Use the subnet_mask to completely define the route. Multiple network routes having an identical destination IP address and address mask can be specified.

Network and host addresses

Each host must have a unique Internet address to communicate with other hosts on the Internet. The network address part of the IP address is centrally administered by the Internet Network Information Center (the InterNIC) and is unique throughout the Internet. Each IP address is made up of two logical addresses:

```
IP address = <network address> <host address>
```

Where:

Network address Represents a specific physical network within the Internet.

Host address Specifies an individual host within the physical network identified by the network address.

For example, 9.12.1.43 is an IP address with 9.12 being the network address and 1.43 being the host address.

IP address classes

In the original Internet routing scheme developed in the 1970s, sites were assigned addresses from one of three classes: Class A, Class B, and Class C. The address classes differ in size and number. Class A addresses are the largest, but there are few of them. Class Cs are the smallest, but they are numerous. Classes D and E are also defined, but not used in normal operation.

- Class A - 0nnnnnnn hhhhhhhh hhhhhhhh hhhhhhhh
  First bit 0; 7 network bits; 24 host bits
  Initial byte: 0 - 127
  126 Class As exist (0 and 127 are reserved)
  16,777,214 hosts on each Class A
- Class B - 10nnnnnn nnnnnnnn hhhhhhhh hhhhhhhh
  - First two bits 10; 14 network bits; 16 host bits
  - Initial byte: 128 - 191
  - 16,384 Class Bs exist
  - 65,532 hosts on each Class B
- Class C - 110nnnnn nnnnnnnn nnnnnnnn hhhhhhhh
  - First three bits 110; 21 network bits; 8 host bits
  - Initial byte: 192 - 223
  - 2,097,152 Class Cs exist
  - 254 hosts on each Class C

10.4 IPv6 addresses

- 128-bit format
  - X:X:X:X:X:X:X:X
  - Examples: Each X representing 16 bits (double IPv4 address) written in hexadecimal as follows:
    - **FEDC:BA98:7654:3210:FEDC:BA98:7654:3210**
    - **1080:0:0:0:8:800:200C:417A**
- If you actually try to calculate the amount of addresses that 128 bits could generate, the following is the number:
  - 340 billion billion billion billion (340 x 10 to the power of 38)

Figure 10-4 IPv6 addresses

IPv6 addresses

IPv6 addresses are 128 bits divided along 16-bit boundaries. Each 16-bit block is converted to a 4-digit hexadecimal number and separated by colons. The resulting representation is called colon-hexadecimal. The major drawback of this addressing system is that, for most people, numbers are difficult to remember.

IPv6 TCP/IP address

An IPv6 TCP/IP address is a 128-bit number written in colon hexadecimal notation. This scheme is hexadecimal and consists of eight 16-bit pieces of the address. For example, x:x:x:x:x:x:x:x represents a single host on a single network. Alternate notations described in RFC 2373 are acceptable. For example:

```
FEDC:BA98:7654:3210:FEDC:BA98:7654:321 or ::1.
```

Note: Due to some methods of allocating certain styles of IPv6 addresses, it will be common for addresses to contain long strings of zero bits. In order to make writing addresses containing zero bits easier, a special syntax is available to compress the zeros. The use of "::" indicates multiple groups of 16 bits of zeros. The "::" can only appear once in an address. It can also be used to compress the leading and/or trailing zeros in an address.

10.5 User login to the z/OS UNIX shell

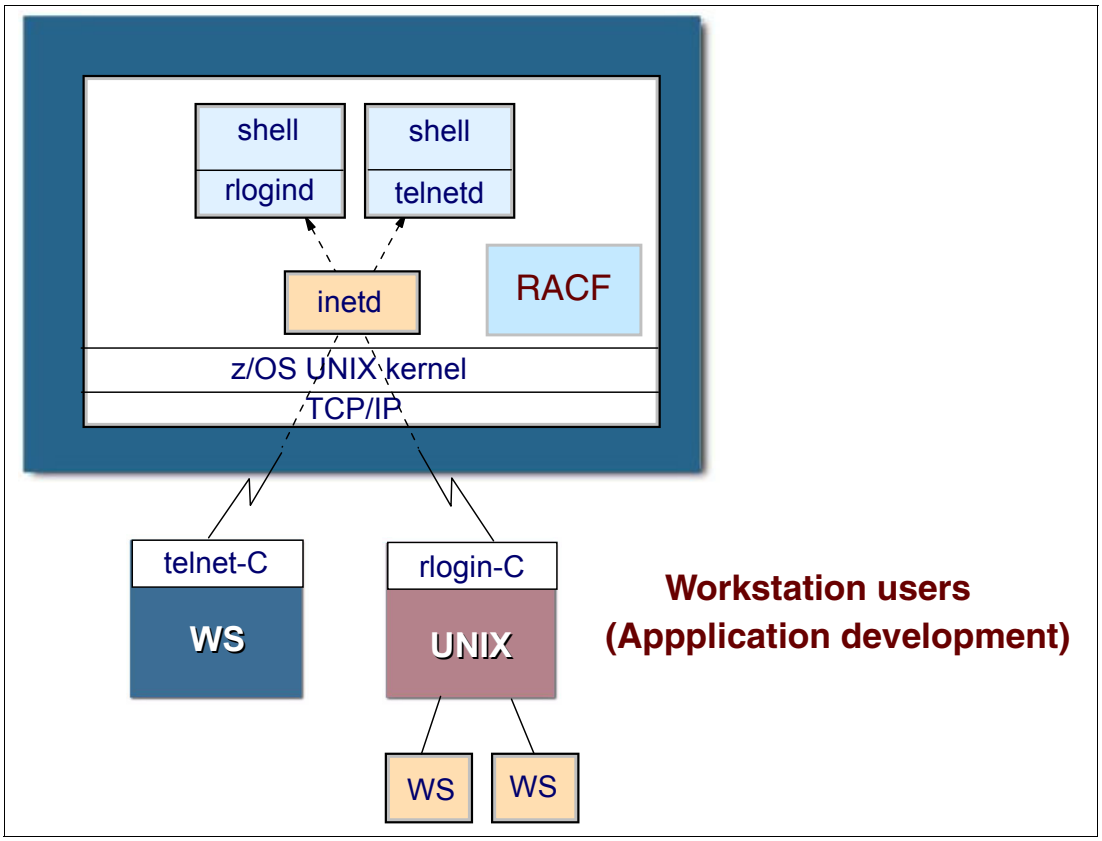


Figure 10-5 Workstation connections to the z/OS UNIX shell

Access to the shell session from workstations

TCP/IP is the transport provider when users rlogin or telnet from a UNIX workstation directly into the z/OS shell.

Users can rlogin or use telnet to log on to a z/OS UNIX system from a remote system. Two daemons are used when processing rlogin requests:

- The inetd daemon handles rlogin requests.
- The rlogind and telnetd daemons are the servers that validate the remote login requests and check the password.

The inetd daemon provides service management for a network. For example, it starts the rlogind program whenever there is a remote login request from a workstation.

The rlogind program

The rlogind program is the server for the remote login command **`rlogin`** commonly found on UNIX systems. It validates the remote login request and verifies the password or password phrase of the target user. It starts a z/OS shell for the user and handles translation between ASCII and EBCDIC code pages as data flows between the workstation and the shell.

The inetd program

When inetd is running and receives a request for a connection, it processes that request for the program associated with that socket. For example, if a user tries to log in from a remote

system into the z/OS shell while inetd is running, inetd processes the request for connection and then issues a fork() and execl() to the rlogin program to process the rlogin request. It then goes back to monitoring for further requests for those applications that can be found as defined in the `/etc/inetd.conf` file.

Security checks for login requesters

The system provides security by verifying a user and verifying that a user or program can access a process or file. It verifies the user IDs and passwords of users when they log on to a TSO/E session or when a job starts. The rlogin user is authenticated by the rlogind daemon before entering the shell. The telnet user is authenticated by the telnetd daemon before entering the shell. The daemons create a process for the user and RACF verifies that the user is properly defined before the system initializes the process.

10.6 Resolver address space

- Routines that act as a client on behalf of applications
  - For DNS - name-to-IP or IP-to-name resolution
- With z/OS UNIX, Resolver has configuration data sets
  - resolv.conf
  - TCPIP.DATA
- API environments
  - Native TCP/IP sockets
  - UNIX System Services callable sockets
    - (BPX1xxxx calls)
  - Language Environment C/C++ sockets

Figure 10-6 Resolver address space support

Resolver overview

A resolver is a set of routines that act as a client on behalf of an application to read a local host file or to access one or more Domain Name Systems (DNSs) for name-to-IP address or IP address-to-name resolution. The resolver function allows applications to use names instead of IP addresses to connect to other partners.

Where the resolver looks for name-address resolution and how it handles requests, is defined in a file called the resolver configuration file. The resolver can exist on any system that supports sockets programs, using name resolution.

Resolver configuration data sets

On a z/OS system, the resolver configuration data is located either in a z/OS data set (TCPIP.DATA) or in a file (`resolv.conf`) in the z/OS UNIX file system. The TCPIP.DATA data set is also known as one of the resolver configuration data sets. In fact, this name is now more commonly used to refer to this important file in the z/OS UNIX System Services environment because the socket library contains a component called the resolver. In a UNIX system, you use the `/etc/resolv.conf` file for the same purpose as you use TCPIP.DATA in an z/OS system.

TCPIP.DATA specifies the name of the TCP/IP address space. Because the data set search order can vary, your installation will determine which data set you can use.

API environments

To initiate a request to the resolver in z/OS, an application executes a set of commands based on the Sockets API Library the application used to generate the socket to connect to the TCP/IP stack. In a z/OS system, this task is more complex, because the applications can be built using one of three main groups of sockets API environments:

- Native TCP/IP sockets
- z/OS UNIX System Services callable sockets (BPX1xxxx calls)
- Language Environment C/C++ sockets

The various resolver libraries supported by the TCP/IP and LE APIs were consolidated into a single resolver component. This allowed for consistent name resolution processing across all applications using the TCP/IP and LE socket APIs. The consolidated resolver is automatically enabled on z/OS and runs in a separate address space that is automatically started during z/OS UNIX System Services initialization, as shown in the following messages during an IPL:

```
BPXF203I DOMAIN AF_UNIX WAS SUCCESSFULLY ACTIVATED.
BPXF203I DOMAIN AF_INET WAS SUCCESSFULLY ACTIVATED.
BPXF224I THE RESOLVER_PROC, RESOLVER, IS BEING STARTED.
IEF196I         1 //BPXOINIT JOB MSGLEVEL=1
IEF196I         2 //STARTING EXEC BPXOINIT
EZZ9298I DEFAULTTCPIPDATA - None
EZZ9298I GLOBALTCPIPDATA - None
EZZ9298I DEFAULTIPNODES - None
EZZ9298I GLOBALIPNODES - None
EZZ9304I NOCOMMONSEARCH
EZZ9291I RESOLVER INITIALIZATION COMPLETE
```

10.7 TCPDATA search order

- Common search order for IP address/hostname resolution
  - Parmlib statement (BPXPRMxx):
    - RESOLVER_PROC(*procname*) or
    - RESOLVER_PROC(NONE) or
    - RESOLVER_PROC(DEFAULT)
  - LFS starts *procname* if INET domain defined in BPXPRMxx
  - TCP/IP supplies the:
    - Default procedure
    - Search order:

1. GLOBALTCPIPDATA
2. RESOLVER_CONFIG environment variable
3. /etc/resolv.conf
4. SYSTCPD DD-name
5. userid.TCPIP.DATA
6. jobname.TCPIP.DATA
7. SYS1.TCPPARMS(TCPDATA)
8. DEFAULTTCPIPDATA
9. TCPIP.TCPIP.DATA

Figure 10-7 Search order for TCPDATA using the resolver address space

Base resolver configuration file

The resolver acts on behalf of programs as a client that accesses name servers for name-to-address or address-to-name resolution. The resolver can also be used to provide protocol and services information. To resolve the query for the requesting program, the resolver can access available name servers, use local definitions (for example, /etc/resolv.conf, /etc/hosts, /etc/ipnodes, HOSTS.SITEINFO, HOSTS.ADDRINFO, or ETC.IPNODES), or use a combination of both. How and whether the resolver uses name servers is controlled by TCPIP.DATA statements (resolver directives). The resolver address space must be started before any application or TCP/IP stack resolver calls can occur.

The base resolver configuration file contains TCPIP.DATA statements. In addition to resolver directives, it is referenced to determine, among other things, the data set prefix (DATASETPREFIX statement's value) to be used when trying to access some of the configuration files specified in this section. TCPIP.DATA is used during configuration of both the TCP/IP stack and applications; the search order to find the TCPIP.DATA data set is the same for both the TCP/IP stack and applications.

RESOLVER_PROC

z/OS UNIX initialization will attempt to start the resolver unless explicitly instructed not to. Using z/OS UNIX is the recommended method since it will ensure that the resolver is available before any applications can make a resolution request.

A BPXPRMxx statement, RESOLVER_PROC, is used to specify the procedure name, if any, to be used to start the resolver address space. If the RESOLVER_PROC statement is not in the BPXPRMxx PARMLIB member or is specified with a procedure name of DEFAULT, z/OS UNIX will start a resolver address space with the assigned name of RESOLVER. The resolver will use the applicable search order for finding TCPIP.DATA statements, but without a GLOBALTCPIPDATA specification. If the address space cannot be started, z/OS UNIX initialization continues.

When z/OS UNIX starts the resolver, it is started so that the resolver does not require JES (that is, SUB=MSTR is used).

BPXPRMxx PARMLIB member

We recommend that you update the BPXPRMxx PARMLIB member and use this to start the resolver address space. Using the BPXPRMxx member and OMVS initialization to start the resolver helps ensure that the resolver API is available before any /etc/rc or COMMNDxx PARMLIB members are executed. The RESOLVER_PROC statement in the BPXPRMxx PARMLIB member is used to start the resolver during z/OS UNIX System Services initialization. The statement specifies how the resolver address space is processed during z/OS UNIX initialization.

```
RESOLVER_PROC(procname|DEFAULT|NONE)
```

procname The name of the address space for the resolver and the procedure member name in SYS1.PROCLIB.

DEFAULT Causes an address space named RESOLVER to start. This also happens if the RESOLVER_PROC statement is not specified in the BPXPRMxx.

NONE Specifies that no address space is to be started.

Ways to start the resolver

There are two ways to start the resolver address space:

- The resolver starts when z/OS UNIX is initialized, unless there are specific instructions that it should not be started. This method of starting the resolver is best because it ensures that the resolver is available before any applications can make a resolution request.

  Use the BPXPRMxx statement, RESOLVER_PROC, to specify the procedure name, if any, to be used to start the resolver address space.

  If the RESOLVER_PROC statement is not in the BPXPRMxx parmlib member or is specified with a procedure name of DEFAULT, z/OS UNIX starts a resolver address space using the system default procedure IEESYSAS with the assigned name RESOLVER. The resolver uses the applicable search order for finding TCPIP.DATA statements, but without a GLOBALTCPIPDATA or a DEFAULTTCPIPDATA specification.

  If the RESOLVER_PROC statement has been used to specify a start procedure name, then the following are true:

  For the procedure to be found, it must reside in a data set that is specified by the MSTJCLxx parmlib member's IEFPDSI DD card specification. The procedure must not contain any DD cards that specify SYSOUT=*. When z/OS UNIX starts the resolver, it is started so that the resolver does not require JES (that is, SUB=MSTR is used).

10.8 Create configuration files used by TCP/IP

- PROFILE.TCPIP - used only for the stack
  - System operation and configuration parameters
  - A sample data set, hlq.SEZAINST(SAMPPROF), can be copied and modified for use as your default configuration
  - PROFILE data set contains:
    - TCP/IP operating characteristics
    - TCP/IP physical characteristics
    - TCP/IP reserved port number definitions
    - TCP/IP network routing definitions
    - TCP/IP diagnostic data statements
- TCPIP.DATA - used by stack and applications
  - Configuration information used by TCP/IP clients

Figure 10-8 Configuration files used by TCP/IP initialization

TCP/IP configuration files

Two configuration files are used by the TCP/IP stack: `PROFILE.TCPIP` and `TCPIP.DATA`. `PROFILE.TCPIP` is used only for the configuration of the TCP/IP stack. `TCPIP.DATA` is used during configuration of both the TCP/IP stack and applications; the search order used to find `TCPIP.DATA` is the same for both the TCP/IP stack and applications.

PROFILE.TCPIP

During TCP/IP address space initialization, of the TCP/IP stack, also referred to as the TCP/IP address space, a configuration profile data set (`PROFILE.TCPIP`) is read that contains system operation and configuration parameters.

A sample data set, SEZAINST(SAMPPROF), can be copied and modified for use as your default configuration profile.

The PROFILE data set contains the following groups of configuration parameters:

- TCP/IP operating characteristics
- TCP/IP physical characteristics
- TCP/IP reserved port number definitions (application configuration)
- TCP/IP network routing definitions
- TCP/IP diagnostic data statements

TCP/IP reads the parameters from the TCP/IP profile data set.

The parameters that need to be changed are:

AUTOLOG Uncomment FTPD or any other daemons that need to be activated.

DEVICE Provide the z/OS address network interface. It could be the OSA address, CTC, IBM 2216 router, or any other supported network device.

LINK Provide the description of the network interface.

HOME Specify the IP address of the z/OS system.

Begin route Specify the IP address of the net and subnet to which this host belongs.

Route default Specify the default gateway IP address. Usually, this is the IP address of the network router to which this host is attached.

TCPIP.DATA

This file is used during configuration of both the TCP/IP stack and applications. It is used to specify configuration information required by TCP/IP client programs.

`TCPIP.DATA` is used by the stack address space as follows:

- The TCP/IP stack's configuration component uses `TCPIP.DATA` during TCP/IP stack initialization to determine the stack's HOSTNAME. To get its value, the z/OS UNIX environment search order is used.
- The TCP/IP stack's TN3270 Telnet server component uses `TCPIP.DATA` statements to resolve a client's IP address to a name. To obtain the resolver-related statements for address resolution, the native MVS environment search order is used.

10.9 Customize the TCP/IP profile data set

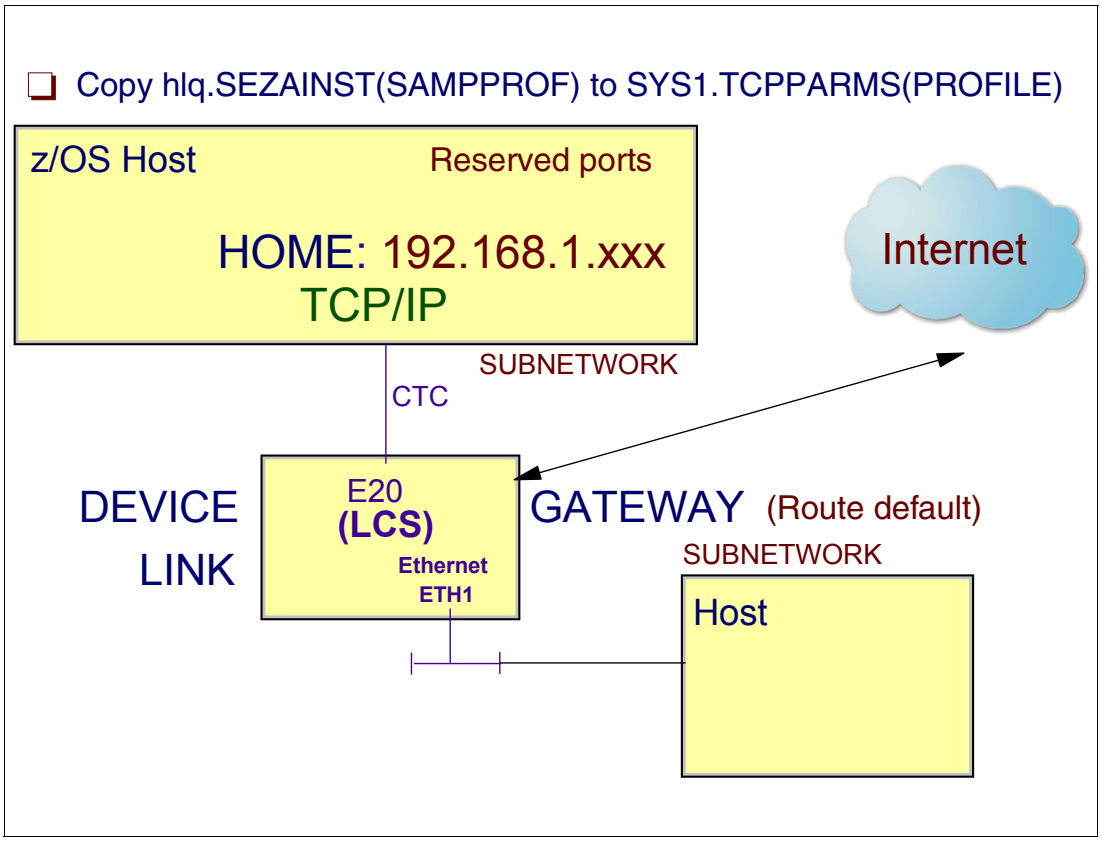


Figure 10-9 Customizing the TCP/IP profile data set

Customize the TCP/IP profile data set

This is a list of PROFILE statements that need to be customized to define your environment. The IP address of the HOME statement for this host, as well as the GATEWAY value such as subnet mask, subnet and DEFAULTNET (or default gateway), can be obtained from your network administrator.

A sample of the PROFILE data sets is provided in hlq.SEZAINST(SAMPPROF), which you can copy to SYS1.TCPPARMS(PROFILE).

TCP/IP reads the parameters from the TCP/IP profile data set.

The parameters that need to be changed are:

AUTOLOG Uncomment FTPD or any other daemons that need to be activated.

DEVICE Provide the z/OS address network interface. It could be the OSA address, CTC, the IBM 2216 router, or any other supported network device.

LINK Provide the description of the network interface.

HOME Specify the IP address of the z/OS system.

Begin route Specify the IP address of the net and subnet to which this host belongs.

Route default Specify the default gateway IP address. Usually, this is the IP address of the network router to which this host is attached.

STEPLIB environment variable

Traditionally, some MVS users have preferred to alter the search order for MVS executable files when they are running a new or test version of an application program, such as a run-time library. To do this, they code a STEPLIB DD statement on the JCL used to run the application. Accessed ahead of LINKLIB or LPALIB, a STEPLIB is a set of private libraries where the new or test version of the application is stored.

The STEPLIB environment variable provides the ability to use a STEPLIB when running a z/OS UNIX executable file. This variable is used to determine how to set up the STEPLIB environment for an executable file. The STEPLIB environment variable should always be exported.

STEPLIB data set restrictions

For executable files that have the set-user-ID or set-group-ID bit set, there are restrictions on the data sets that can be built into the STEPLIB environment for the file to run in. The system programmer maintains a STEPLIB sanction list of data sets that can be included in the STEPLIB environment for such executable files. Only data sets on that list are built into the STEPLIB environment for such files. If you need a data set added to the list, contact your system programmer.

10.10 Customize TCPDATA

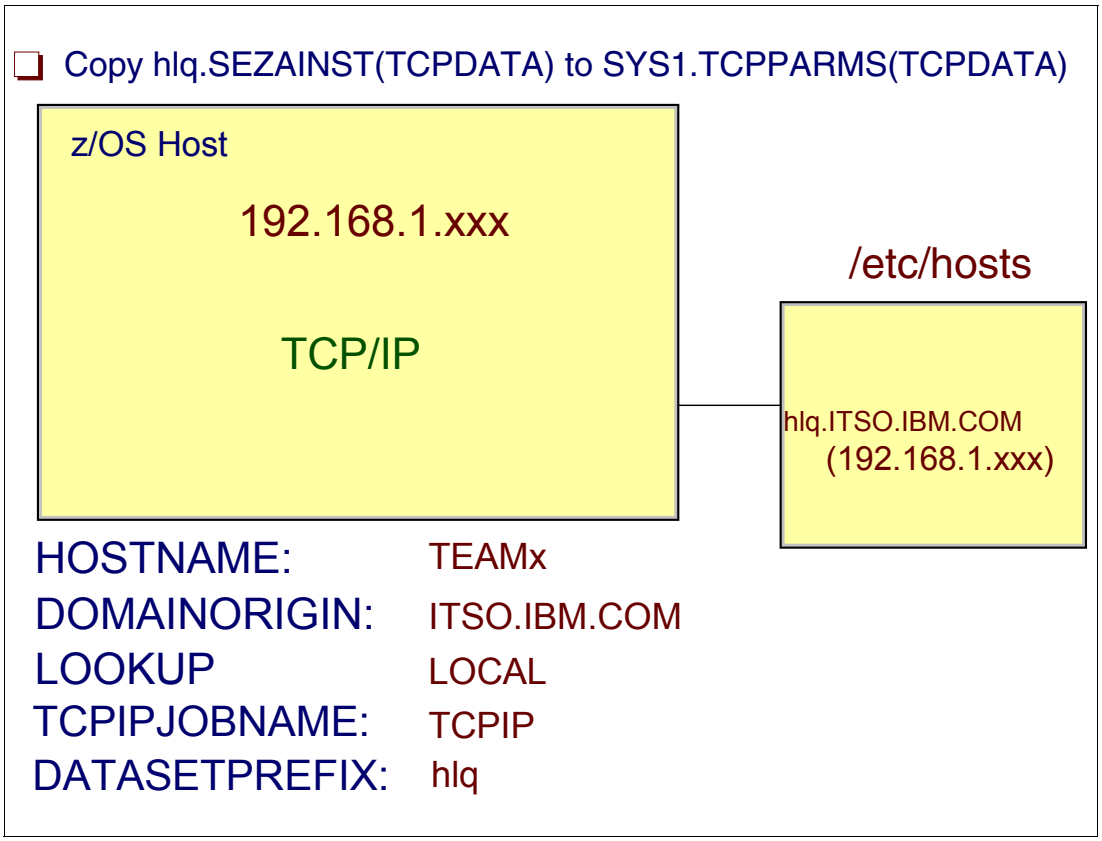


Figure 10-10 Customizing the TCPDATA data set

Customizing the TCPDATA data set

Parameters that need to be customized for TCPDATA are shown in Figure 10-10. Create a `TCPIP.DATA` file by copying the sample provided in SEZAINST(TCPDATA) and modifying it to suit local conditions.

The SYSTCPD DD explicitly identifies which data set is to be used to obtain the parameters defined by `TCPIP.DATA`. The SYSTCPD DD statement should be placed in the TSO/E logon procedure or in the JCL of any client or server executed as a background task. The data set can be any sequential data set or a member of a partitioned data set (PDS).

```
//SYSTCPD  DD DSN=SYS1.TCPPARMS(TCPDATA),DISP=SHR
```

Parameters that need to be changed for the `TCPDATA` file are:

- TCPIPJOBNAME specifies the TCPIP started task jobname.
- HOSTNAME specifies the hostname (SYSNAME on IEASYSxx) or IEASYMxx.
- DATASETPREFIX specifies the hlq you have selected before.

For the DOMAINORIGIN and NSINTERADDR statements, the values can be obtained from the network administrator. Optional parameters shown are:

- DOMAINORIGIN specifies your domain (ITSO.IBM.COM).
- LOOKUP specifies to search either local host files or the domain name server.

Note: The `TCPIP.DATA` data set is the only one of the TCP/IP data sets for which a unique copy is needed for each transport provider. This is because the TCPIPJOBNAME statement identifies the TCP/IP address space and the HOSTNAME statement identifies the host name of the TCP/IP address space.

The following example shows a typical TCP/IP syntax:

```
Datasetprefix      TCPIP          ; This stack's data set prefix
TCPIPjobname       TCPIP          ; Stack name
NSinterAddr        127.0.0.1      ; Name server (on this system)
NSportAddr         53             ; Name server port number
ResolveVia         UDP            ; Use UDP for Name server
ResolverTimeout    30             ; 30-second name server timeout;
ResolverUdpRetries 1              ; Retry name server once
HostName           TEAMx          ; host name
DomainOrigin       ITSO.ibm.com   ; My domain origin
Messagecase        mixed          ; Issue mixed-case messages
```

The system processes this information during the initial request for service. It accepts either format for the information supplied regardless of the source selected. When they are mixed, only the last domain or DOMAINORIGIN data is used and up to 16 name server addresses are used for initialization. However, if you set up `/etc/resolv.conf` to supply resolver information, you must specify the DATASETPREFIX information in `/etc/resolv.conf` unless you have also set up `/etc/services`, `/etc/protocol`, and `/etc/hosts` files.

TCPDATA considerations

Any mix of MVS data sets and z/OS UNIX files can be used. For example, you could use the `TCPIP.DATA` information from SYS1.TCPPARMS, the service and protocol information from ETC.SERVICE and ETC.PROTO and use the `/etc` directory in the z/OS UNIX file system to record host names and addresses of the z/OS UNIX hosts file.

You can use any name for the `TCPIP.DATA` data set if you access it using the //SYSTCPD DD statement, or use ENVAR to set RESOLVER_CONFIG in the JCL for all the servers, logon procedures, and batch jobs that execute TCP/IP functions. If you are not using the //SYSTCPD DD statement, the environment variable, or `/etc/resolv.conf`, then the data set name must conform to the conventions described in Configuration files for the TCP/IP stack. Another alternative is to use the well-known data set name SYS1.TCPPARMS(TCPDATA). You will eventually issue the HOMETEST command with TRACE RESOLVER activated to verify the actual data set name the system finds for `TCPIP.DATA`. However, because HOMETEST is an MVS sockets application, it does not use RESOLVER_CONFIG or `/etc/resolv.conf` in its search order. For this reason, it is recommended that `/etc/resolv.conf` and TCPIP.DATA contain exactly the same information or consider using the resolver GLOBALTCPIPDATA setup statement.

Rule: Since `TCPIP.DATA` statements might need to be read and used multiple times by the resolver, the FREE=CLOSE JCL parameter should not be used when allocating SYSTCPD. To allow TCPIP.DATA statements to be changed while still allocated for long-running programs, consider using a member of an MVS partitioned data set instead of an MVS sequential data set. For these long-running applications, the resolver MODIFY REFRESH command should then be used to indicate that `TCPIP.DATA` statements have been changed.

10.11 z/OS IP search order

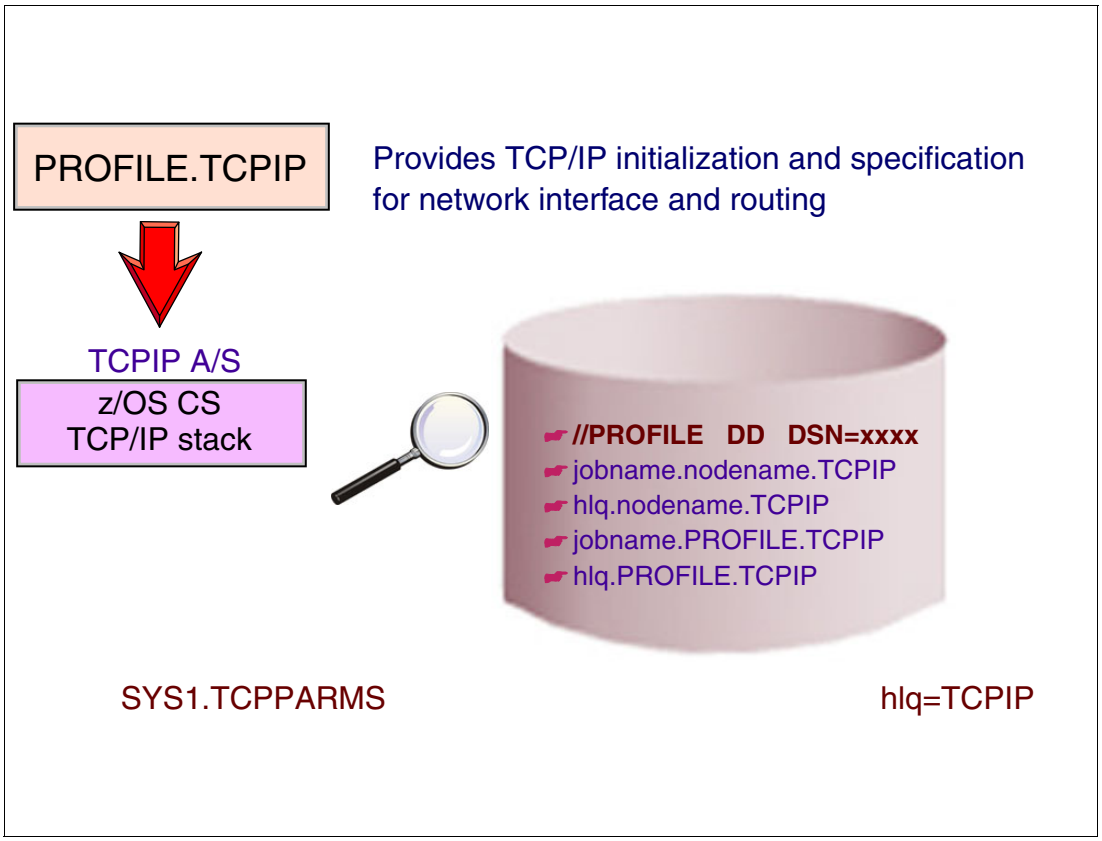


Figure 10-11 Search order used during TCP/IP initialization

TCP/IP initialization search order

During initialization of the TCP/IP stack, system operation and configuration parameters for the TCP/IP stack are read from the configuration file PROFILE.TCPIP. The search order used by the TCP/IP stack to find PROFILE.TCPIP involves both explicit and dynamic data set allocation as identified in the following discussion.

Search order

In the TCP/IP procedure in SYS1.PROCLIB, the PROFILE DD statement specifies the data set containing the TCP/IP configuration parameters. If the PROFILE DD statement is not supplied, a default search order is used to find the PROFILE data set. A sample profile is included in member SAMPPROF of the SEZAINST data set. When TCP/IP starts, it looks for the PROFILE data set in the following order:

- //PROFILE DD explicitly specified in the PROFILE DD statement of the TCP/IP started task procedure.
- jobname.nodename.TCPIP data set
- hlq.nodename.TCPIP data set
- jobname.PROFILE.TCPIP data set
- hlq.PROFILE.TCPIP data set

The search stops if one of these data sets is found.

Most PROFILE parameters required in a basic configuration have default values that will allow the stack to be initialized and ready for operation. There are, however, a few parameters that must be modified or must be unique to the stack.

Note: Explicitly specifying the PROFILE DD statement in the TCPIP PROC JCL is the recommended way to specify PROFILE.TCPIP. If this DD statement is present, the data set it defines is explicitly allocated by MVS and no dynamic allocation is done. If this statement is not present, the search order continues to use dynamic allocation for the PROFILE.TCPIP.

10.12 z/OS IP search order (2)

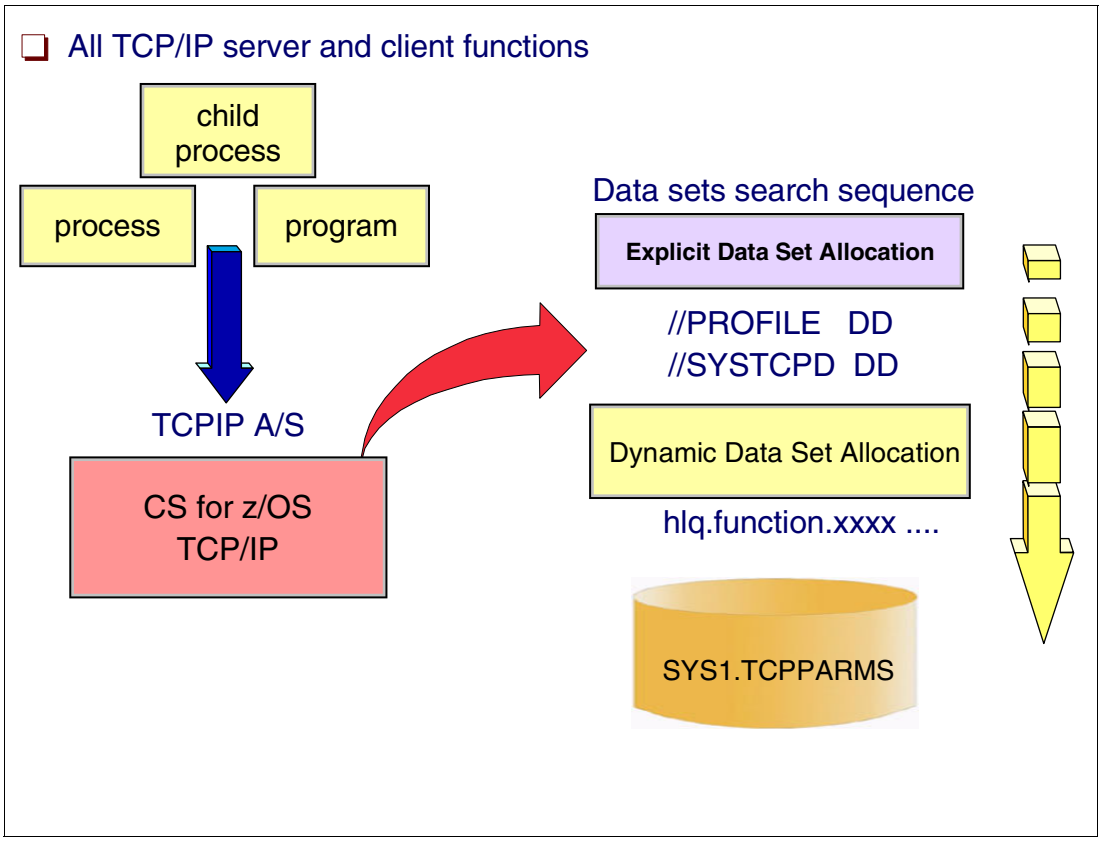


Figure 10-12 Search order for z/OS UNIX applications

z/OS UNIX application search order

The CS for z/OS environment consists of the CS for z/OS stack, CS for z/OS applications (z/OS UNIX System Services applications such as Telnetd, FTPD, and so forth), and the z/OS TCP/IP native MVS applications.

The TCP/IP stack and set of applications have some common configuration files, but they also use configuration files that are different.

Different configuration files may be used for a TCP/IP stack where there is a need to understand the search order for each z/OS UNIX application.

The search order is applied to any configuration file, and the search ends with the first file found, as follows:

- **Explicit Data Set Allocation** consists of those data sets that you specify through the use of DD statements in JCL procedures.
- **Dynamic Data Set Allocation** consists of multiple versions of a data set, each having a different high-level qualifier or middle-level qualifier, and some data sets that can only be dynamically allocated by TCP/IP (they cannot be allocated using DD statements in JCL).

There is a naming convention for dynamically allocated data sets.

10.13 Customize the TCP/IP procedure

- Update SYS1.PROCLIB
  - Create TCP/IP started task procedure
    - Look in TCPIP.SEZAINST(TCPIPROC)
  - Create EZAZSSI
    - Look in TCPIP.SEZAINST(EZAZSSI)
- Update TSO logon procedure - SYSTCPD statement
  - Place in the TSO/E logon procedure or in the JCL of any client or server executed as a background task

Figure 10-13 Customizing the TCP/IP procedure

TCP/IP procedure

Create the PROCLIB member, as follows:

- TCPIP started task procedure - a sample is provided in hlq.SEZAINST(TCPIPROC)
- EZAZSSI procedure to start TCP Subsystem Interface

Add procedure EZAZSSI to your system PROCLIB. A sample of this procedure is located in the data set hlq.SEZAINST (where hlq is the high-level qualifier for the TCP/IP product data sets in your installation).

```
//EZAZSSI    PROC   P=''
//STARTVT    EXEC   PGM=EZAZSSI,PARM=&P
//STEPLIB    DD     DSN=hlq.SEZALINK,DISP=SHR
//           DD     DSN=hlq.SEZATCP,DISP=SHR
```

You can remove the STEPLIB DD if these data sets are defined on LNKLSTxx, as follows:

- Modify your TSO/E logon procedure:
  - To include hlq.SEZAHELP in //SYSHELP DD
  - To include hlq.SEZAMENU in //ISPMLIB DD
  - To include hlq.SEZAPENU in //ISPTLIB DD and //ISPPLIB DD

– Optionally, add a //SYSTCPD DD to point to the TCPDATA data set in order to use TCP/IP client functions and some administrative functions such as OBEYFILE under TSO/E.

  SYSTCPD explicitly identifies the data set used to obtain parameters defined by TCPIP.DATA.

SYSTCPD statement

The SYSTCPD statement should be placed in the TSO/E logon procedure or in the JCL of any client or server executed as a background task. The data set can be any sequential data set or a member of a partitioned data set (PDS). TSO client functions can be directed against any of a number of TCP/IP stacks. Obviously, the client function must be able to find the TCPIP.DATA appropriate to the stack of interest at any one time. Two methods are available for finding the relevant TCPIP.DATA:

- Add a SYSTCPD DD statement to your TSO logon procedure. The issue with this approach is that a separate TSO logon procedure per stack is required, and users have to log off TSO and log on again using another TSO logon procedure in order to switch from one stack to another.
- Use one common TSO logon procedure without a SYSTCPD DD statement. Before a TSO user starts any TCP/IP client programs, the user has to issue a TSO ALLOC command wherein the user allocates a TCPIP.DATA data set to DDname SYSTCPD. To switch from one stack to another, the user simply has to de-allocate the current SYSTCPD allocation and allocate another TCPIP.DATA data set.

Combine the first and second methods. Use one logon procedure to specify a SYSTCPD DD for a default stack. To switch stacks, issue TSO ALLOC to allocate a new SYSTCPD. To switch back, issue TSO ALLOC again with the name that was on the SYSTCPD DD in the logon procedure. The disadvantage to this approach is that the name that was on the SYSTCPD DD is “hidden” in the logon procedure and needs to be retrieved or remembered.

10.14 Customizing PARMLIB members for TCP/IP

- Choose a High-Level Qualifier for TCP data set
  - TCPIP
- Update SYS1.PARMLIB members
  - IEAAPFxx or PROGxx to authorize TCP/IP data set
  - LNKLSTxx - TCPIP.SEZALOAD
  - LPALSTxx - TCPIP.SEZALPA
  - IECIOSxx - Set MIH TIME=00:00,DEV=(cuu-cuu)
  - IEFSSNxx
    - TNF - VMCF

Figure 10-14 PARMLIB members to customize for TCP/IP

PARMLIB members to customize

Steps for customizing TCP/IP are as follows:

- Choose a High-Level Qualifier (hlq).
- TCP/IP uses dynamic allocation with this hlq to get parameters from several TCP/IP data sets. The DATASETPREFIX statement in TCPIP.DATA can be used to override the default hlq. However, it is used as the last step in the search order for most configuration files.
- Update the following PARMLIB members:

  IEAAPFxx Authorizes the following libraries:

  `hlq.SEZADSIL`
  `hlq.SEZALOAD - hlq.SEZALNK2`
  `hlq.SEZALPA - hlq.SEZAMIG`

  PROGxx `LNKLST Add Name(LNKLST)`
  `Dsname(TCPIP.SEZALOAD)`

  LNKLSTxx hlq.SEZALOAD

  LPALSTxx hlq.SEZALPA

  IECIOSxx Use to disable the MIH processing for communication device used by TCP/IP

  `Set MIH TIME=00:00,DEV=(cuu-cuu)`

  IEFSSNxx Define subsystems to use restartable VMCF and TNF

  `SUBSYS SUBNAME(TNF)`
  `SUBSYS SUBNAME(VMCF)`

10.15 PARMLIB members to customize for TCP/IP

- Update SYS1.PARMLIB members
  - COMMNDxx
    - Start TCP/IP
  - IFAPRDxx
    - Enable TCP/IP
  - BPXPRMxx
    - Define PFS

Figure 10-15 PARMLIB members to customize for TCP/IP

Updating SYS1.PARMLIB

Update the following PARMLIB members:

COMMNDxx Add the following command to start the SSI

```
COM='S EZAZSSI,P=sys_name'
```

Where sys_name is the SYSNAME in IEASYSxx or specified in IEASYMxx using the SYSDEF statement.

IFAPRDxx Enable TCP/IP base by adding this:

```
NAME(z/OS) ID(5647-A01)
```

BPXPRMxx Activate TCP/IP support for transport provider as follows:

```
FILESYTYPE TYPE(INET) ENTRYPOINT(EZBPFINI)
    NETWORK DOMAINNAME(AF_INET)
        DOMAINNUMBER(2)
        MAXSOCKETSR(10000)
        TYPE(INET)
```

10.16 RACF customization for TCP/IP

- Define RACF profile for TCP/IP started task
  - Define TCPIP userids and groupid
    - TCPGRP
  - Define STARTED class profile for TCP/IP
  - Program control
    - SETROPTS WHEN(PROGRAM)
  - Use the following commands to create RACF data set profiles:

```
ADDSD 'CEE.SCEERUN' UACC(READ)
ADDSD 'SYS1.LINKLIB' UACC(READ)
ADDSD 'TCPIP.SEZALOAD' UACC(READ)
ADDSD 'TCPIP.SEZATCP' UACC(READ)
```

Figure 10-16 Customizing TCP/IP with RACF profiles

RACF profiles for TCP/IP

There are additional security concerns when you are loading programs that are considered trusted into the z/OS UNIX file system. Program control facilities in RACF and z/OS UNIX provide a mechanism for ensuring that the z/OS UNIX program loading process has the same security features that APF authorization provides in the native MVS environment.

We recommend that you enable program control in your installation. If you define the BPX.DAEMON FACILITY class profile, you must enable program control for certain z/OS Communications Server load libraries. Review the section on program control in z/OS UNIX System Services Planning to decide whether program control is appropriate for your installation.

You need to define a RACF user ID with an OMVS segment for TCPIP, PORTMAP, NFS, and FTPD. You also need to define the following data sets to program control:

- SYS1.LINKIB
- hlq.SEZALOAD

  Executable load modules for concatenation to LINKLIB
- CEE.SCEERUN (or Language Environment run-time modules)

RACF program control

In a z/OS UNIX environment, there are additional security concerns related to the HFS and the loading of programs that are considered trusted. Program control facilities in RACF and z/OS provide a mechanism for ensuring that the z/OS program loading process has the same security features that APF authorization provides in the native MVS environment.

We recommend that you enable program control in your installation. If you define the BPX.DAEMON facility class, then you must enable program control for certain Communications Server for z/OS load libraries. Review the section on program control in *z/OS UNIX System Services Planning*, GA22-7800 to understand that program control is appropriate for your installation.

When you use program control, make sure that all load modules that are loaded into an address space come from controlled libraries. If the MVS contents supervisor loads a module from a noncontrolled library, the address space becomes dirty and loses its authorization. To prevent this from happening, define all the libraries from which load modules can be loaded as program-controlled.

Note: At a minimum, this should include the TCP/IP load libraries SEZALOAD and SEZATCP, the C run-time library SCEERUN, the system library SYS1.LINKLIB, and any load libraries containing FTP security exits.

Use the following commands:

```
RDEFINE PROGRAM * ADDMEM('SYS1.LINKLIB'/'volser'/NOPADCHK) UACC(READ)
RDEFINE PROGRAM * ADDMEM('SYS1.SIEALNKE'/'volser'/NOPADCHK) UACC(READ)
RALTER PROGRAM * ADDMEM('CEE.SCEERUN'/'volser'/NOPADCHK) UACC(READ
RALTER PROGRAM * ADDMEM('TCPIP.SEZALOAD'/'volser'/NOPADCHK) UACC(READ)
RALTER PROGRAM * ADDMEM('TCPIP.SEZATCP'/'volser'/NOPADCHK) UACC(READ)
RALTER PROGRAM * ADDMEM('db2.DSNLOAD'/'volser'/NOPADCHK UACC(READ)
RALTER PROGRAM * ADDMEM('db2.DSNEXIT'/'volser'/NPPADCHK UACC(READ)
RALTER PROGRAM * ADDMEM('ftp.userexits'/'volser'/NOPADCHK UACC(READ)
```

Note: If you define load libraries as RACF program controlled, make sure you do not specify a universal access (UACC) of NONE for any of the PROGRAM class resources. Setting a UACC of NONE for a data set such as SYS1.LINKLIB could prevent you from successfully IPLing your z/OS system. For complete information on the program control facility, refer to *z/OS Security Server RACF Security Administrator's Guide*, SA22-7683-11.

10.17 Customizing TCP/IP

- /etc/services, contains the service names and port assignments of specific z/OS UNIX applications
- RPC interface enables programmers to write distributed applications using high-level RPCs rather than lower-level calls based on sockets
- Customize the socket and RPC call data sets
  - Copy the hlq.SEZAINST(SERVICES) to /etc/services
  - Copy the hlq.SEZAINST(ETCRPC) to /etc/rpc
- Start TCP/IP

Figure 10-17 HFS data sets to customize for TCP/IP

Data sets for TCP/IP

The z/OS UNIX file /etc/services contains the service names and port assignments of specific z/OS UNIX applications. The MVS data set ETC.SERVICES can also be used to contain the same information. The source for this example is shipped in SEZAINST(SERVICES) and copied to hlq.ETC.SERVICES by the Installation Verification Procedure (IVP). The source is also installed in /usr/lpp/tcpip/samples/services for use in copying it to /etc/services. It is important that /etc/services and hlq.ETC.SERVICES be kept identical so that MVS and z/OS UNIX applications use the same port assignments. The shipped file contains the most current assignments. The RPC interface enables programmers to write distributed applications using high-level RPCs rather than lower-level calls based on sockets.

SERVICES and RPC data sets are as follows:

- Copy hlq.SEZAINST(SERVICES) to hlq.ETC.SERVICES. This file specifies the combination of port and services (UPD or TCP) used by TCP/IP. To establish a relationship between the servers defined in the `/etc/inetd.conf` file and specific port numbers in the z/OS UNIX System Services environment, ensure that statements have been added to ETC.SERVICES for each of these servers. See the sample ETC.SERVICES installed in the `/usr/lpp/tcpip/samples/services` directory for how to specify ETC.SERVICE statements for these servers. An HFS file, `/etc/services`, could also be created instead of this file.
- Copy hlq.SEZAINST(ETCRPC) to hlq.ETC.RPC. This file specifies the *port mapper*, which used to be called the portmap daemon.

10.18 TCP/IP shell commands

- UNIX variations of the familiar TSO commands
  - Executed from the UNIX OMVS Shell or from the ISHELL
  - Documented in the SecureWay CS IP User's Guide
- PING - Is the node specified active
  - Allows specification of TCP/IP stack name + other options
- NSLOOKUP - Used to query a name server
  - Valuable for testing resolver files (LE vs. MVS)
- TRACERTE - Used to debug network problems
  - Can specify a source address
  - Can specify an interface name as origin
  - Can specify the TCP/IP stack name
  - Can specify type of service for testing router configurations
- NETSTAT - Display network status of local host
  - To determine stack component status from UNIX environment

Figure 10-18 Commands issued from the shell for TCP/IP

SHELL commands for TCP/IP

You can use the TSO PING, TRACERTE, NETSTAT, and NSLOOKUP commands from the UNIX environment.

The z/OS UNIX **`ping`** command sends an echo request to a foreign node (remote node) to determine whether the computer is accessible. When a response to a **`ping`** command is received, the elapsed time is displayed. The time does not include the time spent communicating between the user and the TCP/IP address space.

Use the **`ping`** command to determine the accessibility of the foreign node.

> **Note:** **`ping`** is a synonym for the **`oping`** command in the z/OS UNIX shell. The **`ping`** command syntax is the same as that for the **`oping`** command.

The z/OS UNIX **`nslookup`** command enables you to query any name server to perform the following tasks from the z/OS UNIX environment:

- Identify the location of name servers
- Examine the contents of a name server database
- Establish the accessibility of name servers

NSLOOKUP has two modes of operation: interactive mode and command mode. Interactive mode enables you to repeatedly query one or more name servers for information about

various hosts and domains and display that information on your terminal. Command mode displays the output from the query supplied as part of the command and then exits.

TSO TRACERTE command

The TSO TRACERTE command is useful for debugging various network problems. The TRACERTE command sends UDP requests with varying TTL (time-to-live) or hop count values and then waits for the routers between the local and remote hosts to send TTL-exceeded messages.

The TRACERTE command displays one line of output for every TTL or hop limit value for which it sent a UDP probe packet. The format of the output is as follows:

```
HOP NAME (IP_ADDRESS) NUM ms !FLAG
```

z/OS UNIX netstat command

Use the z/OS UNIX **netstat** command to display the network configuration and status on a local TCP/IP stack.

netstat is a synonym for the **onetstat** command in the z/OS UNIX shell. The **onetstat** command syntax is the same as that for the **netstat** command.

11

TCP/IP applications

This chapter provides information on how to customize z/OS UNIX in order to use TCP/IP. It also discusses how to enable remote UNIX logins to the z/OS UNIX System Services shell using rlogin or Telnet, as well as FTP daemon and SYSLOGD daemon.

The chapter begins by describing the remote login procedure and the socket file systems in BPXPRMxx for z/OS UNIX.

Then it provides the details of how to set up the following z/OS UNIX daemons: inetd, rlogin, Telnet, FTPD, and SYSLOGD.

Finally, it explains the search sequence for CS TCP/IP.

© Copyright IBM Corp. 2011. All rights reserved.

11.1 Overview of z/OS UNIX data access

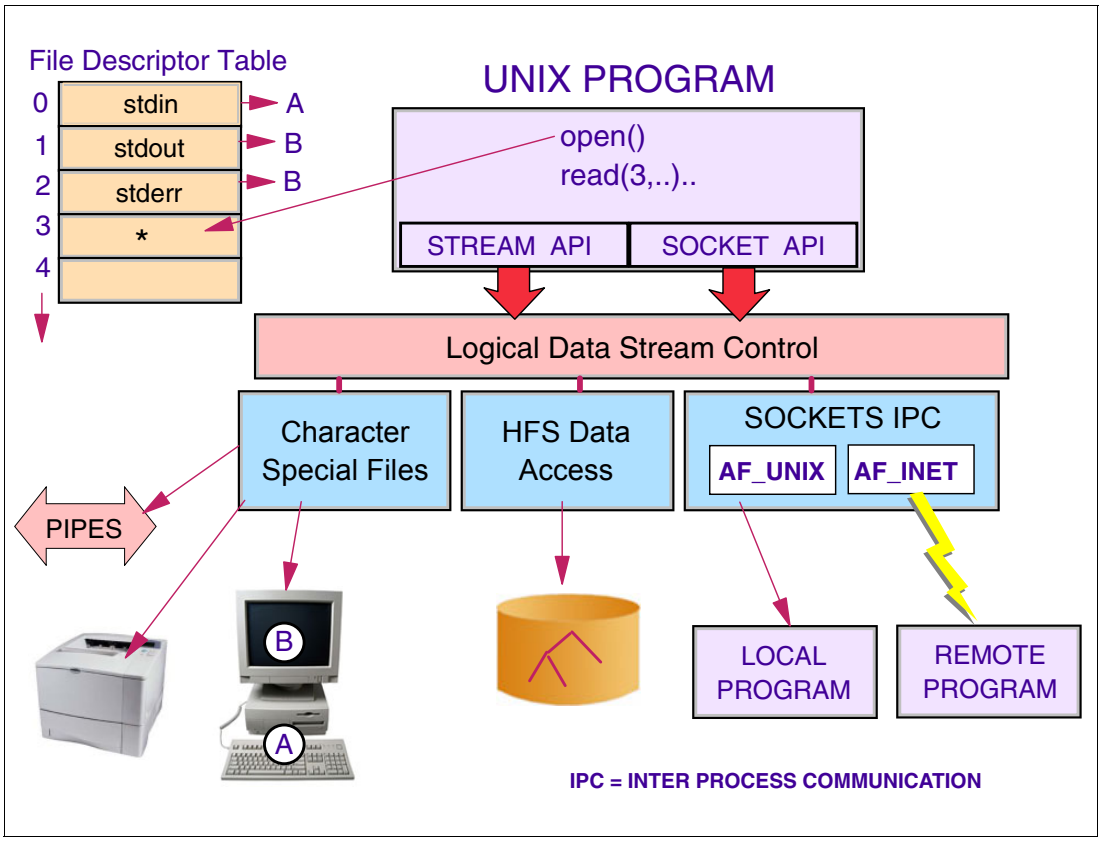


Figure 11-1 User access to z/OS UNIX overview

z/OS UNIX data set access

To do work in a UNIX system, a user requires a program, and data on which the program is to operate. The program is represented by a system process, and the data to manipulate resides in files or streams.

UNIX can use streams to represent many physical data objects:

- *Data files* stored in the UNIX hierarchical file system
- *Special character files* which represent physical devices such as printers and terminals, as well as logical devices such as data pipes
- *Sockets* or message streams used for Inter Process Communication (IPC)

File descriptors

When an application OPENs any stream to process data, the UNIX OPEN routines return a *file descriptor* that is used to identify the stream in future processing calls.

The file descriptor value indexes an entry (file handle or socket handle) stored in a logical table of file handles for files/streams opened by this program.

When an application starts, default file descriptors are automatically opened for:

- STDIN (FD = 0) - Standard Input - user terminal keyboard
- STDOUT (FD = 1) - Standard program output - to user terminal
- STDERR (FD = 2) - Standard error message output - to user terminal

z/OS UNIX sockets

Sockets come in 2 types:

- AF_UNIX sockets (IPC between programs on same host)
- AF_INET sockets (IPC with remote program over network)

Because handling sockets requires more complex functionality than simple data streams, UNIX programs typically use a different Application Programming Interface (API) (set of program calls) to work with sockets.

z/OS UNIX files

STDIN, STDOUT, and STDERR should not be new concepts. However, it is important to point out that they get their unique attributes from occupying the first three slots in this file descriptor table (FD=0, FD=1, and FD=2).

For those MVS internals gurus among you, the file descriptor table is similar in concept to the MVS TIOT (Task Input/Output table).

11.2 Sockets

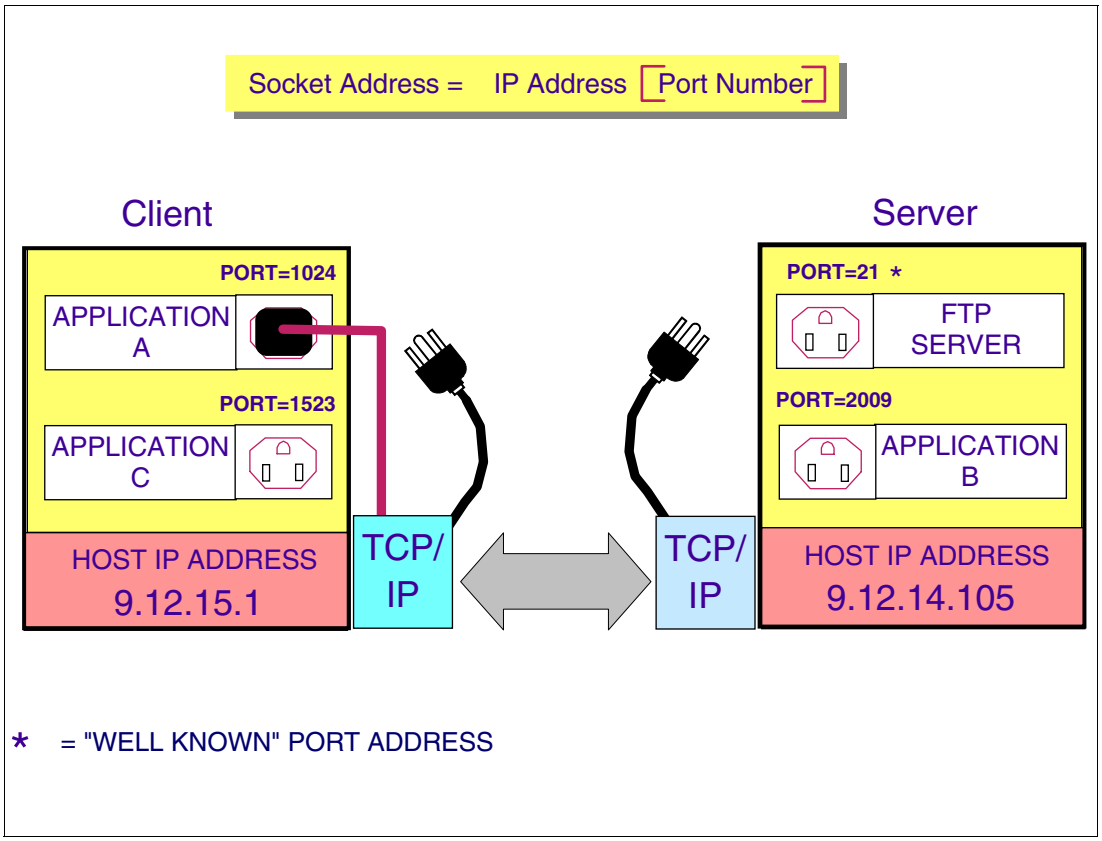


Figure 11-2 z/OS UNIX sockets overview

z/OS UNIX sockets

This should be a refresher topic. It is intended to stress that sockets (at least in the classic UNIX interpretation) are not to be treated any differently than ordinary data files insofar as program logic is concerned. Also, it should be reiterated that sockets are used for local (intrahost) as well as remote (interhost or network) communication between UNIX processes.

Simply position the concept of an API as being the set of C library service calls used to work with a particular resource. Although sockets and HFS files are both streams, the complexity of socket handling requires more powerful syntax and options for handling socket data. This is relevant in a later discussion where we see that z/OS has perhaps 8 different socket APIs!

A powerful design principle was used in UNIX data processing: any type of data manipulated by a process can be represented by a standard file/stream structure. Also, a single set of program calls is used to open streams, read and update data, and close streams, insulating the program from dealing with the physical origin of stream data.

UNIX technologies

Of course, as with many things in UNIX, this is an idealized picture which fits various commercial UNIX “flavors” to a greater or lesser extent. The currently available UNIX flavors are derived from two main UNIX technologies:

- **Berkeley Software Division (BSD)** - This is UNIX produced by the University of California. It is the strongest influence in UNIX systems produced by DEC Ultrix, SunOS,

HP-UX from Hewlett Packard, and, in the early stages, IBM OpenEdition which is now named z/OS UNIX.

- **AT&T** - This is UNIX produced by AT&T, and after that, USL UNIX system laboratories. The current UNIX version is UNIX System VR4. System VR4 is a major influence in IBM AIX, and Apple AUX.

In addition to many other differences, the two flavors of UNIX differ in the way that they see sockets, as follows:

- Berkeley was the original developer of socket protocols, which it connected with TCP/IP network protocols to provide remote process-to-process communication over the network. As a new add-in to the UNIX kernel, and because functionality was considerably more complex in BSD sockets compared to ordinary file handling, BSD sockets use a set of new program calls to manipulate data over sockets. For example, a program would open a file, read or write data to it, and close it. To initialize socket processing, on the other hand, the program issues a socket call, uses **`send`** and **`recv`** commands to transfer data, and then will close the socket.
- The AT&T System V approach is more classical. Both files and sockets are opened, data can be manipulated in both cases using read and write commands, and both are closed to terminate stream usage. Also, AT&T pioneered the idea of the *UNIX-domain* or *local socket*, used to communicate between two UNIX processes in the same system— what MVS would call cross-memory communication. A UNIX domain socket is considerably simpler than a network socket, and can be easily handled with the classic socket approach.

Other UNIX platforms

In modern UNIX systems, there has been considerable cross-pollination between the two UNIX flavors, such that, for example, AIX now supports many BSD features, including BSD sockets. Also, BSD sockets provide both local domain and Internet-type socket support. However, when porting the C source code for a new application to a UNIX system, allowances have to be made for the fact that, for example, program calls made to sockets could be written using either the BSD or the System V approach. To prevent code rewrite, system compiler and run-time support must be able to handle both flavors of calls. This principle must apply to z/OS UNIX as well. The original (4.3) approach in OpenEdition was oriented to BSD socket support rather than to System V.

In the classical (System V) approach, all open streams, whether files, terminals, sockets, and so on, are pointed to by descriptors (file handles) in the same file descriptor table. The contents of a socket descriptor will be different from that of a file descriptor in the FDT, but the index of a socket descriptor is an integer number identical to the type of index pointing to a file descriptor for a file. So, for example, a program might have an open data file with file descriptor FD=4, and an initialized socket at socket descriptor SD=5, in consecutive slots of the table. Obviously, the index numbers cannot be allowed to overlap.

11.3 z/OS Communications Server

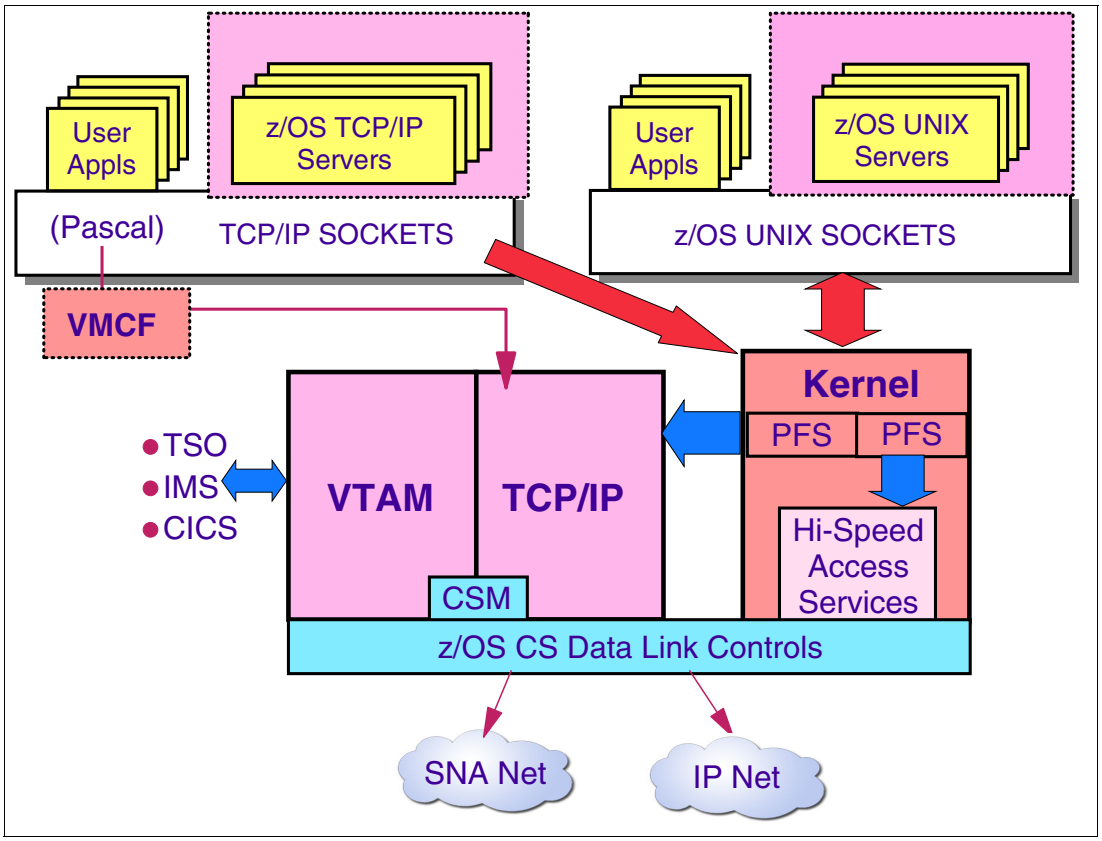


Figure 11-3 Overview of the Communications Server

z/OS Communications Server (CS)

Since z/OS V2R5 was shipped, the z/OS Communications Server (CS) has been included as a base element. The TCP/IP z/OS component, called z/OS CS, is a reconstructed stack that is able to support both UNIX and non-UNIX socket APIs. It is often called the converged IP stack.

All the TCP/IP socket APIs that supported HPNS are now transparently redirected by run-time support to call the UNIX kernel LFS. The REXX socket API has also been directed to call the kernel. HPNS support is no longer required.

Pascal API

The Pascal API, however, still requires the VMCF/IUCV address space to be started, which links the API to the new stack. VMCF and TNF do not respond to commands. This is probably because one or both of the on-restartable versions of VMCF or TNF are still active. To get them to respond to commands, stop all VMCF/TNF users, FORCE ARM VMCF and TNF, then use the EZAZSSI procedure to restart.

The Pascal application programming interface enables you to develop TCP/IP applications in Pascal language. Supported environments are normal MVS address spaces. The Pascal programming interface is based on Pascal procedures and functions that implement conceptually the same functions as the C socket interface. The Pascal routines, however, have different names than the C socket calls. Unlike the other APIs, the Pascal API does not

interface directly with the LFS. It uses an internal interface to communicate with the TCP/IP protocol stack.

CS stacks

The SNA and IP networking stacks have been integrated to a considerable extent. Both stacks use common Data Link Control (DLC) routines to access network hardware, and both types of protocols can flow over the same hardware link. Also, common service routines such as Communications Storage Manager (CSM) exploit use of buffers in common storage for both IP and SNA performance.

11.4 z/OS UNIX sockets support

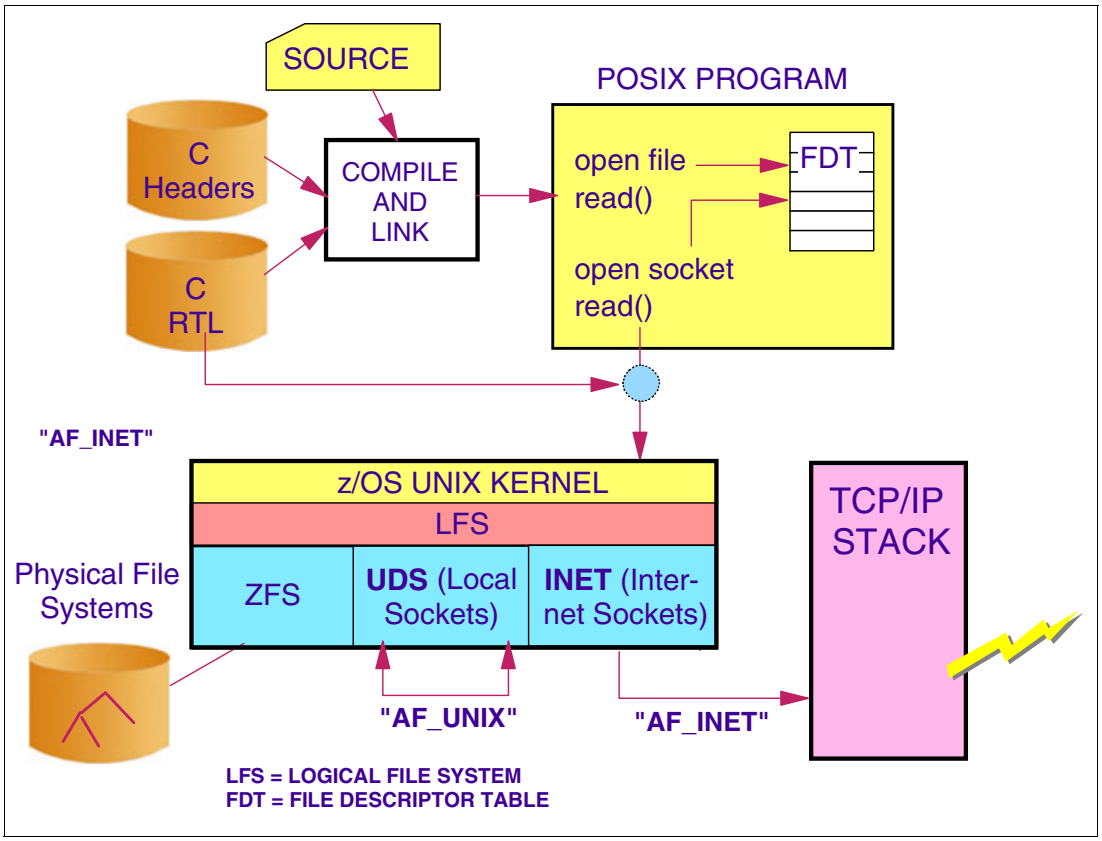


Figure 11-4 Sockets overview with z/OS UNIX

z/OS UNIX sockets support

Integrated Socket Support was the UNIX socket API supplied with TCP/IP V3.x. This API makes it transparent to a POSIX process what type of software underpins the C socket calls.

Integrated sockets merged the C run-time library support for TCP/IP (z/OS) and UNIX socket calls. A read() call is supported by one module for files and sockets.

Two Socket Physical File Systems (PFSs) are part of the z/OS UNIX component. One supports AF_UNIX local domain sockets, and the other supports TCP/IP AF_INET network sockets. Both file and socket access are channeled through the standard Logical File System (LFS) to the relevant PFS. This enables z/OS UNIX to use a single pool of numeric IDs to be allocated to both file descriptors and socket descriptors, avoiding any confusion.

Statements to define the PFS components, and a NETWORK statement, are coded in the z/OS UNIX BPXPRMXX PARMLIB member, defining the type of socket support associated with each file system.

A program that uses files and sockets, and uses identical calls (that is, read() and write() and so on) to process both resources can be compiled and linked as a single unit.

11.5 Customizing sockets

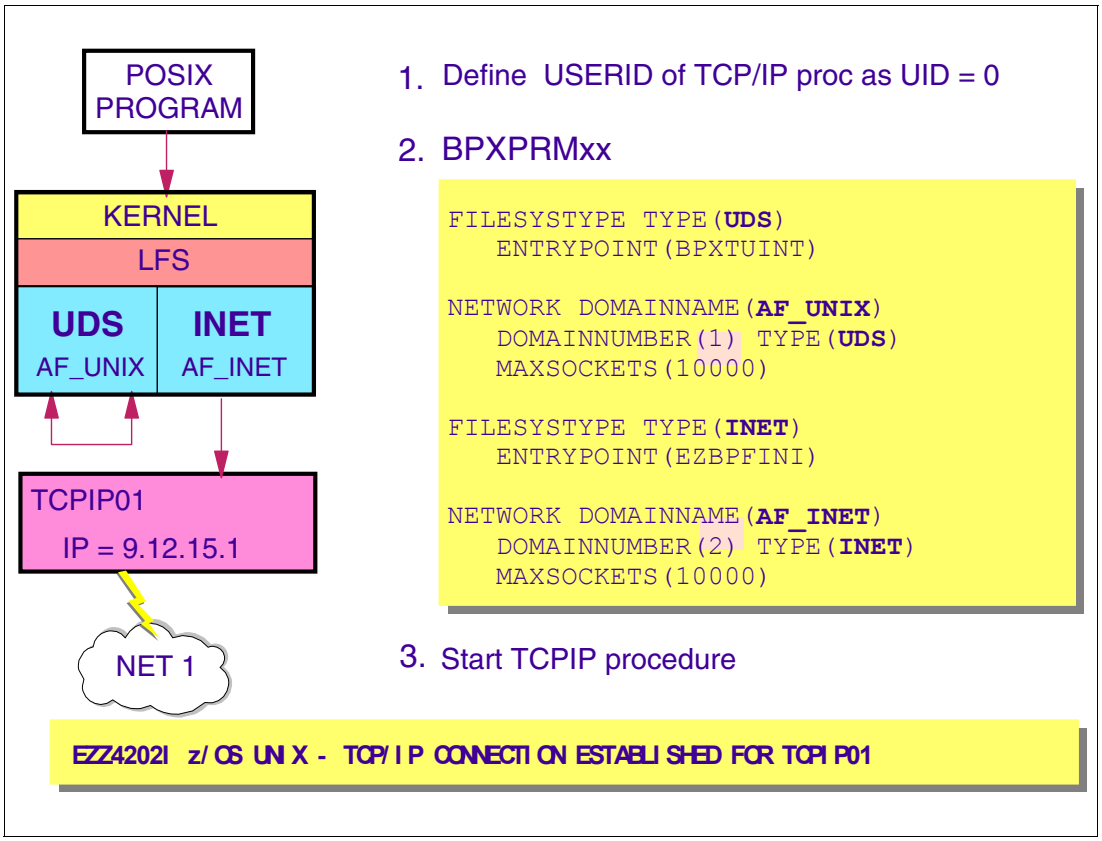


Figure 11-5 Customizing the use of z/OS UNIX sockets

Customizing sockets

This is the first scenario for supporting z/OS UNIX sockets. It assumes a simple environment with only one TCP/IP stack used for network communication.

The user ID owning the TCP/IP address space (as represented by the name of the TCP/IP procedure) must be defined as a z/OS UNIX superuser, by inserting UID=0 in the RACF OMVS segment.

The physical file systems to support socket communication must be defined in the BPXPRMxx member in SYS1.PARMLIB. Two types of statements are required:

- **FILESYSTYPE** - Defines a z/OS UNIX PFS:
  - TYPE - UDS=local domain sockets, INET=networking sockets
  - ENTRYPOINT of z/OS UNIX program supporting PFS - code for IP level:
    - BPXTUINT - AF_UNIX (UDS) socket support
    - EZBPFINI - IP stack
- **NETWORK** - Describes a group of sockets allocated to a PFS:
  - DOMAINNAME - Name of socket file system domain associated with group - local (AF_UNIX) or TCP/IP (AF_INET)
  - DOMAINNUMBER - Number matches name - 1=AF_UNIX, 2=AF_INET
  - MAXSOCKETS - Max sockets supported - default=100, max=64498
  - TYPE - INET or UDS - points to related FILESYSTYPE entry

11.6 Logging in to the z/OS UNIX shell

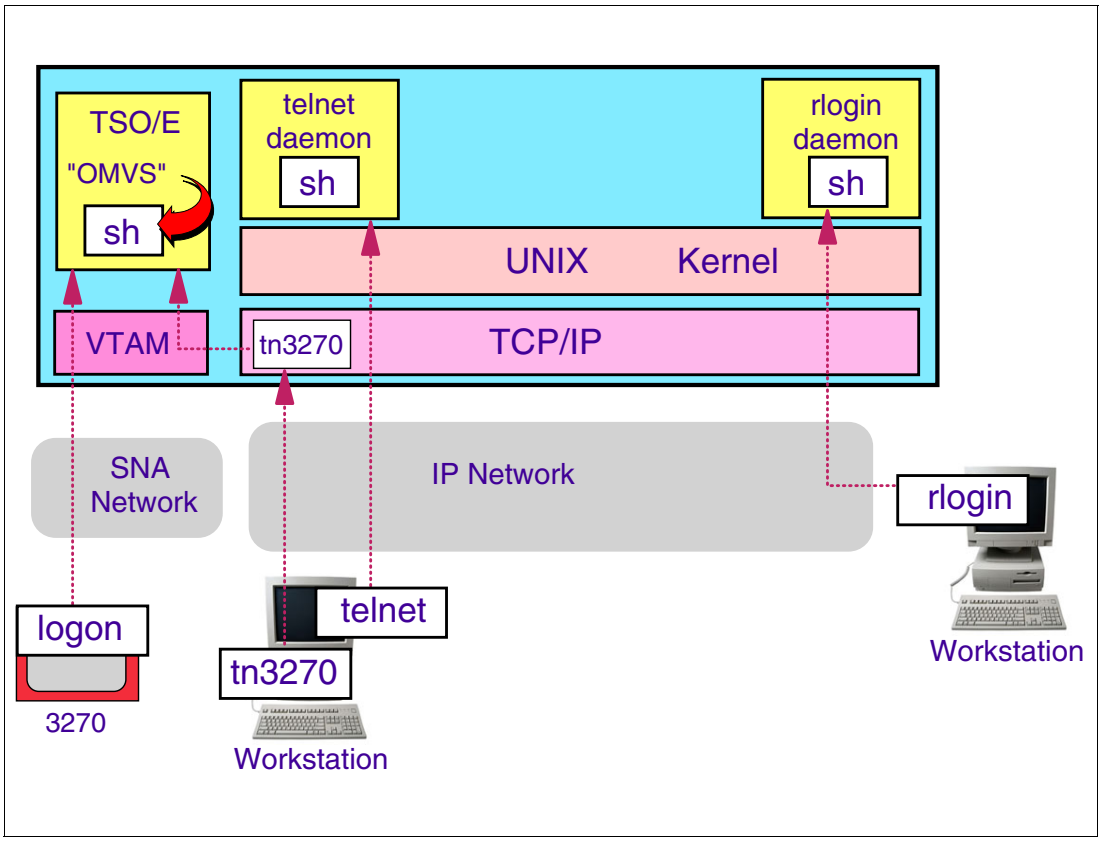


Figure 11-6 Logging in to the z/OS UNIX shell

Login to z/OS UNIX shell

The following are mechanisms to start a z/OS UNIX shell session:

- Via the TSO OMVS command:

  A TSO user, logged in via SNA and VTAM, can issue the OMVS commands directly from the TSO command line.

- A workstation user can use TCP/IP tn3270 protocol to log on to a TSO user ID via IP 3270(E) Telnet server. From TSO, use OMVS to access the shell.

- Via **`rlogin`** done directly from a workstation:

  A UNIX/AIX user can use a standard rlogin client to do remote login to z/OS UNIX. A z/OS UNIX rlogin daemon initiates the shell.

- Via Telnet done directly from a workstation:

  Any platform can use a standard Telnet client to do remote login to z/OS UNIX. A z/OS UNIX Telnet daemon initiates the shell.

- Both direct rlogin/Telnet and CS assisted logins support raw mode.

11.7 Using inetd - master of daemons

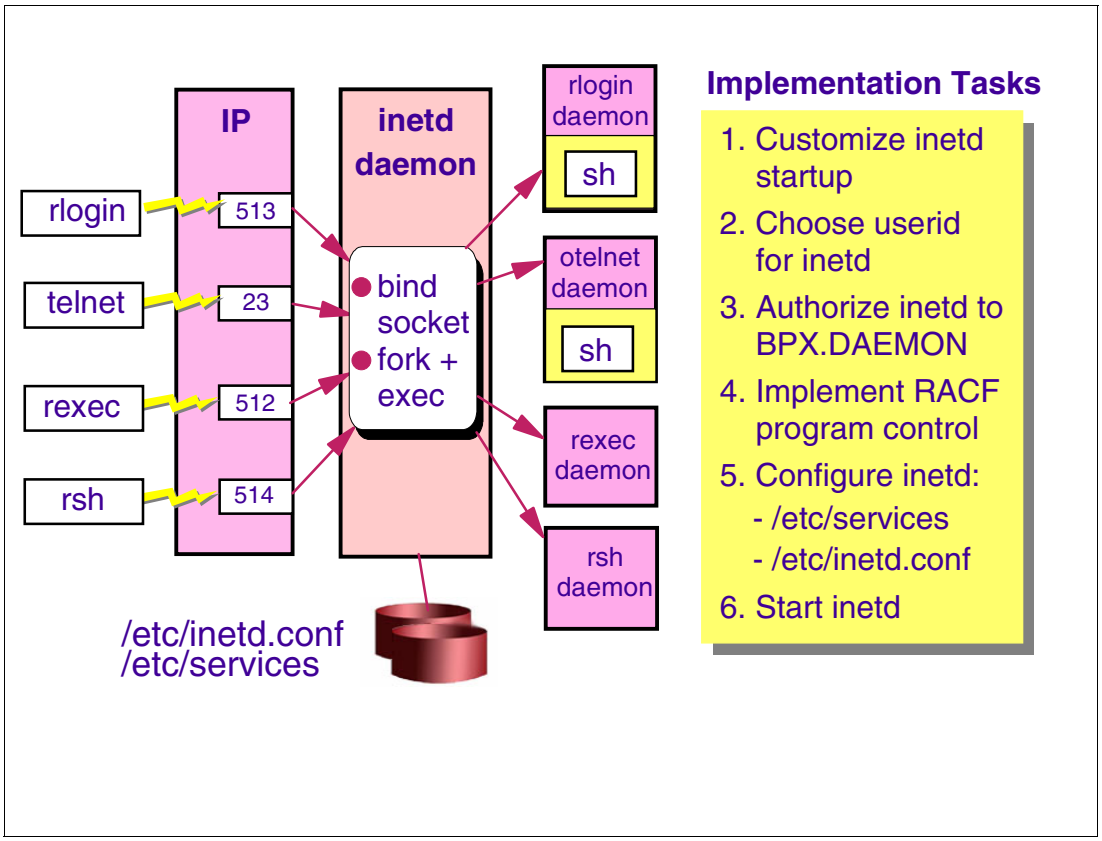


Figure 11-7 Overview of the inetd daemon

Customize and start the inetd daemon

The inetd daemon, usually known as inetd or InetD, is a master of other daemons that execute in z/OS UNIX. The function of inetd is to listen on certain well-known network ports for a request to run one of a number of daemons. When a request is received, inetd creates a new socket for remote connection, and then fork()s a new address space and uses exec() to start the requested daemon program.

The daemon started by inetd relates to the port where the request arrived. The correlation between port number and daemon is stored in the configuration file /etc/inetd.conf.

The daemons started by inetd include:

- The rlogin daemon starts a shell session for a user rlogin request.
- The telnet daemon starts a shell session for a user Telnet request.
- The rexec daemon executes a single command on z/OS UNIX requested by a remote user entering a rexec command.
- The rsh daemon starts a shell session and runs a script generated by a remote user entering an rsh command.

Customization is needed to enable inetd to run on your system. You must decide how to start it, and what RACF ID it will execute under. If you have implemented enhanced daemon security with BPX.DAEMON, you must define inetd to the BPX.DAEMON and implement program control. Finally, you have to configure the relationship between the ports that inetd listens on and the daemons to be started.

11.8 Customize inetd

1. Start inetd

/etc/rc
```
_BPX_JOBNAME='INETD' /usr/sbin/inetd /etc/inetd.conf &
```

2. Establish inetd userid

```
RDEFINE STARTED INETD.* STDATA(USER(OMVSKERN) GROUP(OMVSGRP)
        TRUSTED(NO))
```

3. Authorize userid to use BPX.DAEMON

```
PERMIT BPX.DAEMON CLASS(FACILITY) ID(OMVSKERN) ACCESS(READ)
```

Figure 11-8 Customizing the inetd daemon

Customize inetd

The inetd daemon program can be found in two places. In the HFS, the program file is /usr/sbin/inetd, but IBM has set the sticky bit on. A copy of this program is found in 'SYS1.LINKLIB(INETD)', so this is the program that is used. Start from a line in the initialization script /etc/rc. In this case, use a command similar to the line shown in Figure 11-8.

The next step is to decide which user ID to associate with inetd. It needs to be a superuser (UID=0), and to have minimum access to MVS data sets. How you do this depends on start mode.

When started from /etc/rc, inetd inherits user ID OMVSKERN, which is a superuser. Starting up via /etc/rc you are effectively locked into using the user ID under which the /etc/rc script is running, as inetd is forked from that script. The user ID for /etc/rc is the kernel ID OMVSKERN.

If you have activated the RACF BPX.DAEMON facility, then the INETD user ID must be authorized to this facility.

11.9 Customize inetd (2)

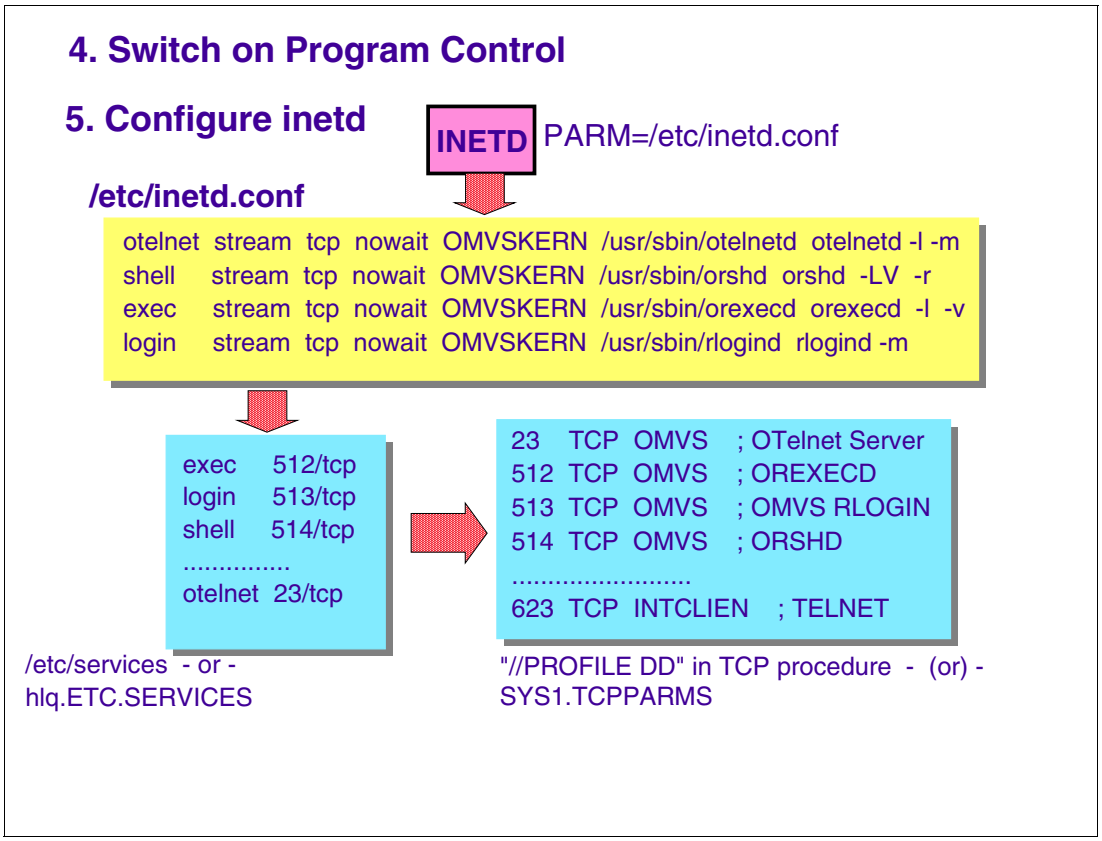


Figure 11-9 Customizing the inetd daemon

Customize inetd

If you have set up the BPX.DAEMON, then you need to make sure that all programs are loaded into the inetd address space. At a minimum, you should protect the following programs:

- SYS1.LINKLIB(INETD).
- CEE.SCEERUN - LE/MVS run time - whole library

There are three configuration files that have to be updated for inetd support:

- The primary file is /etc/inetd.conf, which is the inetd configuration file. There is one entry (line) in this file for each daemon controlled by inetd. The fields are interpreted as follows:
  - Field (1) - Service name - match daemon entry in /etc/services file
  - Field (2) - Daemon socket type - stream or dgram
  - Field (3) - Daemon socket protocol - TCP or UDP
  - Field (4) - Wait_flag - can be wait (single thread server - 1 request at a time) or nowait (multiple requests queued)
  - Field (5) - Login_name - RACF user ID 0nder which daemon will run
  - Field (6) - Server_program - name of daemon program in HFS
  - Field (7) - Server-arguments - first string is jobname for daemon address space, and the rest is the parm string to pass to daemon

- There is a corresponding entry in /etc/services for each daemon in inetd.conf. The entry lists the port where inetd listens for daemon requests.
- The TCPIP PROFILE configuration must list the same ports in the PORT section. This entry identifies the job name authorized to open the socket to this port and the type of socket allowed.

The two TCP/IP files usually exist already; you must make sure that inetd.conf corresponds with the values listed. Also, you may want to change the port number for a daemon.

After all configuration is complete, start inetd.

11.10 Login to a Unix system

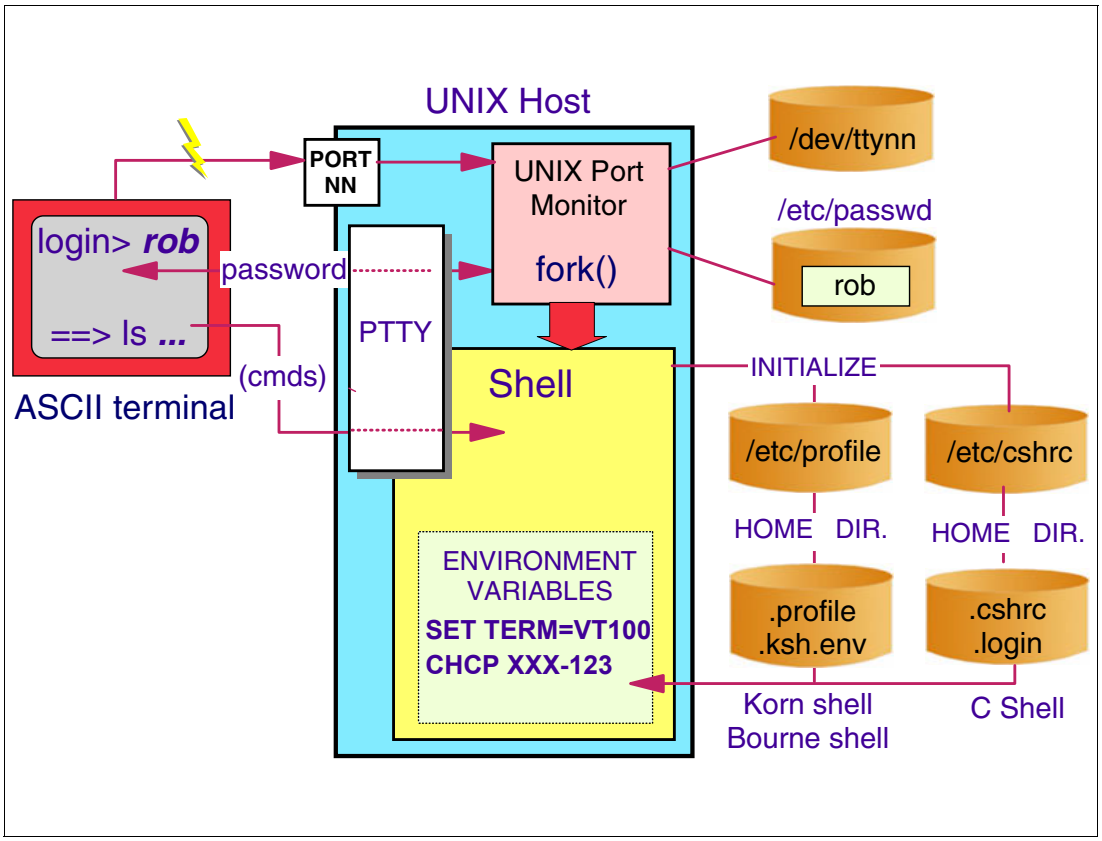


Figure 11-10 Log into a UNIX system on other platforms

UNIX system login

Let's look at a real UNIX login approach first for comparison. When a locally attached ASCII terminal is powered on, an interrupt is generated by the hardware port by which the terminal is connected. This interrupt is detected by the port monitor software, which then retrieves a terminal definition file related to that hardware port. It reads terminal characteristics from this file, initializes a pseudo-TTY terminal interface, and sends the login prompt to the screen.

The user does a login to a locally defined user ID. The port monitor retrieves an account record related to this user from the /etc/passwd file, which includes the user password. The monitor then prompts for and validates the user password.

After validation, the terminal monitor sets the user UID and GID from data in the account file. It then forks a shell process, using the shell program name defined in account data. The PTTY is now the shell terminal interface.

When the shell initializes, it uses the contents of certain system default files, as well as files in the user ID home directory, to initialize the shell environment variables. The files used depend on the type of shell invoked.

In particular, the user may want to set the TERM environment variable in order to define the type of ASCII terminal that the user has. Also, if the user is using a non-US keyboard, they will want to automatically issue the **chcp** (change code page) command to ensure keystrokes are correctly interpreted.

11.11 rlogin to z/OS UNIX services

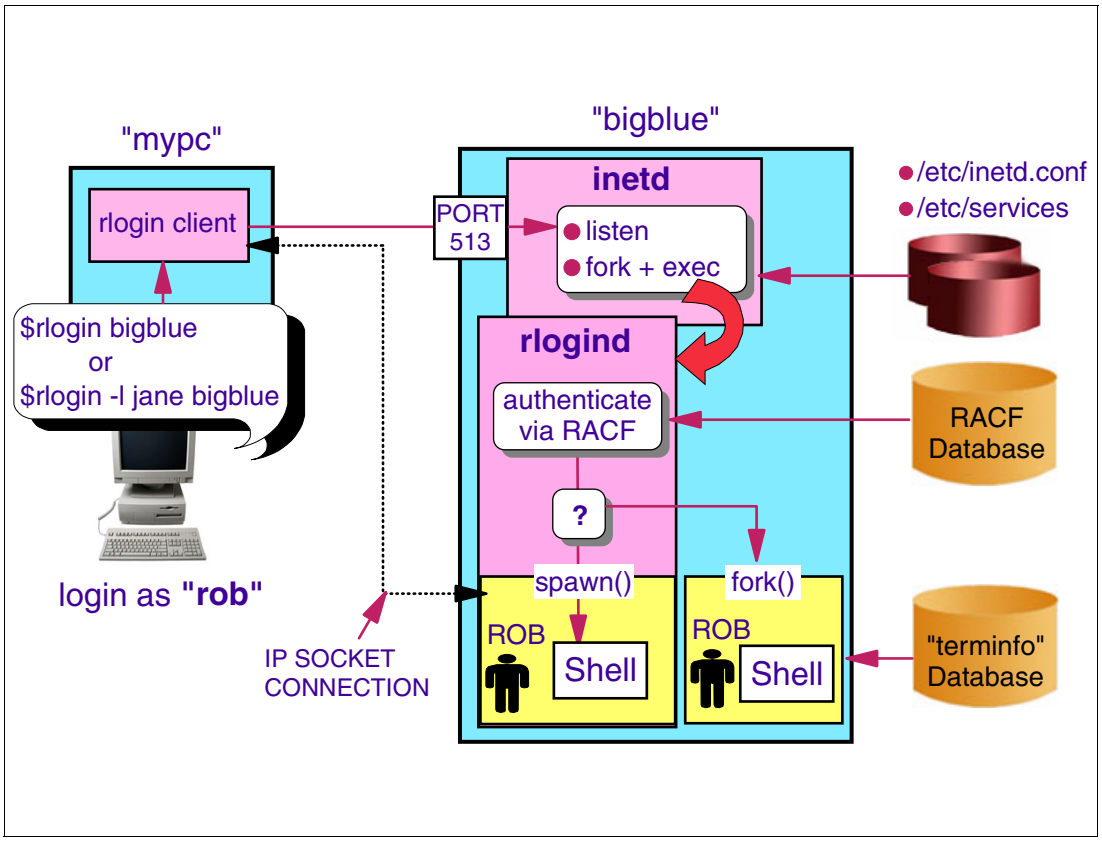


Figure 11-11 rlogin to a z/OS UNIX system

rlogin to z/OS UNIX

Figure 11-11 illustrates a similar flow through an rlogin request made through TCP/IP to z/OS UNIX. Assume that the user has already done a login to the local host as `rob`. The user issues **`rlogin`** from the shell session. The format will depend on the local host. z/OS UNIX accepts an rlogin under the current ID (rob) or the new ID (jane).

The AIX/UNIX rlogin client sends the request to port 513 on the host, monitored by the inetd daemon. inetd forks a new address space and initializes the rlogind server.

The rlogind server uses a z/OS UNIX socket, created by inetd and passed via fork, to communicate with the rlogin client. The server then proceeds to validate the rlogin request, as follows:

- It reads the RACF user profile for the rlogin user ID passed (the current or new user ID). It also reads the contents of the RACF OMVS segment.
- It prompts the remote client for the correct (RACF) password. Note that z/OS UNIX does not support the use of either /etc/equiv.hosts or $HOME/.hosts files defined in HFS to bypass authentication.

If authentication is good, rlogind allocates a standard z/OS UNIX pseudo-tty terminal pair, and then initiates the client shell in one of two ways:

- It creates a child shell process using local_spawn() and the validated user ID.
- It forks a copy of itself in a new address space, uses setuid() and seteuid() commands to set RACF security to a valid user ID, and then runs an exec() shell program.

11.12 Activating z/OS UNIX rlogin daemon

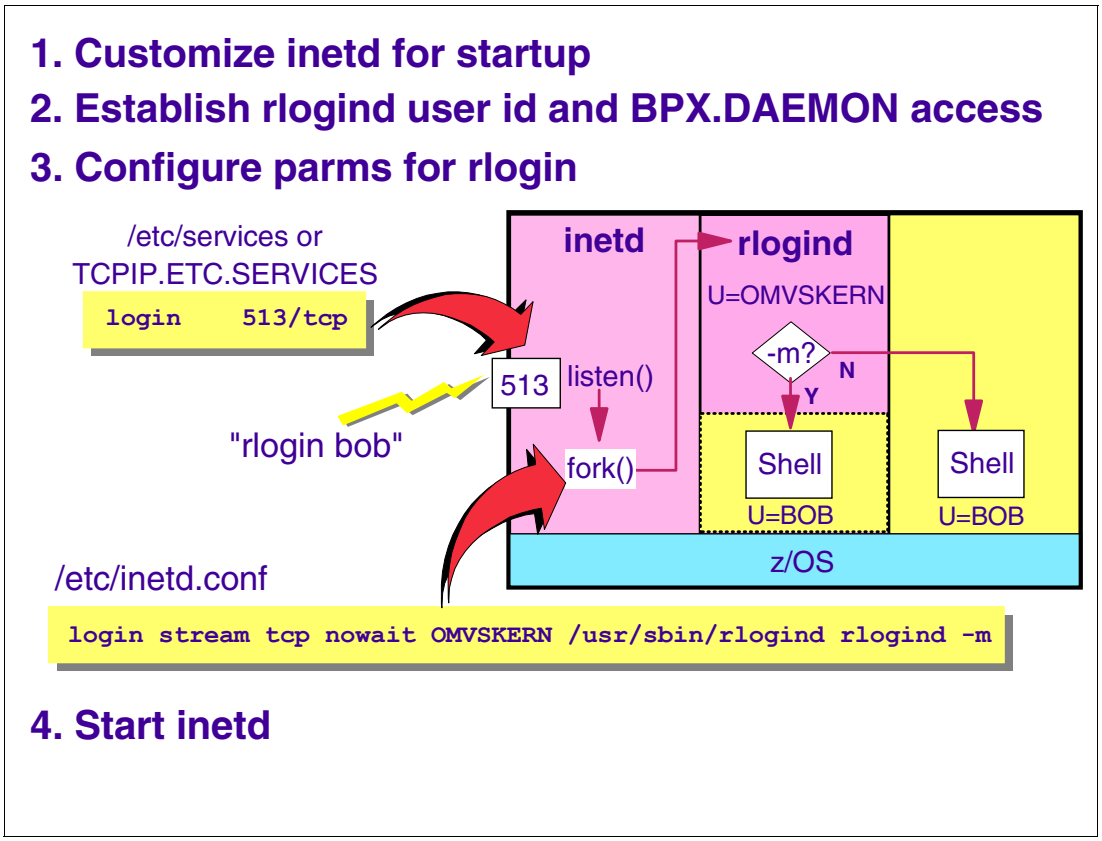


Figure 11-12 Activating the rlogin daemon

rlogin daemon

Figure 11-12 illustrates the steps to customize z/OS UNIX for the rlogin daemon, as follows:

1. Go through the steps to customize the z/OS UNIX inet daemon INETD, and test that the daemon is able to start.
2. Identify the user ID under which rlogind (the login daemon) will run. The rlogind program as a daemon needs to be a superuser (UID=0), and authorized to access the BPX.DAEMON RACF facility, if used. The kernel user ID is typically used.
3. Configure parms for starting rlogind as follows:
   - Ensure that the TCPIP.ETC.SERVICES file has active entry, as shown in the figure. This assigns port 513 to the rlogin daemon.
   - Update the inetd configuration file, /etc/inetd.conf, to include the entry for the rlogin daemon.
     - **login** - The ID of the entry for rlogin; must match TCPIP.ETC.SERVICES.
     - **stream tcp** - Identifies the daemon socket protocol (this is required).
     - **nowait** - INETD accepts multiple current connections on behalf of rlogind.
     - **OMVSKERN** - The user ID under which the rlogin daemon runs.
     - /usr/sbin/rlogind - Pathname of the rlogin daemon program. Sticky bit on means that the system actually fetches SYS1.LINKLIB(RLOGIND).
     - Remaining string = parameters for rlogin daemon (see the following item).

– Parameters in the rlogind parameter string can include:
  - **rlogind** - (jobname) of server process.
  - **-m** - If specified, the shell process shares address space with the rlogin daemon.
  - **-d** - Switches on debug - extra messages are written to the system log.

4. To start rlogind support, you need to start the inetd daemon.

rlogin steps

Let's walk through the process of doing an rlogin:

- First the inetd daemon starts up, either when the z/OS UNIX kernel is started from the **/etc/rc** script, or via a **`start`** command and procedure.
- The inetd daemon reads the configuration file and discovers that it must listen on TCP/IP port 513 for incoming requests for the rlogind daemon (entry login).
- When an incoming request is received on port 513, inetd BINDS a new socket for the request and then forks inetd copy in a new address space.
- inetd copy sets the jobname for the new address space to RLOGIND (from inet.conf parm 7), does setuid to the user ID for rlogon (OMVSKERN), and then does exec () to call the rlogind program. It passes the rest of the argument string from inetd.conf as a parameter.
- The rlogind daemon uses the supplied socket to contact the client and validate the incoming login request. If the client gives a valid ID, rlogind reads the contents of the OMVS segment for the user ID and allocates a PTY/TTY virtual terminal pair for the session.
- Then rlogind tests for the -m parameter. If this is supplied, it runs the shell as a child process in the rlogind address space. Otherwise, rlogind forks a new address space and execs the shell in that address space. In either case, the shell runs under the client user ID.

11.13 Comparing shell login methods

	OMVS	rlogin, telnet
Default code-page Convert	IBM-037 -> IBM-1047	ISO8859 -> IBM-1047
Change codepage?	"CONVERT" ON OMVS	"chcp"
# LOGINS per ID	1	MULTIPLE
File Editors	ISPF(oedit), sed, ed	vi, ed, sed
Toggle to TSO	YES	NO
Shell Modes	LINE	LINE, RAW
Shared Storage	"SHAREAS" ON OMVS	"-m" OPTION ON rlogin

Figure 11-13 Comparison of the logins to the shell

Shell login method comparisons

Figure 11-13 compares and contrasts the different methods of accessing the shell, as follows:

- **Default code page translation** - The OMVS shell interface translates between IBM-037 and IBM-1047. However, it does not handle C square brackets or the accent correctly. All other logins translate between ISO-8859 ASCII and IBM-1047.
- **Changing code page** - For OMVS, use the `CONVERT` option on the **`OMVS`** command. For all other methods, use the **`chcp`** shell command after login.
- **Number of concurrent logins** - OMVS users can only start one shell session at a time. All asynchronous users can log in to multiple shell sessions concurrently.
- **File editors** - In OMVS, the ISPF full screen editor can be used via the **`oedit`** command. No asynchronous modes can use ISPF, but they can use the **`vi`** full screen editor. **ed** and **sed** line editors are commonly available.
- **Shell mode** - OMVS can only work in line (canonical) mode. Asynchronous login supports line mode or *raw* (non-canonical) mode.
- **Toggle to TSO?** - Only OMVS shell provides this option.
- **Shared address space** - Use a single address space to support both terminal I/O and shell processes. For OMVS, specify the SHAREAS parameter on the OMVS command. For rlogin, use the **`-m`** option on the **`start`** command for the lm daemon.

11.14 Define TCP/IP daemons

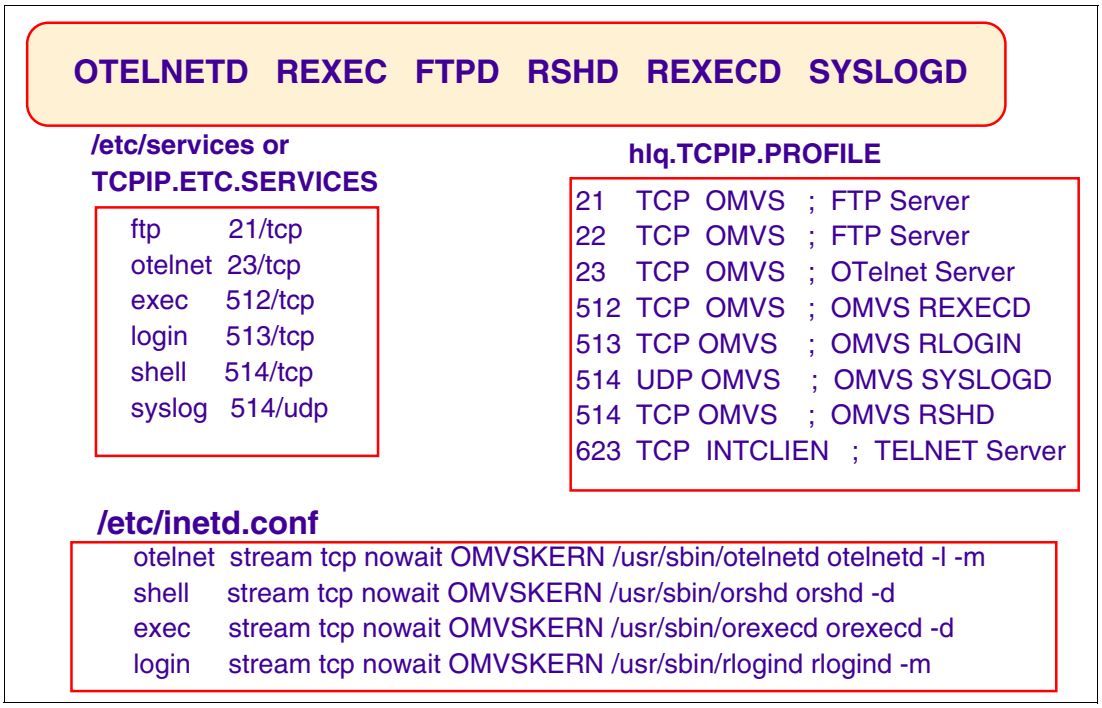


Figure 11-14 Defining the TCP/IP daemons

Defining TCP/IP daemons

The TCP/IP z/OS UNIX Application feature provides several other TCP/IP functions that you might want to configure, as follows:

TELNET Allows remote users to log in to z/OS using a Telnet client. The z/OS UNIX Telnet server is started for each user by the INETD listener program.

REXEC Remote execution client and server support the sending and receiving of a command.

FTP File transfer program supports transfer into and out of the HFS.

RSH Provides remote execution facilities with authentication based on privileged port numbers, user IDs, and passwords.

SYSLOGD Supplies the logging functions for programs that execute in the z/OS UNIX environment.

Ports need to be assigned to the functions that you choose to configure. The hlq.TCPIP.PROFILE data set has an entry for each function and its port and protocol. If you will be configuring both the z/OS UNIX version and the standard TCP/IP version, you will need to decide which one will use the well-known port assignment.

The TCP/IP resolver function also needs to have the port assignments. These can reside in either the TCPIP.ETC.SERVICES data set or the /etc/services file.

Each daemon then has its own configuration information. The inetd program comes with z/OS UNIX and is the listener program for several of the TCP/IP daemons. The commands inetd will use to initiate each program are put in the /etc/inetd.conf file.

The SYSLOG and FTP daemons have their own configuration files, /etc/syslog.conf and /etc/ftpd.data respectively, and each requires a startup procedure.

11.15 The syslogd daemon

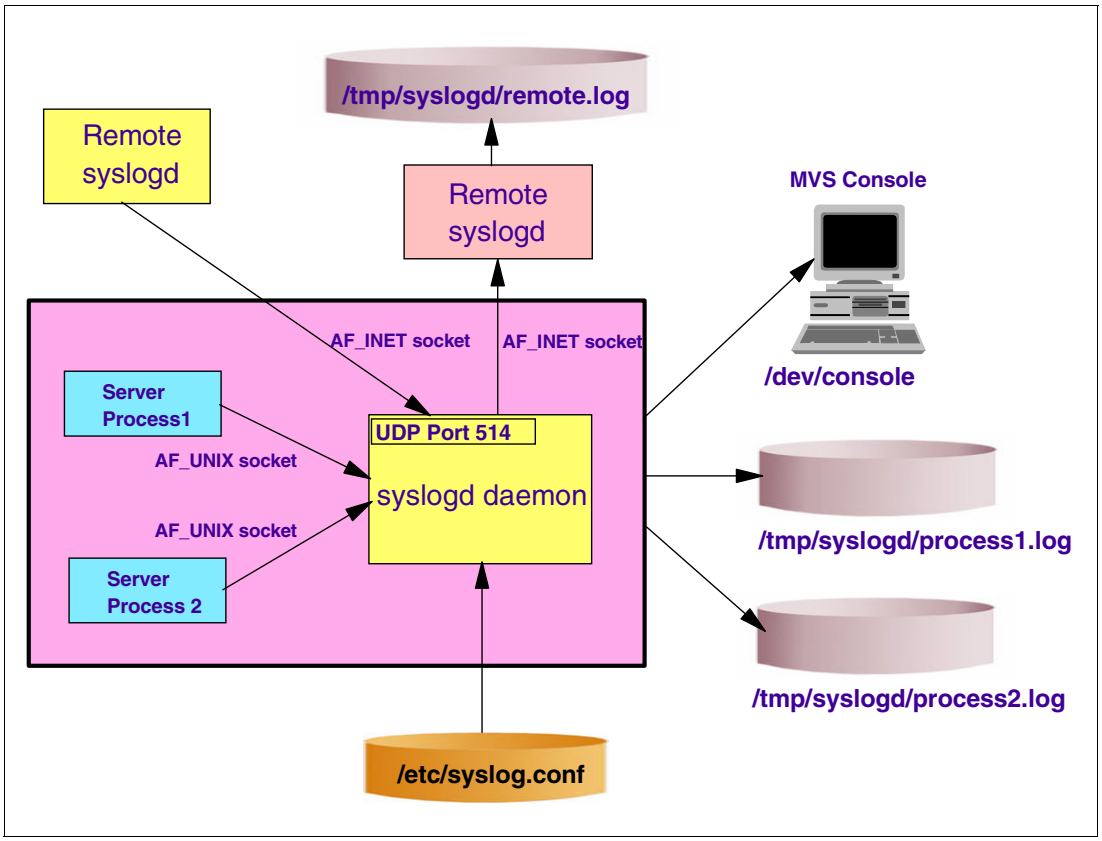


Figure 11-15 Overview of the syslogd daemon

Syslog daemon

The syslog daemon (syslogd) is a server process that must be started as one the first processes in your z/OS UNIX environment. CS for z/OS server applications and components use syslogd for logging purposes and can also send trace information to syslogd. Servers on the local system use AF_UNIX sockets to communicate with syslogd; remote servers use the AF_INET socket.

The syslogd daemon reads and logs system messages to the MVS console, log files, other machines, or users, as specified by the configuration file /etc/syslog.conf. A sample is provided in /usr/lpp/tcpip/samples/syslog.conf.

If the syslog daemon is not started, the application log may appear on the MVS console. The syslog daemon must have a user ID; for example, SYSLOGD defined in RACF with UID=0. The syslogd daemon uses the following files:

/dev/console	Operator console
/etc/syslog.pid	Location of the process ID
/etc/syslog.conf	Default configuration file
/dev/log	Default log path for z/OS UNIX datagram socket
/usr/sbin/syslog	Syslog server

syslogd can only be started by a superuser. It can be terminated using the SIGTERM signal. If you want syslogd to receive log data from or send log data to remote syslogd servers, reserve UDP port 514 for the syslogd job in your PROFILE.TCP/IP data set and enter the syslog service for UDP port 514 in the services file or data set (for example, `/etc/services`).

Starting syslogd

Syslogd can be started only by a task or user with superuser authority (UID 0). You can start syslogd from a procedure or from the UNIX shell. If you start syslogd from a procedure that does not use BPXBATCH, the resulting job name is the same as the procedure name. If you start syslogd from the UNIX shell or from a procedure that uses BPXBATCH, the resulting job name is the user ID or the value of the _BPX_JOBNAME environment variable.

If you start syslogd from the UNIX shell or from BPXBATCH, start it as a background shell command by specifying an ampersand (&) as the last character on the command. If you do not specify the ampersand, control does not return to the shell until syslogd ends. Specifying an ampersand is especially important if syslogd is started from a shell script, such as /etc/rc.

If you start syslogd from a cataloged procedure that uses BPXBATCH, you need to include a sleep command in your script after the start for syslogd to provide syslogd the time to initialize before the shell script ends.

11.16 The FTPD daemon

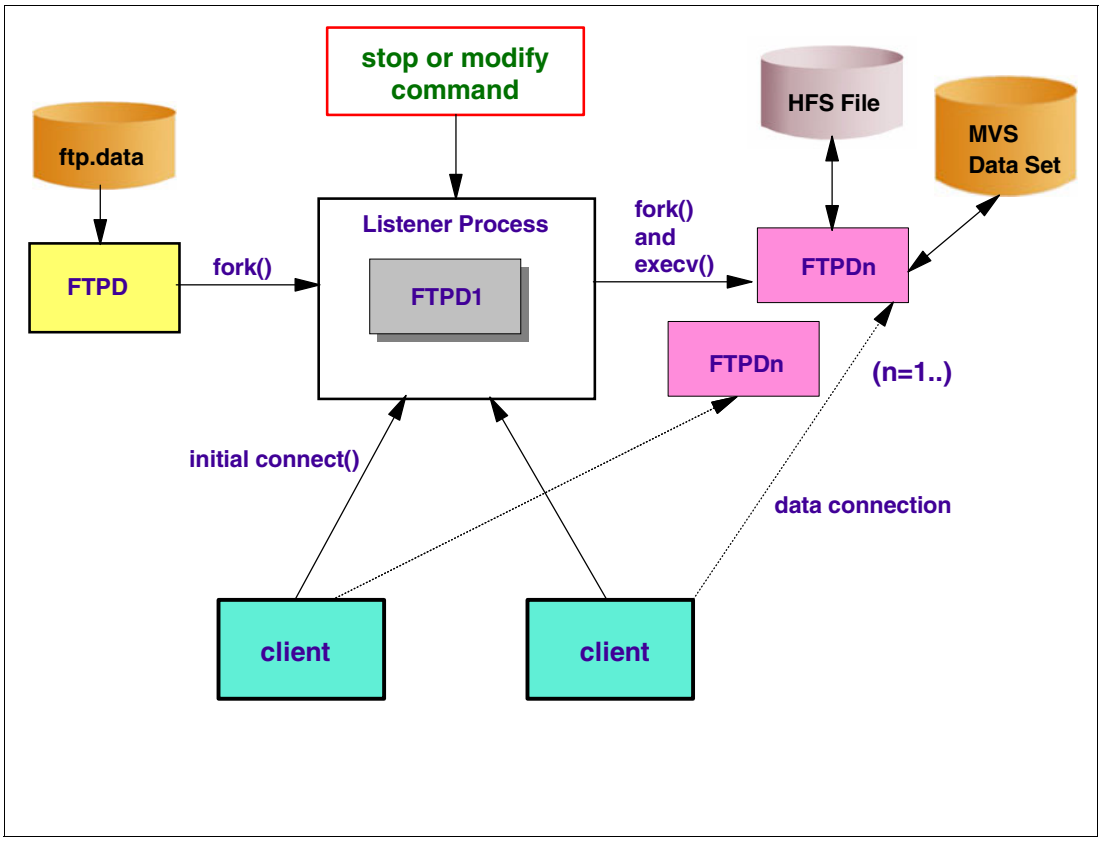


Figure 11-16 Overview of the FTPD daemon

FTPD daemon

File Transfer Protocol (FTP) is used to transfer files between TCP/IP hosts. The FTP client is the TCP/IP host that initiates the FTP session, while the FTP server is the TCP/IP host to which the client connects.

The FTP server uses two different ports and manages two TCP connections as follows:

- Port 21 is used to control the connection (user ID and password).
- Port 20 is used for actual data transfer based on the FTP client's requests.

The FTP server in z/OS IP consists of the daemon (the listener) or **ftpd** and server address space (or processes). The daemon performs initialization, listens for new connections and starts a separate server address space for each connection.

When a new client FTP connects to the FTPD daemon process, ftpd forks an FTP server process; thus, a new jobname is generated by z/OS UNIX System Services.

11.17 z/OS IP search order for FTP

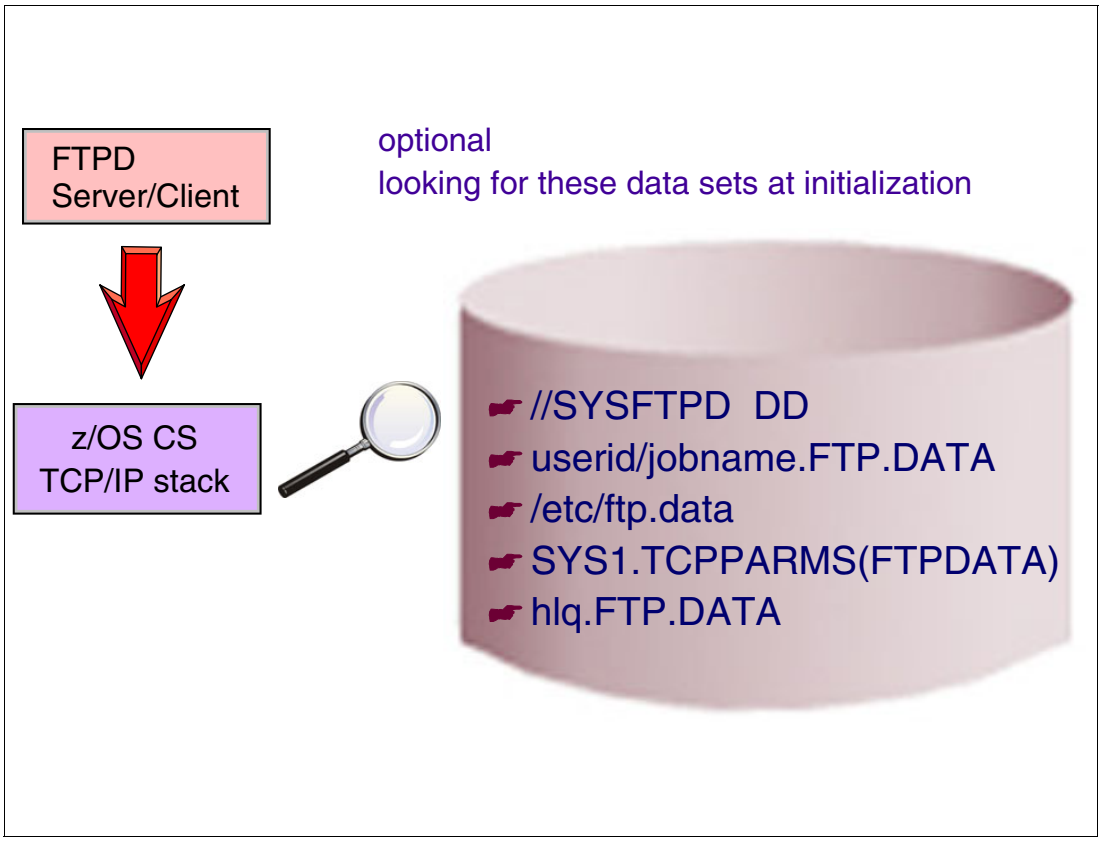


Figure 11-17 Search order for the FTPD daemon

FTP search order

FTP.DATA is used to override the default FTP client and server parameters for the FTP server.

You may not need to specify the FTP.DATA data set if the default parameters are used.

A sample is provided in hlq.SEZAINST(FTPDATA) for the client and hlq.SEZAINST(FTPSDATA) for the server.

When an FTPD daemon or started task is started, it searches the FTP.DATA file in the following order:

- //SYSFTPD DD in FTPD started task procedure
- userid/jobname.FTP.DATA
- /etc/ftp.data
- SYS1.TCPPARMS(FTPDATA)
- hlq.FTP.DATA

The search stops if one of these data sets is found.

11.18 z/OS IP search order for /etc/services

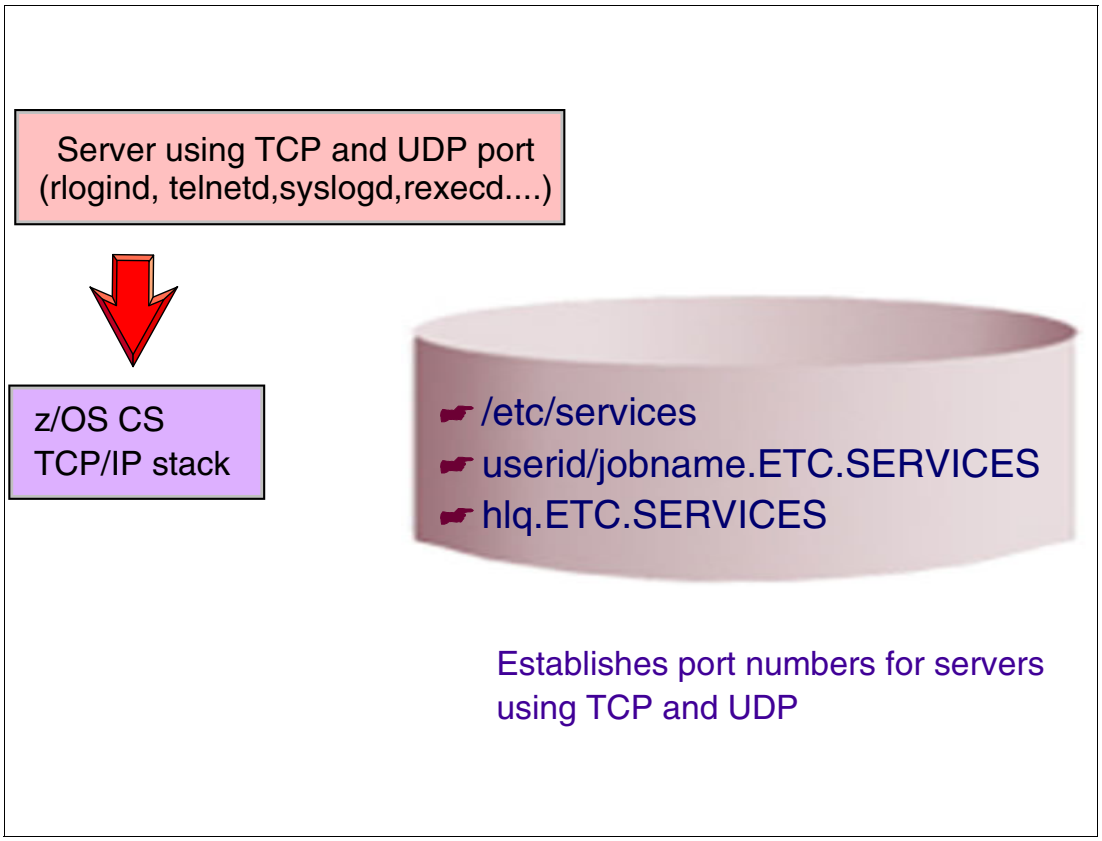


Figure 11-18 Search order for /etc/services

Search order for /etc/services

The ETC.SERVICES data set is used to establish port numbers for UNIX application servers using TCP and UDP. This file or data set is required for any daemon or application that needs the use of a specific port.

Standard applications, like telnet or FTP, are assigned port numbers in the well-known port number range. You can assign port numbers to your own server applications by adding entries to the /etc/services file.

For example, rlogind listens on 513/TCP and telnetd listens on port 23/TCP, while syslogd listens on port 514/UDP. This specification is provided in the ETC.SERVICES data set.

When TCP/IP and the daemons start, they look for the ETC.SERVICE file or data set in the following order:

- /etc/services (HFS file)
- userid/jobname.ETC.SERVICES
- hlq.ETC.SERVICES

The search stops if one of these data sets is found.

11.19 Start the TCP/IP daemons

- Start Daemons at Initialization **/etc/rc**

```
BPX_JOBNAME = 'SYSLOGD'  /usr/sbin/syslogd -f /etc/syslog.conf &

BPX_JOBNAME='FTPD' /usr/sbin/ftpd /etc/ftp.data &

BPX_JOBNAME = 'INETD'  /usr/sbin/inetd  /etc/inetd.conf &
```

- Daemons started by INETD
  /etc/inetd.conf

```
otelnet stream tcp nowait OMVSKERN /usr/sbin/otelnetd otelnetd -l -m
shell   stream tcp nowait OMVSKERN /usr/sbin/orshd orshd -d
exec    stream tcp nowait OMVSKERN /usr/sbin/orexecd orexecd -d
login   stream tcp nowait OMVSKERN /usr/sbin/rlogind rlogind -m
```

- Daemons started by a BPXBATCH job

Figure 11-19 Starting the TCP/IP daemons

Starting TCP/IP daemons

After the configuration files have been completed, the daemons need to be started before any remote requests can be processed.

The /etc/rc script is a good place to put the start command, as Figure 11-19 shows. In this case, the daemons will be started during the initialization processing for z/OS UNIX. The _BPX_JOBNAME environment variable will give the daemon an MVS jobname.

Since inetd is responsible for starting the other daemons (Telnet, rlogin, remote shell and remote execution), start commands for them are in inetd's configuration file.

In case any of these daemons fail, you should have other procedures created to restart them since /etc/rc is only used at z/OS UNIX initialization. You could use shell scripts or MVS procedures for this.

11.20 Message integration support

- z/OS V1R8 support for logging messages from multiple source (Linux, z/OS) into one common place
  - Allows Linux messages to be integrated into z/OS OPERLOG
  - Provides /dev/operlog support
    - Write messages to OPERLOG without issuing WTOs
    - Syslog daemon (Communications Server) receives messages from remote systems, applies a host filter, translates all text to EBCDIC and writes them to the /dev/operlog file

Figure 11-20 Message integration support with Linux®

SYSLOGD message integration

In the current z/OS releases, the syslog daemon (syslogd) is a server process that must be started as one of the first processes in your z/OS UNIX environment. Communications Server server applications and components use syslogd for logging purposes and can also send trace information to syslogd. Servers on the local system use AF_UNIX sockets to communicate with syslogd; remote servers use the AF_INET socket. The syslogd daemon reads and logs system messages to the MVS console, log files, other machines, or users, as specified by the configuration file /etc/syslog.conf. A sample is provided in /usr/lpp/tcpip/samples/syslog.conf.

With z/OS V1R8, the Communication Server syslog daemon now supports the following changes:

- Introduce a new filter Hostname/IP address for the syslog daemon. This allows administrators to limit what remote messages are logged and where.
- Issue a WTO indicating that OPERLOG is not active if OPERLOG is inactive.
- A new file is created, /etc/syslog_net.pid.

The /dev/operlog support provides a much lower path length to /dev/console, and a WTO is then issued for each message in that file. The resulting messages from /dev/operlog do not appear on a console or the SYSLOG.

11.21 Message routing to z/OS

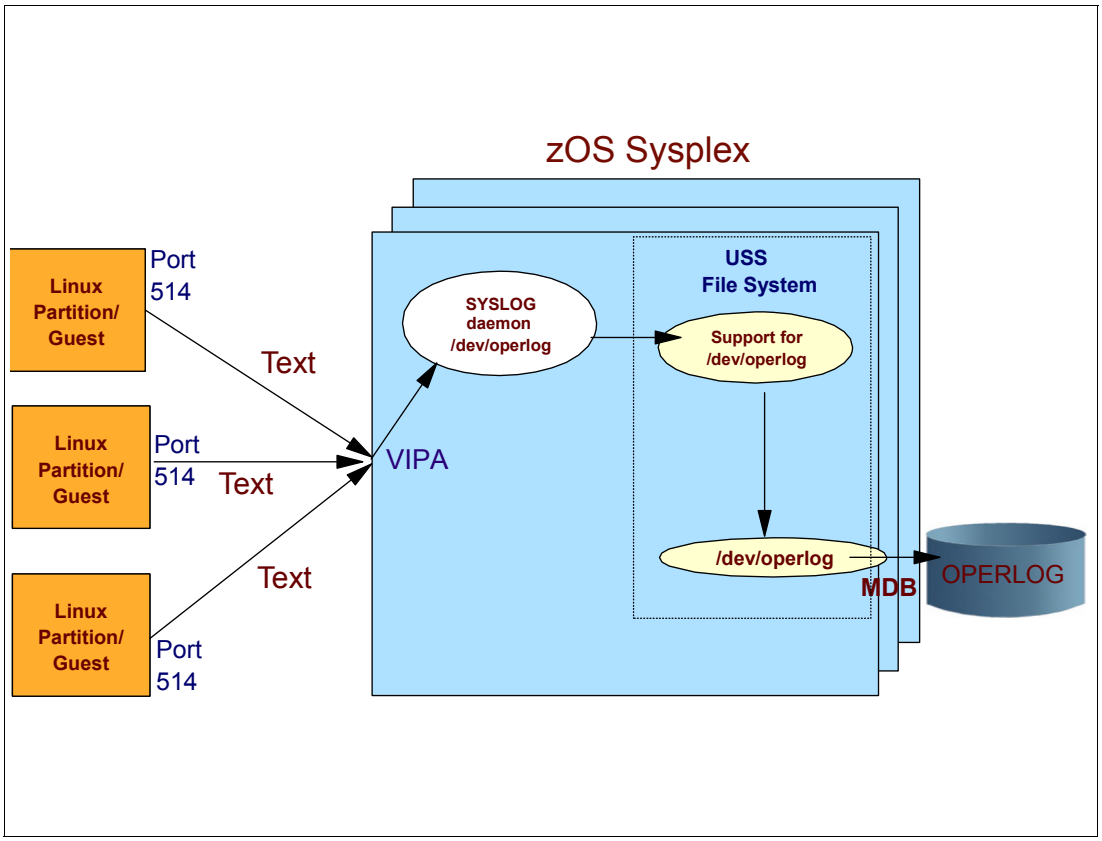


Figure 11-21 Message routing from Linux to z/OS

Message routing from Linux

Figure 11-21 displays an overview of how the new message routing works. You can use the message routing from every platform that supports the SYSLOG standards. Shown are the following:

- Three Linux partitions and z/OS are running on the network.

  zSeries customers are deploying functions and workloads on Linux on zSeries. Some of these functions and workloads may interact with z/OS images or perform functions on behalf of z/OS. For example:

  – Middle-tier servers such as Web servers or application servers

  – Communications Controller for Linux (CCL)

  In these scenarios it may be desirable to be able to integrate certain key messages from these Linux hosts into the z/OS environment and allow the z/OS operator to monitor key events that occur on the Linux hosts that may have an effect on z/OS operations as well.

- For messages from the Linux systems and some of the local z/OS messages, the new message integration support allows a common logging in z/OS for messages from multiple systems and provides a quick alternative way to log messages to the OPERLOG. Normally you would need to go through the system log of each system separately.

- To be able to log messages to OPERLOG, /dev/operlog builds a data block in a 4K storage area (around 40 lines).

11.22 syslogd command options

- It is now possible to start two instances of syslogd
  - One instance in local only mode (-i option)
  - One instance in network only mode (-n option)
- syslogd [-f conffile] [-m markinterval] [-p logpath] [-c] [-d] [-i] [-n] [-x] [-u] [-?]
  - -i option starts syslogd in local only mode - old option
  - -n option starts syslogd in network only mode
  - -x option causes syslogd to avoid resolver calls for converting IP addresses to hostnames
- Using both local and network logging, recommended to use two instances of syslogd
  - This ensures that local syslogd logging is not adversely affected by the amount of remote messages being forwarded to z/OS

Figure 11-22 syslogd command options

syslogd command options

The **syslogd** command has some new options in z/OS V1R8, as follows:

```
syslogd [-f conffile ] [-m markinterval ] [-p logpath ] [-c ] [-d ] [-i ] [-n ]
[-x ] [-u ] [-? ]
```

Where:

-i This option starts syslogd in local only mode - old option.

-n This option starts syslogd in network only mode.

-x This option causes syslogd to avoid resolver calls for converting IP addresses to hostnames.

Using both local and network logging, we recommend that you use two instances of syslogd, as shown in Figure 11-23 on page 598. This ensures that local syslogd logging is not adversely affected by the amount of remote messages being forwarded to z/OS.

11.23 syslogd defined instances

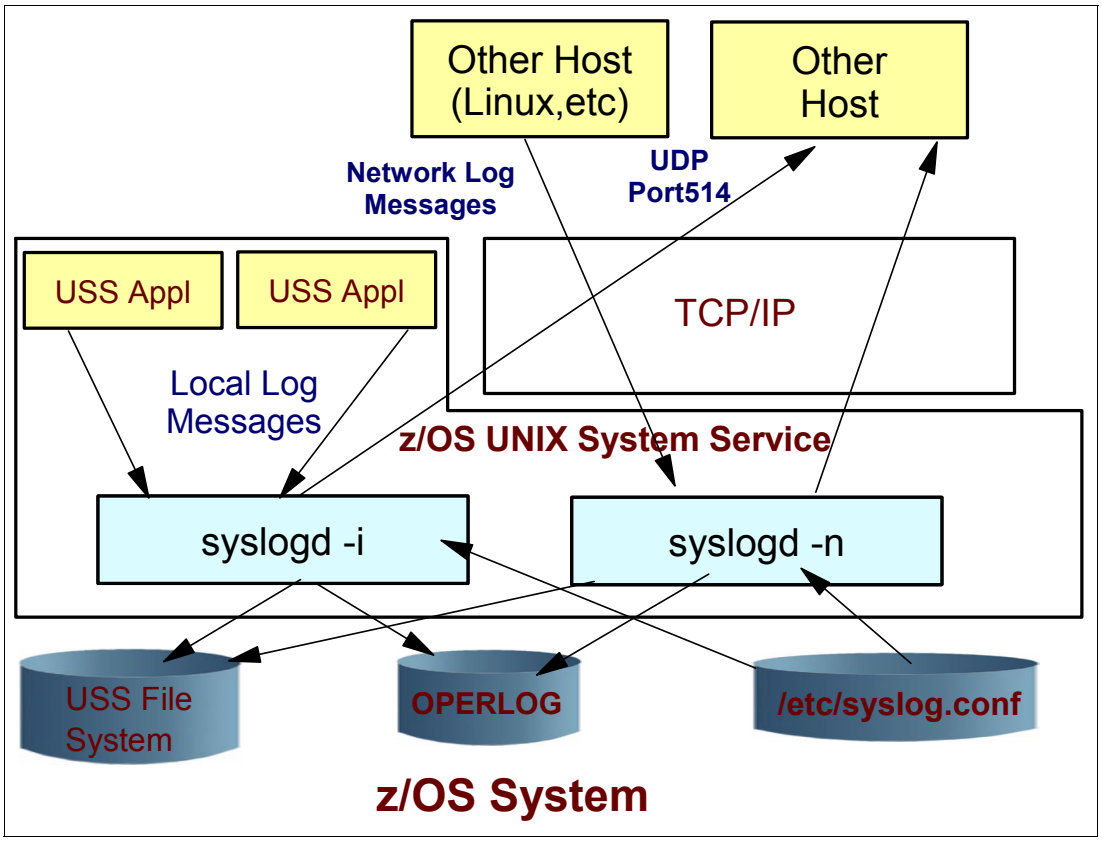


Figure 11-23 Flow of syslogd messages to the defined instances

syslogd message flow

Figure 11-23 shows how the Linux messages to be placed into the z/OS OPERLOG are passed to the defined instance. The normal syslogd messages are passed to the existing instance.

The ability to receive messages from remote syslogd instances can be very useful in providing a consolidated log of messages from multiple hosts into a consolidated z/OS message log. For example, in scenarios where you are running Linux hosts on zSeries processors, and these Linux hosts are performing processing in cooperation with or on behalf of z/OS systems, you might want to have certain important messages generated on the Linux hosts visible from a z/OS system. This capability would enable z/OS operators to be alerted of specific conditions on the Linux hosts that might require actions to be taken locally or on the Linux hosts.

When deciding that this remote logging capability is useful in an environment, there are several configuration considerations that should be examined prior to enabling this function. It is also important to note that the syslogd remote logging capability can work in both directions: the z/OS syslogd can forward some of its messages to another remote syslogd instance, or the z/OS syslogd instance can be the receiver of remote syslogd messages. The considerations described in this section focus primarily on the latter scenario, where the z/OS syslogd is the recipient of remote messages.

11.24 syslogd configuration file

- /etc/syslog.conf - configuration file
- Write all messages with priority crit and higher that arrive from host 192.168.0.6 to the OPERLOG log stream:
  - (192.168.0.6).*.crit /dev/operlog
- Write all messages with priority crit and higher that arrive from any host with IP address in the range 192.168.0.0 to 192.168.0.255 to the OPERLOG log stream:
  - (192.168.0.6/24).*.crit /dev/operlog
- Configuration file for syslogd client (Linux)
  - All messages are routed to z/OS V1R8 system console
    - *.* @10.1.40.4

Figure 11-24 Defining the Linux message integration in the syslogd configuration file

syslogd configuration file

With z/OS V1R8, the format of the syslog.conf entry is extended. With this change filtering of messages for specified hosts or networks is possible. This allows administrators to limit what remote messages are logged and where. You can identify which messages the z/OS syslogd should process by identifying the known remote syslogd instances from which you expect to receive messages by defining them in the syslog.conf configuration file, as follows:

- Write all messages with priority crit and higher that arrive from host 192.168.0.6 to the OPERLOG log stream:

  ```
  (192.168.0.6).*.crit /dev/operlog
  ```

- Write all messages with priority crit and higher that arrive from any host with IP address in the range 192.168.0.0 to 192.168.0.255 to the OPERLOG log stream:

  ```
  (192.168.0.6/24).*.crit /dev/operlog
  ```

Note: If using IP addresses, as shown in previous examples, the IP address can be followed by an optional forward slash and a number representing the number of significant bits of the address. This is called the prefix length. The prefix length provides a means to indicate that a condition applies to all IP addresses that have the bit pattern for the specified number of bits.

This device file, /dev/operlog, is reserved for syslog daemon use only. However, there is no way to restrict someone from writing messages directly to /dev/operlog.

Customize the parameter file for the syslogd client

The configuration for the syslogd is similar to the server. We used a z/OS V1R7 system to act as a client in our test scenario. Figure 11-25 shows an example of the client configuration for syslogd. The meaning of this file is that all messages are routed to the z/OS V1R8 system. At the z/OS V1R8 system the messages will appear on the system console.

```
*.*                    @10.1.40.4
```

Figure 11-25 Example of syslog.conf for the client

11.25 Start procedure for syslogd

- Now recommended as a started task
- Starting syslogd instances
  - One instance in local mode (-i option)
  - One instance in network mode (-n option)

```
//TCPLOG    PROC
//TCPSYS  EXEC PGM=SYSLOGD,REGION=30M,TIME=NOLIMIT,
//  PARM=('POSIX(ON) ALL31(ON)',
//  '/-f /etc/operlog.server ')
//SYSPRINT  DD SYSOUT=*
//SYSIN     DD SYSOUT=*
//SYSERR    DD SYSOUT=*
//SYSOUT    DD SYSOUT=*
//
```

Figure 11-26 syslogd start procedure

syslogd start procedure

We recommend that the SYS1.PROCLIB procedure that follows be used when specifying multiple syslogd instances.

```
//TCPLOG    PROC
//TCPSYS  EXEC PGM=SYSLOGD,REGION=30M,TIME=NOLIMIT,
//  PARM=('POSIX(ON) ALL31(ON)',
//  '/-f /etc/operlog.server ')
//SYSPRINT  DD SYSOUT=*
//SYSIN     DD SYSOUT=*
//SYSERR    DD SYSOUT=*
//SYSOUT    DD SYSOUT=*
//
```

11.26 syslogd availability considerations

- Ensure that there is a process in place to restart a z/OS syslogd after a failure that results in the termination of syslogd
- This can be accomplished by placing syslogd in the:
  - AUTOLOG list in the TCP/IP profile
  - This enables TCP/IP to initially start the syslogd instance as a started task
  - Also enables TCP/IP to monitor whether syslogd has a socket bound to the syslogd port used for remote message receipt (UDP port 514)

Note: Use only for single-stack INET configurations. For multiple stacks, use automation to restart syslogd.

Figure 11-27 syslogd availability considerations

syslogd availability

syslogd availability is important to the logging of both local and remote messages. The following configuration considerations can help improve the availability of the z/OS syslogd remote logging services:

- Ensure that there is a process in place to restart a z/OS syslogd after a failure that results in the termination of syslogd. This can be accomplished by placing syslogd in the AUTOLOG list in the TCP/IP profile. This enables TCP/IP to initially start the syslogd instance as a started task, and also enables TCP/IP to monitor whether syslogd has a socket bound to the syslogd port used for remote message receipt (UDP port 514). This approach works well for single-stack INET configurations. If multiple TCP/IP stacks are deployed on a single system (that is, a CINET configuration), you should not use the AUTOLOG list. Alternatively, an installation can use any available automated operations software package to automate the start and restart of syslogd. For more information about the AUTOLOG statement, refer to *z/OS Communications Server: IP Configuration Reference*, SC31-8776.
- On MVS systems that are not part of a sysplex and that have TCP/IP stacks with multiple network interfaces, using a static VIPA address can provide for better availability characteristics should a specific network interface or network experience an outage. This involves configuring a static VIPA in the TCP/IP profile or reusing an existing one. No special configuration of the local syslogd is needed. However, you should configure the remote syslogd instances to use the static VIPA address as the destination address when forwarding messages. This can typically be accomplished by either specifying the static

VIPA as the destination address, or specifying a host name that maps to the static VIPA in the remote syslogd configuration.

- When the recipient syslogd instance is running in a sysplex environment, additional availability features are available that should be explored. For example, you can use a multiple application-instance dynamic VIPA (DVIPA) to represent the syslogd instance on a given system, and the remote syslogd instances are configured to use that DVIPA address as the destination IP address for the messages they forward. In this configuration, if a failure to the MVS system or TCP/IP stack on which the syslogd instance is running occurs, the DVIPA is automatically moved to a predefined backup system that is currently active. This enables the syslogd instance on that backup system to begin processing these remote messages in a transparent manner. In this configuration, using the MVS operations log (OPERLOG) as the destination provides additional benefits, because the operations log is implemented as a Coupling Facility log stream that any system in the sysplex can access.

12

z/OS UNIX PARMLIB members

This chapter describes the z/OS UNIX PARMLIB members. It explains the differences between the BPXPRMxx and BPXPRMFS members and defines the UNIX System Service parameters in each member.

© Copyright IBM Corp. 2011. All rights reserved.

12.1 BPXPRMxx PARMLIB member

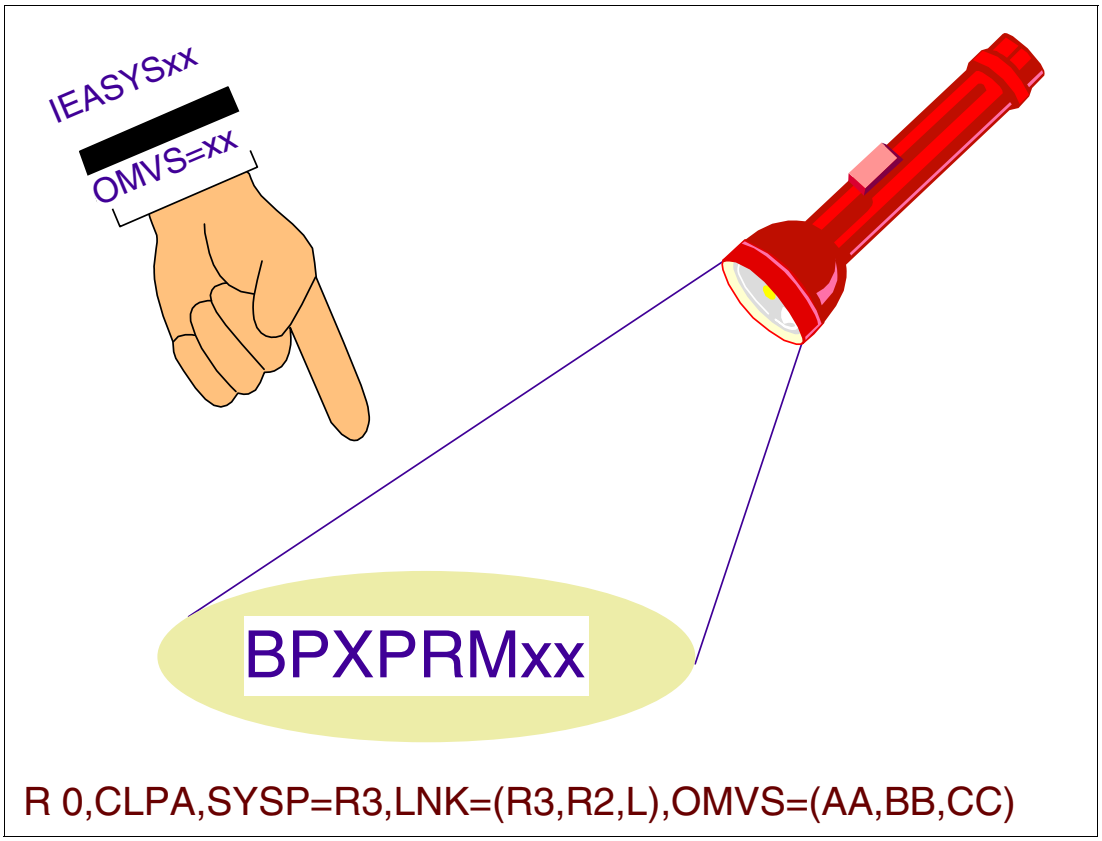


Figure 12-1 z/OS UNIX BPXPRMxx PARMLIB member

BPXPRMxx PARMLIB member

BPXPRMxx contains the parameters that control the z/OS UNIX System Services (z/OS UNIX) environment.

To specify which BPXPRMxx PARMLIB member to start with, the operator can include OMVS=xx in the reply to the IPL message or can include OMVS=xx in the IEASYSxx PARMLIB member. The two alphanumeric characters, represented by xx, are appended to BPXPRM to form the name of the BPXPRMxx PARMLIB member.

You can use multiple PARMLIB members to start OMVS. This is shown by the following reply to the IPL message:

```
R 0,CLPA,SYSP=R3,LNK=(R3,R2,L),OMVS=(AA,BB,CC)
```

The PARMLIB member BPXPRMCC would be processed first, followed by and overridden by BPXPRMBB, followed by and overridden by BPXPRMAA. This means that any parameter in BPXPRMAA has precedence over the same parameter in BPXPMRBB and BPXPRMCC.

You can also specify multiple OMVS PARMLIB members in IEASYSxx. For example:

```
OMVS=(AA,BB,CC)
```

To modify BPXPRMxx PARMLIB settings without re-IPLing, you can use the SETOMVS operator command, or you can dynamically change the BPXPRMxx PARMLIB members that are in effect by using the SET OMVS operator command.

12.2 BPXPRMFS PARMLIB member

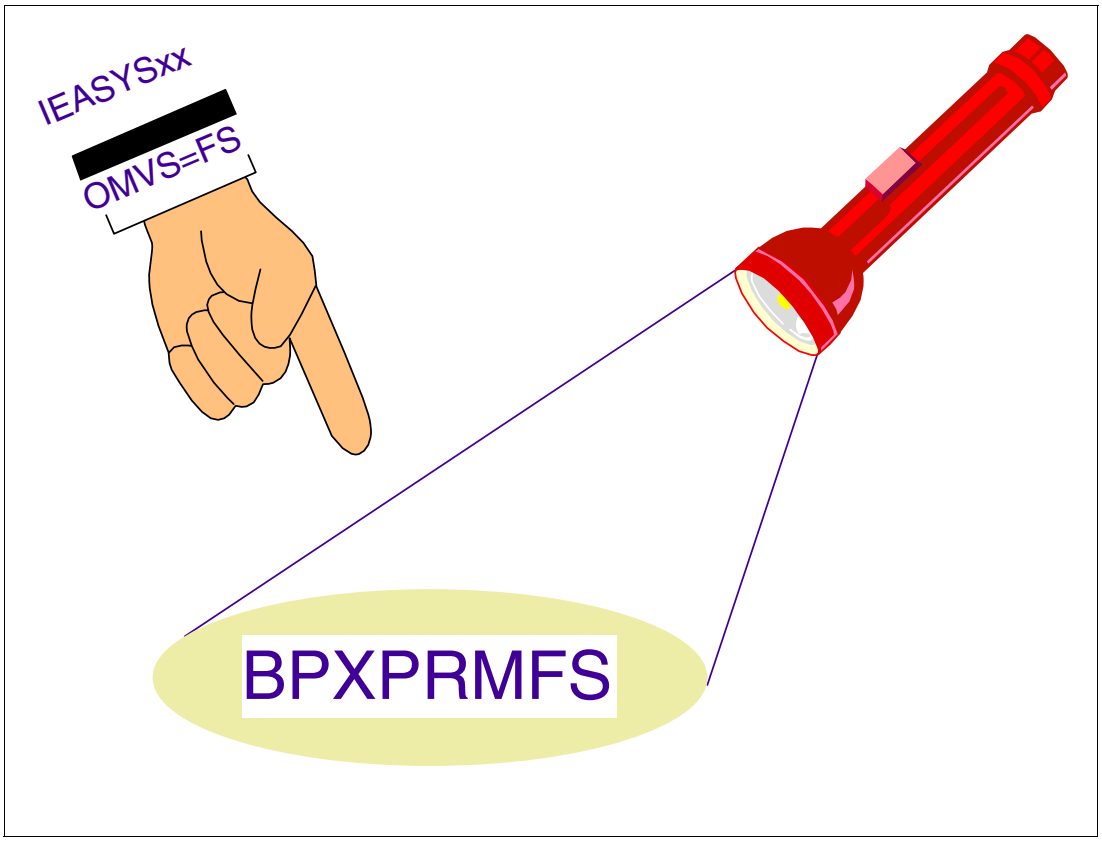


Figure 12-2 BPXPRMFS PARMLIB member

BPXPRMFS PARMLIB member

Starting with the OS/390 V2R6 ServerPac, IBM provides a z/OS UNIX System Services PARMLIB member for HFS and socket definitions, BPXPRMFS.

The BPXPRMFS member reflects the restored file system. The BPXPRMFS PARMLIB member is shipped in CPAC.PARMLIB and contains:

```
FILESYSTYPE  TYPE(HFS)
ENTRYPOINT(GFUAINIT)
PARM(' ')
```

The ALLOCDS job of the ServerPac allocates HFS data sets as you defined in the installation variables option of the installation dialog. The RESTFS job MOUNTs the new ROOT file system and creates mount points for and MOUNTs the additional file systems. The BPXPRMFS member of CPAC.PARMLIB is updated to reflect the file system structure.

It is up to the customer whether to use this additional PARMLIB member or not. Because you have to IPL when one of these parameters is changed, you should provide the HFS and socket information in a separate member.

12.3 BPXPRMxx control keywords

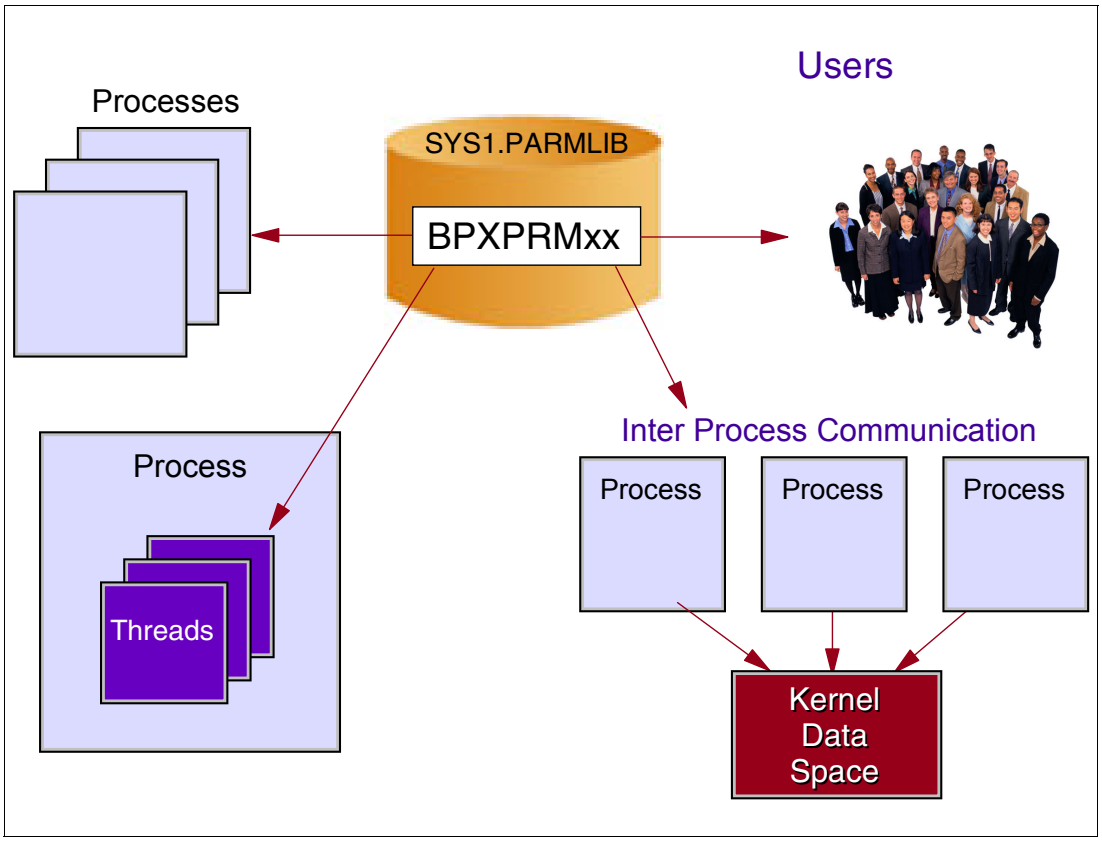


Figure 12-3 BPXPRMxx keywords that control z/OS UNIX processing

Using BPXPRMxx PARMLIB members

The BPXPRMxx PARMLIB member contains the parameters that control z/OS UNIX processing and the file system. The MVS system uses these values when initializing z/OS UNIX System Services.

A sample is shipped in SYS1.SAMPLIB(BPXPRMxx). Copy and rename (if necessary) this member to PARMLIB. Customize this PARMLIB member according to your installation requirements.

IBM recommends that you have two BPXPRMxx members, one that specifies system limits and one that specifies file system setup. This makes it easier to migrate from one release to another, especially when using the ServerPac method of installation.

You can also define system symbols in the IEASYSxx PARMLIB member to enable common BPXPRMxx members to be shared by systems. Symbols like system name (&SYSNAME.) can be used in the BPXPRMxx member, specifically when referring to HFS data set names, in order to have different HFS data sets mounted as the root on each system in the sysplex.

12.4 BPXPRMxx PARMLIB member

```
{MAXPROCSYS(nnnnn)}
{MAXPROCUSER(nnnnn)} - Can specify in RACF OMVS segment
{MAXUIDS(nnnnn)}
{MAXFILEPROC(nnnnn)} - Can specify in RACF OMVS segment
{MAXTHREADTASKS(nnnnn)}
{MAXTHREADS(nnnnnn)} - Can specify in RACF OMVS segment
{MAXPTYS(nnnnn)}
{MAXRTYS(nnnnn)}
{MAXFILESIZE(nnnnn|NOLIMIT)}
{MAXCORESIZE(nnnnn)}
{MAXASSIZE(nnnnn)}       - Can specify in RACF OMVS segment
{MAXCPUTIME(nnnnn)}      - Can specify in RACF OMVS segment
{MAXMMAPAREA(nnnnn)} - Can specify in RACF OMVS segment
{MAXSHAREPAGES(nnnnn)}
```

Figure 12-4 BPXPRMxx members that control active processes

Controlling active processes

Figure 12-4 shows all BPXPRMxx PARMLIB member parameters you can set up to influence user logon, active processes, file handling, and storage requirements.

You should handle these parameters with care because the values you specify for these statements are interrelated. For example:

- MAXPROCSYS
- MAXPROCUSER
- MAXUIDS

You should always monitor all MAXxxx settings before you change any statements. It makes no sense to allow more users to access z/OS UNIX System Services when you do not provide enough TTYs. Each user or process entering z/OS UNIX System Services needs a pseudo-terminal (pseudo-TTY).

Beginning in OS/390 V2R8, you can specify the parameters shown in the OMVS segment of a user profile.

For example, if you specify MAXPROCSYS, that means define the maximum number of processes that can be active at the same time, using 3000, and in the MAXPROCUSER statement you allow 200 processes per user and you allow 3000 users to be active at the same time (MAXUIDS), this would not fit.

12.5 Controlling the number of processes

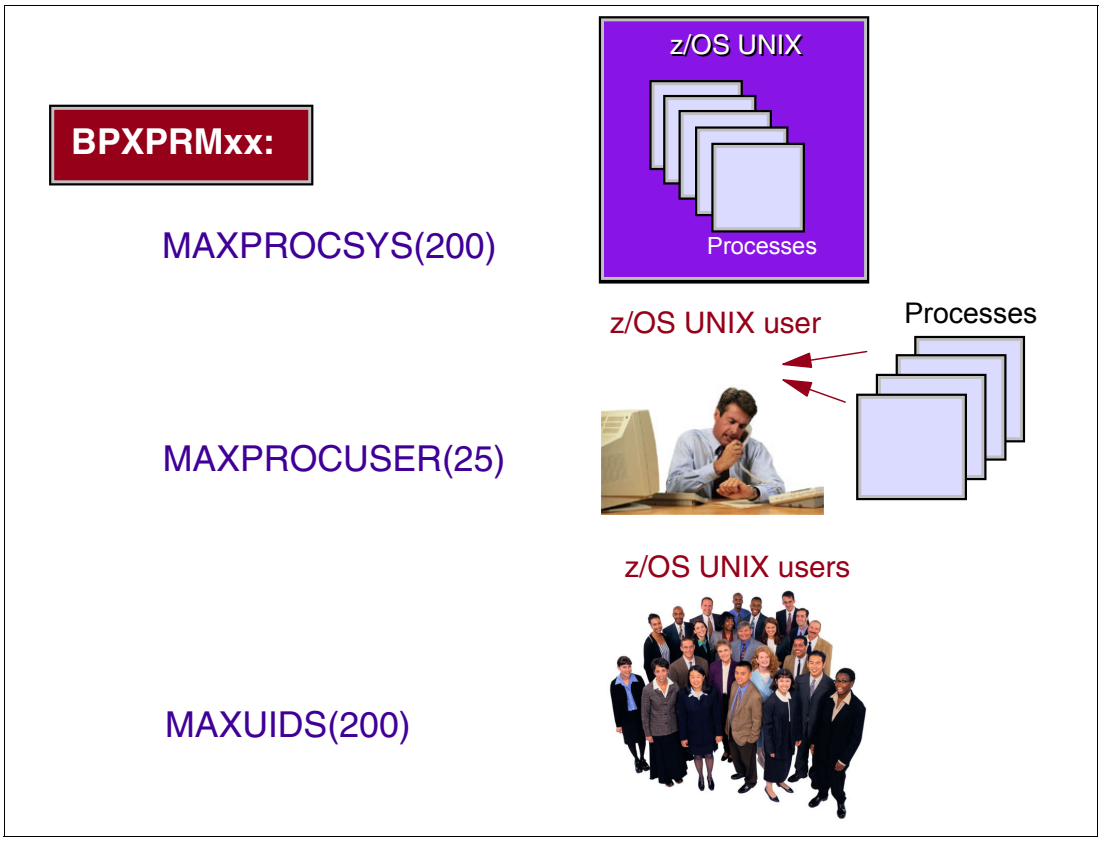


Figure 12-5 BPXPRMxx members that control the number of processes

Statements controlling processes

The MAXPROCSYS statement specifies the maximum number of processes that z/OS UNIX will allow to be active at the same time in the system. The range is 5 to 32767; the default is 200. Specify only the maximum number of processes you expect z/OS UNIX MVS to support, because:

- Some storage allocations in the kernel address space are based on the MAXPROCSYS value. A larger value means that more pageable storage will be allocated.
- Most z/OS UNIX MVS processes use an MVS address space, which uses storage. Therefore, avoid specifying a higher value for MAXPROCSYS than the system can support.
- If MAXPROCSYS is set too high relative to the maximum number of initiators the Workload Manager (WLM) can provide, response time delays or failures for fork() or spawn() could occur because users have to wait for initiators.

The MAXPROCUSER statement specifies the maximum number of processes that a single OMVS user ID (UID) is allowed to have active at the same time, regardless of how the process became a z/OS UNIX MVS process. The range is 3 to 32767; the default is 25.

If system resources are constrained, setting a low MAXPROCUSER value will limit a z/OS UNIX user's consumption of processing time, virtual storage, and other system resources.

Note: PROCUSERMAX can be specified in the OMVS segment of a user profile to set the maximum number of processes for this UID:

```
ALTUSER userid OMVS(PROCUSERMAX(nnnn))
```

The MAXUIDS statement specifies the maximum number of unique OMVS user IDs (UIDs) that can use z/OS UNIX MVS services at the same time. The UIDs are for interactive users or for programs that requested z/OS UNIX MVS services. The range is 1 to 32767; the default is 200.

12.6 Resource limits for processes

BPXPRMxx:

- MAXFILEPROC(64)
- MAXPTYS(256)
- MAXCORESIZE(4194304)
- MAXASSIZE(41943040)
- MAXCPUTIME(1000)

Figure 12-6 Keywords that control resource limits for processes

Controlling resource limits

Some of the system resource limits assigned to processes can be modified by a process using the **`setrlimit`** callable service. Only an authorized process can change the limits beyond what is specified in BPXPRMxx. A resource limit is a pair of values; one specifies the current (soft) limit and the other a maximum (hard) limit. The values assigned in the BPXPRMxx which can be changed by the setrlimit callable service.

MAXFILEPROC value

The MAXFILEPROC value specifies the maximum number of files that a single z/OS UNIX user is allowed to have concurrently active or allocated. The range is 3 to 65535; the default and the value in BPXPRMXX is 64.

- The minimum value of 3 supports the standard files for a process: stdin, stdout, and stderr.
- The value needs to be larger than 3 to support shell users. If the value is too small, the shell may issue the message `File descriptor not available`. If this message occurs, increase the MAXFILEPROC value.

Use the RACF ADDUSER or ALTUSER command to specify the FILEPROCMAX limit on a per-user basis as follows:

```
ALTUSER userid OMVS(FILEPROCMAX(nnnn))
```

MAXCORESIZE value

The MAXCORESIZE value specifies the maximum core dump file size (in bytes) that a process can create. The default is 4 MB. If 0 is specified, no core files will be created by the process.

MAXASSIZE value

The MAXASSIZE value indicates the address space region size. The range is from 10 MB to 2 GB. The default is 40 MB. If there are multiple processes within an address space, the processes share the same limits for MAXASSIZE. Use the RACF ADDUSER or ALTUSER command to specify the ASSIZEMAX limit on a per-user basis as follows:

```
ALTUSER userid OMVS(ASSIZEMAX(nnnn)
```

MAXCPUTIME value

The MAXCPUTIME is the time limit (in seconds) for processes that were created by rlogind and other daemons. You can set a system-wide limit in BPXPRMxx and then set higher limits for individual users. Use the RACF ADDUSER or ALTUSER command to specify the CPUTIMEMAX limit on a per-user basis as follows:

```
ALTUSER userid OMVS(CPUTIMEMAX(nnnn))
```

MAXPTYS value

Use MAXPTYS to manage the number of interactive shell sessions, where each interactive session requires one pseudo-TTY pair. Do not specify an arbitrarily high value for MAXPTYS. But, because each user may have more than one session, it is recommended that you allow four pseudo-TTY pairs for each user (MAXUIDS * 4). Specify a MAXPTYS value that is at least twice the MAXUIDS value.

12.7 MAXFILEPROC statement

- Large servers were approaching the original descriptor limit of 64K - such as TN3270 - CICS
- The limit was raised from 64K to 128K in V1R6 for short term relief
- The limit is being raised from 128K to 512K along with performance improvements in z/OS V1R8
  - Default for MAXFILEPROC is now 64000
  - The new limit is 524287
    - Descriptor values start at 0, so the highest descriptor number is 524286
- Benefit is constraint relief for servers with very large numbers of clients

Figure 12-7 New support for the MAXFILEPROC statement

MAXFILEPROC statement

The file descriptor limit is the maximum number of files per process for a z/OS UNIX user. In some z/OS UNIX environments with very large numbers of clients, the file descriptor limits are too small. In z/OS V1R6, the limit was increased from 64 KB up to 128 KB. This limit was increased again in z/OS V1R8 to 512 KB. Also, the performance was improved in this release.

Use MAXFILEPROC to set the maximum number of file descriptors that a single process can have open concurrently, such as all open files, directories, sockets, and pipes. By limiting the number of open files that a process can have, you limit the amount of system resources a single process can use at one time. The limit that was increased is the limit per user, not the whole system. The system itself has no limit. There are some ways to define and modify this limit in your system. The default is set in the BPXPRMxx PARMLIB member with the MAXFILEPROC parameter. Figure 12-8 on page 615 shows a sample entry from the BPXPRMxx PARMLIB member.

The default process descriptor limit is set at IPL or on an OMVS restart with the MAXFILEPROC() keyword of the BPXPRMxx PARMLIB member. The new limits are as follows:

- The default in z/OS V1R8 for MAXFILEPROC is now 64000.
- The new limit is 524287.

```
MAXFILEPROC(300000)                /* Allow at most 300000 open files per user
```

Figure 12-8 Sample of MAXFILEPROC parameter

12.8 Setting file descriptors

- To modify the system limit there are three ways to do:
  - Use the SET OMVS=xx command with BPXPRMxx MAXFILEPROC(nnnnnn)
  - Use the SETOMVS RESET=(xx) command with BPXPRMxx MAXFILEPROC(nnnnnn)
  - Use the SETOMVS MAXFILEPROC=nnnnnn command
- The limit for an individual process can be changed with:
  - SETOMVS PID=pid,MAXFILEPROC=nnnnnn

Figure 12-9 Commands to set the file descriptors using MAXFILEPROC

Modifying the MAXFILEPROC limit

There are three ways to modify the system limit. The first two choices will dynamically activate a whole BPXPRMxx PARMLIB member in your system, which means all statements you place into a BPXPRMxx member specified by the xx in the command will be activated.

1. Use the SET OMVS=xx command with BPXPRMxx MAXFILEPROC(nnnnnn).
2. Use the SETOMVS RESET=(xx) command with BPXPRMxx MAXFILEPROC(nnnnnn).
3. Use the SETOMVS MAXFILEPROC=nnnnnn command.

The third choice just modifies the MAXFILEPROC limit in your system and nothing else. The following shows the successful activation of a new descriptor limit:

```
SETOMVS MAXFILEPROC=10000
 BPXO015I THE SETOMVS COMMAND WAS SUCCESSFUL.
```

When you use a wrong value to modify the descriptor limit, you see an error message as shown in Figure 12-10 on page 617.

```
SETOMVS MAXFILEPROC=1000000
BPXO006I ERROR IN SETOMVS COMMAND. THE MAXFILEPROC 059
PARAMETER VALUE IS OUT OF THE ALLOWED RANGE OF 3 TO 524287.
BPXO012I ERRORS OCCURRED IN THE PROCESSING OF THE 060
SETOMVS COMMAND; NO VALUES WERE SET.
```

Figure 12-10 Wrong size to modify MAXFILEPROC

Set limit for a process

If you want to change the descriptor limit for an individual process, this can be done with the SETOMVS system command. Figure 12-11 shows the z/OS system command to modify the descriptor limit for a selected process by PID.

```
SETOMVS PID=33555255,MAXFILEPROC=3000
BPXO015I THE SETOMVS COMMAND WAS SUCCESSFUL.
```

Figure 12-11 Modify MAXFILEPROC with the SETOMVS command

12.9 Setting file descriptor for a single user

- Use of the FILEPROCMAX parameter in the users OMVS segment
  - ALU LUTZ OMVS(FILEPROCMAX(10000))
- Use scalable services or system APIs
- Use the SETOMVS system command
  - SETOMVS PID=33555255,MAXFILEPROC=3000
  - BPXO015I THE SETOMVS COMMAND WAS SUCCESSFUL.
  - SETOMVS MAXFILEPROC=10000
  - BPXO015I THE SETOMVS COMMAND WAS SUCCESSFUL.

Figure 12-12 Commands to set the file descriptor for a single user or process

Set file descriptor maximum for a single user

Use MAXFILEPROC to set the maximum number of file descriptors that a single process can have open concurrently, such as all open files, directories, sockets, and pipes. By limiting the number of open files that a process can have, you limit the amount of system resources a single process can use at one time.

The following commands modify the system limit:

- Use of the FILEPROCMAX parameter in the users OMVS segment.
- Use scalable services or system APIs.
- Use the SETOMVS system command.

You can set the limit for individual users by modifying their RACF OMVS segment.
Figure 12-13 shows the RACF command to modify the descriptor limit for individual users. In this example we increase the limit for the user ID LUTZ to 10,000 open objects.

```
ALU LUTZ OMVS(FILEPROCMAX(10000))
```

Figure 12-13 Sample command for modifying the limit

Figure 12-14 on page 619 shows the modified OMVS segment for user ID LUTZ.

```
LU LUTZ NORACF OMVS
OMVS INFORMATION
----------------
UID= 0000001014
HOME= /u/lutz
PROGRAM= /bin/sh
CPUTIMEMAX= NONE
ASSIZEMAX= NONE
FILEPROCMAX= 00010000
PROCUSERMAX= NONE
THREADSMAX= NONE
MMAPAREAMAX= NONE
```

Figure 12-14 Sample listing of the OMVS segment

Note: You can use the USS_MAXSOCKETS_MAXFILEPROC check provided by IBM Health Checker for z/OS to determine whether the MAXFILEPROC value is set too low.

12.10 Display BPXPRMxx limits

- BPXPRMxx LIMMSG=NONE | SYSTEM | ALL
  - NONE - No console messages are to be displayed when any of the parmlib limits have been reached
  - SYSTEM - Console messages are to be displayed for all processes that reach system limits and in addition, messages are to be displayed for each process limit of a process if:
    - The process limit or limits are defined in the OMVS segment of the owning User ID
    - The process limit or limits have been changed with a SETOMVS PID=pid,process_limit
  - ALL - Console messages are to be displayed for the system limits and for the process limits, regardless of which process reaches a process limit

Figure 12-15 How to display BPXPRMxx parmlib limits being reached

LIMMSG statement

Use the LIMMSG statement to control the display of console messages that indicate when parmlib limits are reaching critical levels. The LIMMSG statement in BPXPRMxx parmlib member controls message activity for limit checking. You can specify whether no console messages are to be displayed when any of the parmlib limits have been reached (NONE); console messages are to be displayed for all processes that reach system limits and for certain process limits (SYSTEM); or console messages are to be displayed for all the system limits and the process limits (ALL).

The LIMMSG options can be changed with the SETOMVS LIMMSG command. The LIMMSG value appears in the D OMVS,O display.

LIMMSG warning messages

If the LIMMSG statement is specified with SYSTEM or ALL, a warning console message appears whenever a limit reaches 85%, 90%, 95%, and 100%, identifying the process that has reached the limit. When the limit reaches the next limit level, the prior message is removed from the console and a new message indicates the new limit level that has been reached. When the limit falls below the 85% threshold, a message indicates that the resource shortage has been relieved.

Change the LIMMSG setting

Changing from LIMMSG(ALL) or LIMMSG(SYSTEM) to LIMMSG(NONE) with the **SETOMVS** command stops any further monitoring of resources. However, existing outstanding

messages are not deleted from the panel for a process until the limit is relieved for that process.

```
SETOMVS LIMMSG=[NONE|SYSTEM|ALL]
BPXO029I LIMMSG CHANGED FROM oldvalue TO newvalue
```

Attention: When LIMMSG(ALL) is in effect, a large number of messages can be issued. This option is best suited for use during the initial configuration of a system, when the installation has not yet determined the optimal settings for the z/OS UNIX parmlib limits.

LIMMSG messages

Several messages are issued for mount table limit monitoring. As the capacity of the mount table is approached, the following message is displayed:

```
BPXI043E MOUNT TABLE LIMIT HAS REACHED <nn>% OF ITS CURRENT CAPACITY OF
<current limit>.
```

where <nn> has the value of 85, 90, 95, or 100; the message is updated when the percentage has changed. The message is DOMed when the percentage decreases below 85%, and the following message is issued:

```
BPXI044I RESOURCE SHORTAGE FOR MOUNT TABLE HAS BEEN RELIEVED.
```

12.11 Memory mapped files

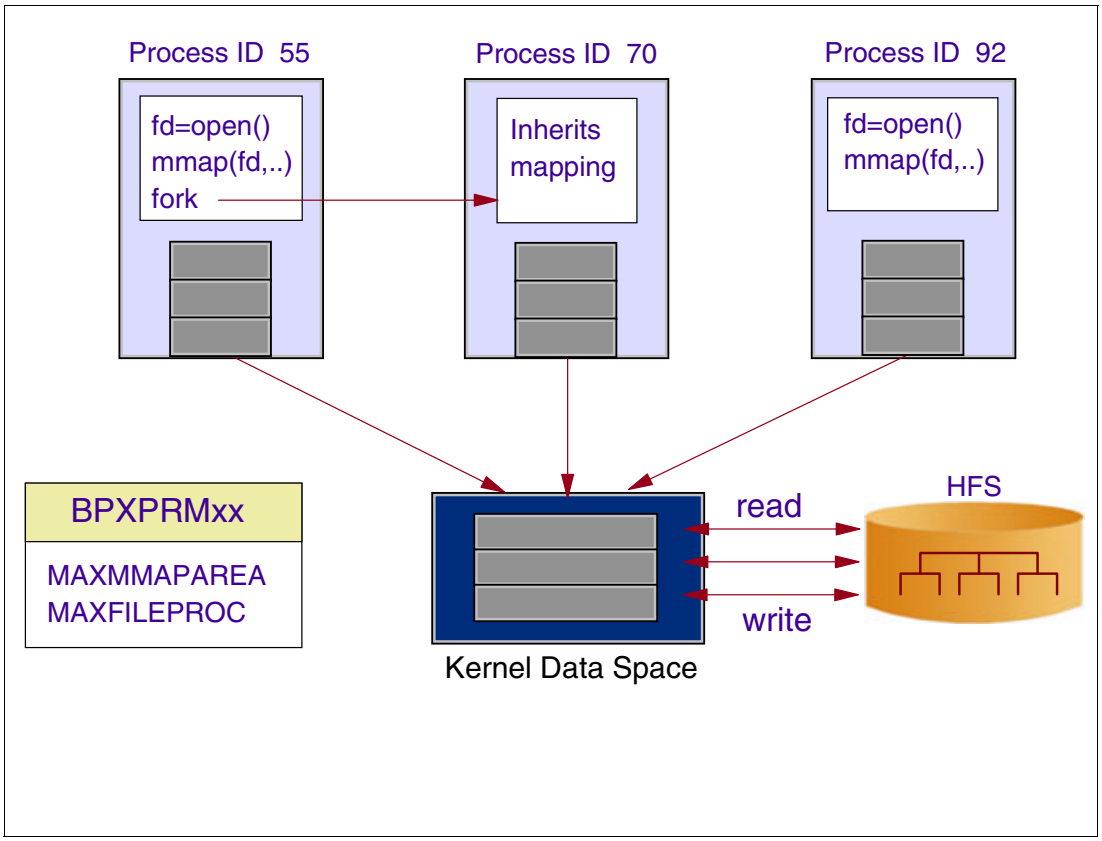


Figure 12-16 BPXPRMxx members for memory mapped files

Memory mapped files

Memory mapped files allow multiple processes to share data in a file that is mapped in storage. Access to the file is through address space manipulation instead of read/write services. The z/OS UNIX kernel stores the pages of data from the file in a data space. A child process will inherit the mapping addresses from the parent and thereby have access to the data. Other processes will need to open the file and can then map to the same data pages in the data space.

There are two BPXPRMxx parameters which control the use of memory mapped files:

- MAXFILEPROC specifies how many files a user can have open or allocated; it also controls the number of memory mapped files a user can have open.
- MAXMMAPAREA specifies the maximum amount of data space storage (in pages) that can be allocated for memory mapped files. Use the RACF ADDUSER or ALTUSER command to specify the MMAPAREAMAX limit on a per user basis as follows:

```
ALTUSER userid OMVS(MMAPAREAMAX(nnnn))
```

Using memory mapped files causes system memory to be consumed. For each page (KB) that is memory mapped, 96 bytes of ESQA are consumed when a file is not shared with anybody else. When a file is shared, each additional user causes 32 bytes of ESQA to be consumed for each shared page. For example, if 4096 pages are used for memory mapped files, the first user will cause 384 KB (96 * 4000) of ESQA to be consumed. Each additional user will cause 128 KB (32 * 4000) of ESQA to be consumed. ESQA storage is consumed when the mmap() function is invoked rather than when the page is accessed.

12.12 Controlling thread resources

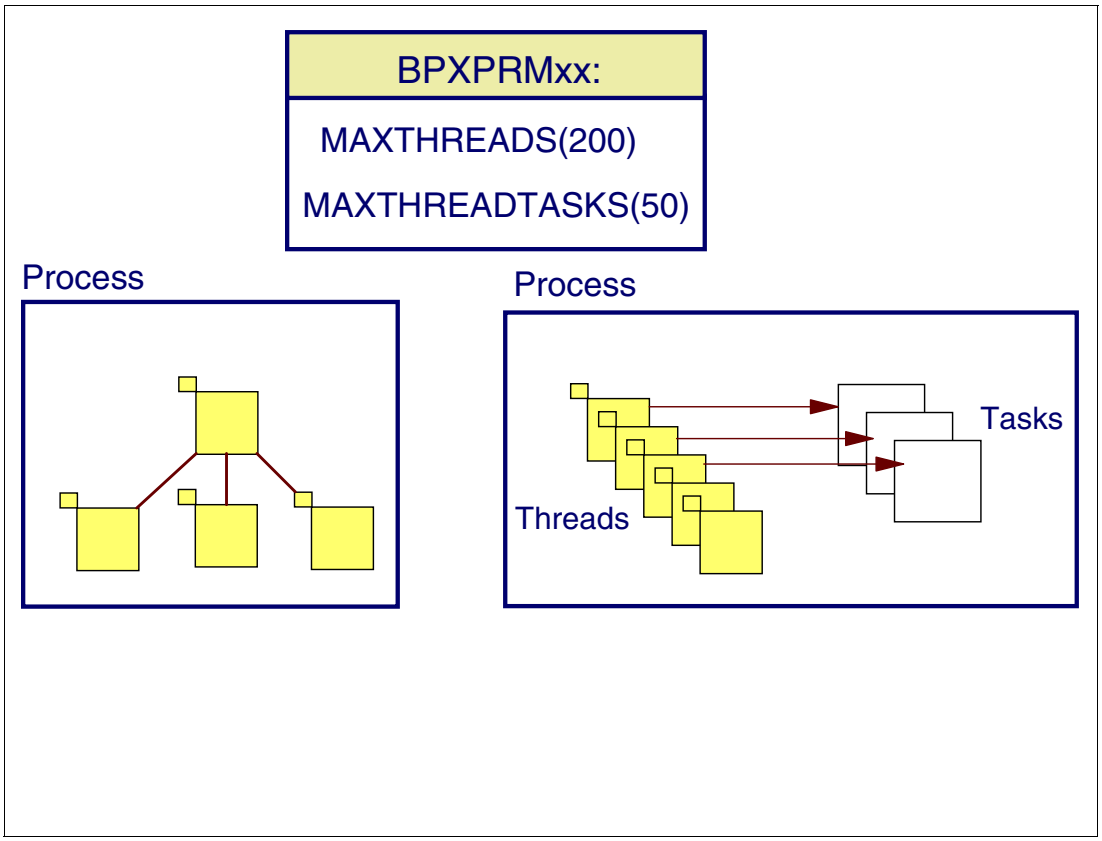


Figure 12-17 Controlling threads

Controlling resources for threads

Threads provide support for multiple separate units of dispatchable work within a process. A z/OS UNIX thread can be compared with an MVS task. Threads allow for concurrent and asynchronous processing without the additional overhead associated with creating a new address space.

The thread support is intended for multitasking server applications that require multiple concurrent execution streams. A program using threads can have significant performance benefits in a multiprocessor environment where each thread in a process can run on an individual processor.

- The MAXTHREADS value specifies the maximum number of pthread_created threads, including those running, queued, and exited but not detached, that a single process can have currently active. Specifying a value of 0 prevents applications from using pthread_create. The range is 0 to 100000; the default is 200.
- The MAXTHREADTASKS value specifies the maximum number of MVS tasks created with pthread_create (BPX1PTC) that a single user may have concurrently active in a process. The range is 0 to 32768; the default is 50.

  MAXTHREADTASKS lets you limit the amount of system resources available to a single user process.

Individual processes can alter these limits dynamically if they have sufficient authority.

MVS tasks and threads

Each thread that is created with pthread_create runs as an MVS subtask of the initial pthread-creating task (IPT). The IPT is the task that issued the first pthread_create call within the address space. When all the threads created with pthread_create and the IPT have ended, the next task in the address space to issue a pthread_create call is made the IPT. The IPT in the example in Figure 12-17 is the thread running Main.

The difference between these two keywords is that MAXTHREADS specifies the limit for how many threads a process can have active. This number includes the number of threads that are executing on MVS tasks and the number that are waiting for execution. MAXTHREADTASKS specifies the limit for how many MVS tasks can be created per process to schedule threads. This number limits the number of threads that can be executing at the same time in a process. In the pictured example, a process has created 5 threads, but only three MVS tasks, so two threads will be queued until a task becomes available.

12.13 Creating a process using fork()

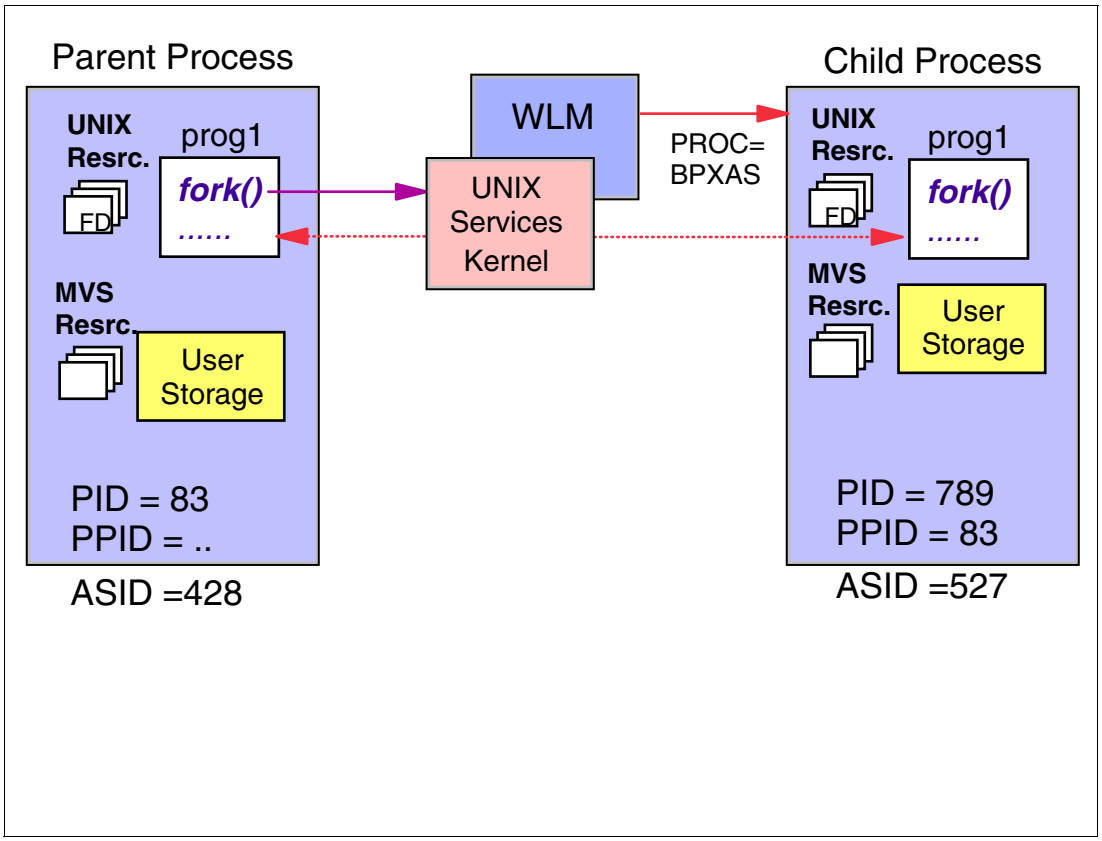


Figure 12-18 Creating a process using the fork() function

Create a process using fork()

Fork() is a POSIX/XPG4 function that creates a duplicate process referred to as a *child* process. The process that issues the fork() is referred to as the *parent* process.

With z/OS UNIX, a program that issues a fork() function creates a new address space which is a copy of the address space where the program is running. The fork() function does a program call to the kernel, which requests WLM to create the child process address space. The storage contents of the parent address space are then copied to the child address space.

After the fork() function completes, the program in the child AS will start at the same instruction as the program in the parent AS. Control is returned to both programs at the same point. The only difference that the program sees is the return code from the fork() function. A return code of zero is returned to the child after a successful fork(). The return code for the parent is the child process ID (PID).

Also, any UNIX resources (pipes, sockets, files) accessible via opened File Descriptors in the parent are propagated to the new address space. z/OS resources such as DD allocations, cross-memory resources, and ENQ serializations are *not* propagated to the child address space.

Program prog1

The program prog1 in the parent AS issues the fork() function, the kernel is called to perform the action. The kernel calls the Workload Manager (WLM) to create a new address space.

WLM uses the BPXAS procedure from PROCLIB to initialize the address space. WLM dynamically manages the number of address spaces started for UNIX processes, in order to meet goals set via ICS/IPS, or WLM policy in goal mode.

Child address space

Once the child address space has been created, the child gets the required storage from a STORAGE request. The kernel then copies the contents of the parent AS to the child AS using the MVCL instruction. After some additional setup, the kernel returns codes to both parent and child programs. The program in the child AS gets control at the same point as the program in the parent AS. The only difference is the return code from the fork() function.

The child address space is almost an identical copy of the REGION storage in parent address space. User data, for example private subpools, and system data, like RTM (Recovery Termination Management) control blocks, are identical.

HFS files are allocated to the kernel and I/O is done by calling the kernel. An open HFS object (file, pipe, socket) for the parent is represented by an open File Descriptor in the File Descriptor Table. On a fork(), all open HFS File Descriptors (that is, files) are inherited by the child address space. The child does not inherit any file locks for the HFS files.

z/OS resources are not propagated to the child address space. Any linkage stack in the parent is not carried over to the child. Also, data spaces and hiperspaces are not carried over since access list and access register contents are not copied. Internal timers together with SMF and SRM accounting data are set to zero in the child address space.

Also, z/OS data sets are allocated to an individual address space, and after a fork() the child address space does not have access to the z/OS data sets allocated by the parent. Also, ENQ resources held by the parent are not propagated.

12.14 Values for forked child process

Job Name	parent jobname + n (n=1 to 9)
Accounting Data	= parent
Environment Vars.	= parent
RACF Userid	= parent
RACF UID and GID	= parent
UNIX PID and PPID	new PID, PPID = parent PID
REGION value	larger of dub/parent REGION
TIME value	larger of dub/parent TIME
Working directory	= parent
UNIX "umask"	= parent
UNIX Resources (file descriptors)	inherit all open file descriptors (files, sockets, pipes)
MVS Resources	**INHERIT: STORAGE, PROGRAMS IN PRIVATE REGION, ESTAE/ESPIE** **Subpools: 0-127, 129-132, 251-252**

Figure 12-19 Propagation of parameters passed to the child process

Child process propagation of parameters

JOBNAME of the parent is propagated to the child and appended with a numeric value in the range of 1-9 if the jobname is 7 characters or fewer. If the jobname is 8 characters, it is propagated as is. Numeric count wraps back to 1 when it exceeds 9.

z/OS accounting data is copied from parent to child. UNIX environment variables are copied from parent to child. RACF security profile (Userid, UID, GID) is also inherited from the parent.

The child is allocated a new process ID (PID). Child PPID = parent PID.

The child REGION size is set by the “dub” process. The current REGION size of the parent is compared with BPXPRMxx MAXASSIZE, and the larger value is set as REGION size.

Child TIME value is also set by the “dub” process. The current TIME for the parent task is compared with BPXPRMxx MAXCPUTIME, and the larger value is set as the TIME value.

Current working directory and umask value for the parent are propagated to the child.

All resources defined by open file descriptors are propagated to open file descriptors on the child.

The only z/OS resources that are propagated to the child are the subpools in the parent REGION area (this includes loaded programs and data), plus the recovery environment for loaded programs established via ESTAE/ESPIE calls.

12.15 Starting a program with exec()

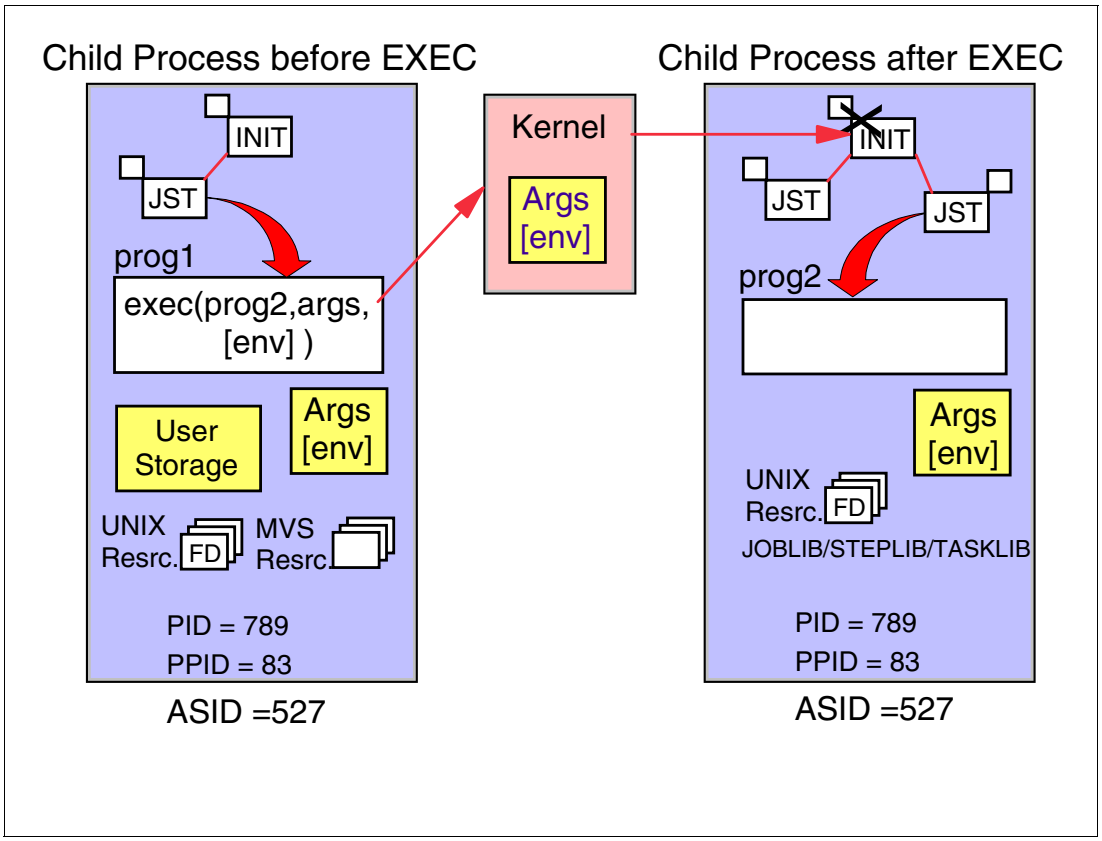


Figure 12-20 Creating a process using the exec() function

Starting a process with exec()

The z/OS UNIX environment provides a family of exec() function calls. These calls load a new program and transfer execution from the calling program to the new program. The difference between the exec calls is based on how the pathname of the new program is specified, and whether the value of environment variables will be passed.

When a program issues an exec() call, it passes these parameters:

- Program name - either partial/full pathname or a file name only
- A series of “arguments” (parameters) to be passed to the new program
- Optionally, an array of environment variables are passed via parm env()

When the kernel receives an exec call, it erases all the current storage contents of the address space. It also terminates the current process task, and starts a new one. The new program is loaded from the HFS, and the values of the arguments and optional environment variables are placed in storage. The new program starts up using only this basic information.

No z/OS resources are propagated from the caller's TCB to the new TCB except JOBLIB, STEPLIB, and TASKLIB. However, open file descriptors from the exec() caller are propagated to the new program, giving it access to resources established by the caller.

Propagated parameters to the child

An exception to no z/OS resources being propagated is that the caller's task JOBLIB/STEPLIB/TASKLIB status may be propagated to the new task. This requires the caller to set the STEPLIB variable before exec(). (We explain more about this later.)

If exec() is used by a process running in an address space created by fork or spawn, the caller of exec() *must* be running under the job step task TCB, with no subtasks, and with no linkage stack. Any other situation will cause an ABEND EC6. For processes created in multi-task address spaces via attach_exec, execmvs, or attach_execmvs, this restriction is relaxed.

fork() and exec() calls

The fork() and exec() calls are usually used together to create a new address space running a child process, and then pass control to a new program in the child. There are some exceptions to this:

- A C program started from a z/OS batch job could use the exec() call to start a program from the HFS. The new program would then replace the original C program executed in batch.
- The BPXBATCH utility can be used to run shell scripts, or C programs resident in the HFS, as a batch job in a JES initiator. After the BPXBATCH job step program has set up the support environment, it then issues exec() to replace BPXBATCH with the program or shell script to be run.

Note: The target of an exec() can be a shell script or REXX exec. In this case, the new program loaded by the kernel is the SHELL program, and the script or REXX exec is automatically executed under the shell.

12.16 Values passed for exec() program

Job Name	(1) = caller (2) $_BPX_JOBNAME
Accounting Data	(1) = caller (2) $_BPX_ACCT_DATA
Environment Vars.	(1) = caller (2) env parm on exec
RACF Userid	= caller
RACF UID and GID	= caller unless target uses SetUID, SetGID
UNIX PID and PPID	= caller PID, PPID
REGION value	larger of dub/parent REGION
TIME value	larger of dub/parent TIME
Working directory	= caller
UNIX "umask"	= caller
UNIX Resources (file descriptors)	Inherit all open file descriptors (files, sockets, pipes) - override with FD_CLOEXEC flag
MVS Resources	Inherits nothing from calling program, except Steplib, Joblib, Tasklib

Figure 12-21 Parameters passed to the new process

Parameters passed to process with exec()

Due to the creation of a new process task for the exec() program, the scheduling environment can change considerably over an exec() call, as follows:

- By default, the address space JOBNAME stays the same. However, an authorized caller (a superuser running in a forked address space) can set a new name envar $_BPX_JOBNAME before exec() - this will change the address space JOBNAME.
- By default, account data is inherited. Alternatively, any exec() caller an set envar $_BPX_ACCT_DATA to new account data prior to issuing a call.
- By default, a new program inherits all caller environment variables. You can use the env() parameter to pass *only* chosen values to the new program.

The RACF Userid remains constant over exec(). UID and GID are also constant, unless the target program has SetUID or SetGID flags set. In this case, the effective UID or GID is changed for the new program. The PID and PPID values are not changed.

As with fork, REGION and TIME values are first set by “dub” values in BPXPRMxx. The inherited values related to the exec() caller TASK are then compared to the dub values, and if inherited values are larger, they override dub.

Working directory and umask are inherited from the caller. All open file descriptors are passed across to the new program for immediate use, except those previously flagged with an FD_CLOEXEC flag.

No z/OS resources are inherited, except possibly JOBLIB/STEPLIB.

12.17 z/OS UNIX processes get STEPLIBs

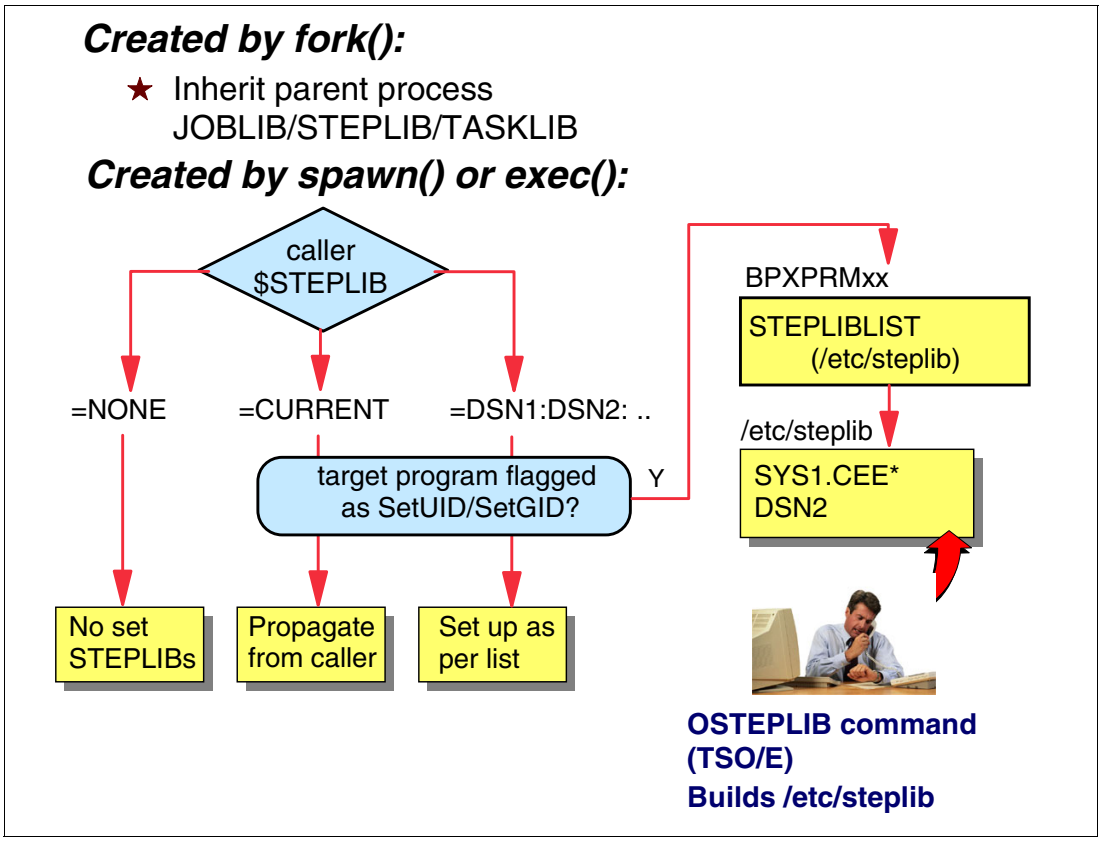


Figure 12-22 The use of STEPLIBs when creating a new process

Propagation of STEPLIBs to a new process

Figure 12-22 illustrates that programs requested by a UNIX application can be loaded from z/OS program libraries, including STEPLIBs.

For a process that has been created by a fork() request, the child process inherits any active parent process JOBLIB/STEPLIB concatenation.

When a process wants to use a spawn() or an exec() call, the caller must set the $STEPLIB envar prior to executing the fork()/spawn() request so that the kernel can control the STEPLIB allocation. The settings have the following meanings:

- **CURRENT** - Propagate and reuse JOBLIB/STEPLIB/TASKLIB from task TCB issuing the exec/spawn to the new process/program.
- **NONE** - No STEPLIB; do not propagate from the caller if present.
- **DSN1:.DSN2:** - Set this as STEPLIB concatenation.

If the $STEPLIB variable is not set, the default assumed is CURRENT.

If the target program for the spawn() or exec() function has the SetUID or SetGID flag set, a further check is made against the sanctioned library list. This list is pointed to by the BPXPRMxx statement STEPLIBLIST. Libraries in the JOBLIB/STEPLIB list to be propagated must also be named in the sanction list. If a library is not sanctioned, it will be dropped from the propagated STEPLIB list.

STEPLIBLIST

STEPLIBLIST specifies the path name of the file in the file system that contains the list of MVS data sets to be used as step libraries for programs that have the set-user-id and set-group-id bit set on.

Step libraries have many uses; one is so that selected users can test new versions of run-time libraries before the new versions are made available to everyone on the system. Customers who do not put the Language Environment Run-Time Library SCEERUN into the linklist should put the SCEERUN data set name in this file.

If your installation runs programs that have the setuid or setgid bit turned on, only those load libraries that are found in the STEPLIBLIST sanction list are set up as step libraries in the environment that those programs will run in. Because programs with the setuid or setgid bit turned on are considered privileged programs, they must run in a controlled environment. The STEPLIBLIST sanction list provides this control by allowing those programs to use only the step libraries that are considered trusted by the installation.

12.18 Locating programs for z/OS UNIX processes

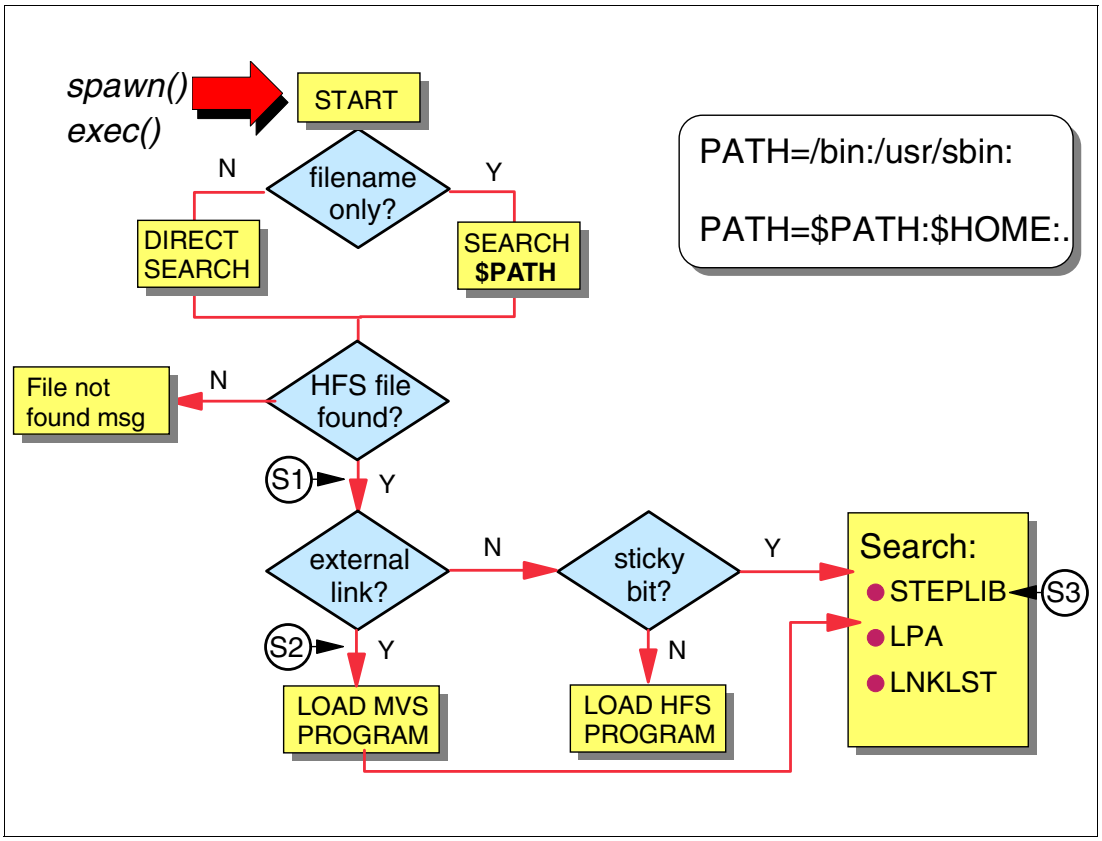


Figure 12-23 Loading programs for z/OS UNIX processes

Loading programs in z/OS UNIX

The spawn() and exec() calls both request z/OS UNIX to find and load a new HFS program. Figure 12-23 shows the overall flow. First scan the supplied pathname of the HFS program for imbedded "/" characters, with the following results:

- First char = "/" - Supplied name is the absolute pathname; search HFS directly.
- Contains one or more "/," not in position 1. Treat this as a relative pathname by suffixing with current working directory, then search HFS for full name.
- No "/" - Supplied name is filename; search directory concatenation supplied in $PATH environment variable, use first object with a matching name.

S1 If the HFS file is located, check permission bits for user execute auth. If access is granted, check to see if object is an external link. If so, attempt to load the z/OS program named in the link.

S2 This is a second security access check. It is made based on RACF user ID and the DSNAME. If the object is not an external link, check to see if the sticky bit is on. If so, switch to do a z/OS search for the program. Otherwise attempt to load and execute the HFS object as a program. For a z/OS search, follow the standard search sequence.

S3 Search any JOBLIB/STEPLIB concatenation pointed to by the process task. If an object is found, make a security check based on RACF user ID and DSNAME before loading the program.

- Search the LPA areas.
- Search the system LNKLST.

12.19 Shared pages for the fork() function

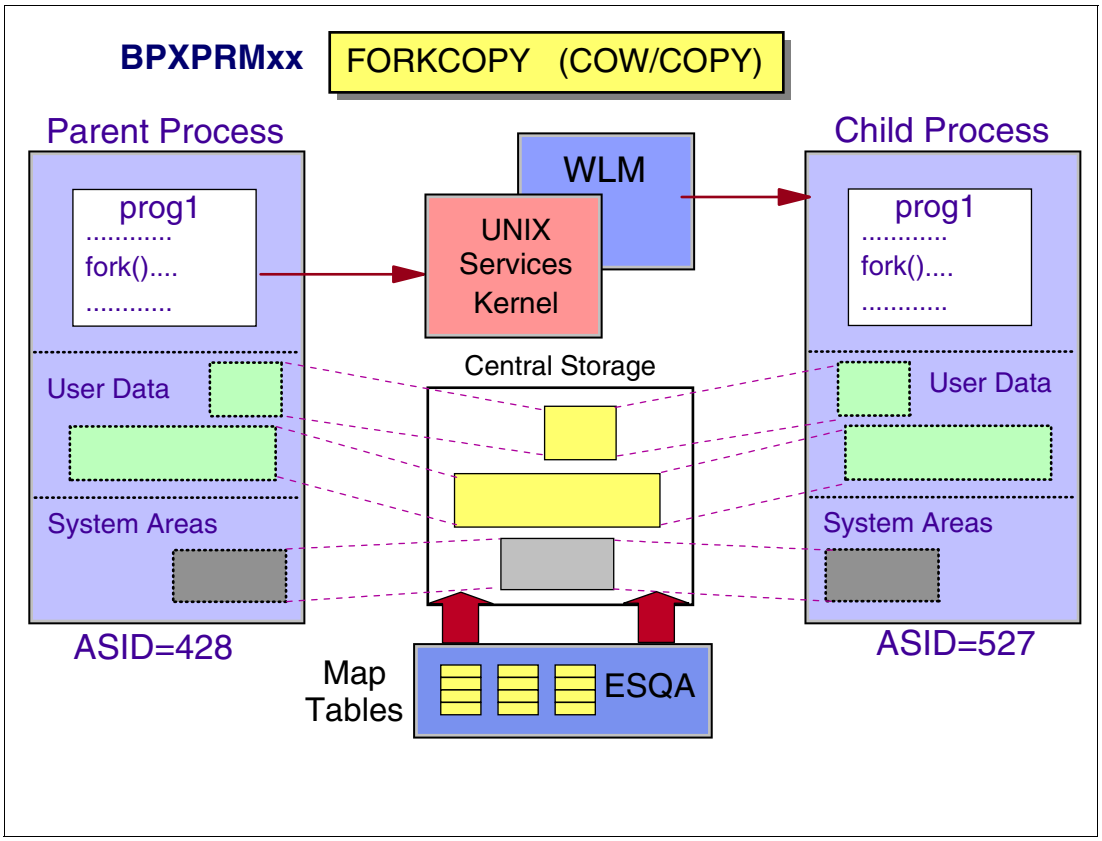


Figure 12-24 Shared pages for the fork() function

Shared pages function

The shared pages function provides the capability to define virtual storage areas through which data can be shared by programs within or between address spaces or data spaces. Sharing reduces the amount of processor storage required and the I/O necessary to support data applications that require access to the same data.

The z/OS UNIX fork function can use shared pages to improve performance. The keyword `FORKCOPY(COW|COPY)` in the BPXPRMxx member specifies how storage is copied from the parent process to the child during fork.

FORKCOPY(COW) Copy On Write (COW). Storage areas will be shared between the parent and child, and they will only be copied to the child if they are modified by either the parent or the child. This requires the suppression-on-protection (SOP) hardware feature.

FORKCOPY(COPY) All storage areas will be copied from the parent to the child. Shared pages will not be used.

FORKCOPY(COW) is the default. Figure 12-24 illustrates how the storage area of a parent process is copied to a child process during a fork operation. For a UNIX system this is not a big deal, but for an MVS system it is a very costly operation. In most cases where a fork function is used, it is followed by an exec function in the child. This means that a new program is started in the child and all the storage areas that were copied from the parent are released.

12.20 Spawn function

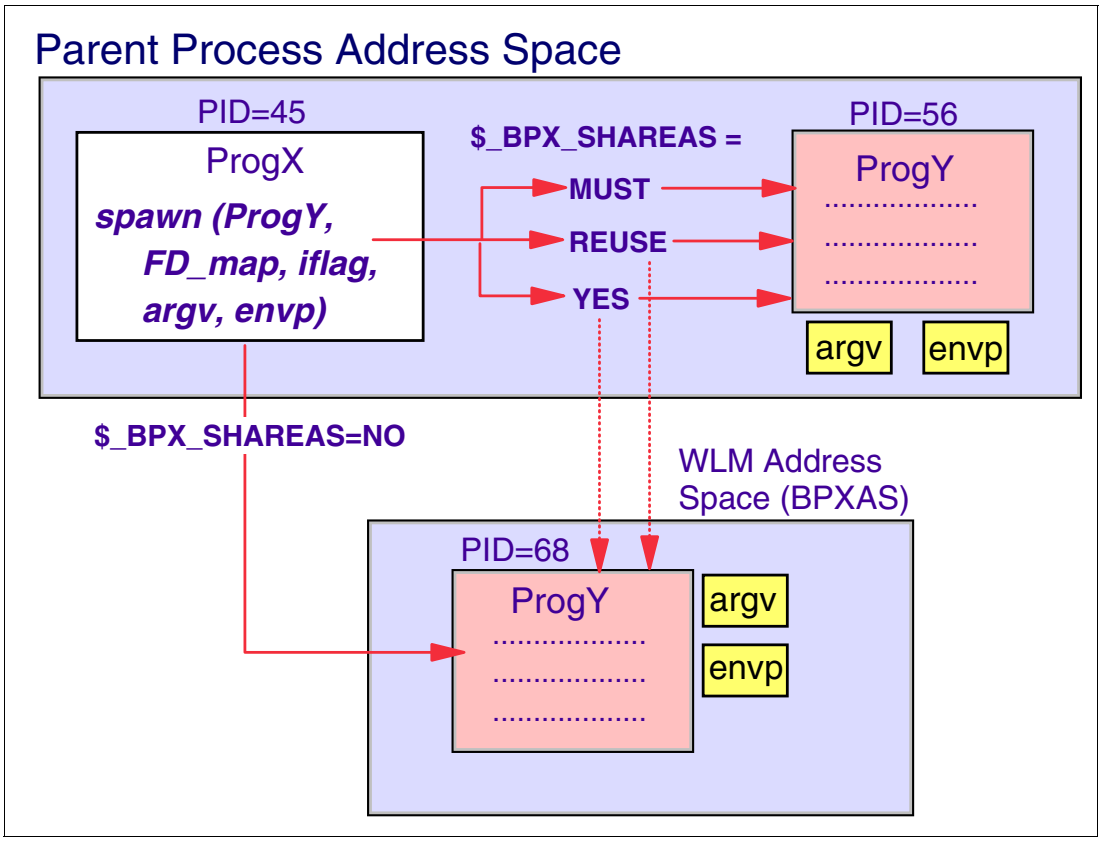


Figure 12-25 The spawn function

Spawn function

Spawn is a combination of fork and exec because spawn will create a new process and start a new program in the child process.

Spawn can create a child process in the same address space as the parent process or in a new address space. An environment variable called _BPX_SHAREAS will decide where the child process will be created.

- _BPX_SHAREAS=YES - The child process will be created in the same address space as the parent. The benefits of using BPX_SHAREAS=YES are:
  - The spawn runs faster
  - The child process consumes fewer system resources.
  - The system can support more resources.

  If YES fails, a new address space is created.

  The side effects are:
  - When running multiple processes with BPX_SHAREAS=YES, the processes cannot change identity information. For example, setuid and setgid will fail.
  - You cannot run a setuid or setgid program in the same address space for another process.
  - When the parent terminates, the child will terminate because it is a subtask.

- _BPX_SHAREAS=MUST - MUST specifies that the child process must be created on a subtask in the parent's address space. If the request cannot be honored, the request will complete unsuccessfully.
- _BPX_SHAREAS=REUSE - REUSE specifies that the child process is to be created on a subtask in the parent's address space and when the process terminates, system structures for the child process are left in place and reused when the parent spawns another process with _BPX_SHAREAS=REUSE.

  Using BPX_SHAREAS_REUSE is similar to specifying YES. In environments where shell commands are invoked over and over, the REUSE option will perform better.
- _BPX_SHAREAS=NO - The child process will be created in a separate address space.

Note: _BPX_SHAREAS=NO is the default setting for this environment variable.

12.21 Interprocess communication functions

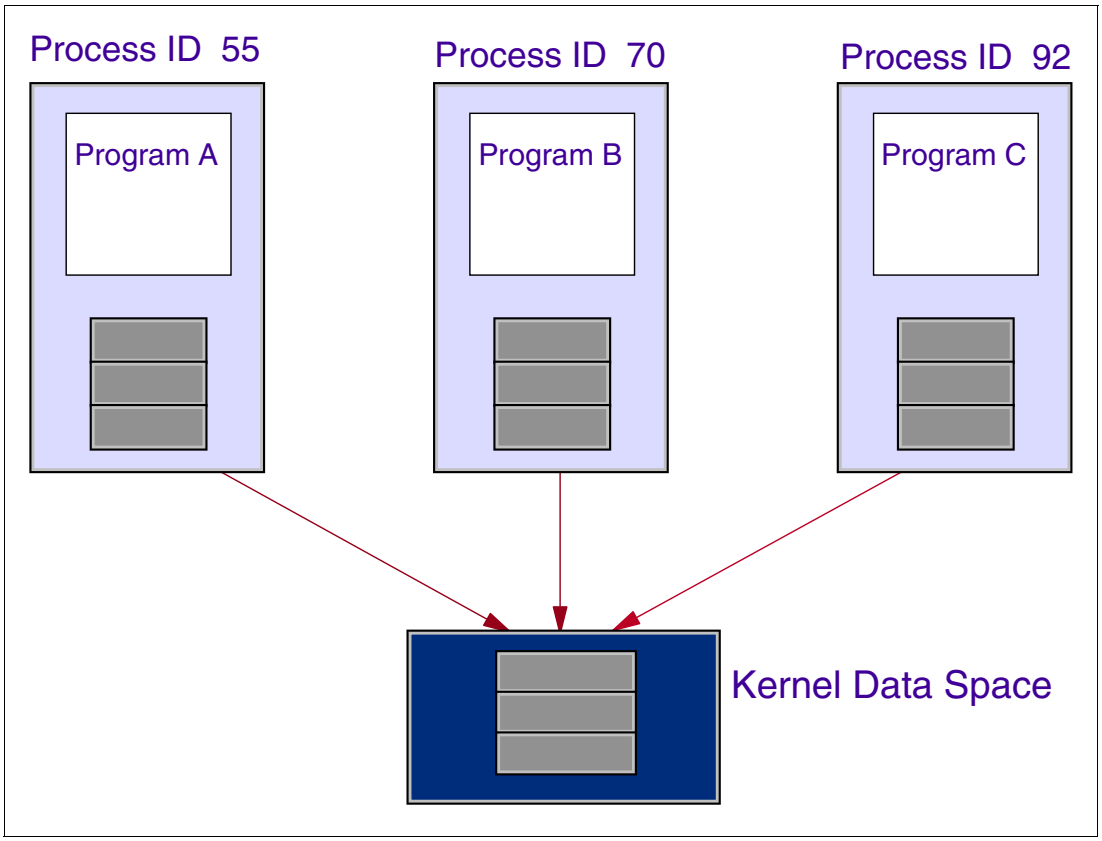


Figure 12-26 Interprocess communication functions

Interprocess communications

Support for XPG4 introduced Interprocess Communication (IPC) functions in z/OS UNIX. These IPC functions were required by many applications, particularly client/server applications.

Message queues - Message queues allow a client and a server process to communicate through one or more message queues in the kernel. A process can create, read from, or write to a message queue. Multiple client and server processes can share the same queue.

Shared memory - Shared memory provides a method of sharing data in storage between multiple processes. The shared data is kept in a data space created by the kernel. The data can be shared between a parent and child process or between unrelated processes.

Semaphores - Semaphores are used for serializing access to shared memory. A program using shared memory must get a semaphore before it allocates shared memory.

12.22 Address Space Memory Map z/OS V1R5

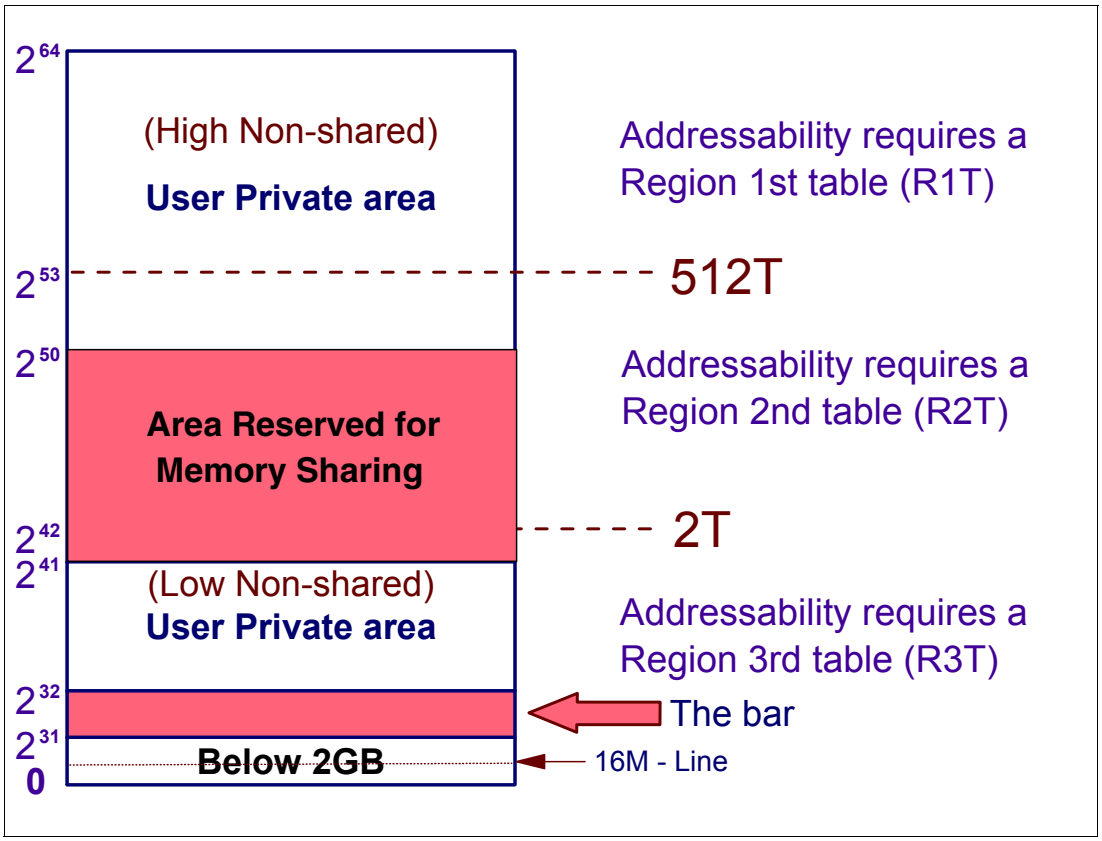


Figure 12-27 64-bit address space with memory sharing support

Memory sharing with z/OS V1R5

Beginning with z/OS V1R5, UNIX applications can obtain memory above the bar in the area reserved for memory sharing.

To get access to the shared memory object, a program uses the SHAREMEMOBJ service. An address space can issue more than one SHAREMEMOBJ request for the same memory object. To separate each of the requests for the same memory object you need to specify a different user token. You must be executing in AMODE64.

If, while running a 64-bit program, you allocate shared memory segments above the bar by using the shmget() service, the shared page limit is not affected.

Shared memory examples

UNIX applications can share memory between address spaces with this support. The following example shows how the first address space creates a shared memory object one megabyte in size. It specifies a constant with value of one as a user token.

Note: USERTKN=usertoken is a required 8-byte token that relates two or more memory objects to each other. Associate the user token with the memory object, so later you can free several shared memory objects at one time.

```
IARV64 REQUEST=GETSHARED,
         SEGMENTS=ONE_SEG,
         USERTKN=USERTKNA,
         ORIGIN=VIRT64_ADDR,
         COND=YES,
         FPROT=NO,
         KEY=MYKEY,
         CHANGEACCESS=LOCAL
```

A second address space can then share the obtained storage in the memory sharing part of storage as follows:

```
IARV64 REQUEST=SHAREMEMOBJ,
       USERTKN=USERTKNS,
       RANGLIST=RLISTPTR,
       NUMRANGE=1,
       ALETVALUE=0,
       COND=YES,
       SVCDUMPRGN=YES
```

12.23 Control IPC resources

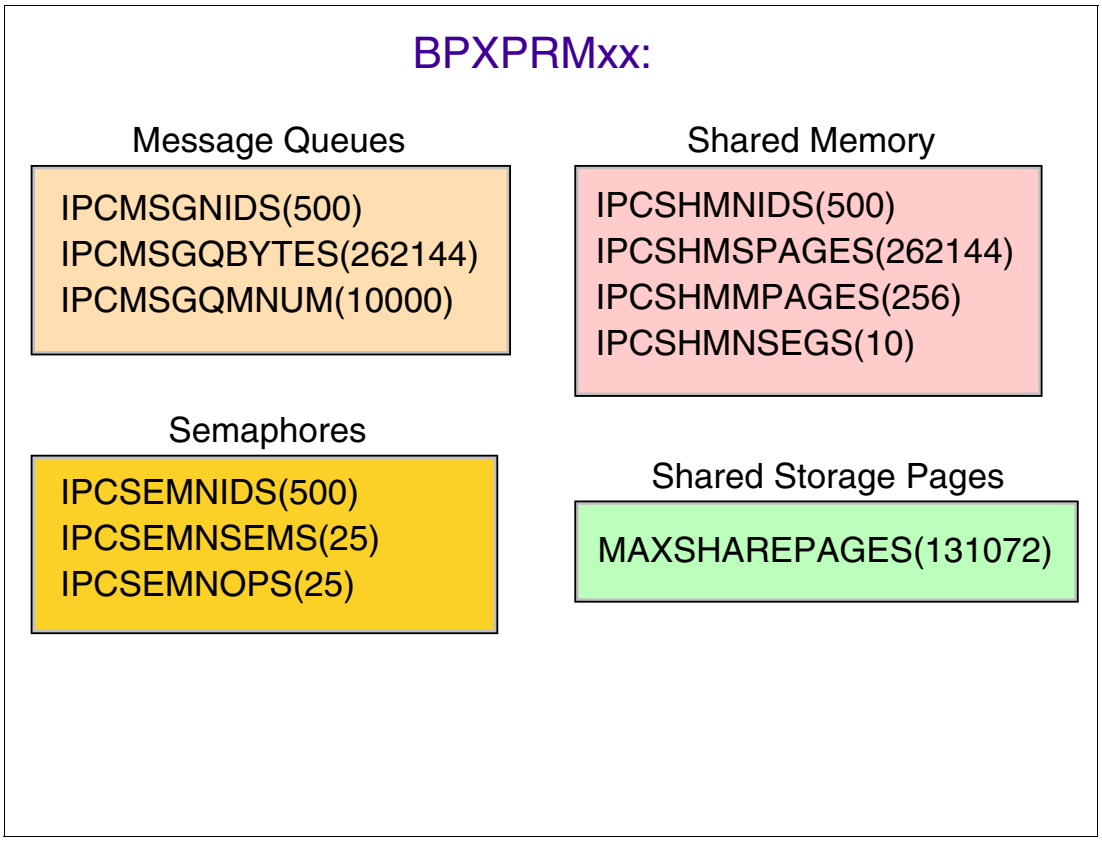


Figure 12-28 Controlling interprocess communication functions

BPXPRMxx values for IPC functions

Message queues are controlled by the following specifications:

- IPCMSGNIDS - Specifies the maximum number of unique message queues in the system. The range is from 1 to 20000. The default is 500.
- IPCMSGQBYTES - Specifies the maximum number of bytes in a single message queue. The range is from 0 to 1048576. The default is 262144.
- IPCMSGQMNUM - Specifies the maximum number of messages for each message queue. The range is from 0 to 20000. The default is 10000.

Shared memory control:

- IPCSHMNID - Specifies the maximum number of unique shared memory segments in the system. The range is from 1 to 20000. The default is 500.
- IPCSHMSPAGES - Specifies the maximum number of pages for shared memory segments in the system. The range is from 0 to 2621440. The default is 262144.
- IPCSHMMPAGES - Specifies the maximum number of pages for a shared memory segment. The range is from 0 to 25600. The default is 256.
- IPCSHMNSEGS - Specifies the maximum number of shared memory segments attached for each address space. The range is from 0 to 1000. The default is 10.

Semaphore control:

- IPCSEMNIDS - Specifies the maximum number of unique semaphore sets in the system. The range is from 1 to 20000. The default is 500.
- IPCSEMNSEMS - Specifies the maximum number of semaphores for each semaphore set. The range is from 0 to 32767. The default is 25.
- IPCSEMNOPS - Specifies the maximum number of operations for each semaphore operation call. The range is from 0 to 32767. The default is 25. This is a system-wide limit.

Controlling shared pages

MAXSHAREPAGES specifies the maximum number of shared storage pages that can be concurrently in use by z/OS UNIX functions. This can be used to control the amount of ESQA consumed, since shared storage pages cause the consumption of ESQA storage. The functions that this limit applies to are fork(), mmap(), shmat(), and ptrace(). The range is from 0 to 32768000. The default is 131072 pages.

12.24 Kernel support for IBM 5.0 JVM

- IBM 5.0 JVM - exploits z/OS UNIX shared memory
- The JVM must be able to run in these environments:
  - A Websphere Application Server environment that runs PSW Key 2
  - A CICS environment that runs PSW Key 9
- Provide ability to use z/OS UNIX shared memory services from Key 2 and Key 9 environments
- Provide enhanced sharing capabilities for address spaces running multiple processes
- z/OS APAR OA11519 available for z/OS R1V6 and up
  - Apply to any z/OS system where shared classes are used

Figure 12-29 Application support with z/OS UNIX with IBM 5.0 JVM

IBM 5.0 JVM version

The IBM 5.0 JVM version of the JVM is designed to exploit UNIX shared memory. In releases prior to z/OS V1R8, UNIX shared memory could only be exploited from environments that ran PSW Key 8. The JVM must now be able to run in the following environments:

- A WebSphere Application Server environment that runs PSW Key 2
- A CICS environment that runs PSW Key 9

z/OS UNIX support for shared memory from Key 2 and Key 9 is now provided with z/OS V1R8. Additionally, to allow for more efficient sharing in an address space such as CICS that can run IBM 5.0 JVM in multiple processes, support for enhanced single address space sharing is provided.

Note: Following are the IBM 5.0 JVM versions:

- IBM 31-bit SDK for z/OS, Java 2 Technology Edition, Version 5, product 5655-I98
- IBM 64-bit SDK for z/OS, Java 2 Technology Edition, Version 5, product 5655-I99.

Java applications

With the introduction of the kernel IBM 5.0 JVM support, Java applications are able to run and exploit the latest features provided by the IBM 5.0 JVM in all z/OS supported environments that now includes critical environments such as WebSphere Application Server and CICS. The z/OS V1R8 support provides the following enhancements:

- Ability to use UNIX shared memory services for WebSphere Application Server and CICS environments
- Enhanced sharing capabilities for address spaces running multiple processes

Using kernel support for IBM 5.0 JVM, applications are able to:

- Get the benefits of the enhanced IBM 5.0 JVM in customer critical environments such as WebSphere Application Server and CICS
- Run Java applications that exploit the latest features provided by the IBM 5.0 JVM in all z/OS supported environments, such as:

  shmgt, shmat, shmdt, shmctl from Key 9 and Key 2 environments

Request for shared memory segments

The user address space storage request for a shared memory segment, an area shown in “Address Space Memory Map z/OS V1R5” on page 638 for 64-bit callers, is normally obtained in storage key 8. In the following special circumstances, the storage key does not have to be 8:

- The caller creating a shared memory segment is running PSW Key 9 or 2 in 64 -bit AMODE and the segment is not of type IPC_MEGA and IPC_BELOWBAR.
- The caller creating a shared memory segment is running PSW Key 9 or 2 in 31-bit AMODE and the shared memory segment is of type IPC_MEGA.

Note: In these circumstances, the user address space storage is obtained in the PSW key of the caller (key 9 or key 2). It is important to note that any subsequent usage of the segment from any address space causes the user address space storage to be obtained in the key the segment was initially created in. This is true regardless of the PSW key the caller is running in at the time of a subsequent attach.

Coexistence support

For z/OS V1R6 and z/OS V1R7, this feature can be exploited by installing APAR OA11519 and APAR PK05350.

Note: It is strongly recommended that z/OS APAR OA11519, available for z/OS R1V6 and onwards, is applied to any z/OS system where shared classes are used. This APAR ensures that multiple **shmat** requests for the same shared segment will map to the same virtual address across multiple processes.

Without this APAR, there is a problem with using shared memory when multiple processes reside in a single address space. Each **shmat** call consumes a separate virtual address range. This is not acceptable because shared classes will run out of shared memory pages prematurely.

12.25 Interprocess communication signals

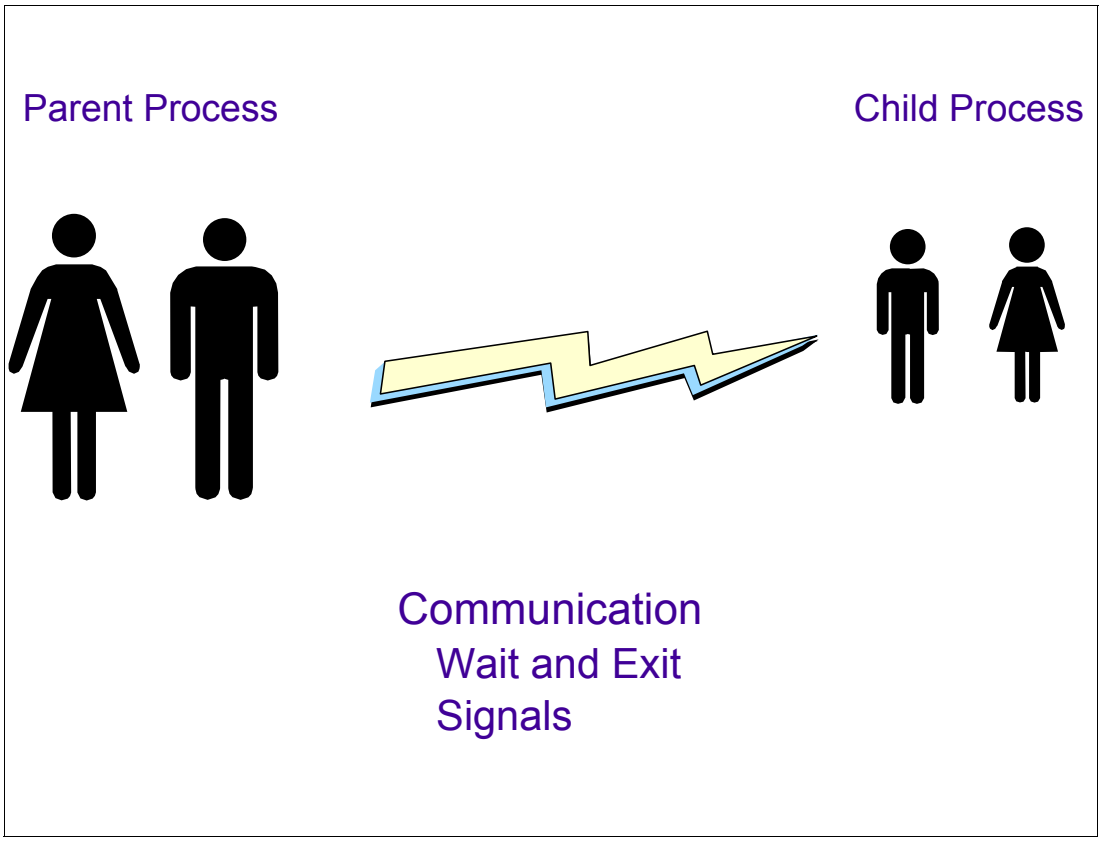


Figure 12-30 Interprocess communication signals

Interprocess communication signals

All signal functions are supported when the task is set up for signals, when it is running with the signal delivery key, and when its current program request block (PRB) is the same PRB as when the task was set up for signals.

Signal delivery also depends on the signal delivery key. Each process has one signal delivery key. The signal delivery key is set to the PSW key of the caller of the first z/OS UNIX call that created the process. A process created by the fork or exec service has key 8.

- **Wait and Exit** - A parent process can wait on the exit of a child. The wait() function is used for waiting for any child process, while the waitpid() function is used for waiting for a particular child process.

  Both exit() and _exit() terminate a process and generate status information which is available for the parent process waiting with wait() or waitpid(). When using exit(), these cleanup routines are not invoked. Generally, exit() is used for a graceful exit from a program, while _exit() is used for abnormal terminations.

- **Signal() and sigaction()** are equivalent functions which are used to catch a signal and determine what to do with it.

  The function for sending signals is called kill(). A process can send a signal to another process or group of processes if it has permission to do so. A process can also send a signal to itself.

Signals are used for system event notification, or they can be used for process synchronization. For example, a process might want to wait for a signal to know that another process has opened a pipe, written a file, or completed a task that the current process needs to wait for.

A process can choose what to do when it receives a signal:

- Execute a signal handling function.
- Ignore the signal.
- Restore the default action of a signal.

Kill signal

The kill callable service sends a signal to a process, a process group, or all processes in the system to which the caller has permission to send a signal.

Kill() accepts several different signal codes. Examples are:

- SIGABND - abend
- SIGCHLD - child termination
- SIGKILL - cancel a process
- SIGSTOP - stop a process

From a program's point of view, signals are asynchronous. That means a program can, in principle, receive a signal between any two instructions.

12.26 Pipes

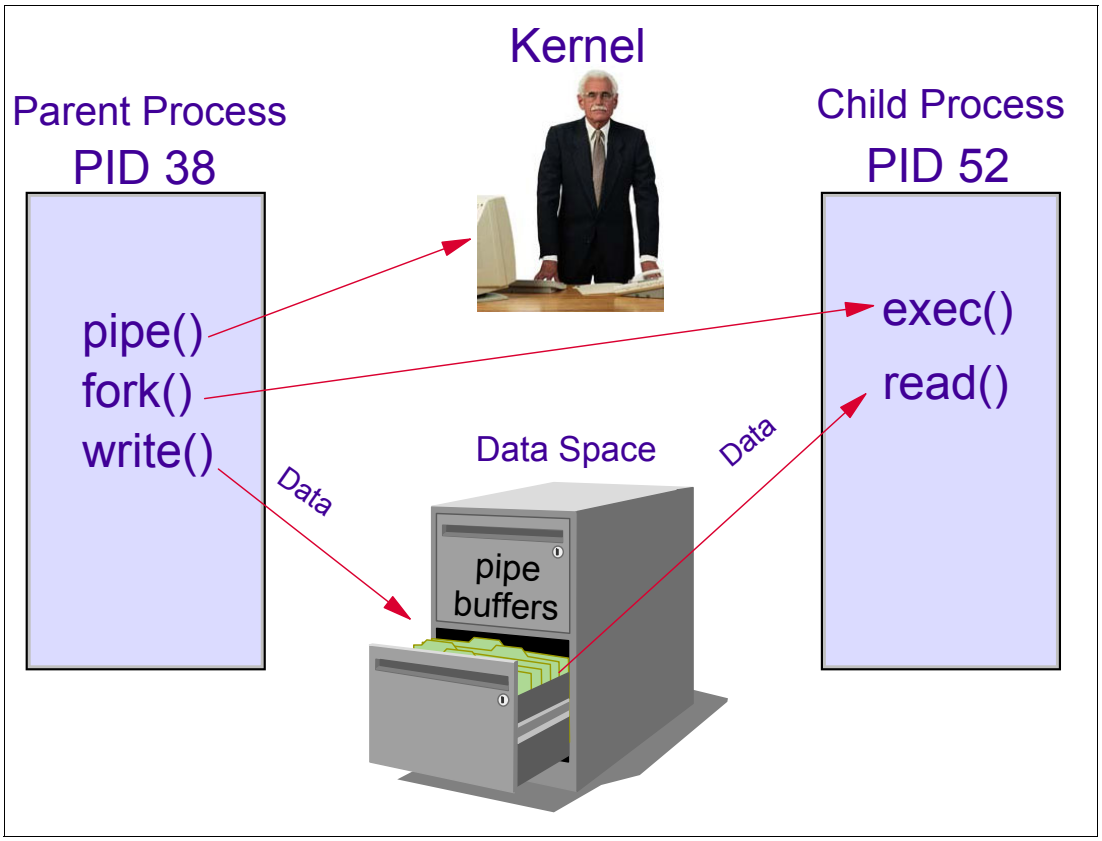


Figure 12-31 Using pipes with z/OS UNIX

Pipe function

In z/OS UNIX, pipes are a communication mechanism for sending arrays of data between two processes. Pipes are conceptually like a sequential file. Processes can write into a pipe and other processes can read from the pipe.

Pipes exist only during the time they are open. The storage needed for the pipes is obtained from a data space associated with the kernel address space.

Data can only be read from the beginning of the pipe. You cannot seek in a pipe. Data is always read from a pipe in the same order it was written into the pipe. The data is discarded when all the processes that read from the pipe have closed the pipe. There are two types of pipes:

- **Unnamed pipes** are used between related processes, for example between a parent and child process. An unnamed pipe is created by the pipe() function. The pipe function also opens the pipe for use. No security checking is performed on unnamed pipes since they are available only to a process and its children.
- **Named pipes** are also called FIFO. A FIFO resides in the hierarchical file system and can be referred to by a name; thus it can be used for communication between any two processes.

12.27 Other BPXPRMxx keywords

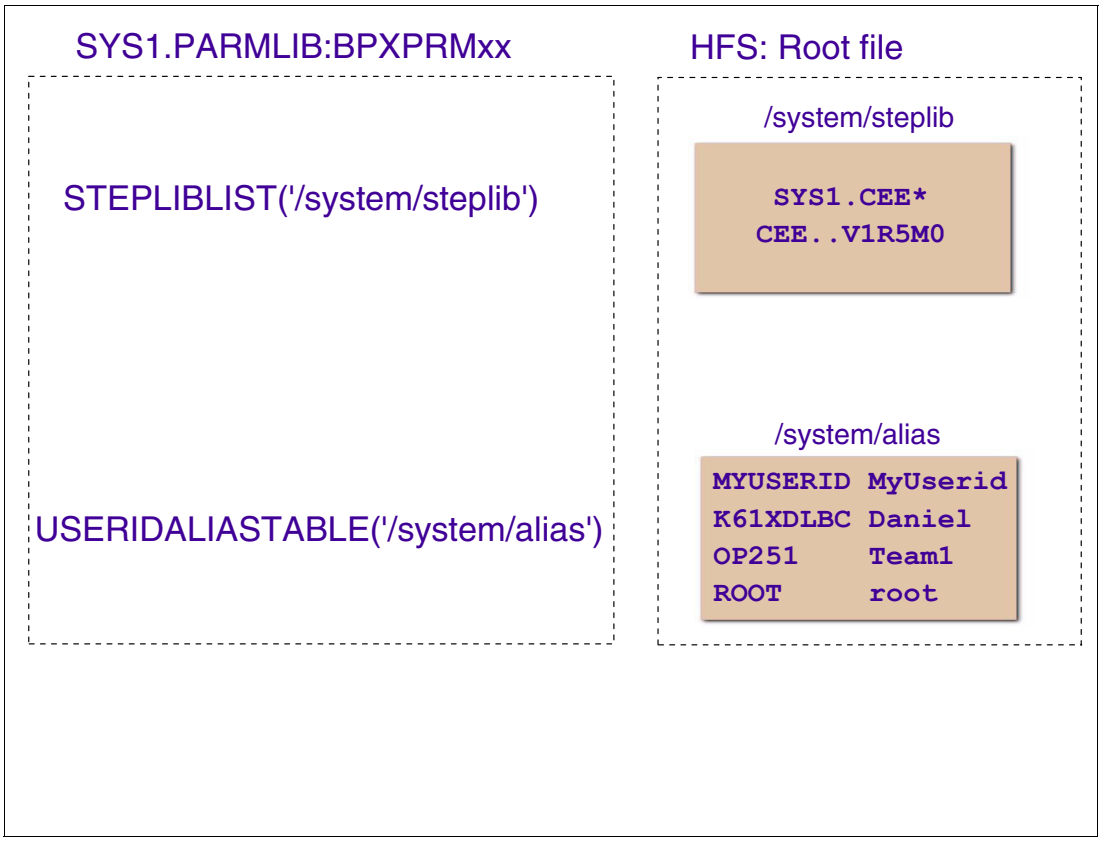


Figure 12-32 Other BPXPRMxx keywords

STEPLIBLST parameter

The STEPLIBLST parameter specifies a pathname of a file in the hierarchical file system. This file is intended to contain a list of MVS data sets that are sanctioned by an installation for use as step libraries for programs that have the set-user-ID and set-group-ID permission bits set.

The TSO/E command OSTEPLIB can be used to build or modify the file which contains the list of MVS data sets that can be step libraries. Superuser authority is required to run this command.

The USERIDALIASTABLE statement allows an installation to associate an alias name with an MVS user ID. If specified, this alias name is used in z/OS UNIX processing for the user IDs listed in the table. The USERIDALIASTABLE statement specifies the pathname of a hierarchical file system (HFS) file. This file is intended to contain a list of MVS user IDs and their associated alias names.

Specifying the USERIDALIASTABLE statement causes poorer performance and increases systems management costs and complexity. Installations are encouraged to continue using uppercase-only user IDs.

12.28 More BPXPRMxx parameters

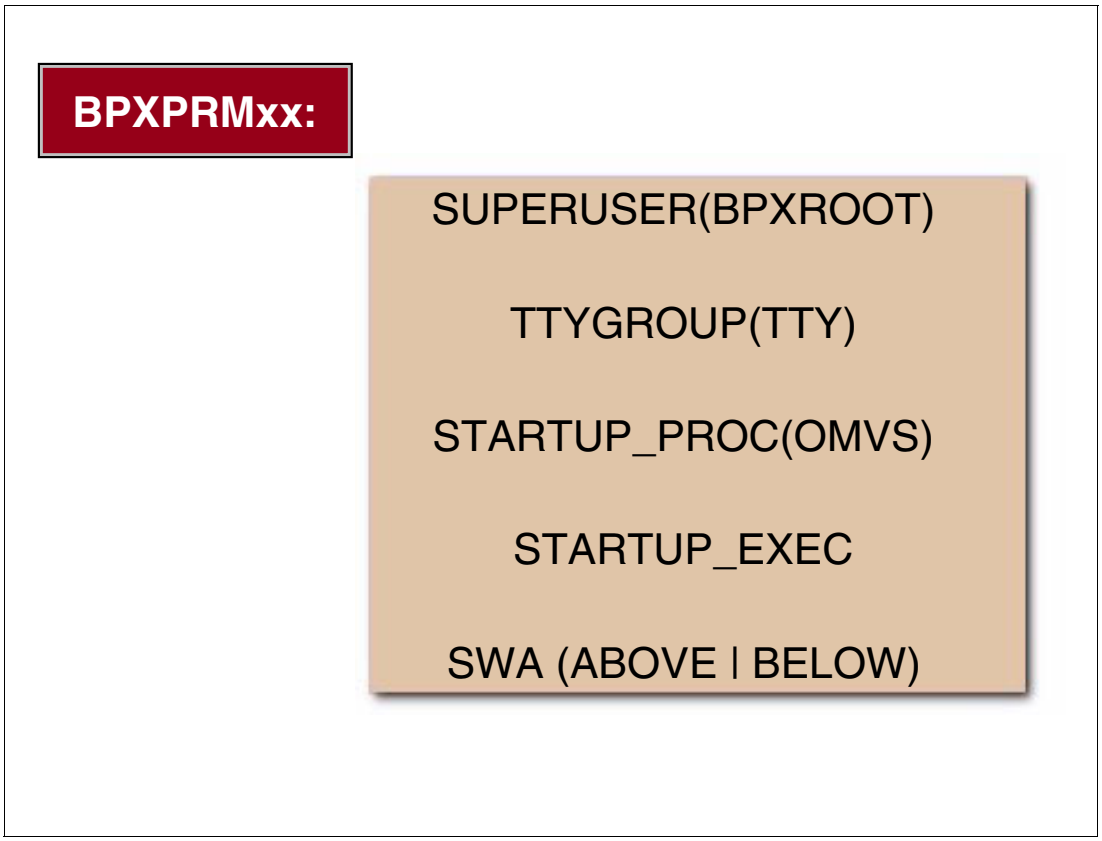


Figure 12-33 More BPXPRMxx parameters

Other BPXPRMxx parameters

The SUPERUSER parameter specifies a superuser name. This will be used when a daemon task issues a setuid() to set a UID of 0 and the user name is not known. It should be an ID defined to the security product with no access to MVS resources and a UID of 0. The default is SUPERUSER(BPXROOT).

TTYGROUP is a group name for the pseudo-terminal files (ptys and rtys) when they are first opened. This group name should be defined to the security product with a GID, but with no users in the group. TTY is the default group name.

The STARTUP_PROC(OMVS) is the started JCL procedure that initializes the kernel. If you change this it must be a single step procedure that invokes the BPXINIT program.

The STARTUP_EXEC is a startup exec that replaces the /etc/init program that BPXOINIT normally invokes. This is used by installations that want to run with a minimal configuration, but would like to populate the TFS with some directories or files.

SWA(ABOVE/BELOW) specifies whether SWA control blocks should be allocated above or below the 16 MB line.

ABOVE All SWA control blocks are to be allocated above the 16 MB line.
BELOW All SWA control blocks are to be allocated below the 16 MB line.

The default is BELOW.

12.29 FILESYSTYPE statement

```
/*****************************************************************
  FILESYSTYPE TYPE(HFS)
          ENTRYPOINT(GFUAINIT)
          PARM(' ')
/*****************************************************************
  FILESYSTYPE TYPE(AUTOMNT)
          ENTRYPOINT(BPXTAMD)
/*****************************************************************
  FILESYSTYPE TYPE(TFS)
          ENTRYPOINT(BPXTFS)
/*****************************************************************/
```

Figure 12-34 The FILESYSTYPE statement in the BPXPRMxx member

FILESYSTYPE statements

The following sections explain where the statements FILESYSTYPE, ROOT, MOUNT, and NETWORK apply.

FILESYSTYPE specifies that:

- A file system program of type HFS is to process file system requests. The system attaches the GFUAINIT load module during z/OS UNIX initialization.
- A physical file system of type AUTOMNT is to handle automatic mounting and unmounting of file systems. The system attaches the BPXTAMD load module during z/OS UNIX initialization.
- A physical file system of type TFS is to handle requests to the temporary file system. The system attaches the BPXTFS load module during z/OS UNIX initialization.

12.30 FILESYSTYPE and NETWORK

```
/*************************************************/
  FILESYSTYPE TYPE(UDS) ENTRYPOINT(BPXTUINT)
  NETWORK DOMAINNAME(AF_UNIX)
        DOMAINNUMBER(1)
        MAXSOCKETS(10000)
        TYPE(UDS)
/*************************************************/
  FILESYSTYPE TYPE(INET) ENTRYPOINT(EZBPFINI)
  NETWORK DOMAINNAME(AF_INET)
        DOMAINNUMBER(2)
        MAXSOCKETS(21100)
        TYPE(INET)
/*************************************************/
```

Figure 12-35 The FILESYSTYPE and NETWORK statements

FILESYSTYPE and NETWORK statements

A physical file system of type UDS is to handle socket requests for the AF_UNIX address family of sockets. The system attaches the BPXTUINT load module during z/OS UNIX initialization.

A physical file system of type INET is to handle requests for the AF_INET address family of sockets. The system attaches the EZBPFINI load module during initialization to start the AF_INET physical file system. This assumes you are using TCP/IP z/OS UNIX. If you want to use this, uncomment out the appropriate line in the BPXPRMxx PARMLIB member.

For more information about this, see *z/OS UNIX System Services Planning,* GA22-7800.

NETWORK

Each NETWORK statement identifies the information needed by the socket physical file system.

The AF_UNIX statement is used to communicate in the local system and AF_INIT to communicate with remote systems.

12.31 ROOT and MOUNT statements

```
ROOT    FILESYSTEM('OMVS.TC1.TRNRS1.ROOT.HFS')
        TYPE(HFS)
        MODE(RDWR)
*********************************************************************
MOUNT FILESYSTEM('OMVS.TC1.ETC.HFS')
        MOUNTPOINT('/etc')
        TYPE(HFS)
        MODE(RDWR)
```

Figure 12-36 The ROOT and MOUNT statements in the BPXPRMxx member

ROOT statement

The ROOT statement defines and mounts the root file system for a zFS (ZFS) or hierarchical file system (HFS).

ROOT identifies and mounts the HFS data set to be used as the root file system.

- TYPE identifies the file system program, which must have been specified on a FILESYSTYPE statement.
- MODE(RDWR) allows read and write access to the file system.

In the ROOT statement, give the name of the root file system. Figure 12-36 shows:

`OMVS.TC1.TRNRS1.ROOT.HFS`

MOUNT statement

The MOUNT statement defines the hierarchical file systems to be mounted at initialization and where in the file hierarchy they are to be mounted. All HFS data sets specified on MOUNT statements in the BPXPRMxx PARMLIB member must be available at IPL time. If an HFS data set is migrated by HSM, then the initialization of OMVS and HSM will deadlock. Neither kernel nor HSM services will be available.

This statement provides a MOUNT statement for each HFS data set to be mounted on the root file system or on another mounted file system. The pictured example shows the data set `OMVS.TC1.ETC.HFS` is to be mounted at directory **/etc**.

12.32 Examples of MKDIR in BPXPRMxx

```
ROOT
   FILESYSTEM('fsroot')
   TYPE(type_name)
   MODE(access)
   PARM('parameter')
   SETUID|NOSETUID
   AUTOMOVE | NOAUTOMOVE
   TAG(NOTEXT | TEXT,ccsid)
   MKDIR('pnfs1')
```

```
MOUNT
   FILESYSTEM('fs1')
   MOUNTPOINT('/pnfs1')
   TYPE(type_name)
   MODE(access)
   PARM('parameter')
   SETUID / NOSETUID
   SECURITY | NOSECURITY
   AUTOMOVE | NOAUTOMOVE
   AUTOMOVE(Ind,S1,...Sn)
   TAG(NOTEXT|TEXT,ccsid)
   MKDIR('pnfs2')
   MKDIR('pnfs3')
```

Figure 12-37 MKDIR examples in the BPXPRMxx PARMLIB member

Creating directories in the BPXPRMxx member

When using the ROOT and MOUNT statements in the BPXPRMxx member during an IPL, these PARMLIB mounts can fail if the mountpoint does not exist. With z/OS V1R5, support is added to the BPXPRMxx PARMLIB member to allow specifying directories to be created during PARMLIB processing.

You can use multiple MKDIR keywords on the MOUNT statement to define mount points in the BPXPRMxx PARMLIB member so that one or more directories are created in the mounted file system during z/OS UNIX initialization.

MKDIR is intended to run during synchronous mounts on the system which is initializing. The directory may not be created if any of these situations exist:

- The file system is mounted asynchronously, such as with NFS.
- The SYSNAME value identifies a remote system.
- The file system is already mounted on a remote system.

Note: If sharing PARMLIB members between a shared file system environment members, this keyword should be omitted unless all are running at V1R5 or above. The directory permissions are set to 755, (rwx r-x r-x).

12.33 Allocating SWA above the line

- SWA control blocks for USS address space are allocated below the 16 megabyte line causing storage constraints when very large numbers of file systems are mounted
- New BPXPRMxx PARMLIB keyword to control from where the SWA control blocks are allocated
  - New BPXPRMxx keyword SWA(ABOVE | BELOW)
  - D OMVS,O displays the setting
  - Only available when starting OMVS during system initialization

Figure 12-38 Allocating SWA control blocks above the 16 MB line

SWA above the line

Scheduler work area (SWA) control blocks for a USS address space are allocated below the 16 megabyte line. This can cause storage constraints when very large numbers of file systems are mounted.

BPXPRMxx PARMLIB keyword

z/OS V1R5 provides a BPXPRMxx PARMLIB keyword to control where the SWA control blocks are allocated, as follows:

```
SWA(ABOVE | BELOW)
```

The operator command D OMVS,O displays all the defined PARMLIB members in the BPXPRMxx PARMLIB members in use.

This keyword can only be changed or specified during an IPL of the system.

12.34 z/OS UNIX Web site

- Web site: v1r3/
  http://www.ibm.com/servers/eserver/zseries/zos/wizards/unix/unixv1r3
- UNIX Wizard
  - z/OS UNIX Configuration Assistant
    - Contains a README file
    - Button to start the Configuration Assistant
  - Builds two BPXPRMxx parmlib members
  - RACF Batch job with commands
  - HFS files

Figure 12-39 Contents of the z/OS UNIX Web site

z/OS UNIX Web site

You can use the z/OS UNIX Configuration Wizard, a Web-based tool, to help you set up z/OS UNIX in full function mode. This wizard begins with a series of interviews in which you will answer questions about your application environment and intentions regarding use of z/OS and TCP/IP.

After you finish answering all of the interview questions, you ask the wizard to build the output. Then the wizard produces a checklist of steps for you to follow, as well as customized jobs and other data sets for you to use. Specifically, it builds two BPXPRMxx members and two HFS files and some RACF ALTUSER commands. The checklist of follow-on actions includes links to sections of *z/OS UNIX System Services Planning*, GA22-7800 and *z/OS Communications Server: IP Configuration Guide*, SC31-8775, thus eliminating the need to reference multiple documents.

Use this wizard to configure z/OS UNIX for the first time or to check and verify some of your configuration settings.

The wizard also allows sysplex users to build a single BPXPRMxx PARMLIB member to define all the file systems used by systems participating in a shared file system environment.

13

Maintenance

This chapter provides information about installing maintenance software with components residing in the z/OS UNIX file system. This chapter discusses the following topics:

- Overview of maintenance issues
- The HFS structure for SMP/E
- The SMP/E DDDEF and MOUNT relationship
- A method to support multiple service levels

© Copyright IBM Corp. 2011. All rights reserved.

13.1 Example of SMP/E SMPMCS

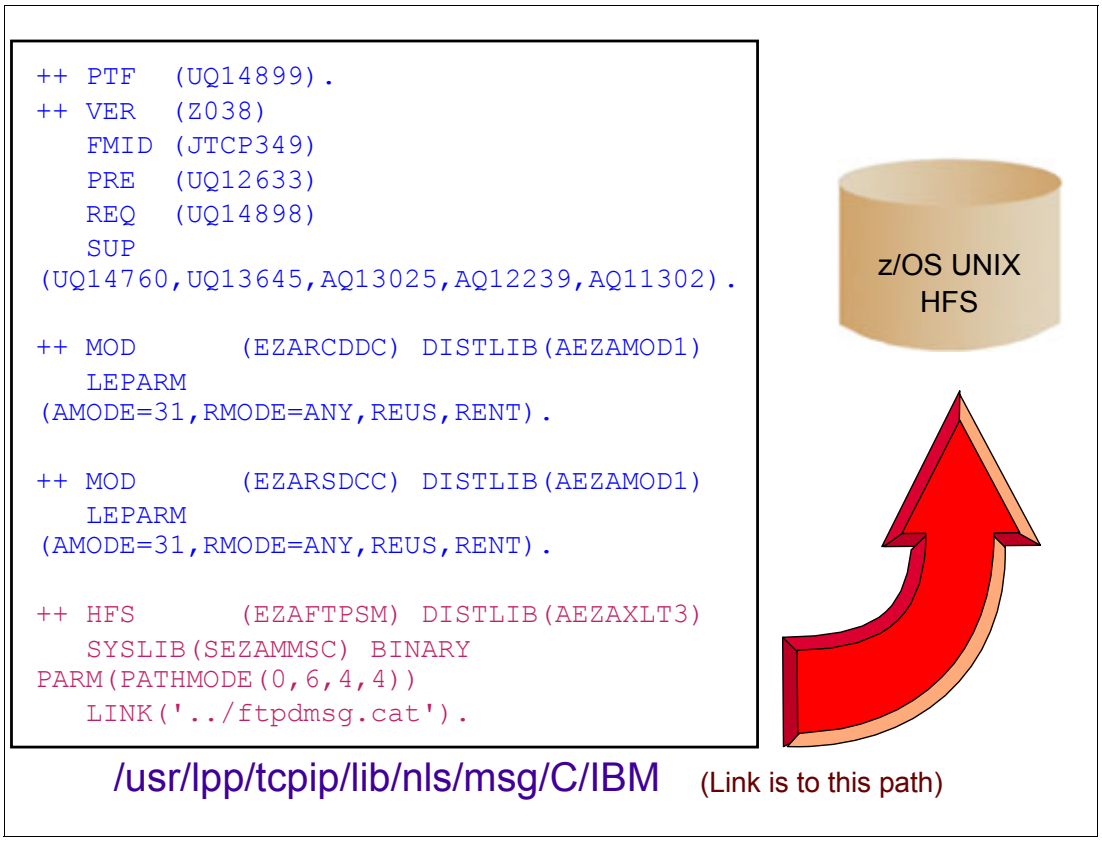


Figure 13-1 Example of an SMP/E SMPMCS

SMP/E SMPMCS example

z/OS now includes UNIX components as a standard part of the operating system. Therefore, it is now mandatory to have z/OS UNIX System Services enabled on any system where SMP/E maintenance is performed. Other traditional MVS products also now contain UNIX components; DB2 and NetView® are two such examples. Figure 13-1 shows a typical PTF for TCP/IP (a component of z/OS). You will notice that it contains typical MCS statements such as `++ PTF`, `++ VER`, and `++ MOD`. You will also notice the `++ HFS` MCS statement, which directs element `EZAFTPSM` as a file into the HFS path pointed to by DDDEF `SEZAMMSC`. In addition, a *hard link* (alias name) will be created to `EZAFTPSM` in the directory one higher in the hierarchy (indicated by the ".." in the LINK parameter), called `ftpdmsg.cat`.

In other words, if the path in DDDEF SEZAMMSC points to:

`/usr/lpp/tcpip/lib/nls/msg/C/IBM`

- Then the element will be written as file:

  `/usr/lpp/tcpip/lib/nls/msg/C/IBM/`**`EZAFTPSM`**

- With a hard link defined as:

  `/usr/lpp/tcpip/lib/nls/msg/C/`**`ftpdmsg.cat`**

Note how the original file (`EZAFTPSM`) is written in the /C/IBM subdirectory, while the hard link (`ftpdmsg.cat`) is created in the /C subdirectory, one directory higher because of the ".." specification in the LINK parameter. This type of packaging allows SMP/E unique element names while also meeting the requirements of UNIX naming conventions.

13.2 Active root file system

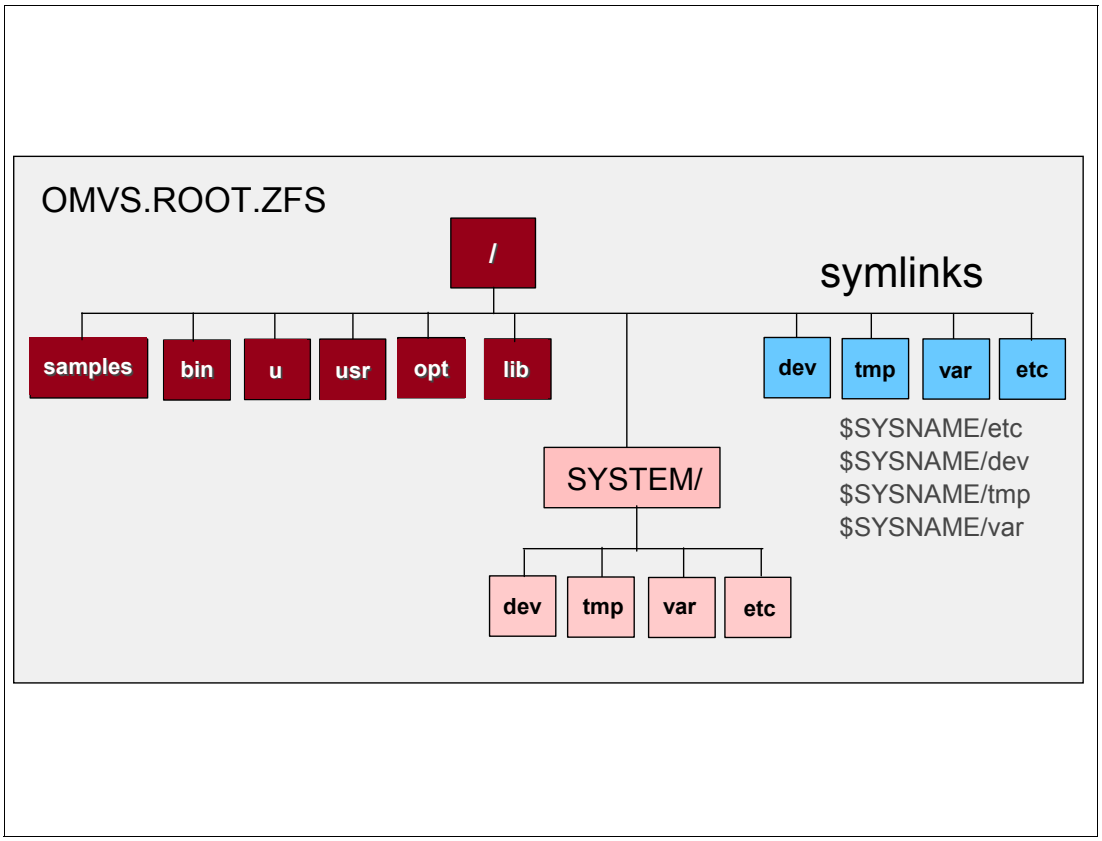


Figure 13-2 Example of an active root file system

Active root file system

Figure 13-2 shows an example of an active file system. The file system is in use by the active system (the system you are logged on to), so applying maintenance directly onto the active file system is undesirable because:

- It could introduce a change which impacts work already running.
- If a problem is encountered during the SMP/E APPLY causing the APPLY to fail, it could damage the active file system and impact work already running.
- If you need to fall back to the point before the service was applied, the process is greatly complicated when it is the active file system.

The problems are similar to those concerning z/OS system residence (SYSRES) volumes, where typically a cloned copy of the active SYSRES is used to receive maintenance, then it is IPLed. This way, application of maintenance does not affect the active system until an IPL is performed. If there is a problem with the new maintenance level, fallback is to re-IPL from the old SYSRES.

13.3 Inactive root file system (clone)

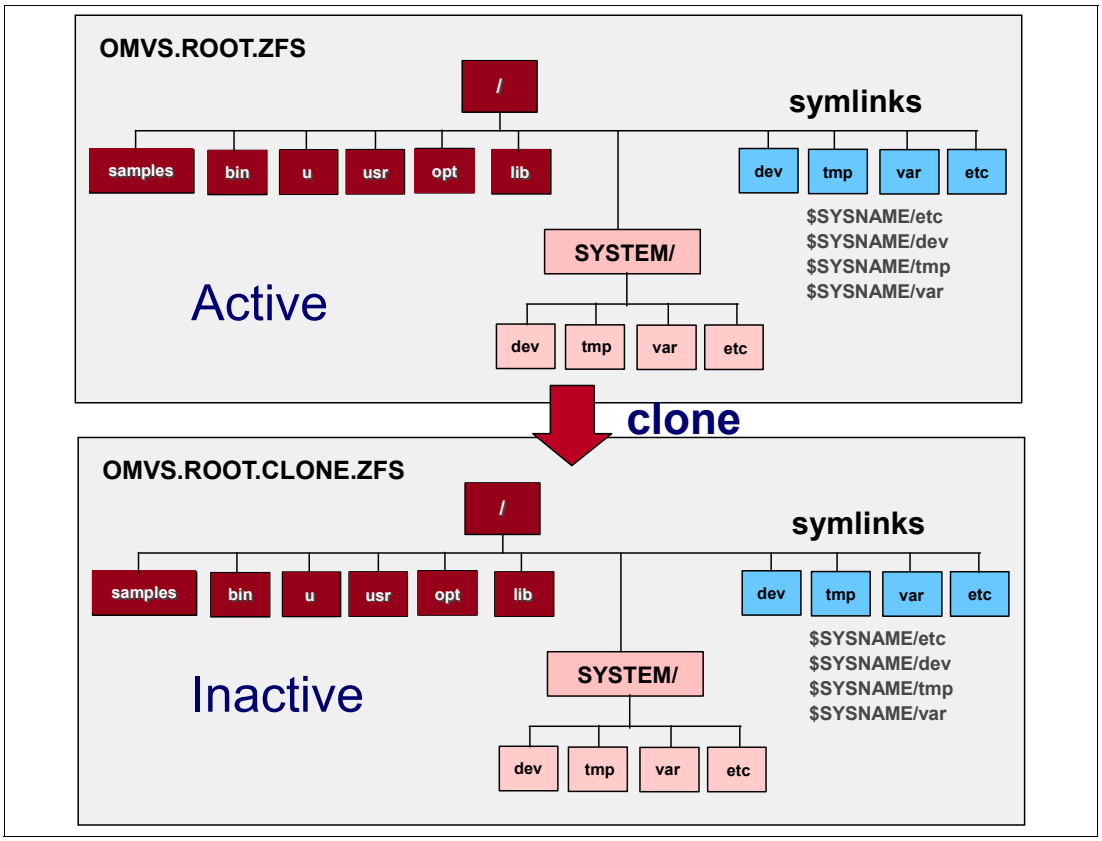


Figure 13-3 Making a clone of the active root file system

Clone the active root file system

To apply maintenance safely without impact to an active system, you need to clone the z/OS UNIX file system, and work with the inactive copy of the file system. This is the same technique used for z/OS system residence (SYSRES) volumes, only the method has to vary because we are only dealing with HFS data sets and not full volumes.

To create a cloned copy of an HFS, DFSMSdss (ADRDSSU) must be used to first DUMP, then RESTORE with the RENAMEU (RENUNC) parameter, to result in a copied file with a different name.

Sample JCL to create a clone

An example of the ADRDSSU JCL required to perform HFS cloning is shown in Example 13-1 on page 659.

Example 13-1 Sample JCL to perform HFS cloning

```
//DUMPREST JOB ,'DUMP/REST',CLASS=A
//DUMP     EXEC PGM=ADRDSSU
//SYSPRINT DD SYSOUT=*
//OUTDD    DD DISP=(NEW,PASS),DSN=&&TEMP,
//            UNIT=SYSALLDA,
//            SPACE=(TRK,(100,100))
//SYSIN    DD *
 DUMP DATASET( INCLUDE( OMVS.RESA01.** ) ) -
OUTDD(OUTDD) -
       CANCELERROR TOL(ENQF)
/*
//*
//*-------------------------------------------------------------------
//*
//RESTORE  EXEC PGM=ADRDSSU
//SYSPRINT DD SYSOUT=*
//INDD     DD DISP=(OLD,DELETE),DSN=&&TEMP
//SYSIN    DD *
RESTORE DATASET( -
           INCLUDE( - ** ) ) -
          INDD(INDD) -
      RENUNC( (OMVS.RESA01.**, OMVS.RESB01.**) ) -
          TGTALLOC(SOURCE) -
          TOL(ENQF)
/*
```

13.4 /SERVICE directory

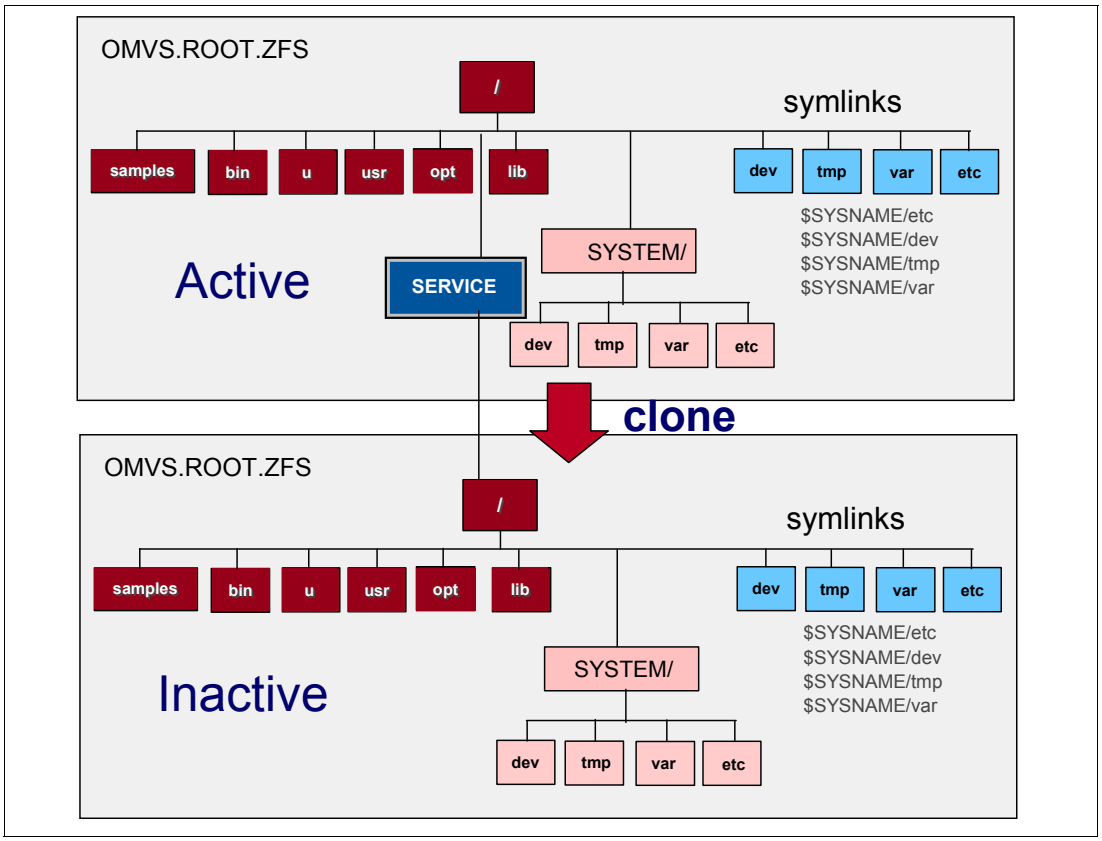


Figure 13-4 Creating a /SERVICE directory

Create a /SERVICE directory

z/OS UNIX can only access files if the HFS that contains them is mounted into the active file system. So, the newly created clone file system needs to be somehow connected into the active file system structure so SMP/E processes can access the data.

The file system could be connected anywhere in the active file system, but the recommended place to do it is under a directory called /SERVICE. Using this directory will ensure no impact to other UNIX processing.

If this directory does not exist on your system, you could create it via a TSO command, shell command, or using the ISHELL. As an example, the TSO command to do this is:

```
MKDIR '/SERVICE' MODE(7,5,5)
```

Once the /SERVICE directory has been created, the cloned file system needs to be mounted there. The mount can happen by specifying the HFS in BPXPRMxx or issuing a TSO MOUNT command.

13.5 Sample SMP/E DDDEFs

```
Entry Type:  DDDEF                          Zone Name:
OS#250T
Entry Name:  SEZALOAD                       Zone Type: TARGET

DSNAME: SYS1.SEZALOAD

VOLUME: RESB01      UNIT: SYSALLDA  DISP:   SHR
SPACE :             PRIM:           SECOND:          DIR:
SYSOUT CLASS:                       WAITFORDSN:
PROTECT:
DATACLAS:                           MGMTCLAS  :
STORCLAS:                           DSNTYPE   :
```

```
Entry Type:  DDDEF                          Zone Name:
OS#250T
Entry Name:  SEZAMMSC                       Zone Type: TARGET

 PATH: '/SERVICE/usr/lpp/tcpip/lib/nls/msg/C/IBM/'
```

Figure 13-5 An example of a SMP/E DDDEFs

SMP/E DDDEFs

Once the HFS data sets are cloned into the /SERVICE directory, some decisions need to be made about the SMP/E configuration that will be used to support the cloned environment. You could either change the DDDEFs of the old SMP/E configuration (meaning that the active environment is no longer maintainable), or you could clone the SMP/E CSIs and change the new CSIs to point to the new HFS.

Although we are mainly talking about HFS data sets in this section, of course you must ensure the integrity of the SMP/E configuration and make sure that all DDDEFs in the CSIs point to data sets with contents that match what is recorded in SMP/E. How this is handled may be different depending on the product and how it is implemented on your system. We discuss this in more detail in the next few sections.

ZONEEDIT command

Regardless of how you choose to solve the problem of SMP/E integrity, you will still have to adjust the SMP/E DDDEFs to point to the cloned file system mounted at /SERVICE. The SMP/E ZONEEDIT command can be used to do this.

Figure 13-5 shows a traditional DDDEF (top box) and an HFS DDDEF (bottom box). The traditional DDDEF SEZALOAD points to a PDS called `SYS1.SEZALOAD` on DASD volume `RESB01`, while the HFS DDDEF SEZAMMSC points to a path called /SERVICE/usr/lpp/tcpip/lib/nls/msg/C/IBM/ (note the /SERVICE directory).

13.6 Prepare for SMP/E

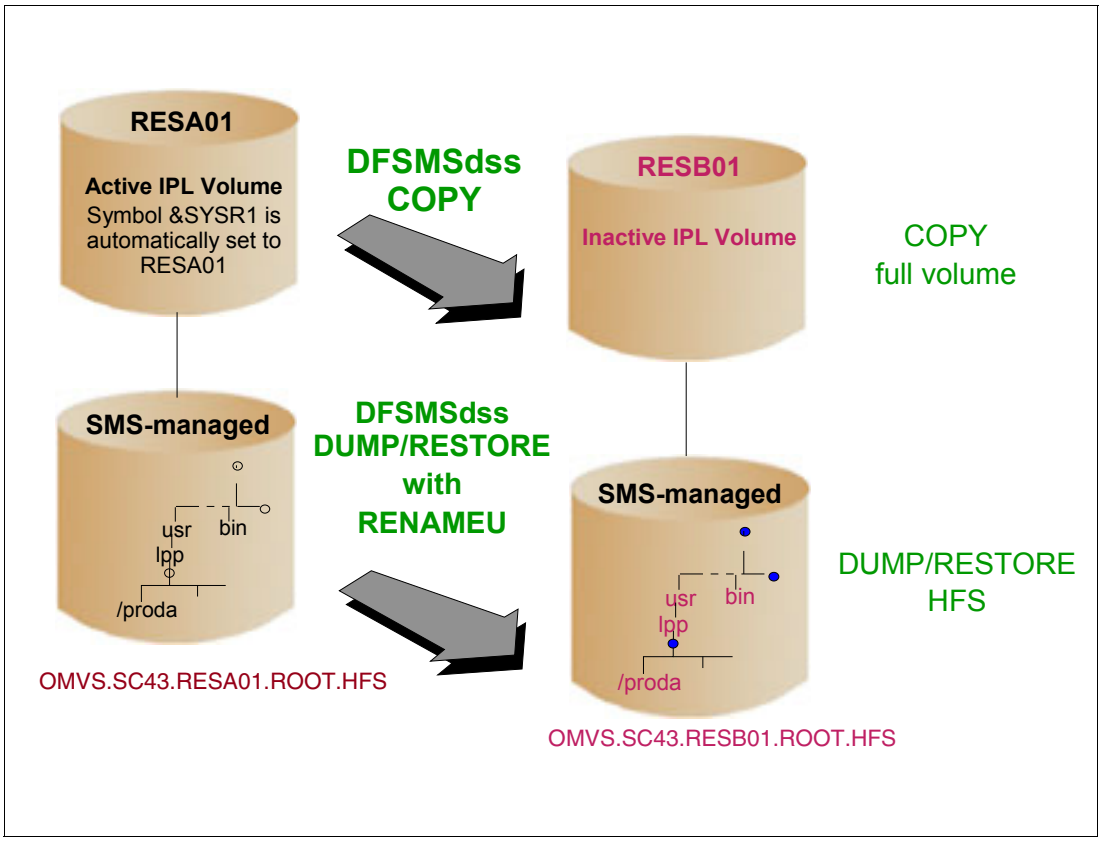


Figure 13-6 Preparing for the SMP/E maintenance

Prepare SMP/E maintenance

The preparation for system maintenance should include the following steps:

- Clone the data sets that the SMP/E DDDEFs point to:
  - For non-HFS data sets, you would DFSMSdss (ADRDSSU) COPY the active files to create inactive equivalents. You should consider that:
    - SYSRES data sets would have the entire volume copied, such that the inactive data sets have the same names but are uncataloged. This figure shows active volume RESA01 being copied to RESB01.
    - Non-SYSRES data sets may either be copied to another volume with the same names, but uncataloged; or they may be copied with other names, allowing them to be cataloged if you choose.
  - For HFS or zFS data sets, you would DFSMSdss (ADRDSSU) DUMP/RESTORE the active files to create inactive equivalents (as described in previous sections). You should consider that:
    - Data sets that belong to SYSRES products need something in their name to associate them to the correct SYSRES. For example, say you clone SYSRES volume RESA01 onto volume RESB01: the HFS data sets that relate to RESB01 should have RESB01 in their name. This allows the use of the &SYSR1. symbol in the BPXPRMxx member so that the HFS data sets that relate to the IPLed SYSRES are always correctly selected.

For example, BPXPRMxx contained a reference to HFS data set OMVS.&SYSNAME..&SYSR1..ROOT.HFS.

- If you IPLed from volume RESA01, symbol &SYSR1. would be substituted with string RESA01 resulting in a data set name of OMVS.SC43.RESA01.ROOT.HFS.
- If you IPLed from volume RESB01, symbol &SYSR1. would be substituted with string RESB01 resulting in a data set name of OMVS.SC43.RESB01.ROOT.HFS.

– This technique simplifies moving back and forth between maintenance levels (test sessions, implementations, and so on).

– Because HFS data sets cannot be shared for write, HFS data set names in a shared DASD configuration may need a system identifier (&SYSNAME.) in them to differentiate which system they belong to. This may be more complicated in a shared DASD configuration where multiple systems are IPLed from the same DASD volume. In this situation, both &SYSNAME. and &SYSR1. may be needed in the affected HFS names. For example: OMVS.&SYSNAME..&SYSR1..ROOT.HFS

► Make SMP/E point to the inactive data. You could choose to:

– Change the DDDEFs in the existing SMP/E CSIs to point to the inactive data. This would result in the active data having no SMP/E CSIs pointing to them, which may be a problem if an emergency fix needs to be applied to the live system.

– Clone the SMP/E CSIs so the old CSIs continue pointing to the active data, while the DDDEFs in the new CSIs can be ZONEEDITed to point to the inactive data.

Note: The DFSMSdss COPY function is supported for HFS data sets beginning in z/OS V1R3.

13.7 SMP/E APPLY process

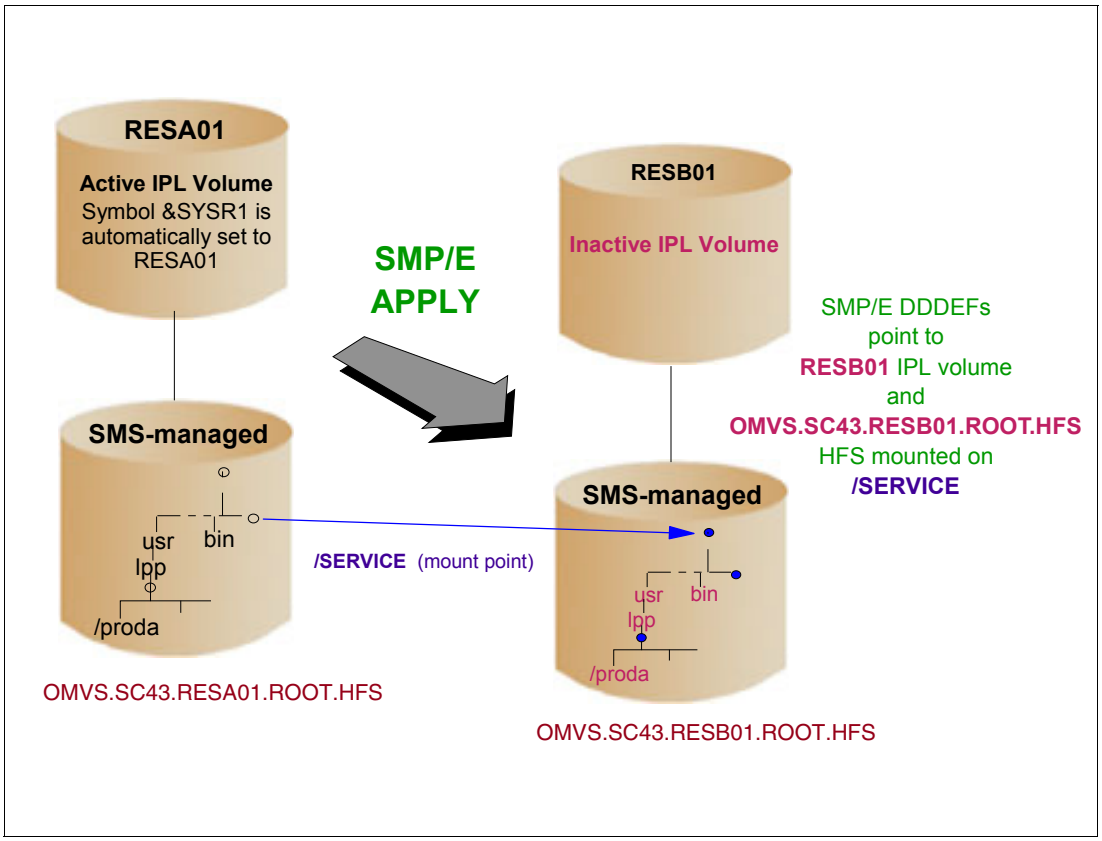


Figure 13-7 Doing the SMP/E APPLY

SMP/E APPLY process

For the SMP/E APPLY process to work as desired, you should have inactive clones of the active data sets.

HFS data sets should be mounted into the active file system off the /SERVICE directory. The data sets could be mounted there permanently via the BPXPRMxx member, or dynamically via a TSO MOUNT command. For the example shown in Figure 13-7, the following command could be used:

```
MOUNT FILESYSTEM('OMVS.SC43.RESB01.ROOT.HFS') +
    TYPE(HFS) MODE(RDWR) MOUNTPOINT('/SERVICE')
```

If you wanted to dynamically remove the file system after maintenance work has been performed, the following command could be used:

```
UNMOUNT FILESYSTEM('OMVS.SC43.RESB01.ROOT.HFS') +
      IMMEDIATE
```

SMP/E ZONEEDIT command

SMP/E CSIs point to the inactive data sets. No matter how you choose to set up your SMP/E CSIs as described in the previous section, you will probably need to use an SMP/E ZONEEDIT command such as the following to get the DDDEFs set correctly:

```
ZONEEDIT DDDEF.
      CHANGE VOLUME(RESA01,
                    RESB01).
      CHANGE PATH('/'*,
                  '/SERVICE/'*).
    ENDZONEEDIT.
```

13.8 Supporting multiple service levels

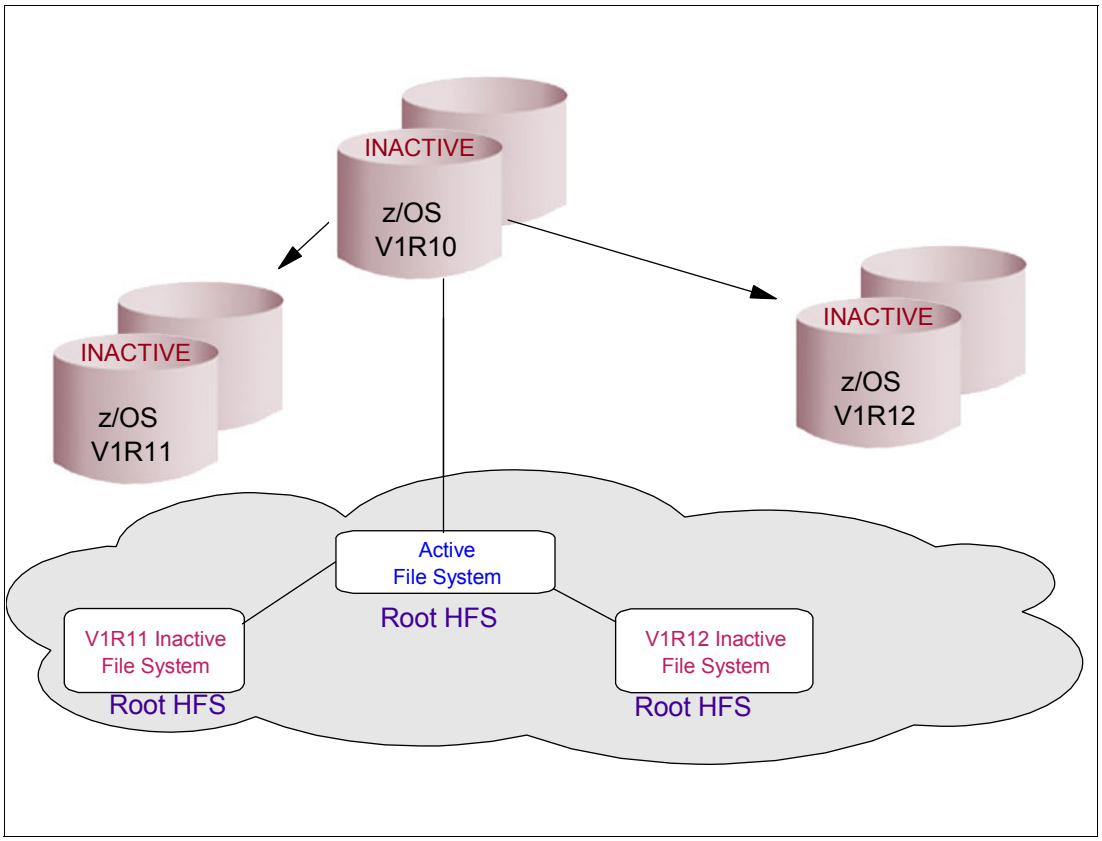


Figure 13-8 Supporting multiple service levels

Support for multiple service levels

Suppose you need to support more than one level of software as you migrate from one level to the next. Or maybe you work in a large organization and have to support many levels of software simultaneously. For this, some additional considerations are necessary, particularly concerning the HFS data sets and file system structure.

Figure 13-8 shows an active z/OS V1R10 system where the SMP/E work is performed in support of V1R11 and V1R12. You will notice there are multiple HFS data sets in SMS to match all the z/OS levels, and which need to be somehow connected into the file system. Until now we have only talked about using the /SERVICE directory to support one inactive file system for maintenance. We could still do this by mounting the correct file system when doing maintenance, then unmounting it afterwards, but there are better ways than that, as shown in Figure 13-9 on page 667.

13.9 Supporting multiple service levels (2)

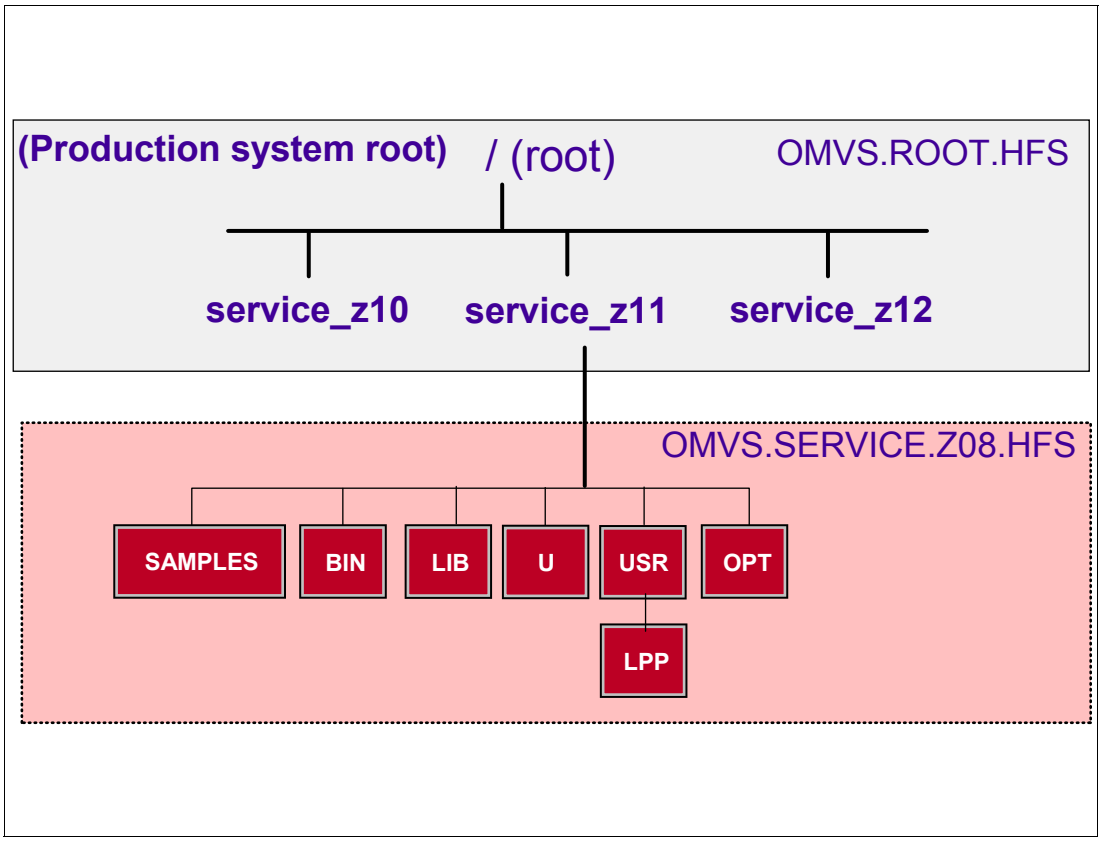


Figure 13-9 A directory structure for multiple service levels

Multiple service level directory structure

This figure shows a possible directory structure for connecting multiple service levels into the **/** directory. The inactive file systems would be mounted something like this:

- z/OS V1R10 at **/service_z010**
- z/OS V1R11 at **/service_z011**
- z/OS V1R12 at **/service_z012**

There is no problem in doing this as long as the path in the DDDEFs match the directory structure. So, an SMP/E ZONEEDIT command something like the following may be required to set the SMP/E DDDEFs correctly:

```
ZONEEDIT DDDEF.
CHANGE PATH('/'*, '/service_z012’).
ENDZONEEDIT.
```

13.10 ISHELL display of root

```
Select one or more files with / or action codes.  If / is used also select an
action from the action bar otherwise your default action will be used.  Select
with S to use your default action.  Cursor select can also be used for quick
navigation.  See help for details.
EUID=0   /
  Type  Filename                                                Row 101 of 191
_ Syml  service_z010_Z10RA1
_ Syml  service_z010_Z10RB1
_ Syml  service_z010_Z10RC1
_ Syml  service_z010_Z10RD1
_ Syml  service_z010_Z10RE1
_ Syml  service_z010_Z10RF1
_ Syml  service_z011
_ Syml  service_z011_MXARX1
_ Syml  service_z011_MXARY1
_ Syml  service_z011_Z11RA1
_ Syml  service_z011_Z11RB1
_ Syml  service_z011_Z11RC1
_ Syml  service_z011_Z11RD1
_ Syml  service_z011_Z11RE1
_ Syml  service_z012
_ Syml  service_z012_Z12RA1
_ Syml  service_z012_Z12RB1
_ Syml  service_z012_Z12RC1
```

Figure 13-10 An ISHELL display of the root directory

Display the root directory using ISHELL

This figure shows the ITSO system where maintenance is done. It shows the directories that were added to the root for each of the system levels of z/OS supported and the individual products where maintenance is applied.

13.11 The chroot command

- Provides a way to test fixes applied to the root HFS
  - Get into production faster
- chroot command allows changing the root HFS
  - Changes to root directory for execution of a command
- chroot Command syntax
  - chroot directory command
- Must be a superuser to issue command

Figure 13-11 Using the chroot command

The chroot command

The **chroot** command allows the system programmer to test fixes and new releases easier and faster. This command is not part of the XPG4 or UNIX98 standard. The externals are somewhat modeled on AIX externals.

Directory path

The directory path name is always relative to the current root. If a nested **chroot** command is in effect, the directory path name is still relative to the current (new) root of the running process.

Command syntax

In order for your process to operate properly after **chroot** is issued, you need to have in your new root all the files that your program depends on. For example, if your new root is /tmp and you issue an **ls**, you will get a `not found` error. To use **ls** with /tmp as your new root, you need a /tmp/bin with **ls** in it before you issue the **chroot** command. The directory path name is always relative to the current root. After **chroot** is issued, your current working directory is the new root (directory).

chroot user authority

If you have appropriate privileges, the **chroot** command changes the root directory to the directory specified by the directory parameter of a specific command. You must either be a superuser (UID=0), or be a member of the BPX.SUPERUSER facility class.

13.12 Testing a root file system

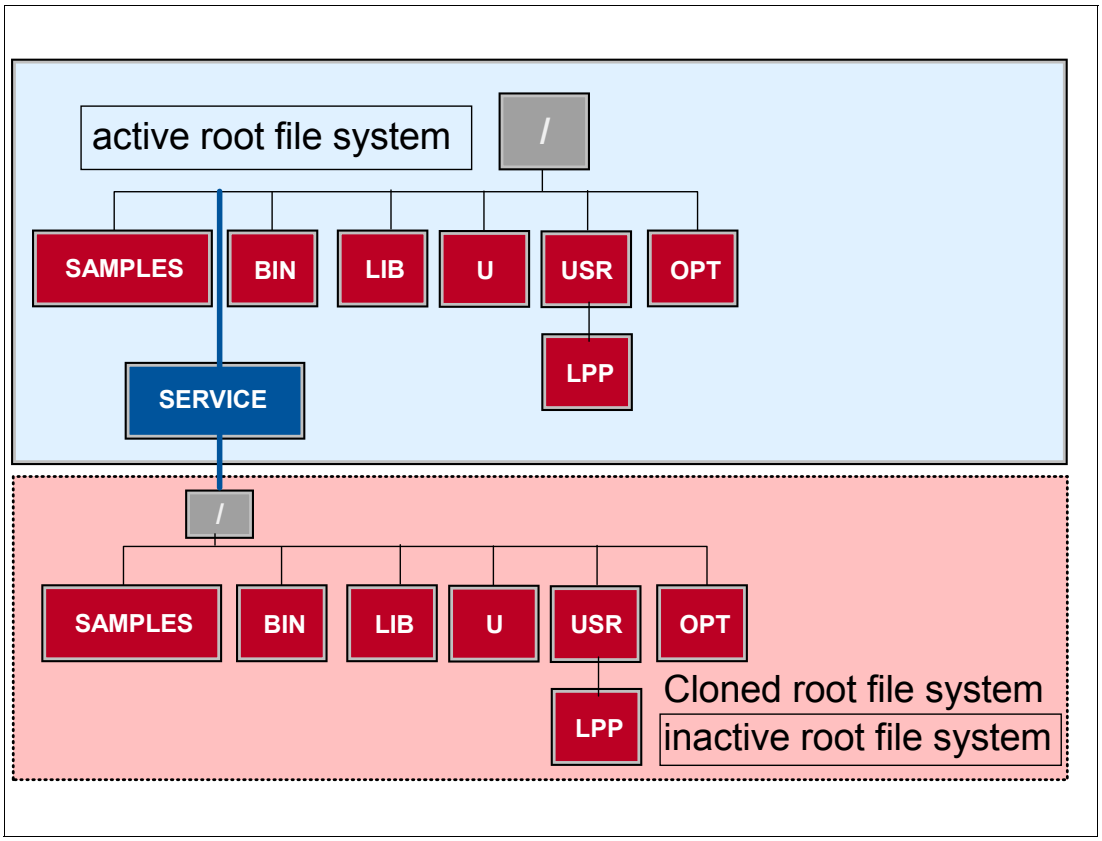


Figure 13-12 Testing the updated clone of the root file system

Testing the clone after maintenance

Figure 13-12 shows the new inactive root file system mounted on the /SERVICE directory in the active root file system.

The **`chroot`** service changes the root directory from the current one to a new one. The root directory is the starting point for path searches of pathnames beginning with a slash. The working directory of the process is unaffected by **`chroot()`**.

13.13 Testing the updated root

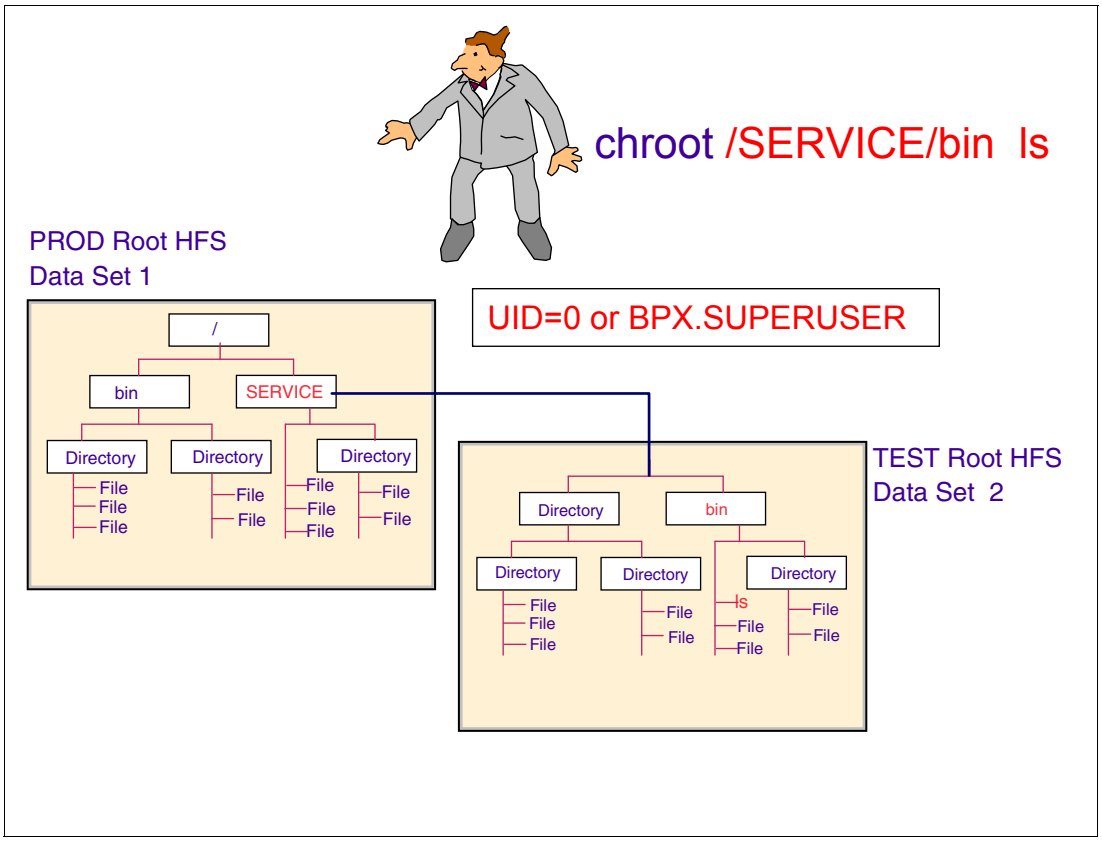


Figure 13-13 Testing the updated root with the chroot command

Testing the root using chroot

If you have appropriate privileges, the **`chroot`** command changes the root directory to the directory specified by the directory parameter of a specific command. The new root directory will also contain its children.

The directory path name is always relative to the current root. If a nested **`chroot`** command is in effect, the directory path name is still relative to the current (new) root of the running process. **`chroot`** does not change environment variables. In order for the command to operate properly after the **`chroot`** is issued, you need to have all the files in the new root directory that the command depends on.

For example, if your new root is /SERVICE and you issue an **`ls`**, you will get a `not found` error. To use **`ls`** with /SERVICE as your new root, you need a /SERVICE/bin with **`ls`** in it before you issue the **`chroot`** command.

13.14 Dynamic service activation

- Previously, you cannot activate maintenance for z/OS UNIX without taking a system outage
- z/OS V1R7 - provide ability to activate maintenance on a running system in a non-disruptive manner
  - Minimize number of planned and unplanned outages
  - Provide higher level of system availability
- Only PTFs with ++HOLD REASON (DYNACT) data will be capable of dynamic activation
- New BPXPRMxx parmlib parameters SERV_LPALIB and SERV_LINKLIB need to be setup to enable the support

```
SERV_LPALIB('dsname','volser')
SERV_LINKLIB('dsname','volser')
```

Figure 13-14 Using dynamic service activation

Dynamic service activation

You can dynamically activate and deactivate service items (PTFs, ++APARS, ++Usermods) that affect the z/OS UNIX System Services component modules without having to re-IPL. This capability is primarily intended to allow an installation to activate corrective service to avoid unplanned re-IPLs of your systems. Additionally, this capability can be used to activate a temporary patch that can be used in gathering additional documentation for a recurring system problem. Although this capability can be used to activate preventive service on an ongoing basis, it is not intended for this purpose as a replacement for the regular application of service that does require a re-IPL.

Identified PTFs

Those PTFs that are capable of being activated dynamically are identified by ++HOLD REASON(DYNACT) data inside their PTFs. In order to properly activate the PTF, whatever instructions that are included with this hold data must be followed.

BPXPRMxx PARMLIB changes

Service items are activated from service activation libraries that have been identified via the SERV_LPALIB and SERV_LINKLIB parameters in the BPXPRMxx PARMLIB member.

13.15 Dynamic service activation commands

- To activate service at any time
  - F OMVS, ACTIVATE=SERVICE
- To deactivate service
  - F OMVS,DEACTIVATE=SERVICE
- To retrieve dynamic service activation information
  - D OMVS,ACTIVATE=SERVICE
- Support to retrieve new PARMLIB settings
  - D OMVS,O
  - C/C++ API get_system_settings()

Figure 13-15 Dynamic service activation commands

F OMVS,ACTIVATE=SERVICE

To activate the component service items, issue:

```
F OMVS,ACTIVATE=SERVICE
```

If a fix capable of dynamic activation is found that cannot be activated due to back-level service found on the active system or missing parts in the target activation libraries, the activation fails.

F OMVS,DEACTIVATE=SERVICE

You can back off a set of dynamically activated service items, if you need to. This may be necessary if, for example, a problem is encountered with a dynamically activated service item or if a particular service item is no longer necessary. To deactivate the service items, issue:

```
F OMVS,DEACTIVATE=SERVICE
```

Only the service items that were activated when F OMVS,ACTIVATE=SERVICE was last issued are backed off. You will see a list of service items to be deactivated and you will be asked whether the deactivation should proceed.

D OMVS,ACTIVATE=SERVICE

Use this command to display all of the service items that were dynamically activated using F OMVS,ACTIVATE=SERVICE. It displays only those service items that are currently active

dynamically. Once a fix has been deactivated, it no longer shows up in this command's display output.

Additionally, D OMVS,ACTIVATE=SERVICE reports the library and volume where each set of fixes was activated from and the amount of ECSA and OMVS address space storage that is being consumed for all dynamically activated fixes. The amount of storage consumed will not decrease when a deactivation is done, because the new modules must remain in storage indefinitely.

13.16 Using the new service

- Stay fairly current on service to best be able to exploit this feature
- Determine the selected PTFs you are interested in activating dynamically for corrective purposes
- Not intended to be used as a complete replacement for regular preventative maintenance application

F OMVS, ACTIVATE=SERVICE

```
BPXM061I THE FOLLOWING SERVICE ITEMS WILL BE ACTIVATED:
         OA09999  OA08888
   ECSA STORAGE BYTES:      24576 AND OMVS PRIVATE STORAGE BYTES:    12468
   WILL BE CONSUMED FOR THIS ACTIVATION.
   *06 BPXM061D REPLY "Y" TO CONTINUE. ANY OTHER REPLY ENDS THE COMMAND.
```

Figure 13-16 Activating PTFs

Activating service

To ensure that you activate only those service items that are of interest, it is recommended that you additionally install these service items into a separate load library from the LPALIB or LINKLIB libraries that are used for your normal install process. This would then be the load library that is regularly used to dynamically activate service on your systems. Although the dynamic service activation feature can be used to activate most z/OS UNIX System Services component PTFs, it is not intended to be used as a way to activate a large set of maintenance for preventive purposes.

Only those service items found in the target libraries that are identified internally by z/OS UNIX System Services as capable of being dynamically activated are activated. Service items that are not explicitly identified as such cannot be activated. If a fix capable of dynamic activation is found that cannot be activated due to back-level service on the active system or missing parts in the target activation libraries, the activation will fail.

Some z/OS UNIX System Services modules and csects will not be capable of dynamic activation due to their location or when they are run. As a result, they still require ++HOLD for IPL documentation in their PTFs (as almost all z/OS UNIX System Services PTFs do currently). It is expected that a very small percentage of PTFs will be affected by this limitation.

The set of service items to be activated and the amount of ECSA and OMVS address space storage consumed for those service items are indicated in messages displayed by the F

OMVS,ACTIVATE=SERVICE command. The issuer of the command is then prompted whether to proceed with the activation based on this information.

13.17 Deactivate service

- F OMVS,DEACTIVATE=SERVICE
  - Enables the backing off of the last set of service items that were activated dynamically
  - Intended for removal of temporary service activated for problem determination, etc. or problematic service
  - Backs off service items but modules remain in storage
  - Prompts user to verify deactivation

```
BPXM063I THE FOLLOWING SERVICE ITEMS WILL BE DEACTIVATED:
     OA09999  0A08888
   *07 BPXM063D REPLY "Y" TO CONTINUE. ANY OTHER REPLY ENDS THE COMMAND.
   .
```

Figure 13-17 Command to deactivate service

Displaying deactivated service

You can back off a set of dynamically activated service items, if you need to. This may be necessary if, for example, a problem is encountered with a dynamically activated service item or if a particular service item is no longer necessary.

This capability can be used to activate a temporary patch that can be used in gathering additional documentation for a recurring system problem. Although this capability can be used to activate preventive service on an ongoing basis, it is not intended for this purpose as a replacement for the regular application of service that does require a re-IPL.

The amount of storage consumed will not decrease when a deactivation is done, because the new modules must remain in storage indefinitely.

The service items are listed in groups based on when they were activated. All service items activated by a given F OMVS,ACTIVATE=SERVICE command are listed together as one set of activated service items. The most recently activated set of service items is listed first, followed in descending order by the next most recent activation set, and so on. The most recent set of service items consists of the only service items that are deactivated if a F OMVS,DEACTIVATE=SERVICE command is issued.

13.18 Display service

- D OMVS,ACTIVATE=SERVICE
  - Provides the display of all sets of service items that are currently activated dynamically
  - Shows the most recently activated set of service items to the oldest set activated

```
BPXO059I 08.51.42 DISPLAY OMVS 284
                  OMVS     000E ACTIVE          OMVS=(6D)
                 DYNAMIC SERVICE ACTIVATION REPORT
           SET  #2:
             LINKLIB=SYS1.DYNLIB.PVT                  VOL=BPXLK1
             LPALIB=SYS1.DYNLIB.LPA                   VOL=BPXLK1
             OA02001  OA02002  OA02003  OA02004
           SET  #1:
             LINKLIB=SYS1.LINKLIB                  VOL=Z17RS1
         LPALIB=SYS1.LPALIB                     VOL=Z17RS1
               OA01001  OA01002  OA01003
          ECSA STORAGE:  68496        OMVS  STORAGE:  268248    .
```

Figure 13-18 Displaying the activated service items

Display activated service items

Use the D OMVS,ACTIVATE=SERVICE command to display all of the service items that were dynamically activated using F OMVS,ACTIVATE=SERVICE. It displays only those service items that are currently active dynamically. Once a fix has been deactivated, it no longer shows up in this command's display output.

Additionally, D OMVS,ACTIVATE=SERVICE reports the library and volume that each set of fixes was activated from, and the amount of ECSA and OMVS address space storage that is being consumed for all dynamically activated fixes. The amount of storage consumed will not decrease when a deactivation is done, because the new modules must remain in storage indefinitely.

14

z/OS UNIX operations

This chapter provides an overview of z/OS UNIX System Services operations. It provides details about:

- z/OS UNIX operator commands
- z/OS UNIX abends and messages
- Sources of debugging information for z/OS UNIX

© Copyright IBM Corp. 2011. All rights reserved.

14.1 Commands to monitor z/OS UNIX

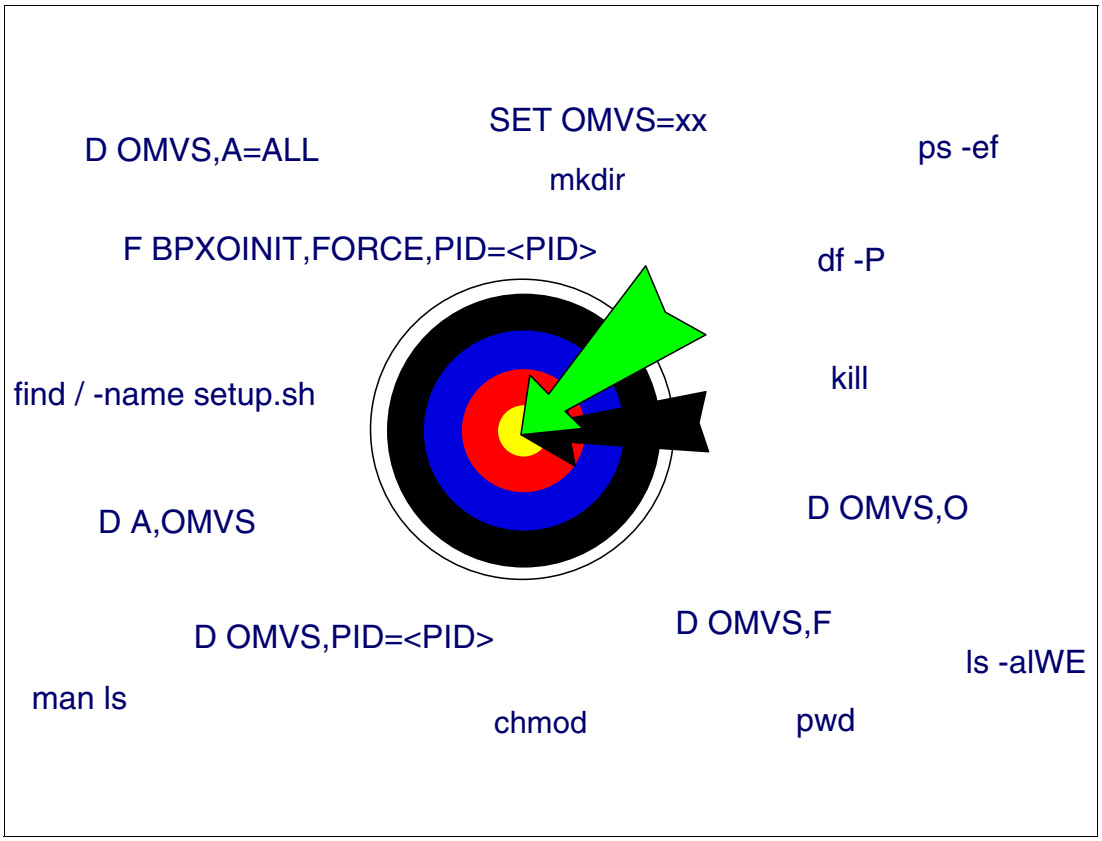


Figure 14-1 Commands used to monitor z/OS UNIX

Monitoring z/OS UNIX

There are many different reasons to monitor z/OS UNIX, for example:

- To kill a process
- To see how much space a directory needs
- To activate a new configuration for z/OS UNIX
- To find a file located in the HFS
- To change your local directory

Therefore, you need to know which commands show the information that you want, and which commands provide useful information without incurring too much overhead.

Commands can be divided into two sections: commands you issue from a console, and commands you issue from the z/OS UNIX shell.

z/OS console commands

The following commands are z/OS (MVS) commands that are issued from MCS or EMCS consoles:

D OMVS,A=ALL	Displays the information about all running processes
D OMVS,O	Displays the information defined in the BPXPRMxx
D OMVS,F	Displays the information about the mounted HFS

D OMVS,PID=<pid>	Displays the information about the process ID
D A,OMVS	Displays the information about kernel and data spaces
F BPXOINIT,FORCE,PID=<PID>	Forces a process ID to end
SET OMVS=xx	Sets a new configuration of BPXPRMxx member and makes a syntax check

z/OS UNIX shell commands

The following commands are issued from the OMVS shell:

pwd	Displays the current pathname
ls -alWE	Displays the contents and extended attributes of the current directory
ps -ef	Displays the information about all running processes
man ls	Displays information about syntax and use of the command **ls**
df -P	Displays the information about the mounted HFS
find / -name setup.sh	Searches the HFS from the root to find the file specified

14.2 Display summary of z/OS UNIX

```
D OMVS
BPXO042I 13.31.39 DISPLAY OMVS 908
OMVS     000E ACTIVE          OMVS=(00,FS)
```

Figure 14-2 Command to display the status of z/OS UNIX

Displaying the status of z/OS UNIX

The **D OMVS** command with no parameters displays the status of z/OS UNIX (hopefully always ACTIVE), and which BPXPRMxx member was defined during IPL. The value 000E shown in Figure 14-2 is the address space identifier (ASID) of the kernel address space.

The status of z/OS UNIX can be:

ACTIVE	z/OS UNIX is currently active
NOT STARTED	z/OS UNIX was not started
INITIALIZING	z/OS UNIX is initializing
TERMINATING	z/OS UNIX is terminating
TERMINATED	z/OS UNIX has terminated
ETC/INIT WAIT	z/OS UNIX is waiting for the /etc/init or /usr/sbin/init program to complete initialization

14.3 Display z/OS UNIX options

```
D OMVS,OPTIONS
BPXO043I 13.21.39 DISPLAY OMVS 923
OMVS     000E ACTIVE          OMVS=(3C)
CURRENT UNIX CONFIGURATION SETTINGS:
MAXPROCSYS      =       4096    MAXPROCUSER     =      32767
MAXFILEPROC     =      65535    MAXFILESIZE     = NOLIMIT
MAXCPUTIME      = 2147483647    MAXUIDS         =        200
MAXPTYS         =        800
MAXMMAPAREA     =       4096    MAXASSIZE       = 2147483647
MAXTHREADS      =     100000    MAXTHREADTASKS  =      32767
MAXCORESIZE     =    4194304    MAXSHAREPAGES   =     331072
IPCMSGQBYTES    =     262144    IPCMSGQMNUM     =      10000
IPCMSGNIDS      =      20000    IPCSEMNIDS      =      20000
IPCSEMNOPS      =      32767    IPCSEMNSEMS     =      32767
IPCSHMMPAGES    =     524287    IPCSHMNIDS      =      20000
IPCSHMNSEGS     =       1000    IPCSHMSPAGES    =    2621440
SUPERUSER       = BPXROOT       FORKCOPY        = COW
STEPLIBLIST     = /usr/lpp/IMiner/steplib
USERIDALIASTABLE= /etc/ualiastable
PRIORITYPG VALUES: NONE
PRIORITYGOAL VALUES: NONE
MAXQUEUEDSIGS   =     100000    SHRLIBRGNSIZE   =   67108864
SHRLIBMAXPAGES  =    3145728    VERSION         = Z08RD1
SYSCALL COUNTS  = NO            TTYGROUP        = TTY
SYSPLEX         = YES           BRLM SERVER     = SC04
LIMMSG          = SYSTEM        AUTOCVT         = OFF
RESOLVER PROC   = DEFAULT
AUTHPGMLIST     = NONE
SWA             = BELOW
SERV_LINKLIB    = SYS1.LINKLIB   Z18RS1
SERV_LPALIB     = SYS1.LPALIB   Z18RS1
```

Figure 14-3 Command to display the z/OS UNIX options defined in the BPXPRMxx member

Displaying the BPXPRMxx PARMLIB definitions

The D OMVS,OPTIONS command displays the same information as the D OMVS command, but in addition, all the option settings that were defined in BPXPRMxx.

This command displays the current settings of the options that were:

- Set during initialization in the PARMLIB member BPXPRMxx
- Set by a SET OMVS or SETOMVS command after initialization, and that can be altered dynamically by these commands

You can use this command to display address space information for a user who has a process that is hung. You can also use the information returned from this command to determine how many address spaces a given TSO/E user ID is using, whether an address space is using too many resources, and whether a user's process is waiting for a z/OS UNIX kernel function to complete.

14.4 Display BPXPRMxx limits

```
D OMVS,LIMITS
BPXO051I 17.50.39 DISPLAY OMVS 356
OMVS     000F ACTIVE          OMVS=(4A)
SYSTEM WIDE LIMITS:         LIMMSG=NONE
                  CURRENT  HIGHWATER     SYSTEM
                    USAGE      USAGE      LIMIT
MAXPROCSYS             50         51        300
MAXUIDS                 0          0         50
MAXPTYS                 0          0        256
MAXMMAPAREA          3572       3572       4096
MAXSHAREPAGES        4516       4516   32768000
IPCMSGNIDS             10         10      20000
IPCSEMNIDS              0          0      20000
IPCSHMNIDS              0          0      20000
IPCSHMSPAGES            0          0    2621440
IPCMSGQBYTES          ---          0     262144
IPCMSGQMNUM           ---          0      10000
IPCSHMMPAGES          ---          0      25600
SHRLIBRGNSIZE    11534336   11534336   67108864
SHRLIBMAXPAGES        472        472       4096
```

Figure 14-4 Command to display the limits specified in the BPXPRMxx member

Display BPXPRMxx member limits

Using the D OMVS,LIMITS command, you can display information about current system-wide PARMLIB limits, including current usage and high-water usage.

An asterisk (*) displayed after a system limit indicates that the system limit was changed via a SETOMVS or SET OMVS command.

The display output shows for each limit the current usage, high-water (peak) usage, and the system limit as specified in the BPXPRMxx PARMLIB member. The displayed system values may be the values as specified in the BPXPRMxx PARMLIB member, or they may be the modified values resulting from the SETOMVS or SET OMVS commands.

You can also use the D OMVS,LIMITS command with the PID= operand to display information about high-water marks and current usage for an individual process.

The high-water marks for the system limits can be reset to 0 with the D OMVS,LIMITS,RESET command. Process limit high-water marks cannot be reset.

14.5 Display address space information

```
D OMVS,ASID=ALL
BPXO040I 16.20.09 DISPLAY OMVS 737
OMVS     000F ACTIVE          OMVS=(8A)
USER     JOBNAME  ASID        PID       PPID STATE   START     CT_SECS
OMVSKERN BPXOINIT 0025          1          0 MRI   08.01.38      2.946
  LATCHWAITPID=         0 CMD=BPXPINPR
  SERVER=Init Process                    AF=    0 MF=00000 TYPE=FILE
STC      NFSCLNT  0028   16777219          1 1R    08.01.54      1.749
  LATCHWAITPID=         0 CMD=BPXVCLNY
TCPIPOE  TCPIPMVS 0053         29          1 MR    08.02.22  19950.289
  LATCHWAITPID=         0 CMD=EZBTTMST
STC      RMFGAT   0059   33554462          1 1R    08.03.20  60749.552
  LATCHWAITPID=         0 CMD=ERB3GMFC
STC      EJWSBKR  0058         31          1 1FI   08.03.18      1.796
  LATCHWAITPID=         0 CMD=EJWSCOM
OMVSKERN INETD1   001D         32          1 1FI   08.03.18      1.512
  LATCHWAITPID=         0 CMD=/usr/sbin/inetd /etc/inetd.conf
STC      PMAPOE1  0022         34          1 1FI   08.03.28      1.657
  LATCHWAITPID=         0 CMD=OPORTMAP
STC      PORTMAP  005B         35          1 1FI   08.03.28      1.832
  LATCHWAITPID=         0 CMD=PORTMAP
STC      RXPROC   005C         36          1 1FI   08.03.28      1.813
  LATCHWAITPID=         0 CMD=RSHD
FTPDOE   FTPDOE1  001E         39          1 1FI   08.03.29      1.684
  LATCHWAITPID=         0 CMD=FTPD
```

Figure 14-5 Command to display information about the active processes

Display z/OS UNIX address spaces

To display information about all users of z/OS UNIX, use the following command:

```
D OMVS,ASID=ALL
```

This will show all z/OS UNIX processes with MVS job name and address space ID (ASID). Information about a specific address space can be displayed by:

```
D OMVS,ASID=31
```

The first line of the display shows the status of z/OS UNIX (Active) and the name of the BPXPRMxx PARMLIB member in use (BPXPRM00). The following status can be displayed:

ACTIVE OMVS is currently active.

NOT STARTED OMVS was not started.

INITIALIZING OMVS is initializing.

TERMINATING OMVS is terminating.

TERMINATED OMVS has terminated.

ETC/INIT WAIT OMVS is waiting for the /etc/init or /usr/sbin/init program to complete initialization.

The remaining information shown about each z/OS UNIX user is:

USER The user ID of the process.

JOBNAME The job name of the process.

ASID The address space ID for the process address space.

PID The process ID, in decimal, of the process.

PPID The parent process ID, in decimal, of the process.

STATE The state of the process or of the most recently created thread in the process. The codes on the figure have the following meanings:

- **1** Process state is for a single thread process.
- **F** File system kernel wait.
- **I** Swapped out.
- **K** Other kernel wait (for example, pause or sigsuspend).
- **M** Process state is for multiple threads and pthread_create was not used to create multiple threads. Process state is obtained from the most recently created thread.
- **R** Running (not kernel wait).

For all other state codes, see *MVS/ESA System Messages, Volume 2* for information about the BPXO001I message.

START The time, in hours, minutes, and seconds, when the process was started.

CT_SECS The total execution time for the process in seconds in the format ssssss.hhh. After approximately 11.5 days, this field will overflow. When an overflow occurs, the field is displayed as ******.***

LATCHWAITPID= Either zero or the latch process ID, in decimal, for which this process is waiting.

CMD= The command that created the process. Truncated to 40 characters and converted to uppercase.

SERVER= The name of the server process.

AF= The number of active server file tokens.

MF= The maximum number of active server file tokens allowed.

TYPE= One of the following:

- **FILE** A network file server
- **LOCK** A network lock server

SERVERLOCKS= The number of active lock processes for this server.

SERVERMAXLOCKS= The maximum number of active lock processes for this server allowed.

14.6 Display process information

```
D OMVS,PID=16777394
BPXO040I 16.32.57 DISPLAY OMVS 753
OMVS     000F ACTIVE          OMVS=(8A)
USER     JOBNAME  ASID        PID       PPID STATE   START     CT_SECS
NICHOLS  NICHOLS  01F7   16777394          1 1RI   14.17.11      1.443
  LATCHWAITPID=         0 CMD=EXEC
 THREAD_ID        TCB@     PRI_JOB  USERNAME   ACC_TIME SC  STATE
 0BA6C1A000000000 007F3338                        .022 NOP  Y
 0BA6D96000000001 007E3460                        .138 LST  Y
```

Figure 14-6 Command to display a specific process

Display a specific process

The operator can display the status of a particular OMVS process and its threads with command:

```
D OMVS,PID=xxxxxx
```

This will show information about each thread of a multi-threaded application. The thread ID would be needed to cancel an individual thread.

The first line of the display shows the status of z/OS UNIX (Active) and the name of the BPXPRMxx PARMLIB member in use (BPXPRM00). The following information about the threads is displayed:

THREAD_ID The thread ID, in hexadecimal, of the thread.

TCB@ The address of the TCB that represents the thread.

PRI_JOB The job name from the current primary address space if different from the home address space, otherwise blank. This is only accurate if the thread is in a wait, otherwise it is from the last time that the status was saved.

USERNAME The username of the thread if a task level security environment created by pthread_security_np exists, otherwise blank.

ACC_TIME The accumulated TCB time in seconds in the format ssssss.hh. When this value exceeds 11.5 days of execution time this field will overflow. When an overflow occurs, the field is displayed as ******.***.

SC The current or last syscall request.

STATE The state of the thread as follows.

- **A** Message queue receive wait
- **B** Message queue send wait
- **C** Communication system kernel wait
- **D** Semaphore operation wait
- **E** Quiesce frozen
- **F** File system kernel wait
- **G** MVS Pause wait
- **K** Other kernel wait (for example, pause or sigsuspend)
- **J** The thread was pthread created rather than dubbed
- **N** The thread is medium weight
- **O** The thread is asynchronous and medium weight
- **P** Ptrace kernel wait
- **Q** Quiesce termination wait
- **R** Running (not kernel wait)
- **S** Sleeping
- **U** Initial process thread (heavy weight and synchronous)
- **V** Thread is detached
- **W** Waiting for child (wait or waitpid callable service)
- **X** Creating new process (fork callable service is running)
- **Y** Thread is in an MVS wait

14.7 Display the kernel address space

```
D A,OMVS
IEE115I 10.21.16 2000.157 ACTIVITY 973
 JOBS     M/S    TS USERS    SYSAS    INITS   ACTIVE/MAX VTAM     OAS
00011    00034    00001      00030    00038    00001/00050       00023
 OMVS     OMVS     OMVS     NSW  *   A=000F   PER=NO   SMC=000
                                      PGN=N/A  DMN=N/A  AFF=NONE
                                      CT=023.135S  ET=00122.24
                                      WKL=SYSTEM   SCL=SYSSTC   P=1
                                      RGP=N/A      SRVR=NO  QSC=NO
                                      ADDR SPACE ASTE=3132C3C0
                                      DSPNAME=BPXDS003 ASTE=5F324C00
                                      DSPNAME=SYSIGWB1 ASTE=4D6EE380
                                      DSPNAME=SYSZBPXC ASTE=52BAD300
                                      DSPNAME=SYSZBPXU ASTE=52BAD280
                                      DSPNAME=HFSDSP04 ASTE=03A31500
                                      DSPNAME=HFSDSP03 ASTE=03A31C00
                                      DSPNAME=HFSDSP02 ASTE=03A31E00
                                      DSPNAME=HFSDSP01 ASTE=03A31C80
                                      DSPNAME=HFSDSPS1 ASTE=03A31B80
                                      DSPNAME=BPXDA002 ASTE=03A31E80
                                      DSPNAME=SYSZBPX3 ASTE=03A31D80
                                      DSPNAME=BPXFSCDS ASTE=04099A00
                                      DSPNAME=BPXDX001 ASTE=03ED2780
                                      DSPNAME=SYSZBPX2 ASTE=04099980
                                      DSPNAME=BPXSMBIT ASTE=03A31480
                                      DSPNAME=SYSZBPXX ASTE=03ED2400
                                      DSPNAME=SYSZBPX1 ASTE=03ED2380
                                      DSPNAME=BPXLATCH ASTE=04099400
```

Figure 14-7 Command to display the OMVS (kernel) address space

Display the kernel address space

The operator can display the kernel address space and its associated data spaces with the command:

```
D A,OMVS
```

Kernel dataspaces

The display output shows the kernel address space identifier (ASID) as `A=nnnn`, where `nnnn` is the hexadecimal ASID value.

The display output shows the data space names associated with the kernel address space as `DSPNAME=BPX...` or `DSPNAME=SYS...`. The system uses these data spaces as follows:

BPXSMBITS For shared memory, memory map, and large message queue buffers. BPXSMBITS should be dumped when you dump BPXD data spaces for these components.

BPXDQxxx For message queues (where xxx can be the number 1 through 9)

BPXDSxxx For shared memory

BPXDOxxx For outboard communications server (OCS)

BPXDMxxx For memory map

BPXFSCDS For couple data set (CDS)

SYSZBPX1 For kernel data (including CTRACE buffers)

SYSZBPX2	For file system data
SYSZBPX3	For pipes
SYSIGWB1	For byte-range locking
SYSGFU01	For DFSMS file system
SYSZBPXC	For Converged INET sockets
SYSZBPXL	For local INET sockets
SYSZBPXU	For AF_UNIX sockets

Include dataspaces in a dump

These data spaces will need to be included in a dump if they have information that could be useful in analyzing a problem.

The kernel data space, SYSZBPX1, is always needed. Dump other data spaces if there is reason to believe they contain data that could be useful in analyzing a problem.

14.8 z/OS V1R7 command options

- D OMVS,MF - displays the last 10 (or less) failures
- D OMVS,MF=A - displays up to 50 failures
  - D OMVS,MF=ALL
- D OMVS,MF=P - deletes the failure information log
  - D OMVS,MF=PURGE

Figure 14-8 Commands to display mount failure messages

Mount failure messages

Failures from prior MOUNT or MOVE file system commands, of any form, will be remembered and the pertinent information is available for display or application retrieval.

This includes, for instance, mounts issued from TSO, ISHELL, those from BPXPRMxx during system startup, and sysplex mounts.

Also, file system new owner failures (**`Move`** commands) that occur during **`setomvs`**, **`chmount`**, shutdown, and member gone event processing will be saved.

To check move and mount file system failures use the following display commands:

- D OMVS,MF displays the last 10 (or less).
- D OMVS,MF=A or D OMVS,MF=ALL displays up to 50 failures.
- D OMVS,MF=P or D OMVS,MF=PURGE deletes the failure information log.

14.9 Mount error messages displayed

```
D OMVS,MF
 BPXO058I 16.05.10 DISPLAY OMVS 356
 OMVS     0011 ACTIVE          OMVS=(7B)
 SHORT LIST OF FAILURES:
 TIME=16.02.09  DATE=2005/08/16        MOUNT RC=007A  RSN=052C018F
   NAME=HAIMO.ZFS
   TYPE=ZFS
   PATH=/SC70/tmp/bpxwh2z.HAIMO.16:01:56.zfs
   PARM=AGGRGROW
 TIME=10.57.24  DATE=2005/08/15        MOUNT RC=0079  RSN=5B220125
   NAME=OMVS.TCPIPMVS.HFS
   TYPE=HFS
   PATH=/u/tcpipmvs
```

Figure 14-9 Mount failure messages displayed

Displaying failure messages for mounts

The D OMVS,MF command displays information about move or mount failures:

- Enter MF to display information about the last 10 or less move or mount failures.
- Enter MF=ALL or MF=A to display information about the last 50 or less move or mount failures.

The system issues message BPXO058I to display the information about mount failures.

14.10 Mount failure messages

```
D OMVS,MF=A
BPXO058I 13.26.37 DISPLAY OMVS 974
OMVS     0011 ACTIVE          OMVS=(TT)
LAST PURGE: TIME=08.49.43  DATE=2005/05/11
ENTIRE LIST OF FAILURES:
TIME=09.36.41  DATE=2005/05/13        MOUNT RC=0081  RSN=EF096055
  NAME=HFS.TEST.MOUNT4
  TYPE=ZFS
  PATH=/TEST/TSTMOUNTZ
  PLIB=BPXPRMTT
(...up to 50 failures)
D OMVS,MF=P
BPXO058I 13.30.25 DISPLAY OMVS 010
OMVS     0011 ACTIVE          OMVS=(TT)
PURGE COMPLETE: TIME=13.30.25  DATE=2005/05/13
D OMVS,MF
BPXO058I 13.42.45 DISPLAY OMVS 041
OMVS     0011 ACTIVE          OMVS=(TT)
LAST PURGE: TIME=13.30.25  DATE=2005/05/13
NO MOUNT OR MOVE FAILURES TO DISPLAY
```

Figure 14-10 Mount failure messages

Mount failure messages

Using D OMVS,MF=ALL or D OMVS,MF=A, displays information about the last 50 or less move or mount failures.

The system issues message BPXO058I to display the information about mount failures.

Using D OMVS,MF=PURGE or D OMVS,MF=P allows you to purge the saved information mount failures displayed in message BPXO058I.

Once you have purged the messages and then issued a D OMVS,MF, there are no messages to display.

14.11 Stopping BPXAS address spaces

```
F BPXOINIT,SHUTDOWN=FORKINIT
BPXM036I BPXAS INITIATORS SHUTDOWN.
$HASP395 BPXAS    ENDED
```

Attempting to shut down active BPXAS address spaces will result in the following message:

BPXM037I BPXAS INITIATOR SHUTDOWN DELAYED

Figure 14-11 Command to shut down all BPXAS address spaces

Shut down all BPXAS address spaces

When closing down a system prior to IPL, the F BPXOINIT,SHUTDOWN=FORKINIT command should be issued prior to stopping JES2. This is because the WLM BPXAS address spaces remain active for 30 minutes after use, and could therefore delay JES2 shutdown for an unacceptable period. The F BPXOINIT,SHUTDOWN=FORKINIT command terminates inactive BPXAS address spaces prematurely.

Attempting to shut down active BPXAS address spaces will result in the following message:

```
BPXM037I BPXAS INITIATOR SHUTDOWN DELAYED
```

14.12 LFS soft shutdown

- F BPXOINIT,SHUTDOWN=FILESYS
  - Unmounts file systems prior to re-IPL
  - Cached buffers synched to disk
- Non-Sysplex:
  - Unmount Force all file systems
- Sysplex:
  - Unmount automounted file systems & file systems mounted on them
  - Move Automove(Yes) file systems to other systems
  - Unmount Force remaining file systems

Figure 14-12 A command to unmount all file systems

LFS soft shutdown

A function to do a soft shutdown for mounted file systems is provided through the MVS operator console MODIFY command, as follows:

```
F BPXOINIT,SHUTDOWN=FILESYS
```

This command terminates UNIX activity on the system. The command also provides a method for unmounting all file systems and hardening the cached buffers to disk.

Other UNIX applications

Many of the different products running under z/OS UNIX System Services, like NFS for z/OS and Communication Server, have their own shutdown procedures. These procedures should be used first to shut down the different products, before any z/OS UNIX System Services termination commands are used.

The distinction between the shared file system environment, file systems in a sysplex, and non-sysplex file systems is based on whether SYSPLEX(YES) is specified in the BPXPRMxx PARMLIB member, and the systems are participating in a shared file system environment.

If it is a non-sysplex environment, all file systems will be unmounted with the FORCE operand. In a sysplex environment, all automounted file systems and file systems mounted on the system you shut down will be unmounted. For file systems mounted with AUTOMOVE parameter set, the ownership is moved to another system in the sysplex. The remaining file systems will be unmounted FORCE.

14.13 z/OS V1R8 file system shutdown

- Automounted file systems are now not mounted during the processing of or after the completion of
  - F BPXOINIT,SHUTDOWN=FILEOWNER
  - Unanticipated automount mounts can be prevented
- File systems can only be mounted on this system by an explicit mount command
- Filtering capability has been added to D OMVS,F that can limit the output display of file systems
- Pertinent filesystem information is displayed when using:
  - D OMVS,F with new operands
    - NAME - OWNER - EXCEPTION - TYPE

Figure 14-13 File system shutdown with z/OS V1R8

Preventing mounts during file system shutdown

Prior to z/OS V1R8, as part of a planned shutdown, you needed to prepare the file systems before shutting down z/OS UNIX by issuing one of these commands:

```
F BPXOINIT,SHUTDOWN=FILEOWNER
F BPXOINIT,SHUTDOWN=FILESYS
```

This synchronizes data to the file systems and possibly unmounts or moves ownership of the file systems. If you use SHUTDOWN=FILEOWNER, the system is disabled as a future file system owner via move or recovery operations until z/OS UNIX has been restarted.

In the previous versions of z/OS UNIX, mounts are permitted to occur during shutdown processing and after file system shutdown has completed. One problem that exists today is that there are file systems still owned by the system that is being shut down.

Functional changes in z/OS V1R8

z/OS V1R8 provides the following solutions for this problem:

- Automounted file systems will not be mounted during or after the file system shutdown processing. Using the following command, that limits additional file systems from being mounted on the system where the command is executed, and changes to the output of the command summarize the number of file systems that are still owned by the system:

  ```
  F BPXOINIT,SHUTDOWN=fileowner
  BPXM048I  BPXOINIT FILESYSTEM SHUTDOWN INCOMPLETE.
  ```

```
3 FILESYSTEM(S) ARE STILL OWNED BY THIS SYSTEM.
2 FILESYSTEM(S) WERE MOUNTED DURING THE SHUTDOWN PROCESS.
```

After the SHUTDOWN=fileowner has been issued, automount-managed file systems will no longer be mounted. File systems can only be mounted on this system by an explicit mount command.

The file system display is easier to use by providing only the specific information that is needed to prepare for a file system shutdown.

- A filtering capability has been added to provide a limit to the message responses of the following command:

```
D OMVS,F
```

New filtering capability added to shutdown process

There are no messages issued when using the F BPXOINIT,SHUTDOWN=fileowner command if an automount was attempted. However, an appropriate error return and reason code will be issued for the mount request. Explicit mounts are still accepted during and after the fileowner shutdown processing.

Behavior for the F BPXOINIT,SHUTDOWN=filesys command is not affected by this change. Automount-managed file systems can be mounted during and after the shutdown process. However, the resulting message, BPXM048I, uses the new format with the additional line that summarizes new mounts that occurred during the shutdown process.

When either shutdown commands are accepted, a timestamp is taken. If the system that is being shut down becomes the owner of a newly mounted file system or one that was acquired due to a move operation (only for SHUTDOWN=filesys) before the command completes, a console message is issued that includes the number of file systems that were acquired during this time frame.

14.14 Options with the D OMVS,F command

- D OMVS,F,name=nn or D OMVS,F,n=nn
  - (where nn is the name of a file system) displays information about the specified file system(s) - One wildcard (*) is permitted in the name and displays all file systems whose names match that template
- D OMVS,F,name=* or D OMVS,F,n=*
  - Displays all file systems (same as 'D OMVS,F')
- D OMVS,F,owner=xx or D OMVS,F,o=xx
  - (where xx is a system name) displays all file systems owned by system xx

Figure 14-14 Options on the D OMVS,F command

Command change benefits

You can benefit from these changes to the operator commands as follows:

- Preventing unanticipated automount-managed mounts.
- Displaying pertinent file system information using the D OMVS,F command and providing only the specific information to prepare for a file system shutdown, as shown in Figure 14-14.
- Limiting additional file systems from being mounted on the system where the command was executed.
- Filtering the output display to a particular file system.

After the SHUTDOWN=fileowner has been issued, file systems can only be mounted on this system by an explicit **`mount`** command.

NAME or N=filesystem Displays information about the specified file system or file systems. You can use one wildcard character (*) in the file system specified. For example, ZOS18.*.HFS or ZOS.L*.HFS. Specifying D OMVS,F,NAME=* results in the system displaying all file systems, which is the same output as if you specified D OMVS,F.

OWNER or O=systemname Displays information for the file systems owned by the specified system name. Specifying D OMVS,F,OWNER displays all the file systems that are owned by this system.

14.15 Options with the D OMVS,F command

- D OMVS,F,owner or D OMVS,F,o
  - Displays all file sytems that are owned by the system where the command was executed
- D OMVS,F,exception or D OMVS,F,e
  - Displays all file systems in an exception state (e.g. quiesced, unowned, in recovery)
- D OMVS,F,type=xx or D OMVS,F,t=xx
  - (where xx is a PFS type) displays all file systems of the type xx

Figure 14-15 Options on the D OMVS,F command

Command change benefits

These new options display a list of file systems that z/OS UNIX System Services is currently using, including the following:

- The status of each file system
- The date and time that the file system was mounted
- The latch number for the file system
- The quiesce latch number for the file system, or 0 if the file system has never been quiesced by z/OS UNIX System Services

OWNER or O=systemname	Displays information for the file systems owned by the specified system name. Specifying D OMVS,F,OWNER displays all the file systems that are owned by this system.
EXCEPTION or E	Displays file systems in an exception state, such as a file system that is quiesced, unowned, or in recovery.
TYPE or T=type	Displays all file systems of the specified PFS type.

14.16 New command examples

```
D OMVS,F,NAME=BECKER.ZFS
BPXO045I 12.51.23 DISPLAY OMVS
OMVS     000E ACTIVE                                    OMVS=(00,FS)
TYPENAME   DEVICE ----------STATUS----------- MODE  MOUNTED    LATCHES
ZFS            14 ACTIVE                      RDWR  05/22/2006  L=30
  NAME=BECKER.ZFS                                   12.35.23    Q=0
  PATH=/u/becker
  AGGREGATE NAME=BECKER.ZFS

D OMVS,F,N=*.ETC
BPXO045I 13.04.42 DISPLAY OMVS 399
OMVS     000E ACTIVE                                    OMVS=(00,FS)
TYPENAME   DEVICE ----------STATUS----------- MODE  MOUNTED    LATCHES
HFS             2 ACTIVE                      RDWR  05/19/2006  L=14
  NAME=OMVS.SC75.ETC                                11.43.25    Q=0
  PATH=/etc
```

Figure 14-16 D OMVS,F,N examples

Using the file system name example

Figure 14-16 shows examples of using the file system name as a filter to display mounted file systems. The use of wildcards is permitted with the file system name.

14.17 New command examples

```
D OMVS,F,O=SC75
BPXO045I 12.57.04 DISPLAY OMVS 388
OMVS     000E ACTIVE                                    OMVS=(00,FS)
TYPENAME   DEVICE ----------STATUS----------- MODE  MOUNTED    LATCHES
AUTOMNT         6 ACTIVE                      RDWR  05/19/2006  L=20
  NAME=*AMD/u                                       11.43.27    Q=0
  PATH=/u
TFS             5 ACTIVE                      RDWR  05/19/2006  L=17
  NAME=/TMP                                         11.43.25    Q=0
  PATH=/SYSTEM/tmp
  MOUNT PARM=-s 100

D OMVS,F,E
BPXO045I 11.22.25 DISPLAY OMVS 832
OMVS     000E ACTIVE                                    OMVS=(00,FS)
TYPENAME   DEVICE ----------STATUS----------- MODE  MOUNTED    LATCHES
ZFS             9 DRAIN UNMOUNT               RDWR  05/15/2006  L=25
  NAME=LUTZ.ZFS                                     10.52.37    Q=0
  PATH=/u/lutz
  AGGREGATE NAME=LUTZ.ZFS
```

Figure 14-17 D OMVS,F with filters for owner and exceptions

Using filters for owner and exceptions

The new commands shown in Figure 14-17 provide filtering syntax for the D OMVS,F command for the file system owner and when file systems have exception conditions.

14.18 New command examples

```
D OMVS,F,T=HFS
BPXO045I 13.01.48 DISPLAY OMVS
OMVS     000E ACTIVE                                      OMVS=(00,FS)
TYPENAME   DEVICE ----------STATUS----------- MODE  MOUNTED    LATCHES
HFS             4 ACTIVE                      RDWR  05/19/2006  L=16
  NAME=OMVS.SC75.VAR                                11.43.25    Q=0
  PATH=/var
D OMVS,F,T=ZFS
BPXO045I 10.16.14 DISPLAY OMVS
OMVS     000E ACTIVE             OMVS=(8B)
TYPENAME   DEVICE ----------STATUS----------- MODE  MOUNTED    LATCHES
ZFS            43 ACTIVE                      RDWR  09/14/2006  L=60
  NAME=HERING.ZFS                                   11.56.13    Q=0
  PATH=/u/hering
  AGGREGATE NAME=HERING.ZFS
  OWNER=SC74     AUTOMOVE=Y CLIENT=N
ZFS            42 ACTIVE                      RDWR  09/13/2006  L=59
  NAME=ROGERS.ZFS                                   13.54.10    Q=0
  PATH=/u/rogers
  AGGREGATE NAME=ROGERS.ZFS
  OWNER=SC75     AUTOMOVE=Y CLIENT=N
```

Figure 14-18 D OMVS,F commands with filter of type

Using the type filter

Figure 14-18 displays the filtering by type of file systems, either HFS or zFS.

14.19 z/OS UNIX shutdown

- OMVS restart
  - Provides shutdown and restart capability of UNIX System Services (USS) environment
  - Like prior P OMVS and S OMVS support with some additional capabilities
- Does not resolve ALL system outage conditions involving UNIX System Services
- Do not use to install maintenance for USS
- More than one command to shut down the system
- Shutdown and restart commands
  - F OMVS,SHUTDOWN
  - F OMVS,RESTART

Figure 14-19 Commands to shut down and restart z/OS UNIX without re-IPL

Shut down z/OS UNIX without IPL

z/OS UNIX provides the possibility to shut down and reinitialize the z/OS UNIX environment without the need to IPL the system. This OMVS option will shut down z/OS UNIX and all the processes that are running under it.

OMVS shutdown allows you to do some reconfiguration that would otherwise have required an IPL, for example:

- If you do a reconfiguration of a system to go from a non-shared file system environment system to a shared file system environment system
- When you implement a new file structure

Restrictions and limitations

There are some restrictions and limitations that are not allowed with the OMVS shutdown. In these cases, you will need to IPL the system as usual. These limitations include:

- During the cleanup of the resources for z/OS UNIX as part of the shutdown process, some internal failures cannot be resolved using this support due to their severity.
- OMVS shutdown support cannot be used to install maintenance against z/OS UNIX because some of the modules are maintained across the shutdown and restart process.
- Installations should avoid using the OMVS restart support as a way to shutdown the system with one single command. This will cause some unexpected abnormal

terminations of address spaces using z/OS UNIX System Services that are not shut down in the manner they recommend.

- OMVS restart support is not intended to be used in an unlimited manner to shut down and restart, because some system resources can be lost during the shutdown phase and due to the disruption it causes to the system.

New operator commands

In order to support OMVS shutdowns, there is a new **Modify** command; support for the OMVS address space has been introduced to provide the ability to shut down and then restart the z/OS UNIX environment with the following commands:

```
F OMVS,SHUTDOWN
F OMVS,RESTART
```

14.20 Recommended shutdown procedures

- Quiesce batch and interactive workloads
- Quiesce major subsystem and application workloads using UNIX System Services
  - DB2, CICS and IMS, and applications such as SAP, Lotus Domino, NetView, and Websphere
- Shut down PFS address spaces (NFS and zFS)

Figure 14-20 Recommended procedures to shut down OMVS

Shutdown procedures

Quiesce your batch and TSO workloads. Having batch jobs and TSO users running during the shutdown may cause these jobs to experience unexpected signals or abends. Additionally, these jobs and users may end up being hung, waiting for z/OS UNIX services to be restarted, if they first access z/OS UNIX services during a shutdown.

Quiesce those application and subsystem workloads using z/OS UNIX services in the manner that each application or subsystem recommends. Doing so will allow subsystems such as DB2, CICS, and IMS™, and applications like SAP, Lotus Domino, NetView, and WebSphere to be quiesced in a more controlled manner than this facility will provide.

Unmount all remotely mounted file systems such as those managed by NFS. Doing so will prevent these file systems from losing data.

Terminate all file system address spaces, such as TCP/IP and DFSS, using their recommended shutdown methods. If you do not shut them down before issuing F OMVS,SHUTDOWN, these system functions may terminate abnormally when the shutdown takes place. Existing colony PFS address spaces will be shut down. This may include NFS, for example.

Note: You can use the D OMVS,A=ALL operator command to determine which applications, if any, require quiescing.

14.21 Application registration

- Shutdown Registration support to request special treatment at shutdown time
  - Applications, jobs, processes, subsystems register as:
    - Permanent
    - Blocking
- SIGDANGER signal to warn of imminent shutdown
  - Signal sent to the registered
  - Who is registered - D OMVS,A=ALL
    - Enhanced to indicate shutdown/restart status

Figure 14-21 Processes register for shutdown processing

Applications register for shutdown

New registration support has been introduced to allow an application to request special treatment when a shutdown is initiated, and to request to receive a new SIGDANGER signal as a warning that shutdown has been initiated and is imminent.

The registration support allows requesting of special treatment in case of shutdown. Different kinds of registrations can be implemented, as follows:

- A process or job registered as “permanent” is not taken down across the shutdown and restart process. Its process-related resources are checkpointed at shutdown time and reestablished at restart time, so the registered permanent process or job can survive the shutdown.
- A process or job registered as “blocking” delays shutdown until it de-registers or ends. This gives the ability for an application to quiesce itself in a more controlled manner before UNIX System Service starts taking down all processes.
- A process or job registered for “notification” is notified that the shutdown process is being planned via a SIGDANGER signal.

The following command has been modified to include information about what type of registration a specific process has:

```
D OMVS,A=ALL
```

14.22 Display application registration

```
D OMVS,A=ALL
BPXO040I 10.02.18 DISPLAY OMVS 543
OMVS     000F ACTIVE          OMVS=(3A)
USER     JOBNAME  ASID        PID       PPID STATE   START     CT_SECS
OMVSKERN BPXOINIT 003C          1          0 MRI--- 07.58.59       .18
  LATCHWAITPID=         0 CMD=BPXPINPR
  SERVER=Init Process                     AF=    0 MF=00000 TYPE=FILE
STC      MVSNFSC5 003B   16908290          1 1R---- 07.59.13       .06
  LATCHWAITPID=         0 CMD=GFSCMAIN
STC      MVSNFSC5 003B   50462724          1 1R---- 07.59.12       .06
  LATCHWAITPID=         0 CMD=BPXVCLNY
STC      MVSNFSC5 003B   50462728          1 1A---- 07.59.14       .06
  LATCHWAITPID=         0 CMD=BPXVCMT
OMVSKERN SYSLOGD5 0041     131081          1 1FI--- 07.59.06       .13
  LATCHWAITPID=         0 CMD=/usr/sbin/syslogd -f /etc/syslog.conf
STC      RMFGAT   0046   84017164          1 1R---P 08.00.01     83.91
  LATCHWAITPID=         0 CMD=ERB3GMFC
TCPIPMVS TCPIPMVS 0043     131085          1 MR---B 08.00.06      8.35
  LATCHWAITPID=         0 CMD=EZBTCPIP
TCPIPMVS TCPIPMVS 0043     131086          1 1R---B 08.00.12      8.35
  LATCHWAITPID=         0 CMD=EZBTTSSL
TCPIPMVS TCPIPMVS 0043     131087          1 1R---B 08.00.12      8.35
```

Figure 14-22 Command to display registered processes

Display of registered processes for shutdown

Issue the normal command to display active BPXAS address spaces.

As shown in the figure, a character `P` or `B`, indicating permanent or blocked, has been included on the STATE field.

14.23 F OMVS,SHUTDOWN

- Initiates shutdown of the UNIX System Services environment
  - *BPXI055I OMVS SHUTDOWN REQUEST ACCEPTED
- SIGDANGER signal sent to registered parties to warn of imminent shutdown
- Shutdown delayed if any blocking processes exist

```
*BPXI064E OMVS SHUTDOWN REQUEST DELAYED
BPXI060I TCPIPMVS RUNNING IN ADDRESS SPACE 0043 IS BLOCKING SHUTDOWN OF OMVS
BPXI060I TCPIPOE RUNNING IN ADDRESS SPACE 0044 IS BLOCKING SHUTDOWN OF OMVS
BPXI060I TCPIPB RUNNING IN ADDRESS SPACE 0052 IS BLOCKING SHUTDOWN OF OMVS
```

Figure 14-23 Issuing the shutdown command

Shutdown of OMVS

The shutdown starts by issuing the F OMVS,SHUTDOWN command and then the following steps are done:

- Once the shutdown command has been accepted, a `BPXI055I` is issued:
  `*BPXI055I OMVS SHUTDOWN REQUEST ACCEPTED`
- SIGDANGER signals are sent to all processes registered for receiving the SIGDANGER signal.

If any blocking processes are found, shutdown is delayed until these processes end or deregister as blocking, or if a F OMVS,RESTART command is issued to restart. If these blocking processes do not end or deregister in a reasonable amount of time, message `BPXI064E` is displayed to the console, indicating shutdown is delayed. In our tests, 12 seconds after the shutdown command was accepted, the `BPXI064E` message was issued. Message `BPXI060I` was also issued for each process found to be holding up the shutdown. This message identified the job and address space involved.

Attention: Use the F OMVS,SHUTDOWN command carefully because this method will take down other system address spaces. As a result, some system-wide resources may not be completely cleaned up during a shutdown and restart. Do not use this command to shut down and restart the z/OS UNIX environment on a frequent basis. (If you do so, you will eventually have to do a re-IPL.)

14.24 Blocking processes completion

- Once blocking processes have ended or deregistered
  - SIGTERM signal sent

```
BPXP010I THREAD 10652BA800000000, IN PROCESS 67239946, WAS 684
TERMINATED BY SIGNAL SIGTERM, SENT FROM THREAD
1065383000000000, IN PROCESS 1, UID 0.
BPXP018I THREAD 1067C3D000000000, IN PROCESS 131109, ENDED 685
WITHOUT BEING UNDUBBED WITH COMPLETION CODE 04EC6000,
AND REASON CODE 0000FF0F.
BPXP018I THREAD 1067AAC000000000, IN PROCESS 131107, ENDED 686
```

Figure 14-24 Messages when blocking processes complete

Applications that block shutdown

Some applications may register to block a shutdown, which delays the shutdown request until the blockers end or unblock. Also, an application exit can be set up to be given control when a shutdown request is initiated in order to allow specific shutdown actions to be taken. This may include initiating the shutdown of the application or sending messages that indicate the specific steps that are required to shut down the application.

If any blocking jobs or processes are active when a shutdown request is initiated, the shutdown is delayed until all blocking jobs or processes either unblock or end. If the delay exceeds a certain time interval, you will receive messages telling you that the shutdown is delayed and which jobs are delaying the shutdown. At this point, you can either attempt to terminate the jobs that are identified as blocking shutdown or issue F OMVS,RESTART to restart the z/OS UNIX environment, which will cause the shutdown request to be terminated.

Once all blocking processes have ended or deregistered as blocking, the shutdown follows by sending a SIGTERM signal to each non-permanent process found and the following messages are received, as shown in the figure.

14.25 Shutdown processing completion

- For processes that do not terminate after SIGTERM
  - Send process a SIGKILL signal

```
BPXP010I THREAD 106C2BA000000002, IN PROCESS 131198, WAS 789
TERMINATED BY SIGNAL SIGKILL, SENT FROM THREAD
1065383000000000, IN PROCESS 1, UID 0.
BPXP010I THREAD 1066EEC800000000, IN PROCESS 84017176, WAS 792
TERMINATED BY SIGNAL SIGKILL, SENT FROM THREAD
1065383000000000, IN PROCESS 1, UID 0.
```

- For processes that still exist
  - Process terminated with a 422-1A3 ABEND

```
IEF450I STEVEZ IKJACCT IKJACCNT - ABEND=S422 U0000 REASON=000001A3 952
        TIME=07.58.28
BPXI061E OMVS SHUTDOWN REQUEST ABORTED
```

Figure 14-25 Steps to terminate active processes

Shutdown processing complete

The best way to end a process is to issue the **kill** command. Use the D OMVS operator command or the **ps** command to display all the active processes. Then issue the **kill** command, specifying the signal and the PID (process identifier) for the process.

- Start by sending a SIGTERM signal:
  `kill -s TERM pid`

  where `pid` is the process identifier.
- If that does not work, try sending a SIGKILL signal:
  `kill -s KILL pid`

  where pid is the process identifier.

If some of the processes still exist after both of these signals are sent, they are terminated with a 422-1A3 ABEND.

```
IEF450I STEVEZ IKJACCT IKJACCNT - ABEND=S422 U0000 REASON=000001A3 952
        TIME=07.58.28
```

If after all of these steps, some non-permanent processes still exist, the shutdown request is aborted and a `BPXI061E` message is issued:

```
BPXI061E OMVS SHUTDOWN REQUEST ABORTED
```

14.26 Shutdown for permanent processes

- Shutdown processing checkpoints processes
  - Processing aborted if processes using:
    - Shared libraries
    - Memory mapped file services
    - Map services
    - SRB services
    - Semaphore services
    - Message queue services
    - Shared memory services
  - BPXI060I message

Figure 14-26 Permanent processes termination

Shutdown of permanent registered processes

After non-permanent processes have been taken down, the shutdown process continues trying to checkpoint all the permanent processes. A permanent process cannot be checkpointed, however, if it is using any of the following resources:

- Shared libraries
- Memory mapped file services
- Map services
- SRB services
- Semaphore services
- Message queue services
- Shared memory services

A permanent process found using any of these resources will cause shutdown to be aborted and message `BPXI060I` is issued indicating what resource for which job is causing the problem.

14.27 Shutdown processing final cleanup

- Take down BPXOINIT after all non-permanent processes have ended
- Unmount and move all file systems - equivalent to:
  - F BPXOINIT,SHUTDOWN=FILESYS
- Clean up most kernel and LFS resources

```
BPXN001I UNIX SYSTEM SERVICES PARTITION CLEANUP IN PROGRESS FOR SYSTEM SC64
```

- Issue message BPXI056E when shutdown completes and wait for restart

```
*BPXI056E OMVS SHUTDOWN REQUEST HAS COMPLETED SUCCESSFULLY
```

Figure 14-27 Final cleanup of shutdown processing

Cleanup of shutdown processing

After all non-permanent processes have ended, BPXOINIT is taken down with a 422-1A3 abend.

File systems on the system where the shutdown was issued are immediately unmounted; data is synched to disk as a result.

For a shared file system environment, one of the following actions is taken on the file systems that are owned by the system where the command was issued:

- Unmount if automounted or if a file system was mounted on an automounted file system.
- Move to another system if an AUTOMOVE(YES) was specified.
- Unmount for all other file systems.

File systems that are not owned by the system on which the shutdown was issued are not affected. The shutdown should be done prior to an IPL. It replaces BPXSTOP.

On the system that you are preparing to shut down, issue the following command:

```
F BPXOINIT,SHUTDOWN=FILESYS
```

All file systems are unmounted and potentially moved to another system. If for some reason it is not possible to unmount some file systems, a `BPXI066E` message is issued and shutdown will proceed to the next phase.

14.28 F OMVS,RESTART

- Restart UNIX System Services environment
- Can optionally specify OMVS=(xx,yy,...) parameter to change PARMLIB
- Redrive normal kernel and LFS initialization
- Start up BPXOINIT address space
  - Reestablish checkpointed processes, if possible
  - Start up /etc/init or /usr/sbin/init to begin full function initialization
  - Reissue BPXI004I message when restart is complete
  - DOM all prior shutdown messages

    BPXI004I OMVS INITIALIZATION COMPLETE

Figure 14-28 Command to restart z/OS UNIX

Restart z/OS UNIX command

The F OMVS,RESTART command restarts the z/OS UNIX environment. This involves the following:

1. Once the restart command has been accepted, indicated by the following message:

   `*BPXI058I OMVS RESTART REQUEST ACCEPTED`

   the first step in the restart process is to re-initialize the kernel and LFS. This includes starting up all physical file systems.
2. BPXOINIT is restarted and it will re-establish itself as process ID 1.
3. BPXOINIT re-establishes the checkpointed processes as follows:

   All checkpointed processes that are still active are re-established and those that are not found are not re-established and will have their checkpointed resources cleaned up.
4. After BPXOINIT completes its initialization, it will restart /etc/init or /usr/sbin/init to begin full function initialization of the z/OS UNIX environment. /etc/init performs its normal startup processing, invoking /etc/rc.
5. After /etc/init has completed full function initialization, a `BPXI004I` message is issued indicating z/OS UNIX initialization is complete.

   `BPXI004I OMVS INITIALIZATION COMPLETE`

14.29 Alternate sysplex root

- System root being a single point of failure in a sysplex
  - If sysplex root becomes corrupted or unavailable and unrecoverable, can become a multisystem outage situation and require a sysplex-wide IPL
- New support provides capability to dynamically replace a failed or failing sysplex root automatically or manually using an external command
- Using alternate sysplex root support, you can:
  - Replace the failed sysplex root automatically or
  - Replace the failed or failing sysplex root manually

Figure 14-29 Alternate sysplex root

Alternate sysplex root

The system root can be a single point of failure in a sysplex because of the lack of an alternate or backup. When the sysplex root gets corrupted and becomes unavailable and unrecoverable, it could lead to a multisystem outage situation and require a sysplex-wide IPL.

In a sysplex configuration, the alternate sysplex root file system becomes a standby for the sysplex root file system. The alternate can be used to replace the current sysplex root file system if the sysplex root file system becomes unavailable.

You establish the alternate sysplex root file system by using the ALTROOT statement in the BPXPRMxx parmlib member during OMVS initialization, or by using the **SET OMVS** command.

Using alternate sysplex root support, you can dynamically replace a failed or failing sysplex root automatically, or replace it manually with an external command.

This support provides the following advantages:

- It eliminates the single point of failure for the sysplex root file system.
- It allows for a dynamic replacement of a failed or failing sysplex root.
- It prevents a single point sysplex root failure from causing a multisystem outage.

14.30 Defining an alternate root and mount

- ❑ Allocate alternate and format
- ❑ Create mountpoint and Mount
- ❑ Use pax utility to copy (or use copytree utility)

```
#> zfsadm define SYSPLEX.ROOTALT1.ZFS -volume BH5ST1 -cylinders 2 2
IOEZ00248I VSAM linear dataset SYSPLEX.ROOTALT1.ZFS successfully created.
#> zfsadm format SYSPLEX.ROOTALT1.ZFS -compat
IOEZ00077I HFS-compatibility aggregate SYSPLEX.ROOTALT1.ZFS has been
successfully created
#> /usr/sbin/chmount -w /
#> mkdir -m 700 /rootalt
#> /usr/sbin/chmount -r /
#> /usr/sbin/mount -f SYSPLEX.ROOTALT1.ZFS /rootalt
#> copytree / /rootalt
Copying / to /rootalt
From filesystem SYSPLEX.ROOT.ZFS
  To filesystem SYSPLEX.ROOTALT1.ZFS
```

Figure 14-30 Define an alternate sysplex root

Defining an alternate susplex root

To set up an alternate sysplex root file system, complete the following steps:

Allocate the alternate sysplex root file system and set up the mount points and symlinks exactly as in the current sysplex root.

- You can use the **pax** shell command instead of the copytree utility to populate the alternate sysplex root.
- The ALTROOT mount point needs to be created before an alternate sysplex root file system can be established using the BPXPRMxx parmlib member. There is no default mount point.

Important: Make sure that the alternate sysplex root does not reside in the same volume, device, and control unit as the current sysplex root.

Following is an example that shows the copytree utility being used to create the alternate sysplex root and followed a mount.

```
#> zfsadm define PLEX75.SYSPLEX.ROOTALT1.ZFS -volume BH5ST1 -cylinders 2 2
IOEZ00248I VSAM linear dataset PLEX75.SYSPLEX.ROOTALT1.ZFS successfully
created.
#> zfsadm format PLEX75.SYSPLEX.ROOTALT1.ZFS -compat
```

```
IOEZ00077I HFS-compatibility aggregate PLEX75.SYSPLEX.ROOTALT1.ZFS has been
successfully created
#> /usr/sbin/chmount -w /
#> mkdir -m 700 /rootalt
#> /usr/sbin/chmount -r /
#> /usr/sbin/mount -f PLEX75.SYSPLEX.ROOTALT1.ZFS /rootalt
#> copytree / /rootalt
Copying / to /rootalt
From filesystem PLEX75.SYSPLEX.ROOT.ZFS
  To filesystem PLEX75.SYSPLEX.ROOTALT1.ZFS
```

ALTROOT mount status

By completing the above tasks, you establish an alternate sysplex root in the shared file system configuration. The alternate sysplex root is mounted in read-only mode at the specified mount point and is designated as AUTOMOVE. When the alternate sysplex root becomes the current sysplex root, it is mounted in read-only mode and is designated as AUTOMOVE regardless of the current sysplex root settings.

Note: Although the internal structure of the sysplex root is more complex, as it should be, the processing works as desired.

14.31 BPXPRMxx parmlib member - ALTROOT

- Altroot statement (BPXPRMRA member)
  - ALTROOT FILESYSTEM ('SYSPLEX.ROOTALT1.ZFS') MOUNTPOINT('/rootalt')
- The ALTROOT file system will be mounted as read-only and with AUTOMOVE=Yes
- Establish altroot sysplex root with SET command

```
SET OMVS=(RA)
IEE252I MEMBER BPXPRMRA FOUND IN SYS1.PARMLIB
BPXO032I THE SET OMVS COMMAND WAS SUCCESSFUL.
IEF196I IEF285I   SYS1.PARMLIB
IEF196I IEF285I   VOL SER NOS= BH5CAT.
IEF196I IEF285I   CPAC.PARMLIB
IEF196I IEF285I   VOL SER NOS= Z19CAT.
IEF196I IEF285I   SYS1.IBM.PARMLIB
IEF196I IEF285I   VOL SER NOS= Z1BRC1.
BPXF013I FILE SYSTEM SYSPLEX.ROOTALT1.ZFS
WAS SUCCESSFULLY MOUNTED.
```

Figure 14-31 BPXPRMxx parmlib member for the alternate sysplex root

BPXPRMxx parmlib member

Establish the alternate sysplex root using the new ALTROOT statement in a BPXPRMxx parmlib member:

```
ALTROOT FILESYSTEM ('PLEX75.SYSPLEX.ROOTALT1.ZFS') MOUNTPOINT('/rootalt')
```

- Mount the ALTROOT file system as read-only and with AUTOMOVE=YES.
- Establish it during OMVS initialization or with the **SET OMVS** command.

```
SET OMVS=(RA)
IEE252I MEMBER BPXPRMRA FOUND IN SYS1.PARMLIB
BPXO032I THE SET OMVS COMMAND WAS SUCCESSFUL.
IEF196I IEF285I   SYS1.PARMLIB
IEF196I IEF285I   VOL SER NOS= BH5CAT.
IEF196I IEF285I   CPAC.PARMLIB
IEF196I IEF285I   VOL SER NOS= Z19CAT.
IEF196I IEF285I   SYS1.IBM.PARMLIB
IEF196I IEF285I   VOL SER NOS= Z1BRC1.
BPXF013I FILE SYSTEM PLEX75.SYSPLEX.ROOTALT1.ZFS 031
WAS SUCCESSFULLY MOUNTED.
```

Alternate sysplex root considerations

The sample BPXPRMxx in the SYS1.SAMPLIB data set provides a detailed explanation of ALTROOT statements and accepted keywords.

The current sysplex root and the alternate sysplex root file system type do not have to be identical; they can be either zFS or HFS.

Restrictions for replacement

The following constraints must be met before using alternate sysplex root support:

- The system must be in a shared file system configuration.
- All systems in the shared file system environment must be at the minimum system level of z/OS V1R11.
- The ALTROOT mount point must not exceed 64 characters, and it must reside in the root directory of the current sysplex root file system.
- All systems in the shared file system configuration must have direct access to the new file system and must be able to locally mount it.
- The file system type for the alternate sysplex root and the current sysplex root must be either HFS or ZFS.
- The alternate sysplex root PFS must be active on all systems in the shared file system configuration.

14.32 Display information about processes

Superuser shell session:

```
# ps -A
       PID TTY        TIME CMD
         1 ?          0:00
    196610 ?          0:20
     65539 ?          0:01
    262148 ttyp0001   0:00 /bin/sh
   1310725 ?          0:00 /usr/sbin/rlogind
   1114118 ?          0:03 /usr/sbin/rlogind
   1179655 ?          0:00 /usr/sbin/inetd
   1048584 ttyp0000   0:00 /bin/sh
    327689 ttyp0002   0:04 /bin/sh
    458762 ?          0:01
    393227 ttyp0002   0:00 /bin/ps
```

Figure 14-32 Displaying information about active processes

Display active processes from the OMVS shell

To get information about z/OS UNIX processes, the shell command **ps** can also be used. An z/OS UNIX user can use this command to display information about all active processes.

A superuser can display information about all the processes in the system by using the command **ps -A**.

Figure 14-32 shows an example of a superuser issuing the command **ps -A** to get information about all the active z/OS UNIX processes. This is an alternative to using the D OMVS command. If there is a problem with a process, the user or system administrator will first try to stop the process using shell commands. To get the PID of the process, the **ps** command is used.

From the example, you can see that there are three shell sessions because there are three different ttyp000x (1, 2, and 3). There are two process IDs that is using the same pseudo-TTY (ttyp0002). The reason for this is that the **ps** command (/bin/ps) is running in a subshell, but is using the same terminal for output as the shell session (/bin/sh) where the command was issued from. These two PIDs belong to the superuser that issued the **ps -A** command.

14.33 Stop a process

Use MVS modify command:

```
F BPXOINIT,TERM,PID=327689
F BPXOINIT,FORCE,PID=327689
```

Use shell command:

```
==> kill -s kill  327689
```

Use ISHELL:

```
                          Work with Processes

Select one or more processes with an action code.
  A=Attributes...    K=Kill    S=Signal...

  Process_ID  State     TTY        Time  Command
K  419430404  RUN                 101.9  EXEC
```

Use MVS cancel command:

```
D OMVS,U=WELLIE5
BPXO001I 13.17.46 DISPLAY OMVS 178
OMVS     ACTIVE                                          BPXPRM00
USER     JOBNAME  ASID        PID       PPID STATE   START     CT_SECS
WELLIE5  WELLIE5  0045     524298          1 1R    13.17.29      6.441

CANCEL WELLIE5,A=45
```

Figure 14-33 Different commands to stop a process

Commands to stop a z/OS UNIX process

There are four ways to stop a z/OS UNIX process:

- The operator MODIFY command. The `TERM` option will allow a signal interface routine to receive control before termination. The `FORCE` option prevents the signal interface routine from receiving control before the process is terminated.
- The **`kill`** shell command can be used to cancel a process. A user can cancel his/her own processes, or a system administrator (superuser) has authority to kill other user's processes. `TERM` sends a SIGTERM signal and `KILL` sends a SIGKILL signal.

  The **`kill`** command sends a signal to a process. One of the signals that can be sent is the kill signal, and that is the reason why kill is used twice in the command. The system administrator can use the **`ps`** command to find the PID of the process. The PID is needed to identify the process in the **`kill`** command.
- The ISHELL **`k`** line command of the Work with Processes menu. You can get to this from the Tools pull-down menu in the ISHELL.
- The MVS CANCEL command can be used to cancel an address space that contains a z/OS UNIX process. If the address space contains multiple processes, CANCEL will cause all of them to terminate. CANCEL is an operator command.

14.34 Superkill function

- USS processes that become hung and cannot be terminated via the kill() service
  - Requires MVS operator intervention to CANCEL the address space containing the USS process
- Using SUPERKILL
  - Cancel hung USS processes using UNIX semantics
  - Cancel their own hung processes from the shell
  - Enhanced console support to give operators and automated console applications additional flexibility
    - BPX1KIL/BPX4KIL - USS callable assembler service
    - __superkill() - C/C++ service
    - kill –K [pid ...][job-identifier ...] - new shell command
    - F BPXOINIT,SUPERKILL=pid – new console command

Figure 14-34 Superkill function

Superkill command with z/OS V1R6

USS processes that use MVS services can defer USS signal processing. Even though these restrictions are documented, a hung process can cause other problems if it cannot be terminated. Therefore, USS processes that become hung and cannot be terminated via the kill() service require MVS operator intervention to cancel the address space containing the USS process.

Therefore, a new superkill function was created that does the following:

- Cancels hung USS processes using UNIX semantics.
- Cancels their own hung processes from the shell.
- Uses the enhanced console support to give operators and automated console applications additional flexibility.

The advantages of this function are that it allows an override of the current restrictions of signal delivery and it provides the ability to use the PID instead of an ASID.

The `-K` option sends a superkill signal to force the ending of a process or job that did not end as a result of a prior KILL signal. The process is ended with a non-retryable abend. The regular KILL signal must have been sent at least 3 seconds before the superkill signal is sent. The superkill signal cannot be sent to a process group (by using pid of 0 or a negative number) or to all processes (by using a pid of -1).

14.35 Superkill example

- The procedural flow of a superkill would be as follows:
  - Send a regular KILL signal by issuing, kill -s KILL pid
  - Wait 3 seconds
  - Then send a superkill to force termination - kill -K pid

Figure 14-35 Using the superkill function

Superkill command example

Superkill is a non-posix interface and is really being geared to provide an unauthorized interface to cancel-like logic. The APIs are fitted around tradition signal interfaces for usability reasons as well as some functionality reasons. The posix and shell APIs will allow users to invoke the superkill interface as they would the other signal interfaces. Also, certain signal logic is used for the authorization checks and delivery mechanisms.

The restrictions have been put in place to ensure that the asynchronous nature of the abend is limited to processes that are truly hung. Limiting the abend to a single process at a time also avoids abusive use.

14.36 Changing OMVS parameter values

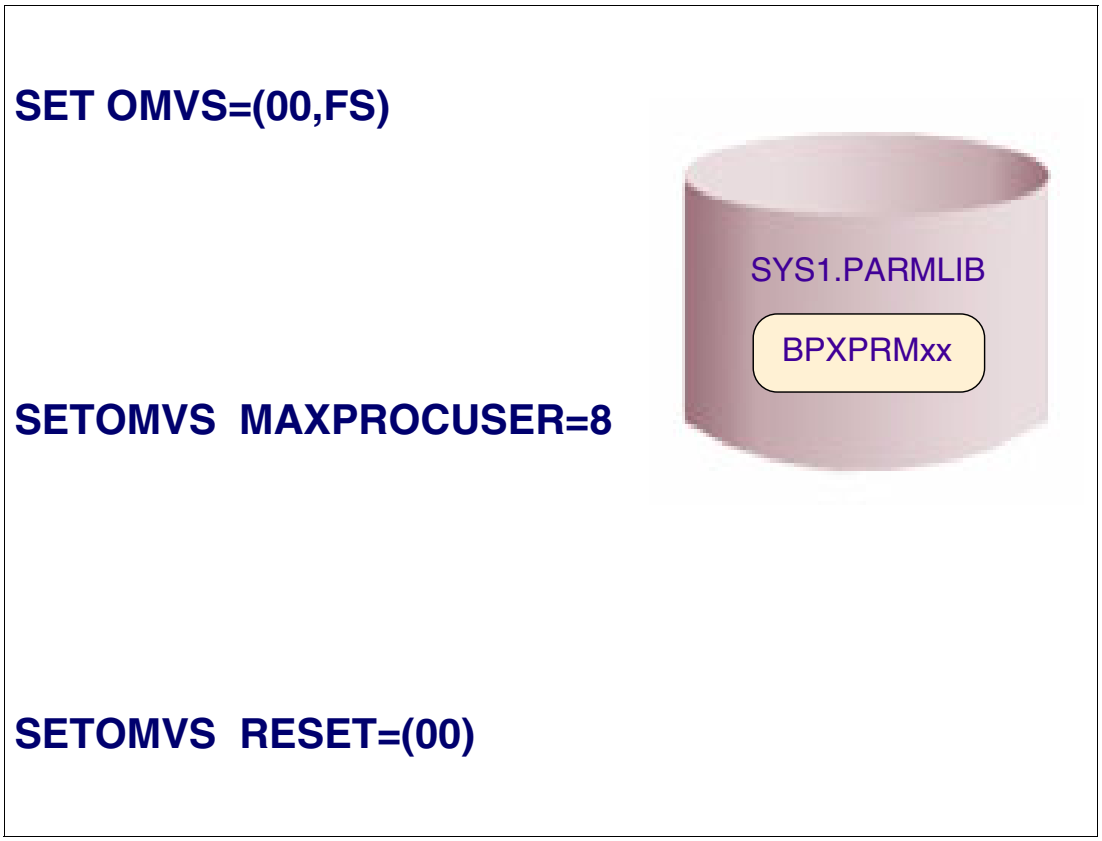


Figure 14-36 Commands to dynamically change BPXPRMxx values

Commands to change BPXPRMxx values

You can change the setting of some of the BPXPRMxx values dynamically using the SETOMVS or SET OMVS commands.

The SET OMVS command lets you dynamically reconfigure the z/OS UNIX system services by specifying one or more BPXPRMxx PARMLIB members to switch to. You can have multiple PARMLIB members stored and use them to establish a new configuration. You can only change the values in the list below.

SETOMVS lets you change specific values independently of others stored in the same PARMLIB member.

Changes to system limits take place immediately (for example, MAXPROCSYS). Changes to user limits (for example, MAXTHREADS) are set when a new user enters the system and they last for the length of the user's connection to z/OS UNIX. Values that can be changed are:

MAXPROCSYS - MAXPROCUSER - MAXFILEPROC - MAXFILESIZE - MAXCPUTIME MAXUIDS - MAXPTYS - MAXRTYS - MAXTHREADTASKS - MAXTHREADS - MAXMMAPAREA - MAXSHAREPAGES - MAXCORESIZE - MAXASSIZE - All IPC values FORKCOPY - STEPLIBLIST - USERIDALIASTABLE - PRIORITYPG - PRIORITYGOAL

14.37 Manage interprocess communication

```
ROGERS @ SC43:/>ipcs -w

IPC status as of Wed Nov 29 15:17:42 EDT 2003
T     KEY       OWNER    GROUP     RCVPID     RCVTYP     SNDPID   SNDLEN
q 0x4107001c OMVSKERN   PRINTQ 1234567890 0x12345678 1234567890  1000000
                                       3 0x00000000          4    10000
                                                             5    90000
                                       4 0xc3c8c1c4
q 0x00003ae8    NANCY     TEST
Shared Memory:
T     KEY       OWNER    GROUP
m 0x0d07021e OMVSKERN   SYSTEM
m 0x0d08c984 OMVSKERN   SYSTEM
Semaphores:
T     KEY       OWNER    GROUP     WTRPID WTRNM  WTROP     AJPID  AJNUM AJVAL
s 0x6208c8ef OMVSKERN   SYSTEM
s 0x00000000 OMVSKERN   SYSTEM                                  2      2     1
s 0x0108c86e OMVSKERN   SYSTEM 1234567890 12345 -12345 1234567890 12345     2
                                       1     3     -1          1     1     1
                                       1     4     -1          2     2     1
s 0x00bc614e     XLIN   VENDOR
s 0x00000058     XLIN   VENDOR
```

ipcrm -Q 0x00003ae8 - remove message

Figure 14-37 Command to display interprocess communication information

Display interprocess communication information

Figure 14-37 shows an example of the output from an IPCS command. The -w option adds information about wait status for message queues and semaphores to the output.

The z/OS UNIX Interprocess Communication (IPC) functions are:

- Shared memory
- Message queues
- Semaphores

Users may invoke applications that create IPC resources and wait for IPC resources. IPC resources are not released when a process terminates or a user logs off. Therefore, it is possible that an IPC user may need assistance to:

- Remove an IPC resource using the shell's **ipcrm** command
- Remove an IPC resource using the shell's **ipcrm** command to release a user from an IPC wait state

The **ipcs** shell command can be used to display the IPC resources in a system, which users own the resources, and which users are waiting for a resource.

You can remove a message queue by using the **ipcrm** command. For example, to remove the message queue with `KEY=0x00003ae8`, which belongs to user `NANCY` (see the figure), use the command:

```
ipcrm -Q 0x00003ae8
```

14.38 System problems

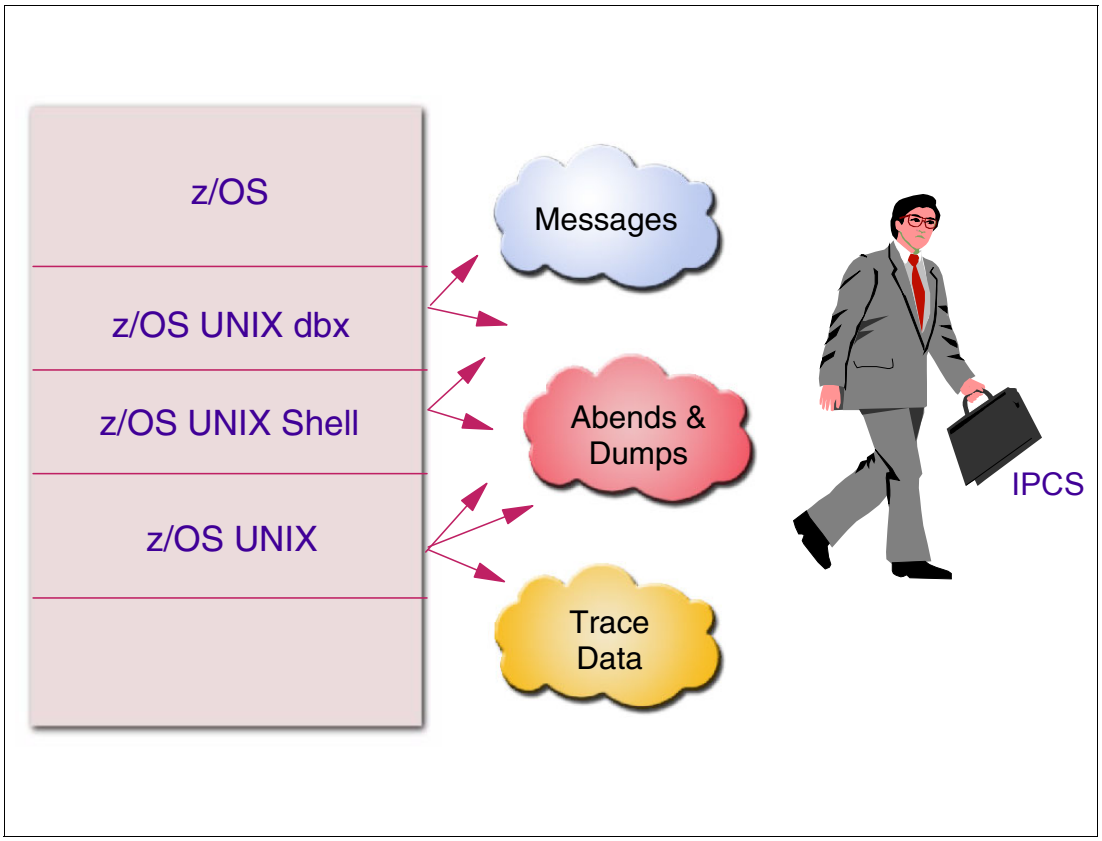


Figure 14-38 Where to find information to solve problems

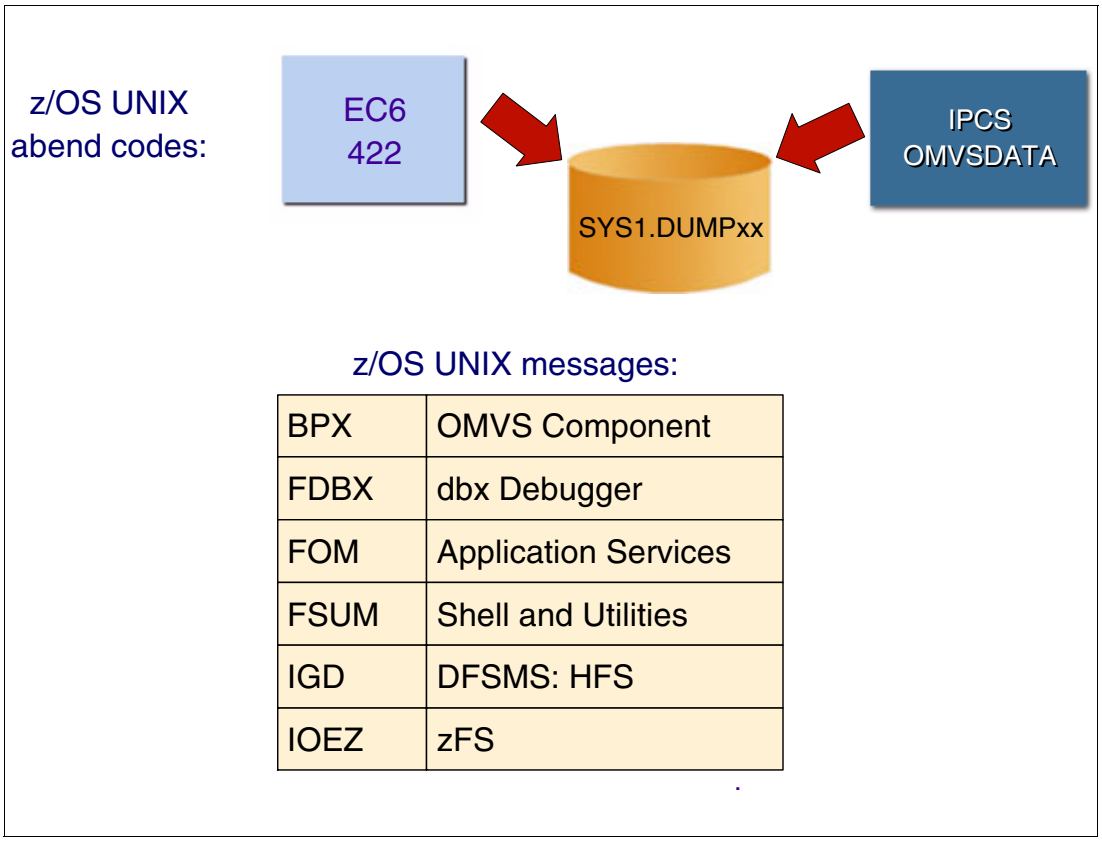
Problem solving information

If a problem occurs with z/OS UNIX System Services, the system writes an SVC dump and may issue messages. The SVC dump can be formatted and analyzed using IPCS. Messages referring to problems with z/OS UNIX kernel services will have the prefix `BPX`, shell messages will have the prefix `FSUM`, dbx messages will have the prefix `FDBX`, and messages relating to the hierarchical file system will have the prefix `GFU`. Problems with z/OS UNIX can be analyzed using the IPCS OMVSDATA keyword.

Problems with the z/OS UNIX shell and z/OS UNIX dbx are treated as application problems. If a SYSMDUMP data set is allocated for the TSO/E session, the system will create a core dump in an HFS file. The core dump must be copied to an MVS data set, and the dump can be formatted and analyzed using IPCS.

For debugging reasons, it is important to have a powerful trace facility. The CTRACE function in z/OS provides support for z/OS UNIX. The resulting trace data can be analyzed using IPCS.

14.39 z/OS UNIX abends and messages



BPX	OMVS Component
FDBX	dbx Debugger
FOM	Application Services
FSUM	Shell and Utilities
IGD	DFSMS: HFS
IOEZ	zFS

Figure 14-39 Understanding z/OS UNIX abends and messages

z/OS UNIX abends and messages

If there is a problem in the z/OS UNIX shell or debugger, the system will treat it as an application program. If a shell user has allocated a SYSMDUMP data set for the TSO/E session, the system will write a core dump in the user's working directory called coredump.pid.

In the example of stopping z/OS UNIX, we saw that `abend 422` was issued for the BPXOINIT process. This is normal. All 422 abends and some EC6 abends may not be accompanied by an SVC dump because the IBM-supplied IEASLP00 PARMLIB member contains SLIP commands to suppress the dumps.

The IPCS command OMVSDATA can be used to analyze a dump from z/OS UNIX. The abend codes are documented in *z/OS MVS System Codes,* SA22-7626. Reason codes can be found in an appendix of *z/OS UNIX System Services Programming: Assembler Callable Services Reference*, SA22-7803.

The reason codes and their descriptions can be found in *z/OS UNIX System Services Messages and Codes*, SA22-7807. Reason codes are listed both alphabetically, by name, and numerically, by value. The value is the lower half of the reason code. Messages with the following prefixes are issued from z/OS UNIX:

- FDBX and FSUM - z/OS UNIX dbx and shell messages. For an explanation of the messages, *z/OS UNIX System Services Messages and Codes*, SA22-7807.
- IGD - DFSMS messages for the hierarchical file system.

14.40 USS errors and codes

- Two high-order bytes of the reason codes are returned by z/OS UNIX - 0000 to X'20FF' or 7100-71FF
  - Contains a value that is used to qualify the contents of the two low-order bytes
- All HFS reason codes are one word (4 bytes) in length
  - The first byte is the subcomponent ID
  - The second byte is the module ID
  - The third and fourth bytes are the module-defined reason code

Figure 14-40 Reason codes and HFS reason codes

Reason codes

The reason code is made up of 4 bytes in the following format:

cccc rrrr

cccc cccc is a half word reason code qualifier. Generally this is used to identify the issuing module and represents a module ID.

rrrr rrrr is the half word reason code described in the appendix cited previously. Only this part of the reason code is intended as an interface for programmers.

The two high-order bytes of the reason codes returned by z/OS UNIX contain a value that is used to qualify the contents of the two low-order bytes. If the contents of the two high-order bytes are within the range of 0000 to X'20FF' or 7100-71FF (but not 7101), the error represented by the reason code is defined by z/OS UNIX. If the contents of the two high-order bytes is outside the range, the error represented by the reason code is not a z/OS UNIX reason code.

HFS reason codes

All HFS reason codes are one word (4 bytes) in length, as follows:

- The first byte is the subcomponent ID. For the HFS, this is X'5B'. For reason codes that do not contain the HFS Component ID X'5B' in the first byte, but do have one of the following patterns: 0Exxxxxx, 18xxxxxx, 1Exxxxxx, 22xxxxxx, 25xxxxxx.
- The second byte is the module ID.

- The third and fourth bytes are the module-defined reason.

To interpret a reason code that begins with X'5B', look at the reason (third and fourth bytes). For reason codes in the range of 0100 - 09FF:

1. The second byte of the reason code is the module ID. Use that hex ID to determine the name of the module in “Module IDs” in the table in the manual cited in the previous topic. The HFS module IDs are listed in the table in numerical order of module IDs for easy reference.
2. Next to the module name, you see the topic where the module reason codes are presented. (Some modules do not issue reason codes.)
3. Go to that topic. Locate the reason (third and fourth bytes) in the list of errors defined for that module.

14.41 CTIBPX00 and CTCBPXxx

CTIBPX00

```
TRACEOPTS
          ON
          BUFSIZE(128K)
/*        OPTIONS ('ALL   ') */
```

CTIBPXxx

```
CTIBPX01
  TRACEOPTS
            WTR(CWTR)
            WTRSTART(CTWTR)
            ON
            BUFSIZE(128K)
            OPTIONS('FILE', 'PIPE')
```

Figure 14-41 SYS1.PARMLIB members for tracing z/OS UNIX

z/OS UNIX trace PARMLIB members

Activating a CTIBPXxx PARMLIB member with additional trace options specified should only be done in situations with problems. Additional trace options will collect more information about z/OS UNIX and this can slow down the system performance noticeably. When requesting additional data to be traced, the trace buffer size should be increased. This cannot be done with TRACE CT commands.

The MVS component trace (CTRACE) provides support for z/OS UNIX. By default, z/OS UNIX performs a minimal amount of tracing at all times. The default tracing records only unusual events in z/OS UNIX to provide trace data when a problem occurs without slowing system performance. An installation can trace additional events by specifying tracing options in a CTIBPXxx PARMLIB member.

The system collects trace entries in a buffer. To analyze these entries, the buffer must be dumped to a data set. The installation can also decide to write the trace entries to a data set by using the component trace external writer.

The trace options are activated when z/OS UNIX is started. Trace options can be changed while the system is running by using the TRACE CT command. The trace options are activated by using IPCS to format the data. Trace data can be located in either:

- A buffer in an SVC or standalone dump
- A trace data set

14.42 Tracing z/OS UNIX events

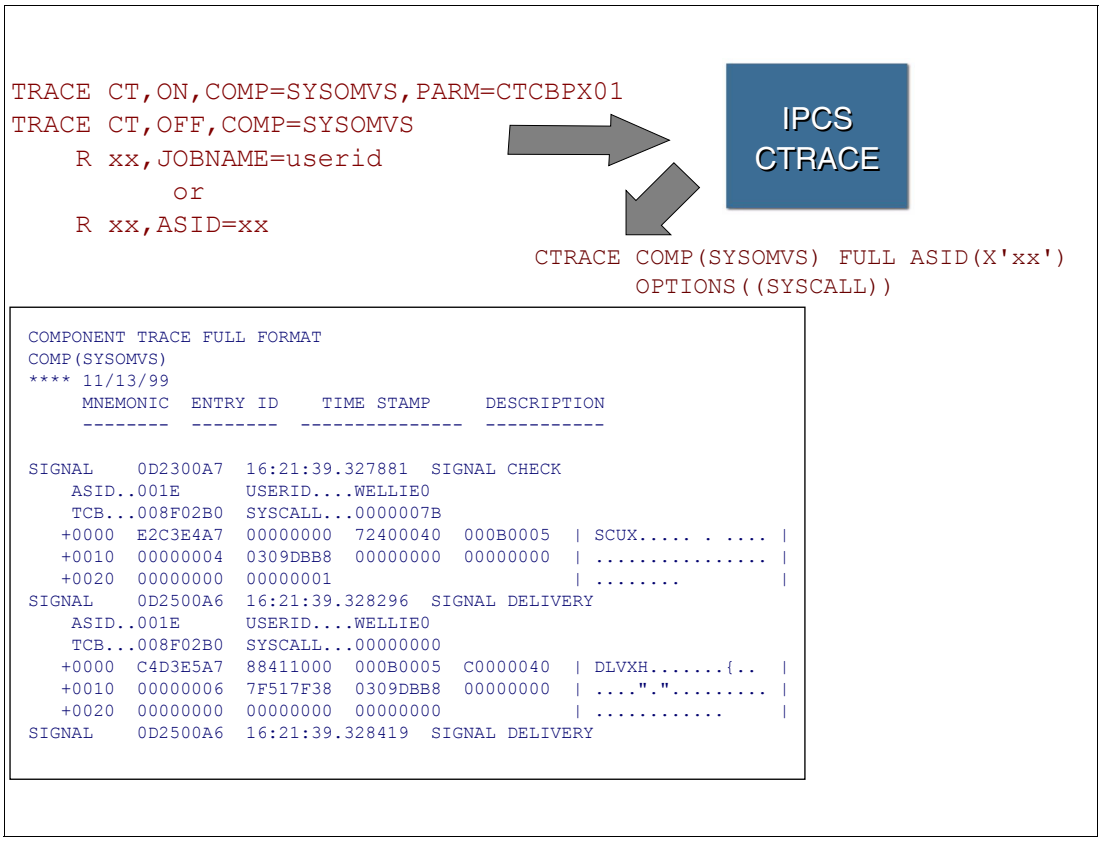


Figure 14-42 Commands to trace z/OS UNIX

Tracing z/OS UNIX events

To provide problem data, events are traced by the z/OS UNIX MVS component trace. When z/OS UNIX MVS starts, the trace automatically starts. The trace cannot be completely turned off while z/OS UNIX is running. The sizes of the trace buffers are specified in the PARMLIB member CTnBPXxx, which is used by z/OS UNIX. The trace buffers require an IPL to be changed. They can be from 16 KB to 4 MB. The following commands control tracing:

- Operator can stop most tracing with the command:

  `TRACE CT,OFF,COMP=SYSOMVS`

- To change the CTnBPXxx PARMLIB member:

  `TRACE CT,ON,COMP=SYSOMVS,PARM=CTnBPXxx`

- To display information about a trace:

  `DISPLAY TRACE,COMP=SYSOMVS`

Use a user ID or address space ID to filter the trace. The IPCS CTRACE subcommand can be used to analyze trace records in a dump.

The previous figure showed the default CTIBPX00 PARMLIB member that starts minimal tracing. An installation can define other members with trace options that can be started if a problem occurs with z/OS UNIX. The trace buffer size cannot be changed while z/OS UNIX is active. To change buffer size, update a CTnBPXxx member with the new BUFSIZE value, stop z/OS UNIX, and restart z/OS UNIX using the new CTnBPXxx PARMLIB member.

14.43 Debugging a z/OS UNIX problem

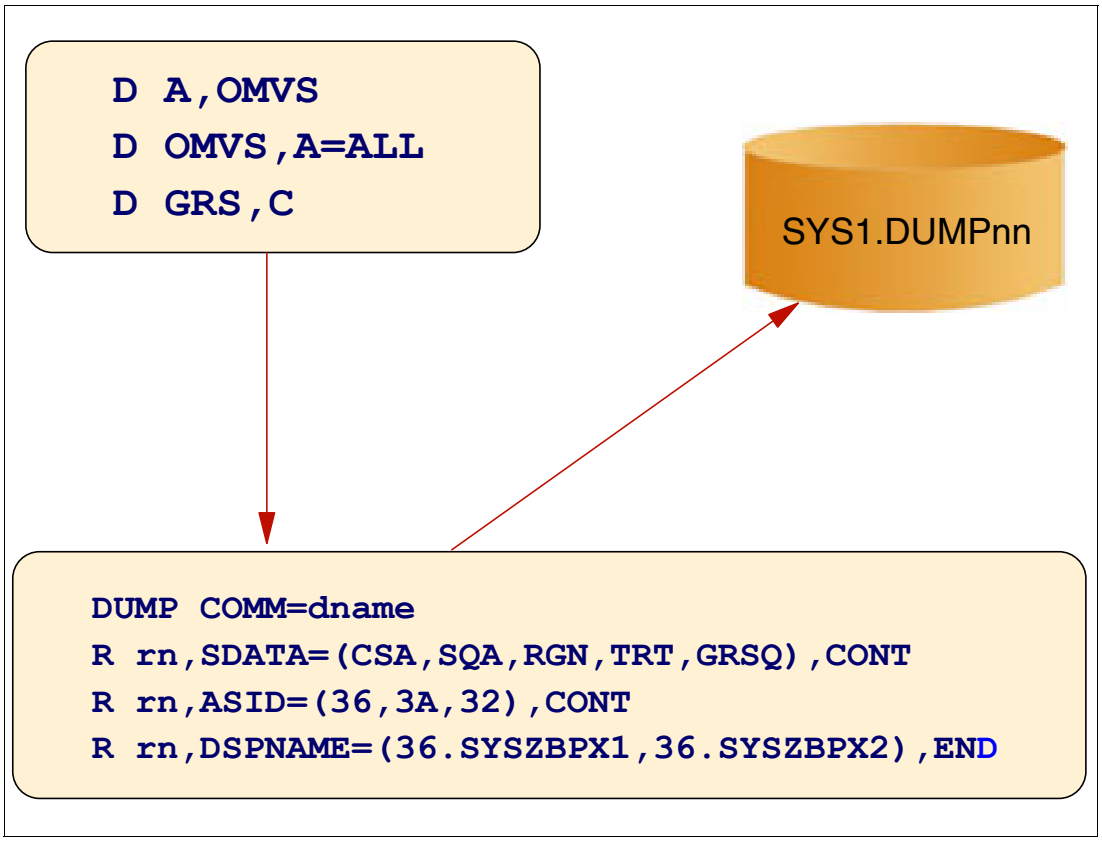


Figure 14-43 Commands to debug z/OS UNIX and take an abend

Commands to debug a z/OS UNIX problem

Information about the kernel address space and its associated data spaces can be obtained from the MVS displays, as we have already seen.

When you need a dump to debug a problem, use the DUMP command to tell MVS what to dump. The dump will go to the SYS1.DUMPnn data set. You need to dump several types of data in order to diagnose a problem, as follows:

- The OMVS kernel address space
- Any OMVS data spaces that may be associated with the problem
- Any OMVS process address spaces that may be associated with the problem
- Appropriate storage areas containing system control blocks

The sample dump command shows dumping the kernel address space, the appropriate user address spaces, and the two major kernel data spaces. You will need to allocate a very large dump data set since you are dumping multiple address spaces and data spaces. The dump title can be up to 100 characters. If you end each line with `CONT`, then the program will prompt you for more input data. When you are finished, code `END`. You will need to use the OMVS displays to determine the appropriate data to dump. After the dump completes, you receive an `IEA911E` message indicating whether the dump was complete.

In Figure 14-43, ASID=36 is the OMVS kernel. Address spaces 3A and 32 are other address spaces believed to be part of the problem. Refer to the data spaces by their names with the kernel address space ID as a preface.

14.44 IPCS OMVSDATA reports

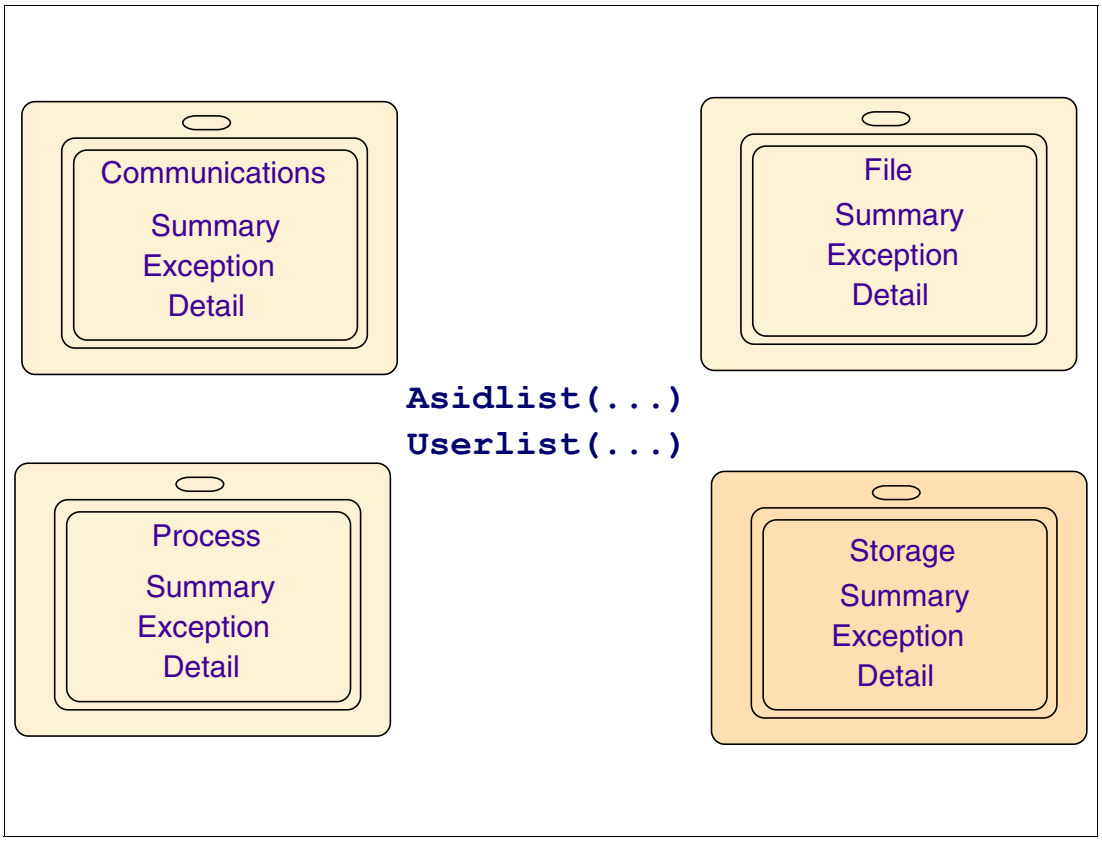


Figure 14-44 The OMVSDATA keyword in IPCS for analyzing problems

IPCS OMVSDATA reports

The IPCS OMVSDATA keyword provides support for analyzing problems with the z/OS UNIX component. SVC dumps or standalone dumps can be formatted using OMVSDATA. The following report types can be created:

- **Communications**: Provides information about z/OS UNIX pseudoterminal user connections.
- **File**: Provides information about each z/OS UNIX file system type and its mounted file systems.
- **Process**: Provides information about z/OS UNIX processes.
- **Storage**: Provides information about z/OS UNIX storage manager cell pools.

The default report type is PROCESS. For each report type, the level of detail can be determined by specifying: summary, exception, and detail. For each report, one or more of the filtering keywords can be used to limit the amount of data in the report, as follows:

- **ASIDLIST(asidlist):** requests that information be included for the ASIDs listed
- **USERLIST(userlist):** requests that information be included for the user IDs listed.

For an application dump (coredump.pid in the HFS), you will have to copy it out to a sequential data set. Then use IPCS to analyze it using the STATUS and SUMMARY FORMAT CURRENT subcommands. The OMVSDATA reports will not work on the application dump as they format system areas in the kernel.

15

z/OS UNIX shell and programming tools

This chapter introduces you to various programming environments, and highlights some of the possibilities for programming under z/OS UNIX.

It describes how to:

- Understand the environment needed
- Choose the most appropriate programming language
- Install the environment needed for programming
- Write short programs in different programming languages

© Copyright IBM Corp. 2011. All rights reserved.

15.1 Language Environment run-time library

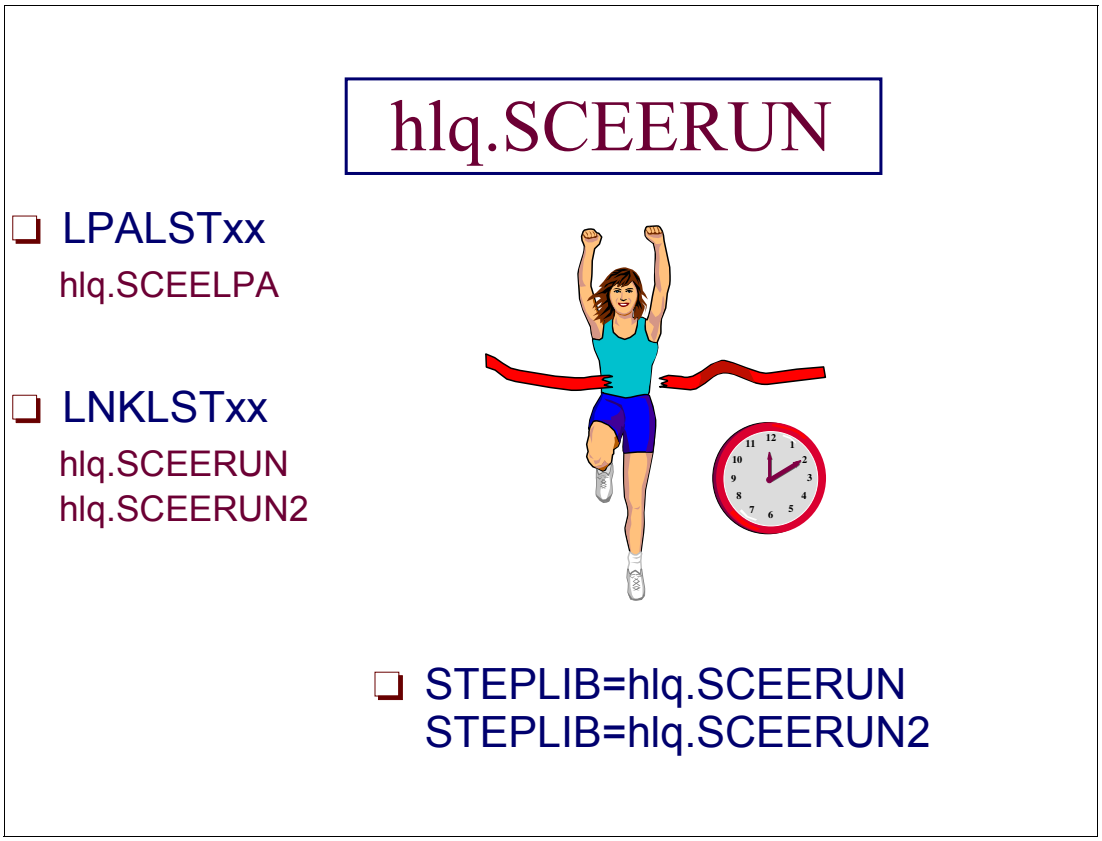


Figure 15-1 The LE run-time library specifications

SCEERUN support

For most z/OS UNIX applications, the Language Environment (LE) run-time library is needed for execution; it comes from the SCEERUN data set. On average, about 4 MB of the run-time library are loaded into memory for every address space running Language Environment-enabled programs, and copied on every fork.

When SCEERUN2 is added, it contains LE load modules that are required to reside in a PDSE. Add this data set to the linklist. The SCEERUN and SCEERUN2 data sets can be:

- Placed in LPA/LNKLST

  If you are using the same compiler for the entire system, then put the compiler data set name in the linklist. By default, the linklist contains the name of the default compiler.

  > **Note:** If the SCEERUN and SCEERUN2 data sets cannot be placed in LNKLST, you can STEPLIB the data sets for each application that requires them. One reason why the Language Environment run-time libraries are not to be placed in LNKLST might be that the pre-Language Environment run-time libraries (VS COBOL II, OS PL/I) are placed in LNKLST and your site has not completed the migration to Language Environment.

- Accessed via STEPLIB

  If you are using a compiler that is not the system-wide default, then you must specify the compiler data set name in the STEPLIB environment variable and export it.

Note: Applications that currently STEPLIB to the SCEERUN data set to gain access to the run-time library provided by Language Environment, do not need to add the SCEERUN2 data set as part of their STEPLIB concatenation. In fact, since SCEERUN2 contains module names that do not intersect with any pre-Language Environment run-time library or any existing library, IBM recommends that SCEERUN2 be added to the LNKLST. This will not result in any adverse effects.

Run-time library access

When choosing a method for run-time library access, you should consider the following:

- Can the Language Environment run-time library be placed in LNKLST without adversely affecting other applications?
- Is the Language Environment run-time library heavily used at your installation?
- Does the RTL require frequent testing or replacement with new versions?

Some installations cannot put the current level of the LE run-time library into the LINKLIST because older Language Environment levels are needed to run key production applications. This means that key run-time library routines cannot be put in the LPA for better performance. In addition, you cannot put the SCEELPA data set as part of the LPALSTxx.

In z/OS V1R6, Language Environment no longer uses the RTLS services provided by the operating system, which was previously used to assist with run-time migration. This includes removal of the RTLS initialization paths and all descriptions of RTLS in the Language Environment publications. The SCEERTLS library will no longer be shipped. The following run-time options are no longer supported:

- LIBRARY
- RTLS
- VERSION

These run-time options are removed from the options reports generated by RPTOPTS(ON), CEEDUMP, and the IPCS verb exit. The CEEXOPT macro has been updated to prevent the use of these run-time options when building new CEEDOPT, CEECOPT, CEEROPT or CEEUOPT CSECTs. Existing CEECOPT and CEEDOPT members that contain these run-time options must be modified to remove them. If these run-time options are encountered in existing CEEROPT or CEEUOPT CSECTs, Language Environment issues CEE3611I informational messages.

Performance considerations

Because the SCEERUN data set has many modules that are not reentrant, you cannot place the entire data set in the link pack area using the LPALSTxx member of SYS1.PARMLIB. However, you can take advantage of a SCEELPA data set that contains a subset of the SCEERUN modules—those that are reentrant, reside above the line, and are heavily used by z/OS UNIX.

To improve performance, put the SCEERUN data set in the link list, using the LNKLSTxx member of SYS1.PARMLIB. Then place the new SCEELPA data set in the LPA list, using the LNKLSTxx member.

Tip: You can also add additional modules to the LPA, using the dynamic LPA capability (SET PROG=). This method is preferable to adding modules to the LPA by using the Modify Link Pack Area (MLPA=) option at IPL, because it avoids the performance degradation that occurs with the use of MLPA.

15.2 Using pre-LE run-time libraries

- Add the SCEERUN data set on a STEPLIB DD statement to OMVS startup procedure found in PROCLIB
- Add the SCEERUN data set to your TSO/E logon procedure
- Add the following statement to the /etc/rc file:
  - export STEPLIB=hlq.SCEERUN
- In /etc/profile, use:
  - export STEPLIB=hlq.SCEERUN
- Add the SCEERUN data set on a STEPLIB DD statement to any job invoking BPXBATCH
- Add the SCEERUN data set to the STEPLIBLIST statement of the BPXPRMxx PARMLIB member

Figure 15-2 Using STEPLIBs with SCEERUN for pre-LE run-time libraries

STEPLIB and SCEERUN

Be aware that putting the Language Environment run-time library in LNKLIST requires the least amount of setup. However, if your applications require the pre-Language Environment run-time library, then make the Language Environment run-time library available through STEPLIB.

Perform the following steps to make the run-time library available through STEPLIB:

- Add the SCEERUN data set on a STEPLIB DD statement to the OMVS startup procedure found in PROCLIB. The STEPLIB data set is then propagated to BPXOINIT and the /usr/sbin/init program, including all programs it invokes using fork or exec.
- Add the SCEERUN data set to the TSO/E logon procedure by concatenating it to the ISPLLIB DD statement (if it exists) and then concatenating it to the STEPLIB DD statement (if it exists). Also, the TSOLIB function can be used to add the SCEERUN data set. After adding the SCEERUN data set, the TSO/E OMVS command can begin to use it.
- Add the following statement to the /etc/rc file:

  ```
  export STEPLIB=hlq.SCEERUN
  ```

  Daemons started in /etc/rc will use the SCEERUN data set.
- In /etc/profile, remove:

  ```
  if [ -z "$STEPLIB" ] && tty -s;
    then
  ```

```
        export STEPLIB=none
        exec sh -L
    fi
```

and replace with:

```
export STEPLIB=hlq.SCEERUN
```

This is used when issuing commands and utilities in the shell environment. If a small number of interactive users need a special version of the run-time library, the STEPLIB environment variable can be set in the $HOME/.profile for each of these users.

- Add the SCEERUN data set on a STEPLIB DD statement to any job invoking BPXBATCH.
- Add the SCEERUN data set to the STEPLIBLIST statement of the BPXPRMxx PARMLIB member. The SCEERUN data set must be APF-authorized.

When this is done, the Language Environment run-time library is made available through STEPLIB. For BPXBATCH processing, you still have to specify the SCEERUN data set on a STEPLIB DD statement even though the RUNOPTS parameter has been set in the BPXPRMxx member.

Note: Place the SCEERUN2 data set in LNKLST, even though the SCEERUN data set is accessed through STEPLIB. Because the SCEERUN data set does not contain module names that conflict with pre-Language Environment run-time libraries, adding it to LNKLST will not have any adverse effects.

15.3 Overview of c89/cc/c++

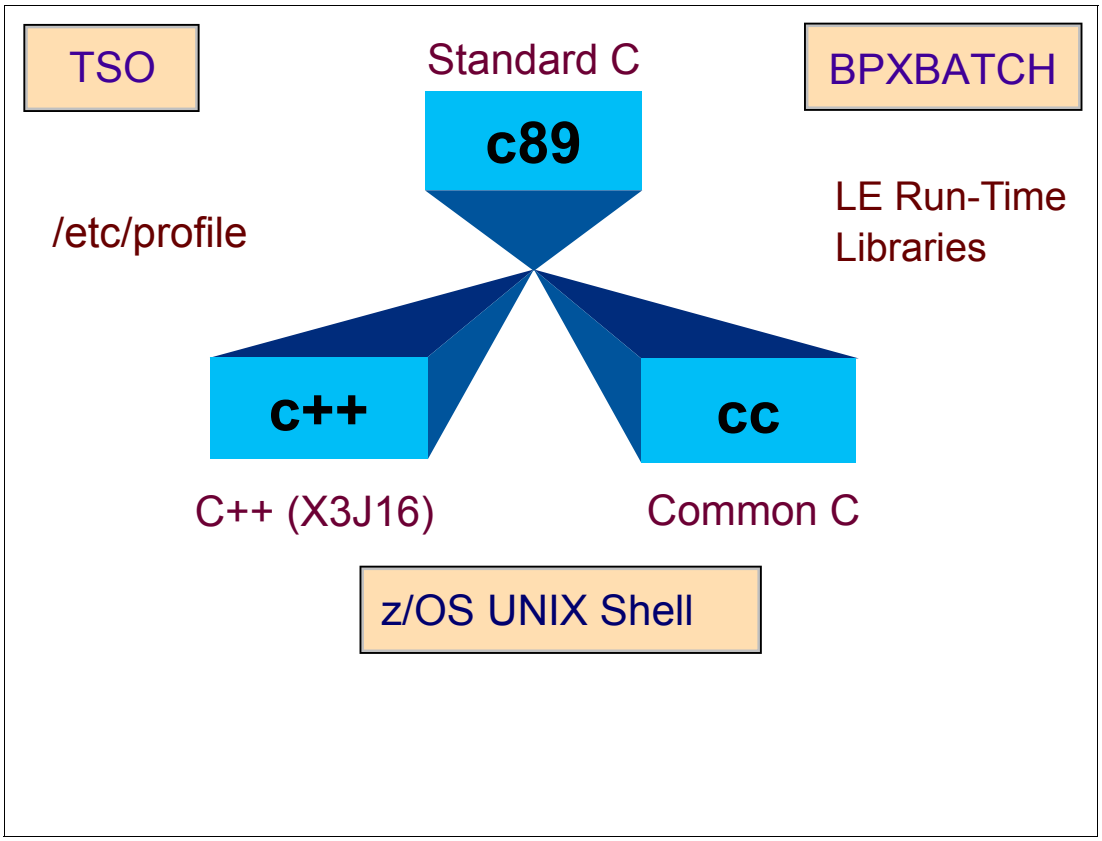


Figure 15-3 Overview of the C/C++ functions with z/OS UNIX

C/C++ functions and assembler calls

z/OS UNIX can be used from C/C++ functions and assembler calls. Applications written in COBOL or PL/I can use z/OS UNIX indirectly through C/C++ functions or assembler calls. If you must use a previous level of the compiler, or target the executables produced by c89 to run on a previous level of the run-time library, then you must customize other environment variables.

c89, cc, and c++ compile, assemble, and link-edit z/OS C and z/OS C++ programs, as follows:

- c89 should be used when compiling C programs that are written according to Standard C.
- cc should be used when compiling C programs that are written according to Common Usage C.
- c++ must be used when compiling C++ programs, which are written according to Draft Proposal International Standard for Information Systems—Programming Language C++ (X3J16). c++ can compile both C++ and C programs, and can also be invoked by the name cxx.

The c89 utility is customized by setting environment variables. The ones that most commonly require setting are specified in the c89 customization section in /etc/profile.

The customization section in /etc/profile assumes that you are using the current level of z/OS C/C++ compiler and Language Environment run-time library.

15.4 Customization of /etc/profile for c89/cc/c++

```
# Start of c89/cc/c++ customization section
# ==================================================================
# High-Level Qualifier "prefixes" for data sets used by c89/cc/c++:
# Compiler:
   export _C89_CLIB_PREFIX="CBC"
# Prelinker and runtime library:
   export _C89_PLIB_PREFIX="CEE"
#
# z/OS system data sets:
   export _C89_SLIB_PREFIX="SYS1"
# Compile and link-edit search paths:
#   Compiler include file directories:
    export ${_CMP}_INCDIRS="/usr/include /usr/lpp/ioclib/include"
#   Link-edit archive library directories:
    export ${_CMP}_LIBDIRS="/lib /usr/lib"
# Esoteric unit for data sets:
#==================================================================
#   Unit for (unnamed) work data sets:
    export ${_CMP}_WORK_UNIT="SYSALLDA"
```

Figure 15-4 Customization of /etc/profile for cc/c++

/etc/profile customization for compilers

If c/c++, LE, or z/OS do not use the installation default for the high-level qualifier, then the appropriate environment variable must be exported to make c89 aware of this. The environment variables in Figure 15-4 are set to the default values for the current level of z/OS, but you will need to set them to your high-level qualifiers.

In order to get the cxx environment up and running, you have to customize the /etc/profile.

The environment variables used by the cc utility have the same names as the ones used by c89, except that the prefix is _CC instead of _C89. Likewise, for the c++ (cxx) utility, the prefix is _CXX instead of _C89. Normally, you do not need to explicitly export the environment variables for all three utilities; the **eval** commands at the bottom of the c89 customization section of the sample /etc/profile can be used. These commands set the variables for the other utilities, based on those set for c89.

Customization for /etc/profile: By placing any customization statements for c89 into /etc/profile and uncommenting those lines, the environment variables will automatically be exported through the **eval** command for cc and c++ as well.

Export statements for compiler versions

These are the export statements for each compiler version, assuming that the default high-level qualifiers are being used. Where the c89 environment variables are shown, the environment variables for c++ and cc must also be set, as follows:

For the current z/OS C/C++ compiler:

- If you are using the z/OS shell, issue the following command:

```
export STEPLIB="CBC.SCBCCMP"
```

- If you are using the tcsh shell, issue the following command:

```
setenv STEPLIB "CBC.SCBCCMP"
```

Because this compiler only supports the C language, it cannot be used with the c++ utility.

The primary Language Environment level must be the default level for the release, and you must be using the default compiler. To verify your Language Environment primary level, check that the library name (for example, CEE.SCEERUN) appears first in the linklist concatenation in the LNKLSTxx member of SYS1.PARMLIB.

On systems where application development is the primary activity, performance may benefit if you put CBC.SCBCCMP in the LPALSTxx concatenation. All compiler modules run above the line, and they consume just over 42 MB in total.

15.5 Compile, link-edit, and run

TSO

```
CXX 'C_library(CBC3UBRC)' ( LSEARCH('H_library') OBJECT(BIO.TEXT)
CXXMOD OBJ(BIO.TEXT(CBC3UBRC)) LOAD(BIO.LOAD(BIORUN))
CALL BIO.LOAD(BIORUN)
```

z/OS UNIX

```
tso oput "'cbc.scbcsam(cbc3ubrc)' '$PWD/cbc3ubrc.C'"
tso oput "'cbc.scbcsam(cbc3ubrh)' '$PWD/cbc3ubrh.h'"
c++ -o bio cbc3ubrc.C
./bio
```

MVS JCL Job

```
//DOCLG     EXEC  CBCCLG,
//          INFILE='PETE.TEST(CBC3UBRC)',
//          CPARM='LSEARCH(''''PETE.TESTHDR.H'''')'
/*
```

Figure 15-5 Ways to compile, link-edit, and execute C/C++ programs

Creating and executing programs with z/OS UNIX

The interface to the linkage editor for z/OS UNIX System Services (z/OS UNIX) C applications is the z/OS UNIX c89 utility or the cc utility, and for C++ applications it is the c++ utility. You can use them to compile and link-edit a z/OS UNIX C/C++ program in one step, or link-edit application object modules after the compilation. You must, however, invoke one of the z/OS UNIX shells before you can run the c89 utility.

Program execution

Following are some examples of using the three different methods of compiling, doing a link-edit, and running the program.

To compile, link-edit, and run under TSO:

- Compile:

  ```
  CXX 'PETE.TEST.C(CBC3UBRC)' (LSEARCH('PETE.TESTHDR.H') OBJECT(BIO.TEXT)
  ```

- Prelink and link:

  ```
  CXXMOD OBJ(BIO.TEXT(CBC3UBRC)) LOAD(BIO.LOAD(BIORUN))
  ```

- Run the program:

  ```
  CALL BIO.LOAD(BIORUN)
  ```

Under z/OS UNIX, put the source into files in the HFS:

- Compile, prelink, and link:

```
c++ -o bio cbc3ubrc.C
```

- Run the program:

```
GGI:/u/ggi:==> ./bio
```

Under batch, use the cataloged procedure CBCCLG to compile, link, and run:

```
//DOCLG    EXEC  CBCCLG,
//         INFILE='PETE.TEST(CBC3UBRC)',
//         CPARM='LSEARCH(''''PETE.TESTHDR.H'''')'
/*
```

Note: As of z/OS V1R8 with XL C/C++, when using the IPA compiler option to compile very large applications, you might need to increase the size of the work file associated with SYSUT5 DD in the IPA Link step. If you are using JCL procedures to compile the application, you can save the file space associated with the work files in the IPA Link step by defining DUMMY files as the work files associated with SYSUT6, SYSUT7, SYSUT8, and SYSUT14.

Including the CXX parameter with user-defined JCL

The CBCC, CBCCL, and CBCCLG procedures, which compile C++ code, include parameter CXX when the following compilers are used:

- C++ for MVS/ESA V3R2
- OS/390 C/C++
- z/OS C/C++

If you have written your own JCL to compile a C++ program, you must include this parameter; otherwise, the C compiler is invoked. When you pass options to the compiler, you must specify parameter CXX. You must use the following format to specify options:

```
run-time options/CXX compiler options
```

15.6 Customization of Java for z/OS

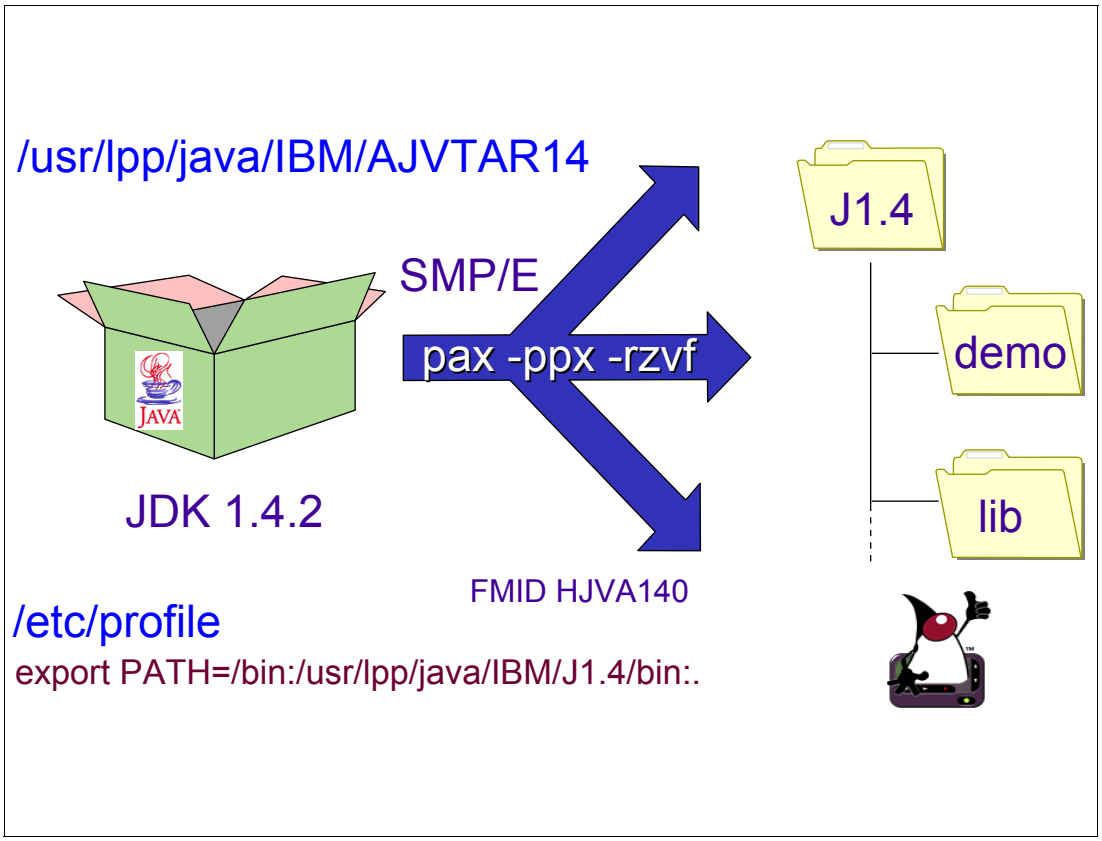


Figure 15-6 Customization of Java for z/OS

Java customization

The Java for z/OS product is IBM's port of Sun Microsystems' Java Development Kit (JDK) to the S/390 platform. The Java for z/OS product at the JDK 1.4.2 level is certified as a fully compliant Java product.

Java for z/OS is a full object-oriented language in 64-bit mode. In contrast to C++, you have to write your programs with object-oriented code. (In C++, you do not have to use object-oriented code.) The big difference between the C++ code and Java code is that Java does not use pointer and pointer arithmetic.

Java for z/OS is operational within any version and release of the z/OS operating system. It provides a Java execution environment equivalent to that available on any other server platform.

Java Programming Language: Java Programs: Write once, run everywhere.

If ordered with ServerPac, the Java code has already been put in the root HFS under the path of /usr/lpp/java. The FMID for Java is HJVA140.

To find the requested files for Java, you have to add the path /usr/lpp/java/IBM/J1.4/bin to the /etc/profile or $HOME/.profile.

15.7 Java virtual machine

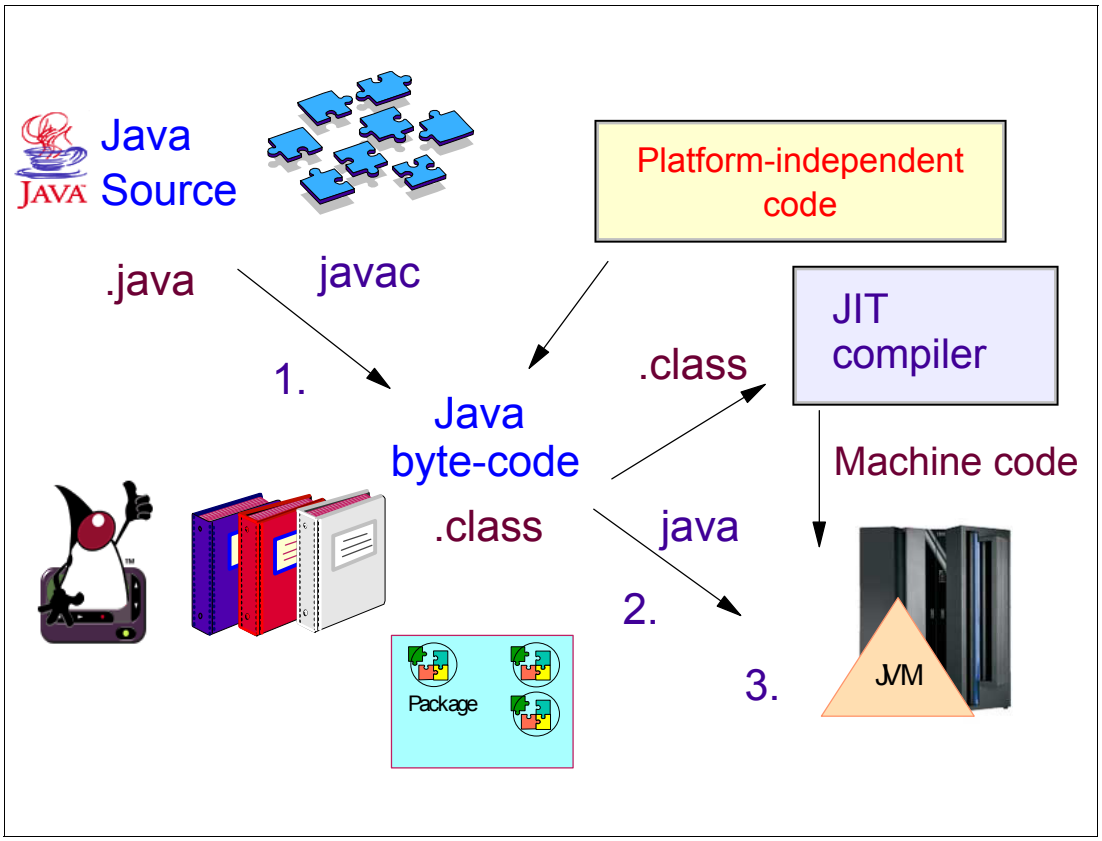


Figure 15-7 Java virtual machine

Java virtual machine (JVM)

A *virtual machine* is the processor on which Java's byte-codes run. It is the instruction set that the interpreter understands.

The compiler, javac, takes the Java source code and produces Java byte-codes. These byte-codes correspond to the language of the virtual machine. The file organization is such that byte-codes are per class, that is, one .class file per Java class definition.

These class files may be loaded across the network.

Because of the need for architecture independence, performance tuning must be performed on the client side. This client-side compilation is known as just-in-time (JIT) compilation.

Since the Java virtual machine (JVM) does not necessarily correspond to any particular hardware/operating system, the .class files are portable to any implementation of the virtual machine. This is the essence of Java's portability.

JIT will take Java byte-code and generate native code for the client processor. Many optimizations are possible that can lead to execution speeds which are competitive with C/C++ (within a factor of 2 to 5).

The JIT compiler is built specifically for z/OS. It provides execution time improvements over the interpreter. By default, the JIT compiler is activated but can be deactivated by setting an environment variable.

15.8 Management of software and the make utility

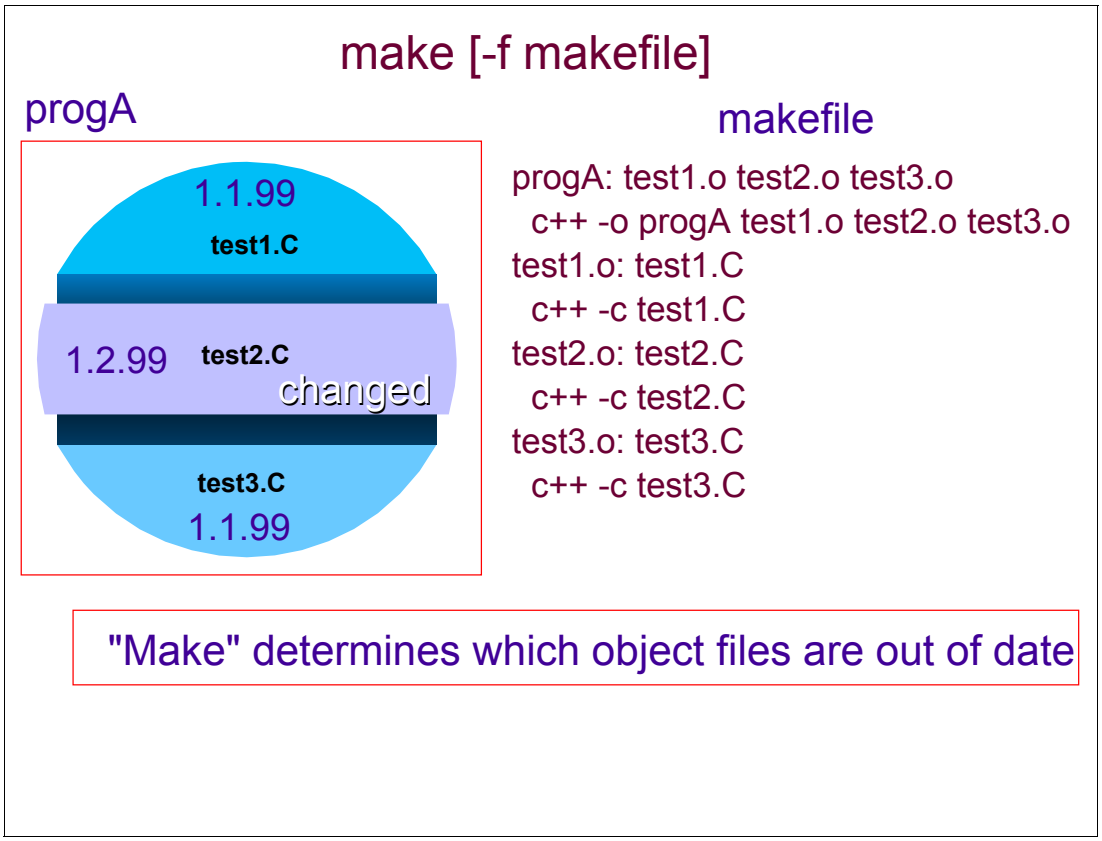


Figure 15-8 Using the make utility to manage software

Make utility

Using the **make** utility can be a key factor in the successful management of software development projects, as well as any other type of project where you must keep a collection of files in synchronization with one another.

For example, suppose a program is built from several separate object files, each of which depends on its own source file. If you change a source file and then run **make**, **make** can automatically determine which object files are out of date (older than their corresponding source files).

The information that the **make** utility uses is in a file called the makefile. To invoke the utility, simply use **make -f makefile**, or, if you are in the same directory as the makefile, use only **make**.

This section introduces the syntax used in the makefile:

- Program: test1.o test2.o test3.o

  This tells the program that progA depends upon the three files with the names test1.o, test2.o and test3.o. If any or all of the .o files have changed since the last time the program was made, make attempts to remake the program. It does this using the recipe on the next line. This recipe consists of a C++ command that links programs from the three objects.

- c++ -o progA test1.o test2.o test3.o

  If the **make** utility determines the changes, it will execute this line and try to link the program from the three objects.

- test1.o: test1.C

  This statement tells the **make** utility that if the test1.C code has changed, it must recompile the code to the object test1.o with the following commands.

- c++ -c test1.C

  As mentioned previously, the **make** utility recompiles the test1.C code and creates the object test1.o.

The next statements are the same as for the test1.C file.

The syntax of the **make** utility is:

```
target target ... : prerequisite
      prerequisite ... <tab> recipe
```

The <tab> syntax is a character with `X'05'`. This is not displayable from the ISPF Editor.

15.9 The dbx debugger

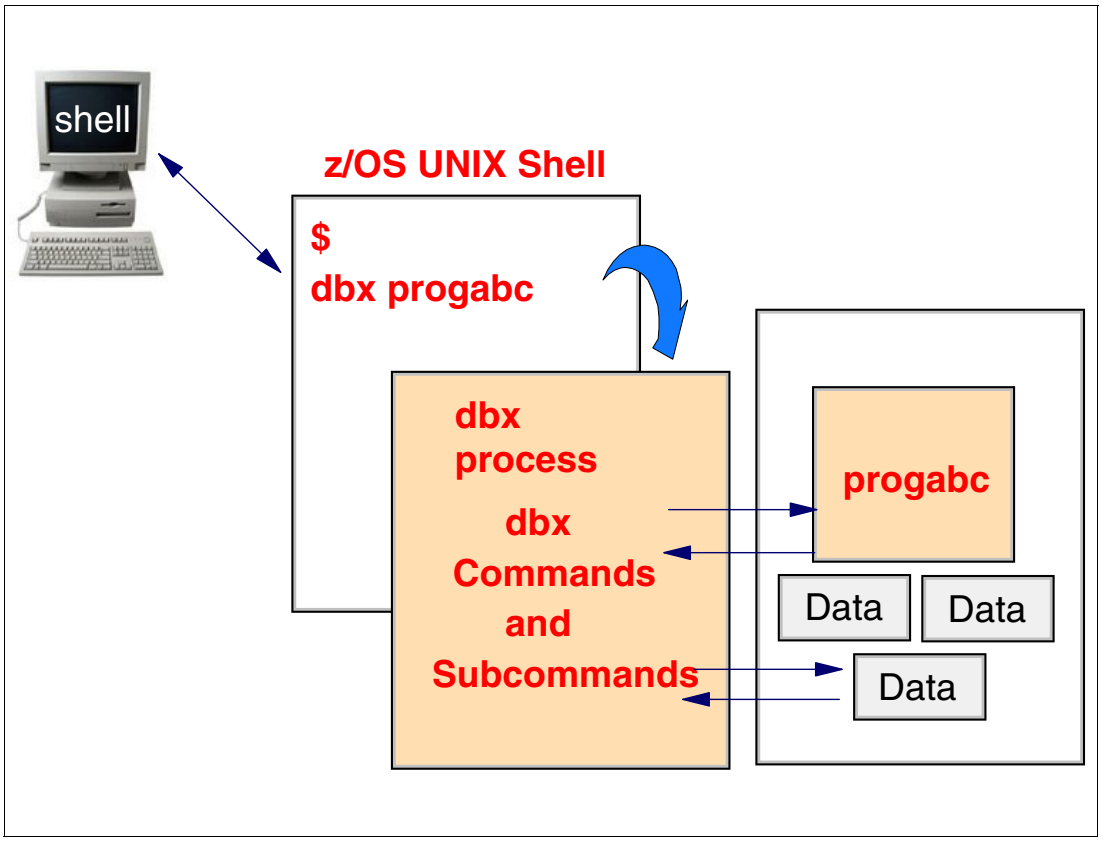


Figure 15-9 z/OS UNIX dbx debugger

z/OS UNIX dbx debugger

The z/OS UNIX dbx debugger is an interactive tool for debugging C language programs that use z/OS UNIX. It is based upon the dbx debugger, which is regarded as an industry standard on UNIX systems.

The dbx debugger provides the options to debug at source level or assembler level. Source-level debugging allows you to debug C language programs. Assembler-level debugging allows you to debug executable programs at machine code level.

The dbx debugger is a utility that is invoked from the z/OS UNIX shell. It cannot be invoked directly from TSO/E. In the shell, dbx is the debugging facility for z/OS C/C++ programs. With dbx, you can debug multi-threaded applications at the C-source level or at the machine level. Support for multi-threaded applications gives you the ability to:

- Debug or display information about the following objects related to multithreaded applications: threads, mutexes, and condition variables.
- Control program execution by holding and releasing individual threads.

When using the interactive dbx debugger you can:

- Run a program one line or one instruction at a time.
- Set breakpoints at selected statements and machine instructions with conditions for activation.
- Access variables symbolically and display them in the correct format.

- Display or modify the contents of registers, variables, and storage.
- Examine the source text using simple search functions.
- Debug processes that contain fork() and exec() functions.
- Interrupt and examine a program that is already in progress.
- Trace processing of a one-task program by line, instruction, routine, or variable.
- Call programs or diagnostic routines directly from the debug program.
- Invoke shell commands from debug session.
- Examine loaded address maps for a process.

15.10 The dbx debugger

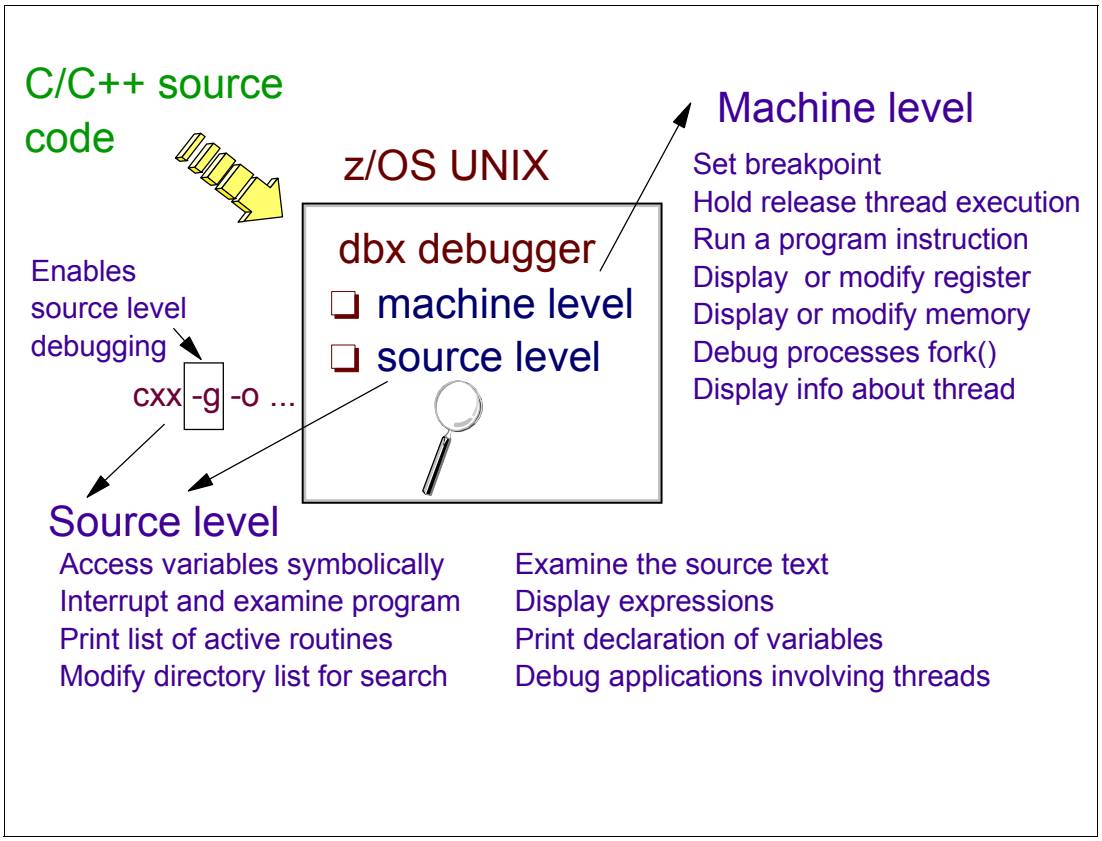


Figure 15-10 The dbx debugger

Using the dbx debugger

You need to create a z/OS UNIX C/MVS™ application program that will compile, link-edit, and run successfully. After your program has been developed, you can take advantage of the z/OS UNIX debugger (with its dbx utility) to debug the program from within the shell environment on an MVS system.

Using dbx, you can debug your program at the source level and at the machine level.

To trace the source level, the programs must be compiled with the -g option:

```
c89 -g -o looper looper.c
```

The dbx debugger also has some restrictions that must be considered before debugging programs. Some of them are:

- It can only debug key(8) programs
- It cannot debug programs in supervisor state
- It cannot source-level debug DLLs below OS390 V2R2.

15.11 Introduction to shells

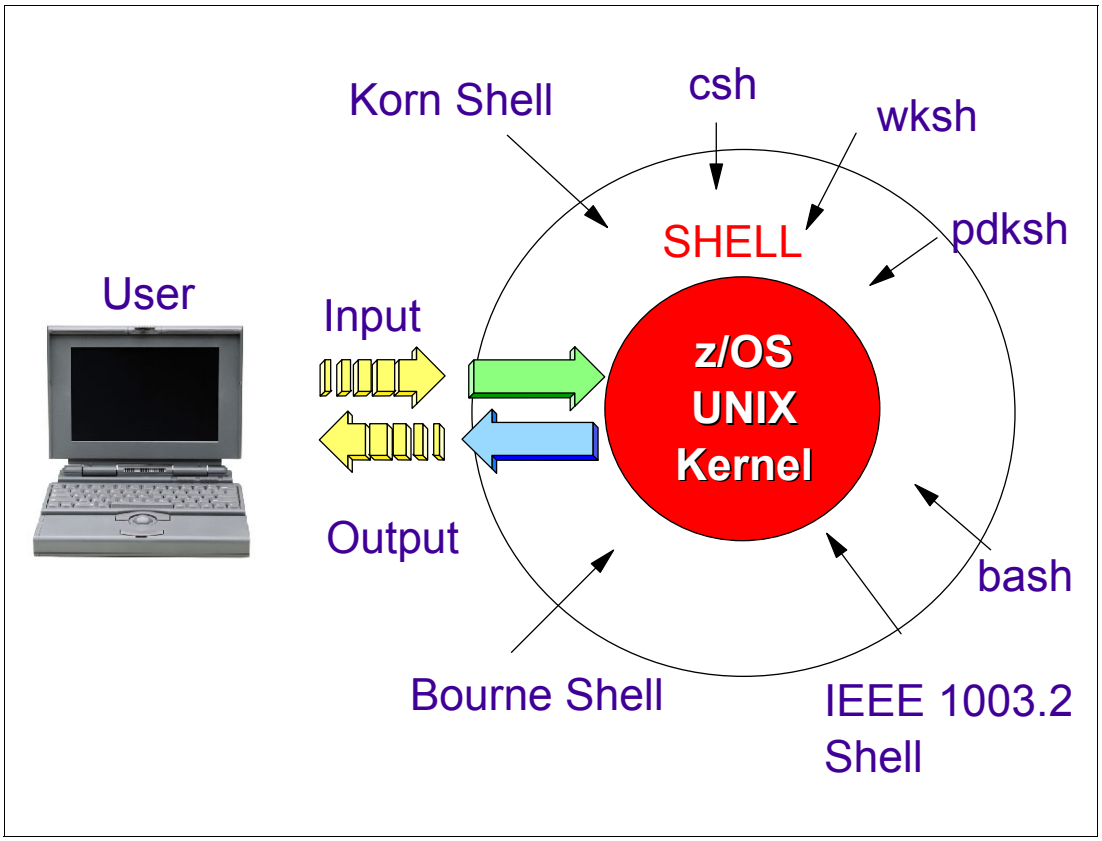


Figure 15-11 An introduction to the many shells available with z/OS UNIX

Introduction to UNIX shells

What is a *shell*? A shell's job is to translate the user's command lines into operating system instructions.

There are many different shells available for UNIX systems, as you can see. This led to the current situation, where multiplicity of similar software has led to confusion, lack of compatibility, and—most unfortunate of all—the inability of UNIX to capture as big a share of the market as other operating platforms.

The differences between the shells are minimal but have a big impact. Under z/OS UNIX, you can supply an alternate shell. The shells shown in Figure 15-11 are:

IEEE POSIX 1003.2	The standard used by z/OS UNIX
Bourne Shell	The first major shell; named after Steven Bourne (sh)
C shell	An alternative to the Bourne Shell, which was written at Berkley
Korn Shell	Compatible with the Bourne Shell; written by David Korn (ksh)
wksh	Windowing Korn Shell, which is the ksh shell with the extension for GUI
pdksh	Public Domain V7-based; written by Eric Gisin
bash	The Bourne-Again Shell; written by Brian Fox and Chet Ramey

You can find a short description of how to set up the Korn Shell in *z/OS UNIX System Services Planning*, GA22-7800.

The steps are as follows:

1. Download the shell program, for example, Korn Shell from the z/OS UNIX home page.
2. Install to a directory like `/ksh`.
3. Consider turning on the sticky bit and putting the program in the Link Pack Area.
4. For users that want this shell as the default shell, change the PROGRAM in the OMVS RACF user profile as follows: /ksh/bin/ksh.
5. Customize /etc/init.options with the following statements:

   ```
   -sh /ksh/bin/ksh
   -e  SHELL=/ksh/bin/ksh
   ```

6. You can test the environment variables after changing the alternate shell with the command **env**:

   ```
   SHELL=/ksh/bin/ksh
   PATH=/ksh/bin:. ..........
   ```

15.12 REXX, CLISTs, and shell scripts

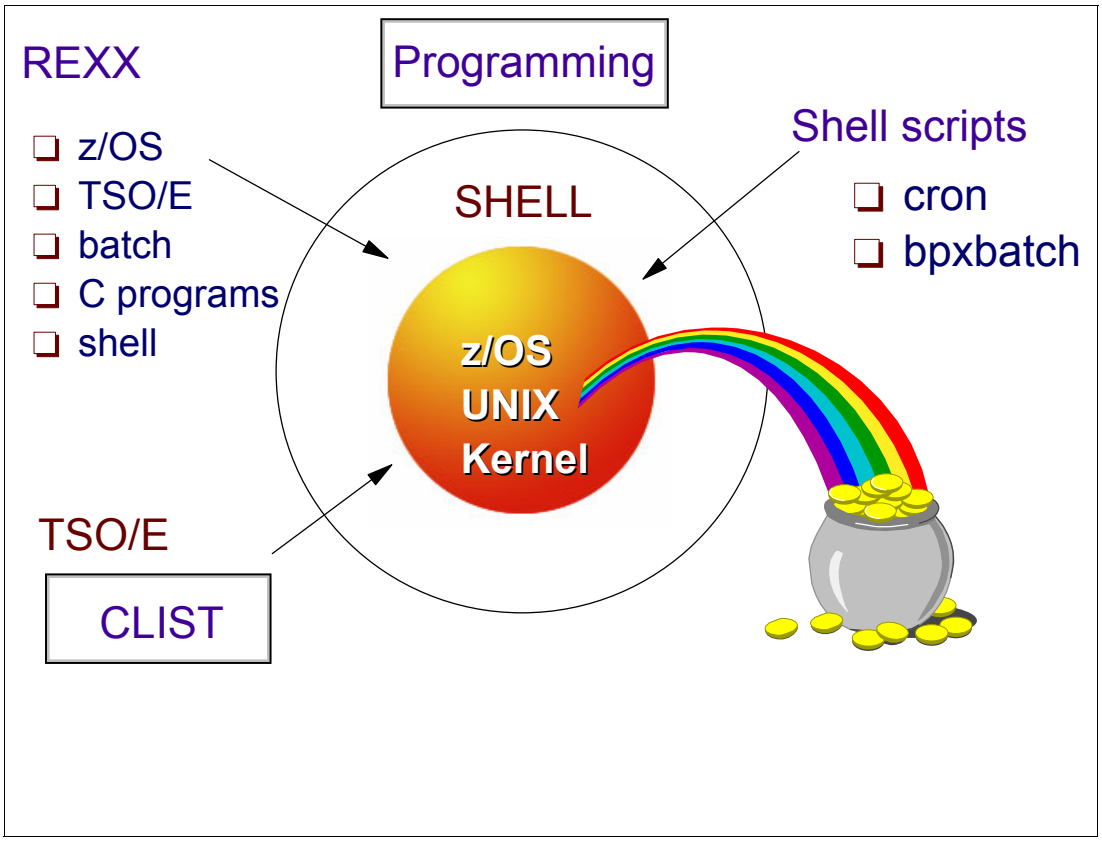


Figure 15-12 Different languages that can be used to create shell scripts

Writing scripts

The shell programming environment with its shell scripts provides function similar to the TSO/E environment with its command lists (CLISTs) and the REstructured eXtended eXecutor (REXX) execs. The CLIST language is a high-level interpreter language that lets you work efficiently with TSO/E. A CLIST is a program, or command procedure, that performs a given task or group of tasks.

The REXX language is a high-level interpreter language that enables you to write programs in a clear and structured way. You can use the REXX language to write programs called REXX programs, or REXX execs, that perform given tasks or groups of tasks. REXX programs can run in any z/OS address space. You can run REXX programs that call z/OS UNIX services in TSO/E, in batch, in the shell environment, or from a C program.

In the z/OS shell, command processing is similar to command processing for CLISTs. You can write executable shell scripts (a sequence of shell commands stored in a text file) to perform many programming tasks. They can run in any dubbed z/OS address space. They can be run interactively, using cron, or using BPXBATCH. With its commands and utilities, the shell provides a rich programming environment.

Performance improvement: To improve performance when running shell scripts, add to the export statement in /etc/profile or $HOME/.profile:

```
_BPX_SPAWN_SCRIPT=YES.
_BPX_SHAREAS=YES
```

15.13 Shell script syntax

- Composed of shell commands
- '**while**' loops
- '**for**' loops
- '**do**' ... '**done**'
- '**if**' ... '**elif**' ... '**else**' ... '**fi**'
- '**test**' for conditions, e.g.:

```
if
   test -d $1
then
   echo "$1 is a directory"
fi
```

Figure 15-13 Typical shell script code

Shell script example code

The shell script is composed of different types of code. In a script file, you can use z/OS UNIX shell commands, flow control constructions like if-then-else, variables, environment settings and so on.

If you are familiar with programming languages, you will not have problems writing your first shell script.

If you are setting environment variables through a shell script, consider the following environment settings:

Any variables set in a shell script are set only while the script is running and do not affect the shell that invoked the shell script (unless the script is sourced by running it with the **.dot** command).

To run a shell script in your current environment without creating a new process, use the **.dot** command. You could run the compile shell script this way:

. script-file-name

You can improve shell script performance by setting the _BPX_SPAWN_SCRIPT environment variable to a value of YES.

15.14 BPXBATCH enhancements

- Limitations in BPXBATCH input and output make it more difficult to create and maintain BPXBATCH jobs compared to other batch utilities
- Enhance BPXBATCH to allow far greater parameter input and to allow output to be directed to traditional MVS data sets
- In z/OS V1R8 you are now able to:
  - Pass up to 65,536 characters of parameter data supplied as input to BPXBATCH - (100 previously)
  - Allow both STDOUT and STDERR to specify traditional MVS data sets to be the target of the output of the BPXBATCH jobs
- Available for z/OS V1R5, V1R6 and V1R7 with a PTF for APAR OA11699

Figure 15-14 BPXBATCH enhancements with z/OS V1R8

BPXBATCH limitations

Currently, limitations on the input and output capabilities of BPXBATCH jobs make it difficult to use and maintain. For example, allowing only 100 characters of input parameter data to be specified makes it difficult to invoke some z/OS UNIX applications that require long path names or input data. Dealing with output from the jobs in z/OS UNIX files requires special processing that is not required for normal batch jobs.

BPXBATCH enhancements in z/OS V1R8

These problems are solved by enhancing BPXBATCH to support greater parameter input data and to allow output to be directed to traditional MVS data sets.

In z/OS V1R8 you are now able to:

- Pass up to 65,536 characters of parameter data supplied as input to BPXBATCH.
- Allow both STDOUT and STDERR to specify traditional MVS data sets to be the target of the output of the BPXBATCH jobs.

Note: The enhancements are available for z/OS V1R5, V1R6 and V1R7 with a PTF for APAR OA11699.

15.15 BPXBATCH implementation

- A new STDPARM DD
  - Points to a MVS data set or UNIX file that contains the parameter data
  - The MVS data sets that can be used include:
    - SYSIN
    - Sequential
    - PDS/PDSE members
  - Specification of the STDPARM DD overrides:
    - Usage of PARM= on the EXEC PGM= JCL statement
- BPXBATCH TSO command enhanced to allow up to 32,754 length input string

Figure 15-15 z/OS V1R8 BPXBATCH implementation changes

STDPARM parameter

Parameters to BPXBATCH can also be supplied via the STDPARM DD up to a limit of 65,536 characters. When the STDPARM DD is allocated BPXBATCH will use the data found in the z/OS UNIX file or MVS data set associated with this DD rather than what is found on the parameter string or in the STDIN DD. The informational message BPXM079I will be displayed indicating that this is occurring, as a warning to the user.

The STDPARM DD will allow either a z/OS UNIX file, or an MVS SYSIN, PDS or PDSE member, or a sequential data set.

The default is to use the parameter string specified on the TSO command line or in the PARM= parameter of the JCL EXEC statement. If the STDPARM ddname is defined, BPXBATCH uses the data found in the specified file rather than what is found in the parameter string or in the STDIN ddname.

Currently, BPXBATCH supports only up to 100 characters of parameter data when invoked from JCL and 500 characters when invoked from TSO. With z/OS V1R8, BPXBATCH now supports up to 65,536 characters of parameter data.

This is supported via a new STDPARM DD that points to an MVS data set or UNIX file that contains the parameter data. The MVS data sets that can be used include SYSIN, sequential, or PDS/PDSE members. Specification of the STDPARM DD overrides the usage of PARM= on the EXEC PGM= JCL statement.

BPXBATCH TSO command

For the BPXBATCH TSO command, up to 32,754 characters are supported in the parameter string specified on the command invocation.

15.16 BPXBATCH summary

	STDPARM	STDOUT/ STDERR
TSO	Allocating STDPARM in TSO	Allocating STDOUT/ STDERR in TSO
JCL	Allocating STDPARM in JCL	Allocating STDOUT/ STDERR in JCL

```
//STDPARM DD DSN=TURBO.PARM.LIBRARY(P1),DISP=SHR
//STDPARM  DD * SH echo "Hello, world!"
```

Figure 15-16 Using TSO and JCL with STDPARM, STDOUT, and STDERR

Ways to use BPXBATCH with STDPARM

You can define the STDPARM parameter file by using one of the following:

- The TSO/E ALLOCATE command

  For example: The parameter data to be passed to BPXBATCH resides in the MVS sequential data set TURBO.ABC.PARMS, as follows:

  ```
  ALLOCATE DDNAME(STDPARM) DSN('TURBO.ABC.PARMS') SHR
  ```

- A JCL DD statement

  To identify a z/OS UNIX file, use the PATH operand and specify PATHOPTS=ORDONLY. For example: The parameter data resides in the z/OS UNIX file /u/turbo/abc.parms.

  ```
  //STDPARM DD PATH='/u/turbo/abc.parms',PATHOPTS=ORDONLY
  ```

  For example: The BPXBATCH parameter data resides in member P1 of the MVS PDSE TURBO.PARM.LIBRARY.

  ```
  //STDPARM DD DSN=TURBO.PARM.LIBRARY(P1),DISP=SHR
  ```

- A JCL in-stream data set

  The BPXBATCH parameter data immediately follows the STDPARM DD statement. Trailing blanks are truncated for in-stream data sets, but not for other data sets.

  For example: The following invokes the echo shell command:

  ```
  //STDPARM  DD * SH echo "Hello, world!"
  ```

BPXBATCH with STDOUT and STDERR

The TSO/E ALLOCATE command, using the ddnames STDOUT, and STDERR can be used as follows:

The following command allocates the MVS sequential data set TURBO.MYOUTPUT to the STDOUT ddname:

```
ALLOCATE DDNAME(STDOUT) DSNAME('TURBO.MYOUTPUT') VOLUME(volser) DSORG(PS)
SPACE(10) TRACKS RECFM(F,B) LRECL(512) NEW KEEP
```

A JCL DD statement, using the ddnames STDOUT, and STDERR can be used as follows:

The following JCL allocates member M1 of a new PDSE TURBO.MYOUTPUT.LIBRARY to the STDOUT ddname and directs STDERR output to SYSOUT:

```
//STDOUT   DD DSNAME=TURBO.MYOUTPUT.LIBRARY(M1),DISP=(NEW,KEEP),
//         DSNTYPE=LIBRARY, SPACE=(TRK,(5,1,1)),UNIT=3390,VOL=SER=volser,
//         RECFM=FB,LRECL=80
//STDERR   DD SYSOUT=*
```

15.17 TSO/E ALLOCATE command for STDPARM

- ALLOCATE DDNAME(STDPARM) DSN('RICH.PARMS') SHR
  - In this case, STDPARM is allocated to the data set 'RICH.PARMS' with DISP set to share
- Contents for STDPARM could look something like this:
  - SH mkdir -m 755 /u/rich/ims
- From the TSO command line issue:
  - BPXBATCH
    - Directory ims is created

Command examples

```
ALLOCATE DDNAME(STDOUT) DSNAME('BPX.MYOUTPUT') VOLUME(volser) DSORG(PS)
SPACE(10) TRACKS RECFM(F,B) LRECL(512) NEW KEEP

ALLOCATE DDNAME(STDERR) DSNAME('BPX.MYOUTPUT1') VOLUME(volser) DSORG(PS)
SPACE(10) TRACKS RECFM(F,B) LRECL(512) NEW KEEP
```

Figure 15-17 Using the TSO/E ALLOCATE command for STDPARM

Using BPXBATCH from TSO/E

Prior to invoking BPXBATCH, you can allocate any of the resources discussed earlier, using the TSO/E ALLOCATE command with the following ddnames:

```
STDIN, STDOUT, STDERR, STDENV, and STDPARM
```

You can invoke BPXBATCH under TSO/E as follows:

```
BPXBATCH SH|PGM program_name [arg1...argN]
```

where:

When `SH` is specified, program_name is the name of a shell command or a file containing a shell script. `SH` starts a login shell that processes your .profile before running a shell command or shell script. `SH` is the default; therefore, you can allocate a file containing a shell script to the STDIN ddname, invoke BPXBATCH without any parameters, and the shell script will be invoked.

When `PGM` is specified, program_name is the name of an executable file or a REXX exec that is stored in a z/OS UNIX file. Inadvertent use of a shell script with `PGM` may result in a process that will not end as expected, and will require use of the **`kill -9 pid`** command to force termination.

You can invoke BPXBATCH without any parameters on the command line and, instead, place the parameter data in a file or data set defined by STDPARM.

15.18 STDERR and STDOUT as MVS data sets

- STDOUT, STDERR DDs supported as MVS data sets
  - Allows output to be directed to SYSOUT, sequential or PDS/PDSE members
- Data sets should have an LRECL that allows all lines of output to fit or truncation will occur
- Only basic format sequential data sets are supported
  - Extended format and large format sequential data sets are not supported
- Previously, STDENV could only be represented by a SYSIN, sequential or PDS data set
  - This limitation is now changed to allow PDSEs

Figure 15-18 STDOUT and STDERR as MVS data sets

Use as MVS data sets

The STDOUT and STDERR DDs were always used previously as z/OS UNIX files. This limitation is changed so that these DDs can be represented by MVS data sets. Previously, STDENV could only be represented by a SYSIN, sequential, or PDS data set. This limitation is now changed to allow PDSEs.

Implementation rules

If you define an MVS data set for stdout or stderr it must follow the following rules:

- It must be a basic format sequential data set, a partitioned data set (PDS) member, a partitioned data set extended (PDSE) member, or SYSOUT.

  Extended format and large format sequential data sets are not supported.
- The data set must have a non-zero logical record length (LRECL) and a defined record format (RECFM); otherwise, BPXBATCH fails with the error message BPXM012I indicating an open failure for the affected ddname.
- If the LRECL of the target STDOUT or STDERR data set is not large enough to hold a line of output, the data is truncated and message BPXM080I is issued. This can happen for both fixed and variable blocked data sets. For variable blocked data sets, the first four bytes of each record, record segment, or block make up a descriptor word containing control information. You must allow for these additional 4 bytes in the specified LRECL if you intend to avoid truncation of the output to the STDOUT and STDERR DDs.

- If you use two members of the same partitioned data set for the STDOUT and STDERR ddnames, then you must use a PDSE (not a PDS). Using a PDS instead of a PDSE can result in a 213 ABEND (and, if running in a batch job, an abnormal end for the job step) or the output does not appear in the members as expected.

15.19 BPXBATCH sample job

```
//LUTZBPX JOB (ITSO,TXX,T870270),LUTZ,MSGCLASS=X,
//            MSGLEVEL=(1,1),REGION=8M,
//            NOTIFY=&SYSUID,CLASS=D
//*=========================================================
//* STDOUT and STDERR pointing to SYSOUT
//*=========================================================
//BPXBATCH EXEC PGM=BPXBATCH,PARM='PGM /u/rogers/programx'
//STDOUT   DD   SYSOUT=D,DCB=(LRECL=133,RECFM=F,DSORG=PS)
//STDERR   DD   SYSOUT=D,DCB=(LRECL=133,RECFM=F,DSORG=PS)
//*=========================================================
//* STDOUT and STDERR pointing to convential MVS data sets
//*=========================================================
//BPXBATCH EXEC PGM=BPXBATCH,PARM='SH ls -la /'
//STDOUT   DD   DISP=SHR,DSN=ROGERS.STDOUT.LIST
//STDERR   DD   DISP=SHR,DSN=ROGERS.STDERR.LIST
//
```

- REGION keyword
- Out of space condition for STDOUT and STDERR

Figure 15-19 BPXBATCH sample job using MVS data sets

BPXBATCH job using PDSEs

This job example, shown in Figure 15-19, for STDOUT and STDERR has these data sets directed to members of an existing PDSE.

Note: If you wish to use two members of the same partitioned data set for STDOUT and STDERR output, then you must use a PDSE (not a PDS).

The JCL shown in the example is as follows:

- The program name to be run is /u/rogers/programx.
- STDOUT is to be written to the PDSE member ROGERS.STDOUT.LIST.
- STDERR is to be written to the PDSE member ROGERS.STDERR.LIST.
- STDIN defaults to /dev/null.

Note: BPXBATCH consumes more private storage for MVS data sets. Therefore, existing jobs may need larger region size. In addition, out-of-space conditions for either STDOUT or STDERR cause an ENOSPC error to the program. The out-of-space condition causes an EPIPE error to be returned to the program and the generation of a SIGPIPE to the program.

15.20 Child process created for MVS data sets

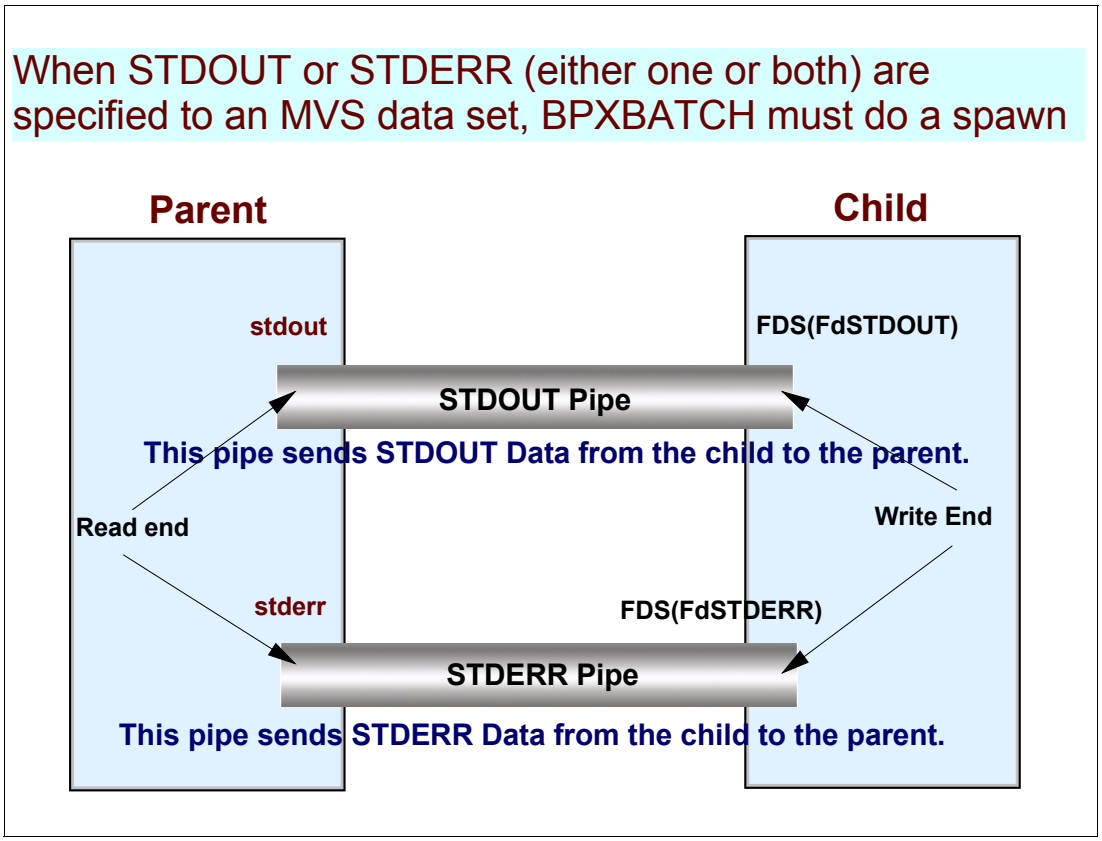


Figure 15-20 Flow of creation of MVS data set output

Creating MVS data sets for STDOUT and STDERR

A program creates a pipe with the pipe() function. A pipe typically sends data from one process to another; the two ends of a pipe can be used in a single program task. A pipe does not have a name in the file system, and it vanishes when the last process that is using it closes it.

When a pipe sends data from one process to another or back to itself, it forks processes. A pipe can be shared by a number of processes—for example, written to by three processes and read by seven.

When STDOUT or STDERR (either one or both) are specified to an MVS data set, BPXBATCH must do a spawn.

During the OpenNonHFS function, a pipe is created to connect the child process to the parent so that the data to be output in STDOUT or STDERR from the child process is sent over to the parent for processing.

When you specify an MVS data set for either the STDOUT or STDERR DDnames, a child process is created to run the target z/OS UNIX program. In some cases, the child process runs in a separate address space from the BPXBATCH job. In such cases, the job log messages for the child do not appear in the job log of the BPXBATCH job. To capture the job log messages of the child process, set the _BPXK_JOBLOG=STDERR environment variable. This will cause the job log messages of the child process to be written to the STDERR data set specified in the BPXBATCH job.

15.21 BPXBATCH utility

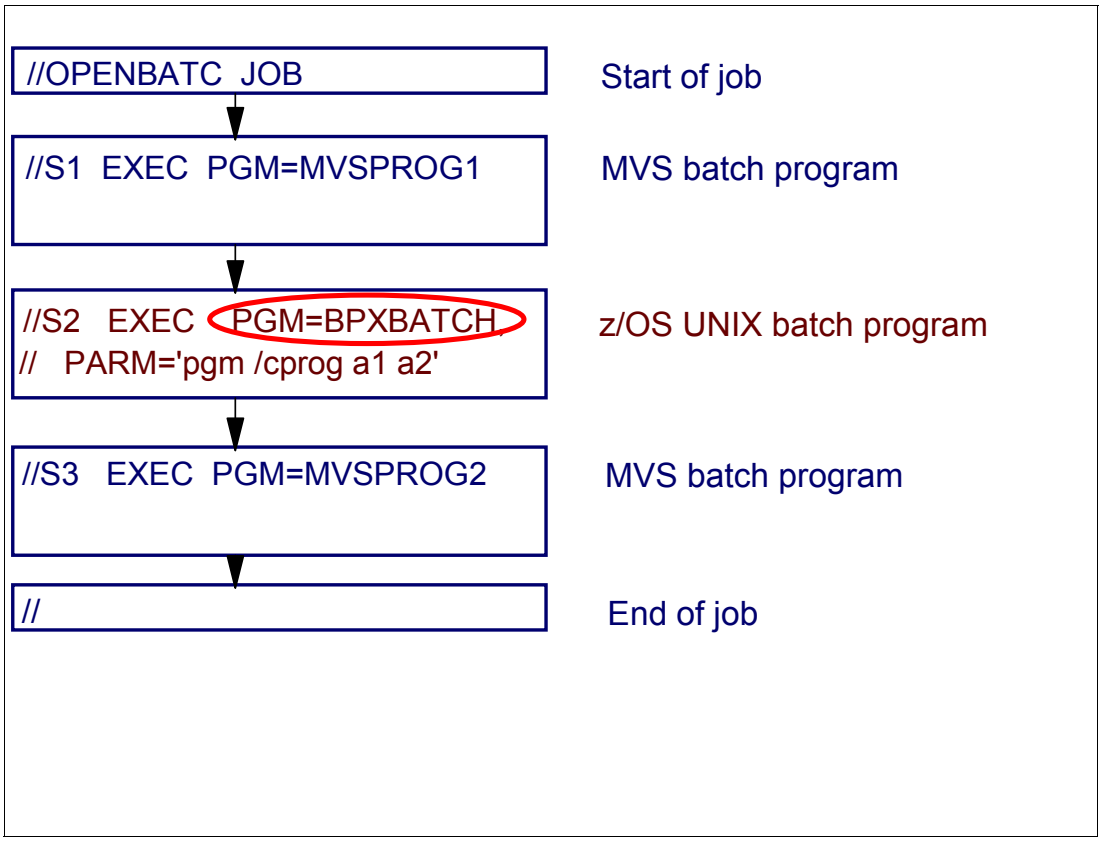


Figure 15-21 JCL used when using the BPXBATCH utility

BPXBATCH utility

You can create a job that uses BPXBATCH to run a shell command, a shell script, or an executable file, or a program as shown in Figure 15-21. When PGM is specified, program_name is the name of an executable file or a REXX exec that is stored in an HFS file.

BPXBATCH can also be invoked in JCL as a shell command:

```
//stepname EXEC PGM=BPXBATCH,PARM='SH!PGM program_name'
```

When SH is specified, program_name is the name of a shell command or a file containing a shell script. SH is the default; therefore, you can omit PARM= and supply the name of a shell script for STDIN, and it will be invoked.

BPXBATCH has logic in it to detect when it is running from a batch job. By default, BPXBATCH sets up the stdin, stdout, and stderr standard streams (files) and then calls the exec callable service to run the requested program. The exec service ends the current job step and creates a new job step to run the target program. Therefore, the target program does not run in the same job step as the BPXBATCH program; it runs in the new job step created by the exec service. In order for BPXBATCH to use the exec service to run the target program, all of the following must be true:

- BPXBATCH is the only program running on the job step task level.
- The _BPX_BATCH_SPAWN=YES environment variable is not specified.
- The STDOUT and STDERR ddnames are not allocated as MVS data sets.

For BPXBATCH, the default for stdin and stdout is /dev/null. The default for stderr is the same file defined for stdout. For example, if you define stdout to be /tmp/output1 and do not define stderr, then both printf() and perror() output is directed to /tmp/output1.

Note: BPXBATCH has logic in it to detect when it is run from JCL. If the BPXBATCH program is running as the only program on the job step task level, it sets up the stdin, stdout, and stderr and execs the requested program. If BPXBATCH is not running as the only program at the job step task level, the requested program will run as the second step of a JES batch address space from JCL in batch. If run from any other environment, the requested program will run in a WLM initiator (BPXAS) in the OMVS subsys category.

16

Performance, debugging, recovery, and tuning

This chapter describes the tuning necessary in a z/OS UNIX environment, and also shows the required and possible settings that improve system performance.

It presents the following considerations for performance tuning:

- The necessity of z/OS UNIX tuning
- Settings for z/OS UNIX with Workload Manager running in goal mode
- Additional settings to improve z/OS UNIX performance
- Where to get data to analyze the performance of z/OS UNIX

© Copyright IBM Corp. 2011. All rights reserved.

16.1 z/OS UNIX performance overview

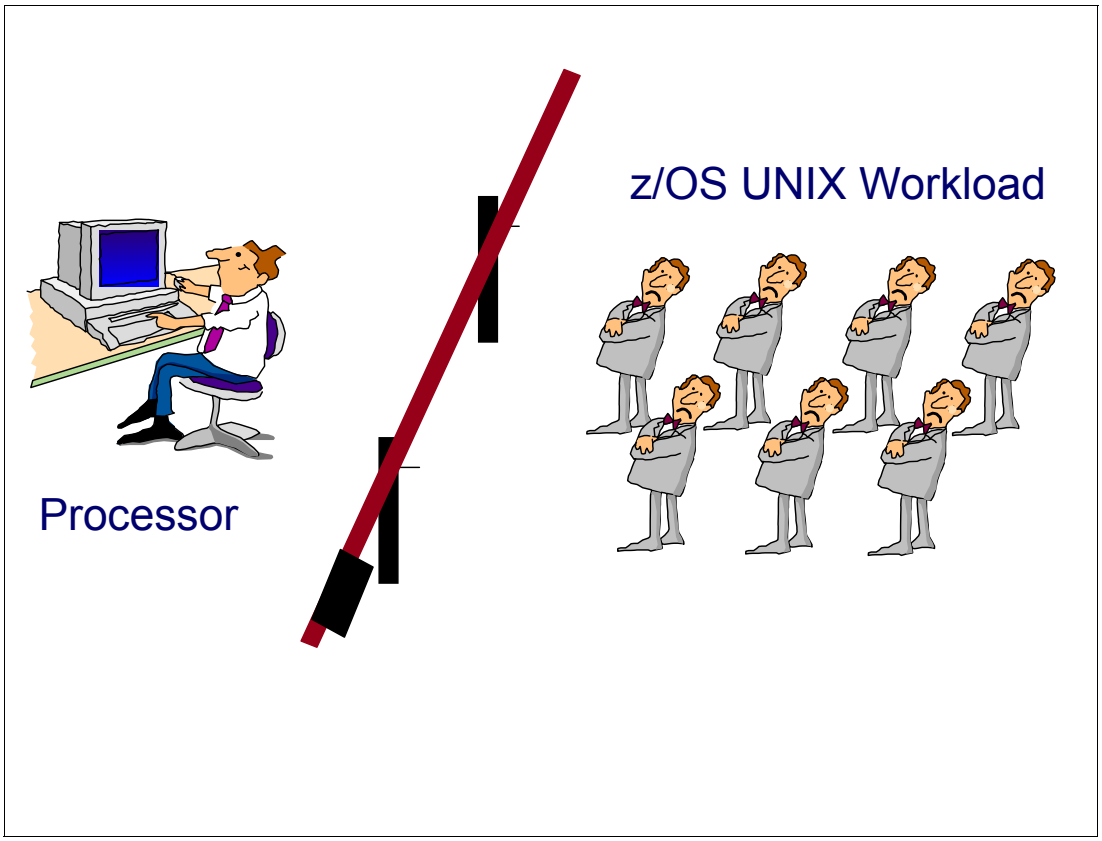


Figure 16-1 A z/OS UNIX performance overview

z/OS UNIX performance overview

It is both important and necessary to tune the z/OS UNIX environment to obtain an acceptable level of performance. Because z/OS UNIX is tightly integrated into the operating system, there are many z/OS UNIX tuning steps necessary to reach maximum interoperability between z/OS UNIX and traditional z/OS services.

As you run more UNIX-based products, you need a z/OS UNIX environment that performs well. Such products are, for instance, the Lotus Domino Server, TCP/IP, SAP R/3 and the newly announced Novell Network Services based on z/OS UNIX.

Some considerations regarding your z/OS UNIX workload are the following:

- Each z/OS UNIX interactive user will consume up to double the system resource of a TSO/E user.
- Every time a user tries to invoke the z/OS UNIX shell, RACF will have to deliver the security information out of the RACF database.
- The average active user will require three or more concurrently running processes that, without tuning, run in three or more concurrently active address spaces.

These are only a few of the considerations regarding performance impacts and how to prevent them. In most cases in our lab, tuning improved throughput by 2 to 3 times and the response time improved by 2 to 5 times. That represents a significant improvement, so let‘s describe how to accomplish this.

16.2 WLM in goal mode

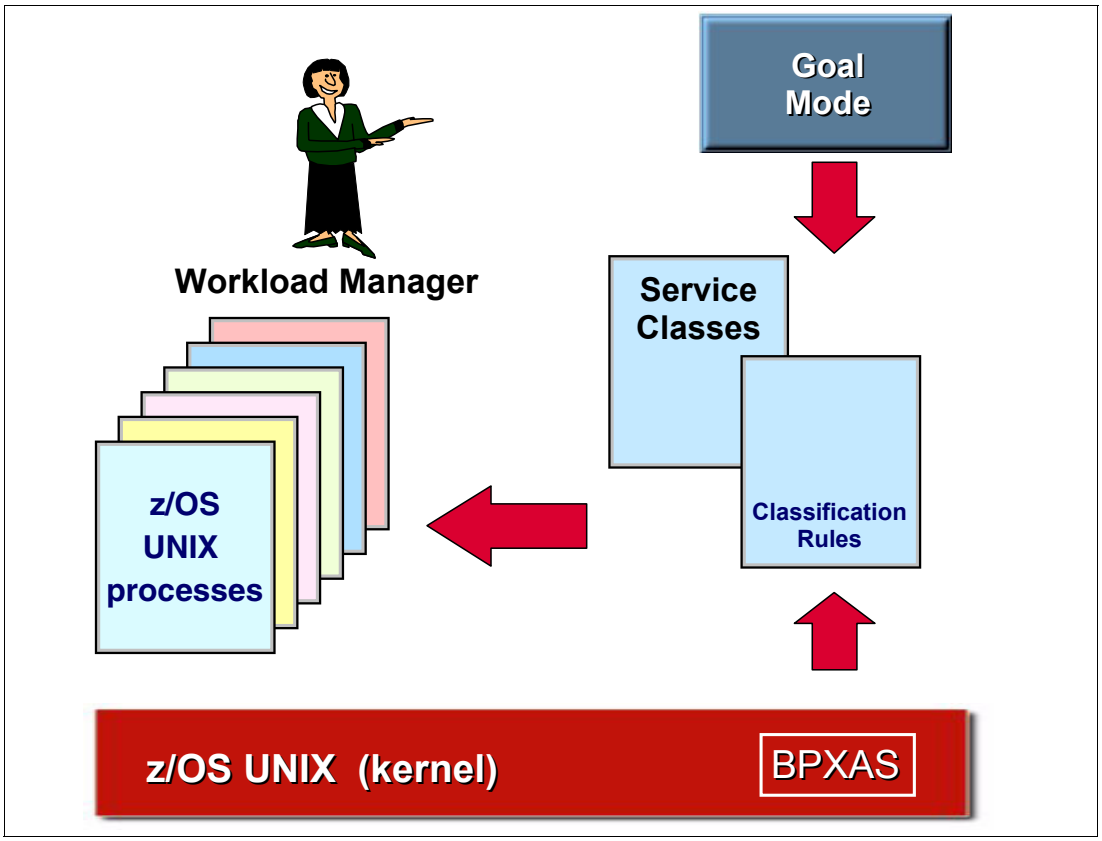


Figure 16-2 Using WLM in goal mode

Goal mode definitions for z/OS UNIX processes and daemons

Installations that run in goal mode can take the following steps to customize service policies in their Workload Manager service definition:

1. Define a workload for z/OS UNIX kernel work.
2. Define service classes for z/OS UNIX kernel work:
   a. Define a service class for forked children.
   b. Define a service class for startup processes.
3. Define classification rules:
   a. By default, put child processes (subsystem type OMVS) into the service class defined for forked children.
   b. Put the kernel (TRXNAME=OMVS) into a high-priority started task (subsystem type STC) service class.
   c. Put the initialization process BPXOINIT (TRXNAME=BPXOINIT) into a high-priority started task (subsystem type STC) service class.
   d. Startup processes that are forked by the initialization process, BPXOINIT, fall under SUBSYS=OMVS.
   e. Other forked child processes (under subsystem type OMVS) can be assigned to different service classes.
   f. Put the DFSMS buffer manager SYSBMAS (TRXNAME=SYSBMAS) into a high-priority started task (subsystem type STC) service class.

16.3 Defining service classes

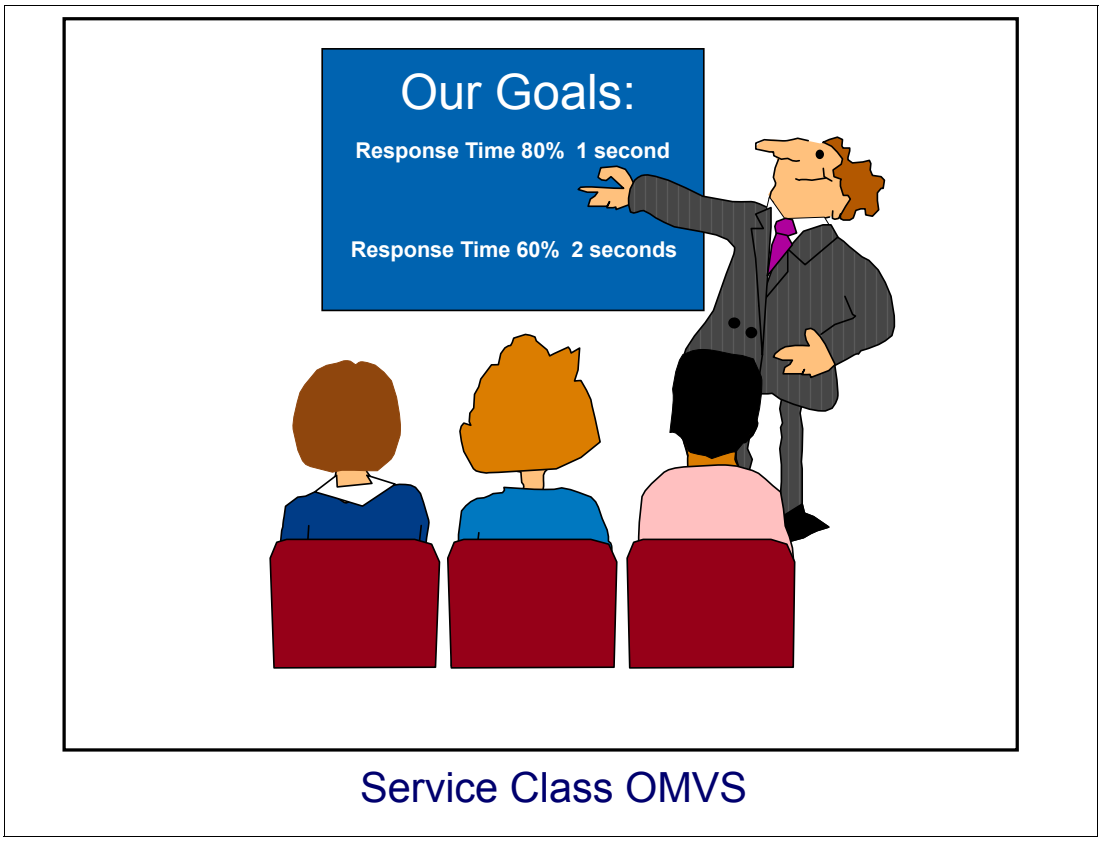


Figure 16-3 Defining service classes for OMVS address spaces

Defining service classes

Define a service class for forked child address spaces. This service class should normally have three performance periods, because it must support all types of kernel work, from short interactive commands to long-running background work. Following is a sample service class for forked children. Change the values as appropriate for the corresponding installations.

Table 16-1 Service class OMVS: Base goals

#	Duration	Importance	Goal Description
1	2000	2	Response Time 80% 1 second
2	4000	3	Response Time 60% 2 seconds
3		5	Execution Velocity of 10

Also define a service class for daemons. This service class should normally have only one period with a velocity goal higher than the velocity goals of other forked children.

Table 16-2 Service class OMVSKERN: Base goals

#	Duration	Importance	Goal Description
1		1	Execution Velocity of 40

You may want to define other service classes for z/OS UNIX kernel work for special users.

16.4 Workload Manager service classes

Service Class OMVS -- OMVS Forked Children

Base goal:

#	Duration	Imp	Goal Description
1	2000	2	Response Time 80% 1 second
2	4000	3	Response Time 60% 2 seconds
3		5	Execution velocity of 10

Service Class OMVSKERN -- OMVS Init & Other Daemons

Base goal:

#	Duration	Imp	Goal Description
1		1	Execution velocity of 40

Figure 16-4 Service class definitions for OMVS processes

Service class definitions

When using Workload Manager in goal mode, the z/OS workloads must be defined to Workload Manager.

Define a service class called OMVS that will include all the forked or spawned child address spaces. You should define three performance periods for this class because it must support all types of z/OS work; from short interactive commands to long running background work.

Define a separate service class for the OMVSKERN userid that will include the initialization process and other daemons started at initialization time by forks from this process. This service class should have only one period with a velocity goal higher than the goals for the forked children (OMVS service class).

The service classes you define to Workload Manager are similar to the definitions in the IEAIPS member. Performance goals for a service class can be defined as either response time, velocity, or discretionary. You can use either response goals or velocity for z/OS UNIX work.

An installation may choose to have other service classes defined for special z/OS UNIX users in addition to the one for OMVSKERN. A good example might be daemons such as INETD or TELNETD. These daemons run under the OMVSKERN ID by default, but it is possible to assign different RACF user IDs to the daemons, which can then be separately controlled.

16.5 Subsystem type panel

```
_  Subsystem-Type  View  Notes  Options  Help
 ------------------------------------------------------------------------
                   Subsystem Type Selection List for Rules     Row 1 to 15 of 15
 Command ===> ____________________________________________________________

 Action Codes: 1=Create, 2=Copy, 3=Modify, 4=Browse, 5=Print, 6=Delete,
               /=Menu Bar

                                                      ------Class-------
 Action  Type      Description                        Service   Report
   __    ASCH      Use Modify to enter YOUR rules               OTHER
   __    CB        WebSphere App Server               WASDF     OTHER
   __    CICS      CICS SERVER                        CICSDFLT  OTHER
   __    DB2       Use Modify to enter YOUR rules     SCDB2STP
   __    DDF       DRDA Stored Procedures             SCDB2STP
   __    IMS       Use Modify to enter YOUR rules     IMS       IMS
   __    IWEB      Webserver Subsystem Type           WEBSRVC   OTHER
   __    JES       JES TYPE                           VEL50     OTHER
   __    LSFM      Use Modify to enter YOUR rules               OTHER
   __    MQ        Workflow Request Classification    DEF_SC
   __    OMVS                                         VEL70     OTHER
   __    SAP       WLM definition for SAP R/3 4.5B    SAPAS     SAP
   __    SOM       Use Modify to enter YOUR rules
   __    STC                                          SYSSTC    OTHER
   __    TSO       Use Modify to enter YOUR rules     ITSOTSU   RTSO
 ******************************** Bottom of data ********************************
```

Figure 16-5 Selecting the OMVS subsystem to create classification rules

OMVS subsystem

When you choose the Classification Rules option from the Definition Menu, you go to the Subsystem Type Selection List for Rules panel. This panel initially contains the reserved names of the IBM-supplied subsystem types.

Figure 16-5 shows the IBM-supplied subsystem types that workload management supports. The ISPF application provides these subsystem types as a selection list on the classification rules panel. You can add any additional subsystem type on the same panel if it supports workload management.

Classification of work depends on having the rules defined for the correct subsystem type; for z/OS UNIX the subsystem type is OMVS.

Although you may want to change the description of the subsystem types, you should not delete any of the entries provided by IBM unless your installation does not plan to ever use them. If your installation later does need to use them, they can be manually added at that time.

16.6 WLM work qualifiers

AI	Accounting Information	PR	Procedure Name
CI	Correlation Information	PRI	Priority
CN	Collection Name	PX	Sysplex Name
CT	Connection Type	SE	Scheduling Environment
CTG	Connection Type Group	SI	Subsystem Instance
LU	LU Name	SIG	Subsystem Instance Group
LUG	LU Name Group	SPM	Subsystem Parameter
NET	Net ID	SSC	Subsystem Collection
NETG	Net ID Group	SY	Sysname
PC	Process Name	SYG	Sysname Group
PF	Perform	TC	Transaction Class
PFG	Perform Group	TCG	Transaction Class Group
PK	Package Name	TN	Transaction Name
PKG	Package Name Group	TNG	Transaction Name Group
PN	Plan Name	UI	Userid
PNG	Plan Name Group	UIG	Userid Group

Figure 16-6 The WLM work qualifiers used to classify work

Work qualifiers

A *work qualifier* is an attribute of incoming work. Figure 16-6 shows a list of the work qualifiers that can be used in defining classification rules that determine the service class a unit of work is assigned.

Each subsystem has its own list of work qualifiers that it supports.

Work qualifiers depend on the subsystem that first receives the work request. When you are defining the rules, start with the service classes you have defined, and look at the type of work they represent. Determine which subsystem type or types process the work in each service class. Then understand which work qualifiers your installation could use for setting up the rules. It may be that your installation follows certain naming conventions for the qualifiers for accounting purposes. These naming conventions could help you to filter work into service classes. Also, understand which work qualifiers are available for each subsystem type. You can then decide which qualifiers you can use for each service class.

16.7 OMVS work qualifiers

```
  Subsystem-Type  Xref  Notes  Options  Help
 ------------------------------------------------------------------------
                  Modify Rules for the Subsystem Type       Row 1 to 9 of 9
 Command ===> ____________________________________________ SCROLL ===> PAGE

 Subsystem Type . : OMVS        Fold qualifier names?   Y  (Y or N)
 Description  . . . ______________________________________

 Action codes:   A=After     C=Copy        M=Move     I=Insert rule
                 B=Before    D=Delete row  R=Repeat   IS=Insert Sub-rule
                                                        More ===>
                 Qualifier Selection  Row 1 to 8 of 8   s--------
 Action          Command ===> ______________________________  Report
                                                             OTHER
  ____   1       Select a type with "/"                      RCOMVS
  ____   1                                                   ________
  ____   1       Sel  Name  Description                      ________
  ____   1        _   AI    Accounting Information           RCGIAN
  ____   1        _   PX    Sysplex Name                     RCANNA
  ____   1        _   SY    Sysname                          ITSOFN
  ____   1        _   SYG   Sysname Group                    ERWCTG1
  ____   1        _   TN    Transaction Name                 WAS
  ____   1        _   TNG   Transaction Name Group           ________
*********         _   UI    Userid                          **************
                  _   UIG   Userid Group
                ***************** Bottom of data ****************
```

Figure 16-7 The work qualifiers usable for OMVS

OMVS work qualifiers

Accounting data is normally inherited from the parent process of a z/OS UNIX System Services address space. In addition, when a daemon creates a process for another user, accounting data is taken from the WORKATTR of the RACF user profile. A user can also assign accounting data by setting the _BPX_ACCT_DATA environment variable or by passing accounting data on the interface to the _spawn service. For more information about z/OS UNIX System Services accounting information, see *z/OS UNIX System Services Planning*, GA22-7800.

Figure 16-7 shows the work qualifiers that can be used to set up the OMVS classification rules.

16.8 Defining classification rules

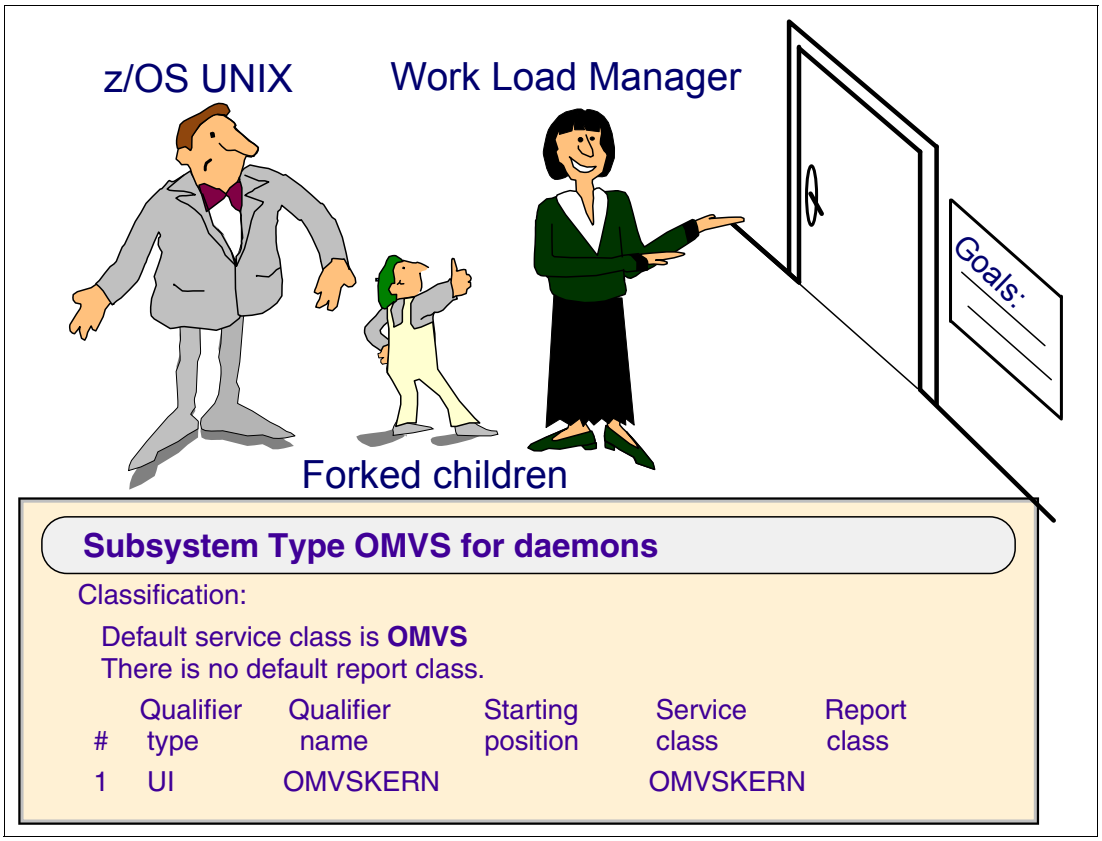


Figure 16-8 Defining classification rules for OMVS

Defining classification rules for OMVS

The workload in a z/OS system must be identified and classified in Workload Manager. The classification rules should specify a subsystem called OMVS, which will cover all forked/spawned child processes. Within this subsystem, the userid OMVSKERN should be defined to use a separate service class than the rest of the work in this subsystem. Other categories can be defined if you have separated daemons out into other service classes. Specify the classification rules needed to separate daemons (for example, inetd) from other forked children. The following is a sample classification for subsystem type OMVS.

- Subsystem Type OMVS
  - Classification:
    - Default service class is OMVS.
    - There is no default report class.

Table 16-3 Classification rules for subsystem type OMVS

#	Qualifier Type	Qualifier Name	Starting Position	Service Class	Report Class
1	UI	OMVSKERN		OMVSKERN	

If no action was taken, OMVS and BPXOINIT will run under the rules for subsystem type STC, which is typically defined to have high priority. If needed, it is possible to define an extra classification rule for subsystem type STC to ensure that the kernel, the initialization process BPXOINIT, and the DFSMS buffer manager SYSBMAS run as high-priority started tasks.

16.9 Classification rules

```
  Subsystem-Type  Xref  Notes  Options  Help
 ------------------------------------------------------------------------------
                 Modify Rules for the Subsystem Type        Row 1 to 4 of 4
 Command ===> ____________________________________________ SCROLL ===> PAGE

 Subsystem Type . : OMVS        Fold qualifier names?   Y  (Y or N)
 Description  . . . OMVS Address Spaces

 Action codes:  A=After    C=Copy         M=Move     I=Insert rule
                B=Before   D=Delete row   R=Repeat   IS=Insert Sub-rule
                                                          More ===>
          -------Qualifier-------------             -------Class--------
 Action    Type       Name     Start                Service     Report
                                          DEFAULTS: OMVS        ________
  ____  1  UI         REDADM   ___                  SAPAS       SAPASREP
  ____  1  TN         MIKE*    ___                  SYSSTC      SAMBA
  ____  1  TN         DB2OL*   ___                  DB2_____    DB2OLAP
  ____  1  UI__       OMVSKERN ___                  OMVSKERN    ________
****************************** BOTTOM OF DATA ******************************
```

Figure 16-9 Panel to define the classification rules

Classification rule panel

When the subsystem receives a work request, the system searches the classification rules for a matching qualifier and its service class or report class. Because a piece of work can have more than one work qualifier associated with it, it may match more than one classification rule. Therefore, the order in which you specify the classification rules determines which service classes are assigned.

A service class default is the service class that is assigned if no other classification rule matches for that subsystem type. If you want to assign any work in a subsystem type to a service class, then you must assign a default service class for that subsystem -- except for STC. You are not required to assign a default service class for STC, even if you assign started tasks to different service classes.

Optionally, you can assign a default report class for the subsystem type. If you want to assign work running in a subsystem to report classes, then you do not have to assign a default service class for that subsystem.

16.10 Classification rules for STC

- STC1 is a high priority started task service class

Subsystem Type STC

```
Classification:
 Default service class is STC2
 There is no default report class.
      Qualifier      Qualifier     Starting     Service    Report
#     type             name        position     class      class
1     TN              OMVS                     STC1
1     TN              BPXOINIT                 STC1
1     TN              SYSBMAS                  STC1
1     TN              ZFS                      STC1
```

Figure 16-10 Panel for defining the STC service classes for OMVS

Classification rules for OMVS address spaces

Specify a classification rule for subsystem type STC to ensure that the OMVS, BPXOINIT, and SYSBMAS procedures run as high priority started tasks, as follows:

- Subsystem Type STC
  - Classification:
    - Default service class is STC2.
    - There is no default report class.

Table 16-4 Classification rules for subsystem type OMVS

#	Qualifier Type	Qualifier Name	Starting Position	Service Class	Report Class
1	UI	OMVS		STC1	
2	UI	BPXOINIT		STC1	
3	UI	SYSBMAS		STC1	
..	...	.....		.....	

16.11 Virtual lookaside facility (VLF)

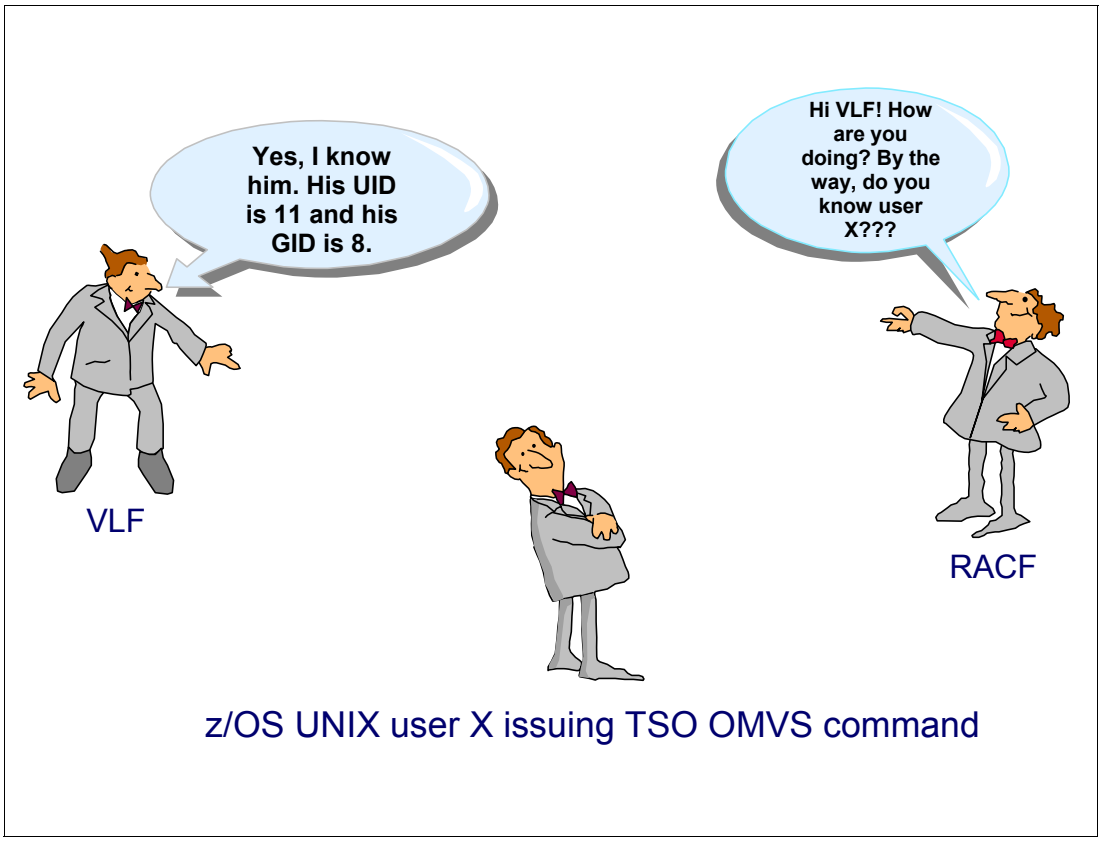


Figure 16-11 Using VLF to cache UIDs and GIDs

VLF for performance

To improve the performance of z/OS UNIX RACF support, you should add the Virtual Lookaside Facility (VLF) classes that control caching of the z/OS UNIX UID and GID information. The mapping of z/OS UNIX group identifiers (GIDs) to RACF group names and the mapping of z/OS UNIX user identifiers (UIDs) to RACF user IDs is implemented using VLF services to gain performance improvements.

To achieve these performance improvements, the two VLF classes IRRGMAP and IRRUMAP should be added to the COFVLFxx PARMLIB member. A COFVLFxx PARMLIB member including the z/OS UNIX information should look like this:

```
CLASS NAME(CSVLLA)         /* Class name for LLA                */
   EMAJ(LLA)               /* Major name for LLA                */
CLASS NAME(IRRUMAP)        /* z/OS UNIX RACF UMAP Table        */
   EMAJ(UMAP)              /* Enable Caching of z/OS UNIX UIDs */
CLASS NAME(IRRGMAP)        /* z/OS UNIX RACF GMAP Table        */
   EMAJ(GMAP)              /* Enable Caching of z/OS UNIX GIDs */
CLASS NAME(IRRGTS)         /* RACF GTS Table                    */
   EMAJ(GTS)               /* Enable caching of RACF GTS        */
CLASS NAME(IRRACEE)        /* RACF saved ACEEs                  */
   EMAJ(ACEE)              /* Enable caching of RACF ACEE       */
CLASS NAME(IRRSMAP)        /* z/OS UNIX Security Package       */
   EMAJ(SMAP)              /* Major Node SMAP                   */
```

The VLF member can be activated by starting VLF using the operator command:

```
START VLF,SUB=MSTR,NN=xx
```

where xx is the suffix of the corresponding COFVLF member.

If a user requests to invoke z/OS UNIX, the z/OS UNIX kernel asks RACF about the permission. RACF checks whether VLF is active. If it is active, it asks VLF about the data of the user that tries to logon. If the user has already been logged to z/OS UNIX since VLF became active, VLF should know about this user and provides the UID and GID to RACF. RACF passes the information to z/OS UNIX for processing.

However, if VLF is not active, or if the user tries to invoke z/OS UNIX for the first time since VLF became active, RACF has to start I/O to the RACF database to get the information.

VLF is able to collect this data in its data spaces if the following two classes were added to the COFVLFxx member in SYS1.PARMLIB:

- IRRUMAP
- IRRGMAP
- IRRSMAP

16.12 VLF for z/OS UNIX

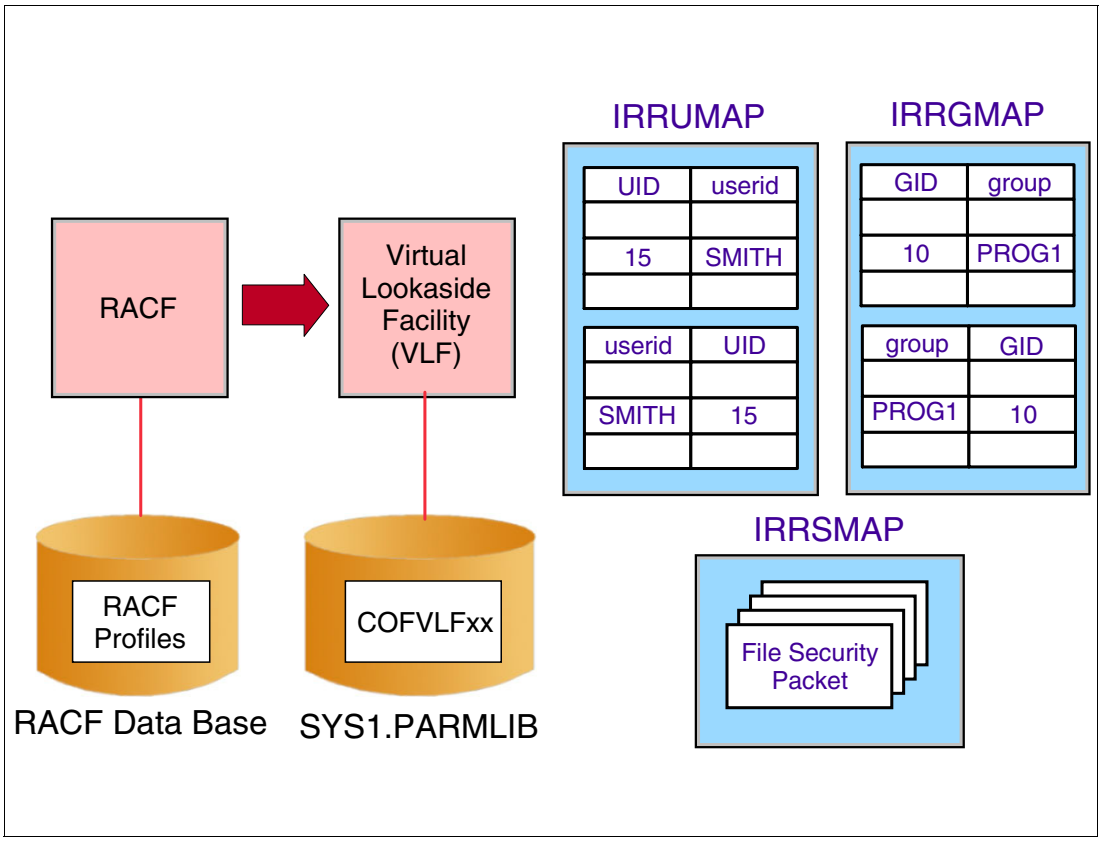


Figure 16-12 The VLF cache of UIDs and GIDs with RACF

VLF caching of UIDs and GIDs

z/OS UNIX uses the UID and the GID to identify users and groups while z/OS uses the userid and group name. Both identifications are defined in the RACF profiles in the RACF database. When RACF is called to do security processing for z/OS UNIX, RACF often needs to find which userid belongs to a UID, or which group belongs to a GID. This could cause excessive I/O to the RACF database. A solution is to build tables with UID-to-userid mapping and GID-to-group mapping and keep the tables in VLF data spaces called IRRUMAP and IRRGMAP.

Also, whenever a file is accessed by a user, RACF must check the permission bits, which are stored in a File Security Packet (FSP) on the same HFS data set as the file. As a user may access the same file many times, RACF can save time and I/O by storing File Security Packets for active files in a VLF dataspace called IRRSMAP. Virtual Lookaside Facility (VLF) is used to build the tables and keep them in processor storage. For this mechanism to work:

- VLF must be active.
- The IRRUMAP, IRRSMAP, and IRRGMAP classes must be defined by adding these VLF options to the COFVLFxx member of SYS1.PARMLIB:

```
CLASS NAME(IRRUMAP)
    EMAJ(UMAP)
CLASS NAME(IRRGMAP)
    EMAJ(GMAP)
CLASS NAME(IRRSMAP)
    EMAJ(SMAP)
```

16.13 COFVLFxx updates for z/OS UNIX

COFVLFxx

```
CLASS NAME(CSVLLA        /* class name for Library Lookaside  */
      EMAJ(LLA)          /* Major name for LIbrary Lookasid   */
CLASS NAME(IRRUMAP)      /* z/OS UNIX RACF UMAP Table         */
      EMAJ(UMAP)         /* Major name = UMAP                 */
CLASS NAME(IRRGMAP)      /* z/OS UNIX RACF GMAP TABLE         */
      EMAJ(GMAP)         /* Major name = GMAP                 */
CLASS NAME(IRRSMAP)      /* z/OS UNIX Security Packet         */
      EMAJ(SMAP)         /* Major name = SMAP                 */
CLASS NAME(IRRGTS)       /* Enable caching of RACF GTS        */
      EMAJ(GTS)          /*                                   */
CLASS NAME(IRRACEE)      /* Enable caching of RACF ACEE       */
      EMAJ(ACEE)         /*                                   */
```

Figure 16-13 The definitions required in the COFVLFxx PARMLIB member

COFVLFxx PARMLIB definitions

This example shows the definitions to add to the COFVLFxx member in SYS1.PARMLIB to support the RACF UMAP, GMAP, and SMAP tables. You are strongly urged to use IRRUMAP and IRRGMAP for any installation. Samples are also included in the member RACPARM in SYS1.SAMPLIB which is delivered with RACF.

It is also recommended for extra performance improvements that you cache the z/OS UNIX security packets by adding the IRRSMAP entries to COFVLFxx.

When working with the z/OS UNIX shell or ISHELL, some of the commands or functions can display output with either the UID or userid as the file owner. z/OS UNIX only knows about UIDs and GIDs. Whenever a userid or group name is displayed, it has been looked up in the RACF data base or the mapping tables to find the corresponding userid for a UID, or group for a GID. The end user does not have to be aware that such a mapping/conversion is done.

Something which causes a bit of confusion is which userid RACF will show when a file/directory is owned by a UID=0. In a system, there will be multiple superusers (defined with UID=0, or to the BPX.SUPERUSER class), and RACF can only pick one of these user IDs from the mapping table to show as the owner. It seems like without VLF active, RACF will pick the first superuser user ID, in alphabetical order, that has logged on. With VLF active, RACF will pick the first superuser userid that has logged on since VLF was started. This is not a problem; it just causes some confusion for people to see a file owned by one user ID one day, and another user ID another day. This situation happens only for users that share the same UID, which should only be the superusers.

16.14 AIM Stage 3

- RACF locates application identities - UIDs and GIDs
  - Using an alias index
  - Allows better authentication and authorization
    - From applications such as z/OS UNIX
    - Eliminates UNIXMAP class and VLF
- Deactivate the UNIXMAP class and remove VLF classes IRRUMAP and IRRGMAP
  - APAR OA02721 - Performance with AIM

Figure 16-14 Conversion of the RACF database to AIM

AIM for performance

You can convert your RACF database to stage 3 of application identity mapping (AIM) using the IRRIRA00 conversion utility. In stage 3, RACF locates application identities, such as UIDs and GIDs, for users and groups by using an alias index that is automatically maintained by RACF. This allows RACF to more efficiently handle authentication and authorization requests from applications such as z/OS UNIX than was possible using other methods, such as the UNIXMAP class and VLF.

You can improve RACF performance when looking up UIDs and GIDs by using virtual lookaside facility (VLF) and alias index entries. For more information on using them, see *z/OS Security Server RACF Security Administrator's Guide*, SA22-7683.

Using alias index entries allows you to use the RACF SEARCH command to determine which users are assigned a specified UID, and which groups are assigned a specified GID, as follows:

```
SEARCH CLASS(USER) UID(0)
SEARCH CLASS(GROUP) GID(100)
```

Migration to AIM stage 3

After migrating to AIM stage 3, some environments may observe increased times required for performing UID to username mapping using the Get_UMAP callable (IRRSUM00).

AIM stage 3 only uses ALIAS processing to locate the userid for a particular UID or GID. The VLF cache is not used. RACF Application Identity Mapping (AIM) stage 3 can cache the UID or GID in VLF with the corresponding userid or group name from ALIAS processing. Subsequently, the cache can be used rather than the ALIAS processing now used all the time.

The installation should define the IRRUMAP and IRRGMAP classes to VLF for use by AIM stage 3 processing.

16.15 Further tuning tips

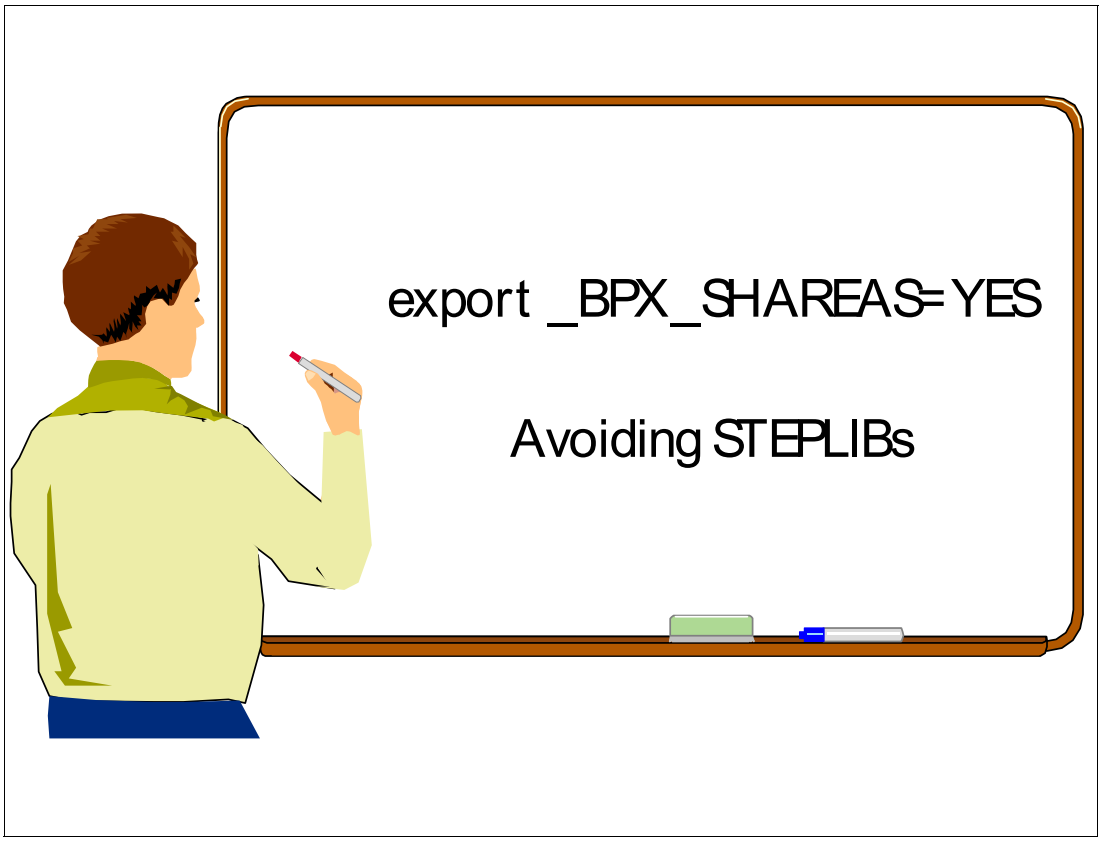


Figure 16-15 Additional tuning tips

Performance tuning tips

To improve z/OS UNIX shell performance, it is recommended that you do the following:

- There are two environment variables that can be used to improve the performance:
  - Set the _BPX_SHAREAS environment variable to YES or REUSE. This saves the overhead of a fork and exec by not creating its own address space for foreground shell processes.
  - Set the _BPX_SPAWN_SCRIPT environment variable to YES. This avoids having the shell invoke spawn after receiving ENOEXEC for an input shell script.
- To improve the performance for all shell users, the /etc/profile should contain the following settings:

```
export _BPX_SHAREAS=YES (or export _BPX_SHAREAS=REUSE)
export _BPX_SPAWN_SCRIPT=YES
```

Avoid STEPLIBs

To improve performance for users who log in to the shell with the OMVS command, do not place any STEPLIB or JOBLIB DD statements in logon procedures. Specify STEPLIB=none in the /etc/profile to avoid excessive searching of STEPLIB data sets. The section in the /etc/profile, that contains the STEPLIB statement, should look like the following:

```
if  -z "$STEPLIB"  &&; tty -s; then
    echo "- Improve performance by preventing the propagation of -"
    echo "- TSO/E or ISPF STEPLIBs                                -"
    export STEPLIB=none
    exec sh -L
fi
```

Storage consumption

Be aware of storage consumption. If the system is running in an LPAR or as a VM guest, the storage size should be at least 64 MB; however, having quite a bit more than this will not be harmful.

- ECSA storage used by z/OS UNIX is based on the following formula:

  (n * 150 bytes) + (m * 500 bytes)

  where n is the number of tasks using z/OS UNIX and m is the number of processes.

  So if your system supports 500 processes and 2000 threads, z/OS UNIX consumes 550 KB of ECSA storage.

- In addition to this:
  - WM uses some ECSA for each forked initiator.
  - The OMVS address space itself uses 20KB ECSA.
  - Spawn usage requires approximately 100 KB of ECSA.
  - Each process that has a STEPLIB that is propagated from parent to child or across an EXEC consumes about 200 bytes of ECSA.

Taking these points into consideration may prevent possible storage shortages.

16.16 zFS performance tuning

- zFS performance features
  - The IOEFSPRM data set provides tuning options
  - Output of the operator modify query commands provides feedback about the operation of zFS
    - f zfs,query,all ------ zfsadm query .....
- zFS performance optimization with cache sizes
  - Reduce I/O rates and path length
  - Hit ratios should be 80% or better
  - Over 90% gives good performance
- Monitor DASD performance

Figure 16-16 zFS performance tuning features

zFS performance

zFS performance is dependent on many factors. zFS provides performance information to help the administrator determine bottlenecks. The IOEFSPRM file contains many tuning options that can be adjusted. The output of the operator **`modify query`** commands provide feedback about the operation of zFS.

zFS and its caches

zFS performance can be optimized by tailoring the size of its caches to reduce I/O rates and path length. It is also important to monitor DASD performance to ensure there are no volumes or channels that are pushed beyond their capacity. The following describes some of the considerations when tuning zFS performance.

In general you should have hit ratios of at least 80% or more, over 90% usually gives good performance. However, the desired hit ratio is very much workload-dependent. For example, a zFS file system exported exclusively to SMB clients by using the SMB server would likely have a low hit ratio since the SMB client and the SMB server cache data, making the zFS cache achieve a low hit ratio in this case. That is expected and is not considered a problem.

16.17 zFS cache

- With **V1R4**, the log file cache is in a dataspace
  - Default - (64 MB)
  - Frees up space for caches in ZFS address space

Cache	Dataspace?	How many?
User file	Yes	32
Log file	Yes	1
Metaback	Yes	1
Metadata	No	
Vnode	No	
Transaction	No	

Figure 16-17 zFS cache

zFS cache specifications

Following are the specified caches used by zFS:

User file cache The user file cache is used to cache all "regular" files. It caches any file no matter what its size and performs write-behind and asynchronous read-ahead for files. It performs I/O for all files that are 7K or larger. For files smaller than 7K, I/O is normally performed through the metadata cache. The user file cache is allocated in data spaces. Its size by default is 256 MB and can be tailored to meet your performance needs based on your overall system memory. The maximum size is 65536 MB (which is 64 GB).

Metadata The metadata cache is used to contain all file system metadata, which includes all directory contents, file status information (such as atime, mtime, size, permission bits, and so on), file system structures, and it also caches data for files smaller than 7K. The metadata cache is stored in the primary address space and its default size is 32 MB. Since the metadata cache only contains metadata and small files, it normally does not need to be nearly as large as the user file cache.

Log file Every zFS aggregate contains a log file used to record transactions describing changes to the file system structure. This log file is, by default, 1% of the aggregate size but is tailorable by the administrator on the **`ioeagfmt`** command. Usually, 1% is sufficient for most aggregates. Especially large aggregates might need less than 1%, while very small

aggregates might need more than 1% if a high degree of parallel update activity occurs for the aggregate. The log file cache is a pool of 8K buffers used to contain log file updates. Log file buffers are always written asynchronously to disk and normally only need to be waited upon when the log is becoming full, or if a file is being fsync'ed. The log file cache is stored in a data space and its default is 64 MB. The log file cache is grown dynamically by adding one 8K buffer for each attached aggregate. This ensures that each aggregate always has one 8K buffer to use to record its most recent changes to file system metadata.

Transaction

Every change to zFS file system metadata is bounded by a transaction describing its changes by using records written to the log file. The transaction cache is a cache of data structures representing transactions. The transaction cache is stored in the zFS primary address space and its default is 2000. zFS dynamically increases the size of this cache based on the number of concurrent pending transactions (transactions that have not been fully committed to disk) in the zFS file system.

Vnode

Every object in the zFS file system is represented by a data structure called a *vnode* in memory. zFS keeps a cache of these and recycles these vnodes in an LRU fashion. The vnode cache is stored in the zFS primary address space, and the default number of vnodes is 16384.

Metaback

Metadata backing cache is an extension to the metadata cache. The size of this extension is controlled by the metaback_cache_size configuration option. The backing cache is stored in a dataspace and is used only to avoid metadata reads from disk. All metadata updates and write I/O are performed from the primary metadata cache.

16.18 zFS cache locations

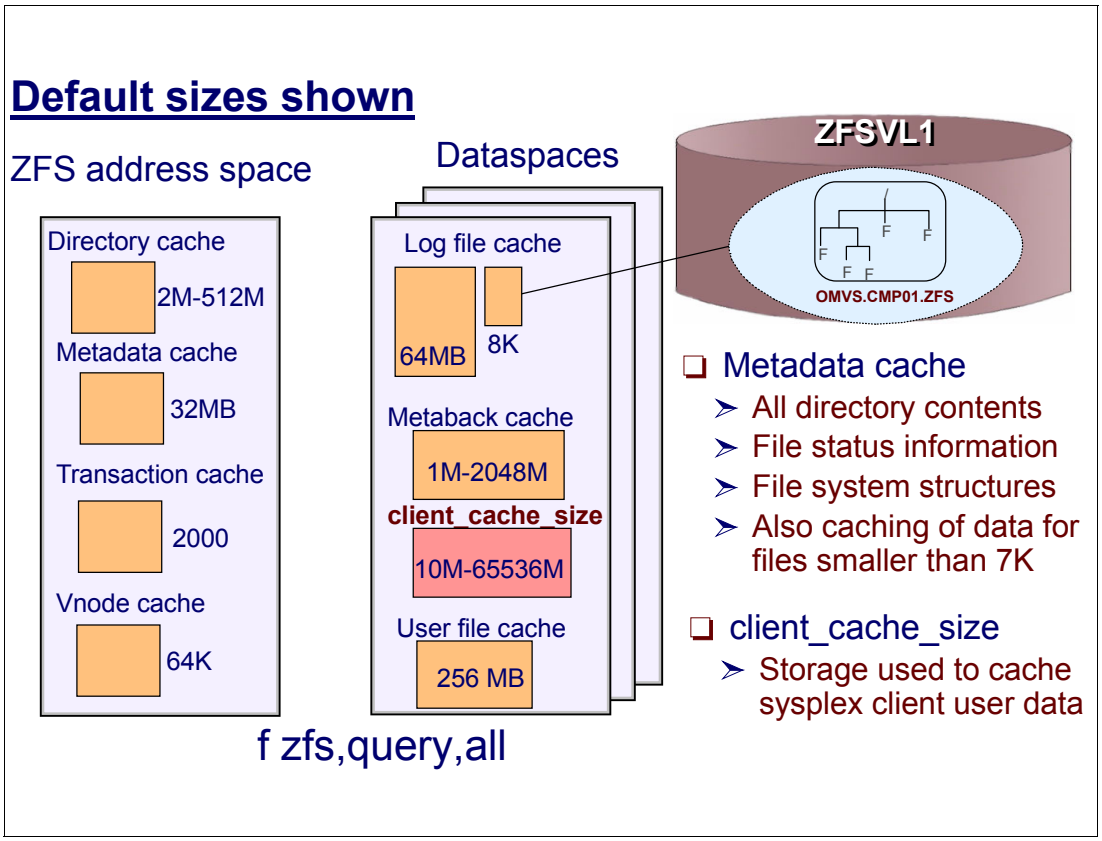


Figure 16-18 zFS caches in dataspaces and zFS address space

zFS cache locations

Currently, the ZFS address space is restricted to 2 GB of total storage. Therefore, the total storage size for all of the caches that reside in the ZFS address space must be less than 2 GB. Since storage is needed in addition to the ZFS address space caches to process file requests and for products zFS might use, generally you should restrict total ZFS address space cache storage to approximately 1.5 GB.

The zFS operator F ZFS,QUERY,ALL command shows total zFS storage allocated, which includes storage allocated for all the caches and everything else zFS might need. zFS terminates during initialization if it cannot obtain all the storage for the caches as directed by the IOEFSPRM file. The ZFS address space caches include the metadata cache, the transaction cache and the vnode cache.

The user data cache, the log file cache, and the metadata backing cache reside in dataspaces and do not use ZFS address space storage.

Metadata cache

The metadata cache is used to contain all file system metadata, which includes all directory contents, file status information (such as atime, mtime, size, permission bits, and so on), file system structures, and it also caches data for files smaller than 7K. Because of the potential size of the metadata cache for very large file systems, the backing cache is stored in a dataspace and is used only to avoid metadata reads from disk. All metadata updates and write I/O are performed from the primary metadata cache.

16.19 Metadata backing cache

- A backing cache contains an extension to the metadata cache and resides in a dataspace - Specify in IOEFSPRM file
  - metaback_cache_size=64M,fixed
    - Values allowed: 1 MB to 2048 MB
- Used as a "paging" area for metadata
- Allows a larger meta cache for workloads that need large amounts of metadata
- Only needed if meta cache is constrained

Figure 16-19 Metadata backing cache used for metadata cache constraints

Metadata backing cache

The metaback_cache_size specifies the size of the backing cache used to contain metadata. This resides in a dataspace and can optionally be used to extend the size of the metadata cache. You can also specify a fixed option, which indicates that the pages are permanently fixed for performance. Note that the fixed option reserves real storage for usage by zFS only.

When specifying the fixed parameter, a number between 1M and 2048M can be used. A 'K' or 'M' can be appended to the value to mean kilobytes or megabytes, respectively. An example is:

```
metaback_cache_size=64M,fixed
```

Use the **zfsadm config** command to change or set a value since there is no default value set by zFS.

16.20 Performance APIs

- Previously, certain zFS performance counters could only be retrieved via operator console command
  - Data could not easily be retrieved by an application
  - Provide an API to obtain statistics on zFS activity
- Now, all zFS performance counters can be retrieved by an application - pfsctl API (BPX1PCT)
- Now, zfsadm command can be used to display/reset performance counters
  - zfsadm query -metacache
- Administrative application programs can use this function
- RMF exploits this function

Figure 16-20 Performance APIs for zFS

Performance APIs for zFS

zFS file systems contain files and directories that can be accessed with z/OS UNIX application programming interfaces (APIs). These APIs are used to manage zFS aggregates and file systems and to query and set configuration options.

pfsctl BPX1PCT

The zFS application programming interface (API) is pfsctl (BPX1PCT). This API is used to send physical file system-specific requests to a physical file system. It is documented in a general manner in the *z/OS UNIX System Services Programming: Assembler Callable Services Reference*, SA22-7803. zFS is a physical file system and supports several (ZFS-specific) pfsctl functions.

zFS pfsctl APIs work across sysplex members. They can query and set information on the current system only. The **`zfsadm`** command has a new subcommand, **`query`**, that allows performance analysis of all caches.

RMF support

Two new Monitor III reports provide overview and performance information about the zFS activity, as follows:

- The zFS Summary Report helps to control the zFS environment and I/O balancing.
- The zFS Activity Report measures zFS activity on the basis of single file systems, for example, a file system's performance and DASD utilization.

16.21 Performance monitoring APIs

- zFS provides six more pfsctl APIs to retrieve performance counters
  - Locks
  - Storage
  - User data cache
  - iocounts
  - iobyaggr
  - iobydasd
- zFS provides a new zfsadm command (**query**) to query/reset the performance counters

zfsadm query [-locking] [-storage] [-usercache] [-iocounts] [-iobyaggregate] [-iobydasd] [-level] [-help] [-reset]

Figure 16-21 Performance monitoring commands with z/OS V1R6

The zfsadm query command

This command displays internal zFS statistics (counters and timers) maintained in the zFS physical file system. It provides performance counters for individual cache and storage areas. The **`zfsadm query`** command has the following options:

```
[-system system name] [-locking] [-reset] [-storage] [-usercache] [-trancache]
[-iocounts] [-iobyaggregate] [-iobydasd] [-knpfs] [-metadata]
[-dircache] [-vnodecache] [-logcache] [-level] [-help]
```

The **`f zfs,query,all`** command was implemented with the very first zFS release and provides statistics for all the performance counters.

The zfsadm query command in z/OS V1R6

This command displays internal zFS statistics (counters and timers) maintained in the zFS physical file system.

The options are as follows:

-locking Specifies that the locking statistics report should be displayed.

-storage Specifies that the storage report should be displayed.

-usercache Specifies that the user cache report should be displayed.

-iocounts Specifies that the I/O count report should be displayed.

-iobyaggregate Specifies that the I/O count by aggregate report should be displayed.

-iobydasd Specifies that the I/O count by DASD report should be displayed.

-level Prints the level of the **zfsadm** command. This is useful when you are diagnosing a problem. All other valid options specified with this option are ignored.

-help Prints the online help for this command. All other valid options specified with this option are ignored.

-reset Specifies the report counters should be reset to zero. Should be specified with a report type.

The f zfs,query,all command

This command, which has always been available with zFS, enables you to query internal zFS counters and values. They are displayed on the system log. It also allows you to initiate or gather debugging information. The zFS PFS must be running to use this command.

The options for using this command are as follows:

```
f zfs,query,{all | settings | storage | threads}
f zfs,reset,{all | storage}
f zfs,trace,{reset | print}
f zfs,abort
f zfs,dump
f zfs,hangbreak
```

16.22 zfsadm query command

- zfsadm query command options with z/OS V1R7
  - knpfs, -metacache, -dircache, -vnodecache, -logcache, -trancache -system system name

```
ROGERS @ SC65:/u/rogers>zfsadm query -dircache
                   Directory Backing Caching Statistics

Buffers    (K bytes)  Requests     Hits    Ratio   Discards
---------- --------- ---------- ---------- ------ ----------
       256      2048    1216679    1216628  99.9%         50

ROGERS @ SC65:/u/rogers>zfsadm query -system sc70 -dircache
                   Directory Backing Caching Statistics

Buffers    (K bytes)  Requests     Hits    Ratio   Discards
---------- --------- ---------- ---------- ------ ----------
       256      2048          0          0   0.0%          0
```

Figure 16-22 zfsadm query command changes with z/OS V1R7

Options on the zfsadm query command

The options are as follows:

-system system name Specifies the name of the system the report request will be sent to, to retrieve the data requested. This is important since you can send the request to other members of a sysplex.

-trancache Specifies that the transaction cache counters report should be displayed.

-knpfs Specifies that the kernel counters report should be displayed.

-metacache Specifies that the metadata cache counters report should be displayed.

-dircache Specifies that the directory cache counters report should be displayed.

-vnodecache Specifies that the vnode cache counters report should be displayed.

-logcache Specifies that the log cache counters report should be displayed.

16.23 The IOEZADM utility from TSO for commands

```
   Menu  List  Mode  Functions  Utilities  Help

-----------------------------------------------------------------------
                                ISPF Command Shell
 Enter TSO or Workstation commands below:

 ===> ioezadm query -dircache

 Place cursor on choice and press enter to Retrieve command

 => ioezadm query -dircache
 => ioezadm aggrinfo
 => omvs
 => ish
 => ishell
 => bpxwh2z
 =>

                   Directory Backing Caching Statistics

Buffers    (K bytes)  Requests     Hits     Ratio   Discards
---------- --------- ---------- ---------- ------ ----------
      8192     65536    2994464    2976066  99.3%      14743
```

Figure 16-23 Using the IOEZADM utility from TSO

The IOEZADM utility

Using this utility from TSO, you can enter commands exactly like you do with the **zfsadm** command. Therefore, the **zfsadm** command and the IOEZADM utility program can be used to manage file systems and aggregates.

Note: Although **zfsadm** and IOEZADM are physically different modules, they contain identical code. Whenever you think of **zfsadm** as a zFS administration command, it also means both **zfsadm** and IOEZADM. Therefore, IOEZADM can be used as a TSO/E command; it performs the same functions as the **zfsadm** command.

16.24 Directory cache

- A zFS IOEPRMxx parameter file statement is available to set the directory cache size
  - dir_cache_size - Size of the directory buffer cache
    - Example: dir_cache_size=4M
  - The minimum and default size is 2M, the maximum value that can be set is 512M
  - Changing this cache size may be useful if you expect situations with many file creates or deletes

Figure 16-24 dir_cache specifications for directory cache

Directory cache specifications

New with z/OS V1R7 is a directory cache. The dir_cache_size parameter can be specified in either the IOEFSPRM member or the IOEPRMxx PARMLIB member, depending on which option is used. This option is specified as follows:

```
dir_cache_size - Specifies the size of the directory buffer cache.
   Default Value       2M
   Expected Value      A number between 2M to 512M
   Example             dir_cache_size=4M
```

Monitoring dircache

This is a parameter that should be monitored since this cache size should be looked at if it is expected that many file creates or deletes may occur during system operation. Use the **`zfsadm query -dircache`** command, as shown in Figure 16-22 on page 794.

This important new cache can be implemented on older levels of z/OS back to z/OS V1R4 with the PTFs shown in Figure 16-24.

16.25 Directory caching display tool

- This z/OS UNIX tool displays whether z/OS UNIX directory caching is active
  - It is important for helping to resolve performance issues when z/OS UNIX directory caching is lost due to file system movement
  - This normally only happens when the shared file system environment is mixed
- You can get the tool from:
  - ftp://ftp.software.ibm.com/s390/zos/tools/wjsip/wjsipndc.txt

Note: Be sure to convert it from ISO8859-1 to IBM-1047 or IBM-037 when placing the file to your z/OS system

Figure 16-25 Directory caching display tool

Directory caching display tool

This tool displays whether z/OS UNIX directory caching is active. It is important for helping to resolve performance issues when z/OS UNIX directory caching is lost due to file system movement. This normally only happens when the shared file system environment is mixed.

You can get the tool from:

```
ftp://ftp.software.ibm.com/s390/zos/tools/wjsip/wjsipndc.txt
```

Note: Be sure to convert it from ISO8859-1 to IBM-1047 or IBM-037 when placing the file into your z/OS system.

16.26 File system monitoring tool (FSMON)

- This wjsfsmon utility is mentioned in APAR OA29619 and can help you to determine which systems are accessing your zFS read-write file systems and whether your zFS read-write file systems are accessed from multiple systems
- See the z/OS UNIX Tools and Toys website - to download the tool and its documentation:
  - http://www.ibm.com/servers/eserver/zseries/zos/unix/tools
- You can also go directly to the UNIX Tools site:
  - http://www-03.ibm.com/systems/z/os/zos/features/unix/bpxa1ty2.html
- Retrieve it from:
  - ftp://ftp.software.ibm.com/s390/zos/tools/wjsfsmon/wjsfsmon.txt

Figure 16-26 File system monitoring tool (FSMON)

File system monitoring tool (FSMON)

This wjsfsmon utility is mentioned in APAR OA29619 and can help you to determine which systems are accessing your zFS read-write file systems and whether they are accessed from multiple systems.

See the z/OS UNIX Tools and Toys website to download the tool and its documentation:

```
http://www.ibm.com/servers/eserver/zseries/zos/unix/tools
```

You can also go directly to the UNIX Tools site:

```
http://www-03.ibm.com/systems/z/os/zos/features/unix/bpxa1ty2.html
```

Or retrieve it from:

```
ftp://ftp.software.ibm.com/s390/zos/tools/wjsfsmon/wjsfsmon.txt
```

Note: The documentation is available as a PDF. The tool itself is available in ASCII. Be sure to convert it from ISO8859-1 to IBM-1047 or IBM-037 when placing the file into your z/OS system.

16.27 The zfsadm query -iobyaggr command

```
ROGERS @ SC65:/u/rogers>zfsadm query -iobyaggr
                     zFS I/O by Currently Attached Aggregate

DASD   PAV
VOLSER IOs Mode  Reads       K bytes     Writes      K bytes    Dataset Name
------ --- ----  ----------  ----------  ----------  ---------- ------------
SBOX42   1  R/O          10          68           0          0 OMVS.TEST.MULTIFS.ZFS
MHL1A0   3  R/O       93432      373760           0          0 ZFSFR.ZFSA.ZFS
MHL1A1   2  R/O       93440      373824           0          0 ZFSFR.ZFSB.ZFS
SBOX32   1  R/W          22         164       82775     331100 OMVS.HERING.TEST.ZFS
SBOX1A   3  R/W           9          60       82747     330988 OMVS.HERING.TEST.DIRCACHE
------ --- ----  ----------  ----------  ----------  ----------  ------------
     5               186913      747876      165522     662088  *TOTALS*


Total number of waits for I/O:     186910
Average I/O wait time:                 0.703 (msecs)
```

Figure 16-27 zfsadm query -iobyaggr command example

The -iobyaggr option

The **`zfsadm query -iobyaggr`** command displays information about the number of reads and writes and the number of bytes transferred for each aggregate. Use this command to determine the number of I/Os and the amount of data transferred on an aggregate basis. The example shown in Figure 16-27 shows five aggregates.

zFS I/O driver

The zFS I/O driver is essentially an I/O queue manager (one I/O queue per DASD). It uses Media Manager to issue I/O to VSAM data sets. It generally sends no more than one I/O per DASD volume to disk at one time. The exception is parallel access volume (PAV) DASD. These DASD often have multiple paths and can perform multiple I/O in parallel. In this case zFS will divide the number of access paths by 2 and round any fraction up. (Example, for a PAV DASD with 5 paths zFS issues at most 3 I/Os at one time to Media Manager). The reason zFS limits the I/O is that it uses a dynamic reordering and prioritization scheme to improve performance by reordering the I/O queue on demand. Thus, high priority I/Os (I/Os that are currently being waited on, for example) are placed up front, and an I/O can be made high priority at any time during its life. Thus the "PAV IOs" column shows how many I/Os are sent in parallel to Media Manager by zFS; non-PAV DASD always shows the value 1. The DASD volser for the primary extent of each aggregate is shown along with the total number of I/Os and bytes read/written. Finally, the number of times a thread processing a request must wait on I/O and the average wait time in milliseconds is shown. By using this information in conjunction with the KN report, you can break down zFS response time into what percentage of the response time is for I/O wait.

16.28 SMF recording

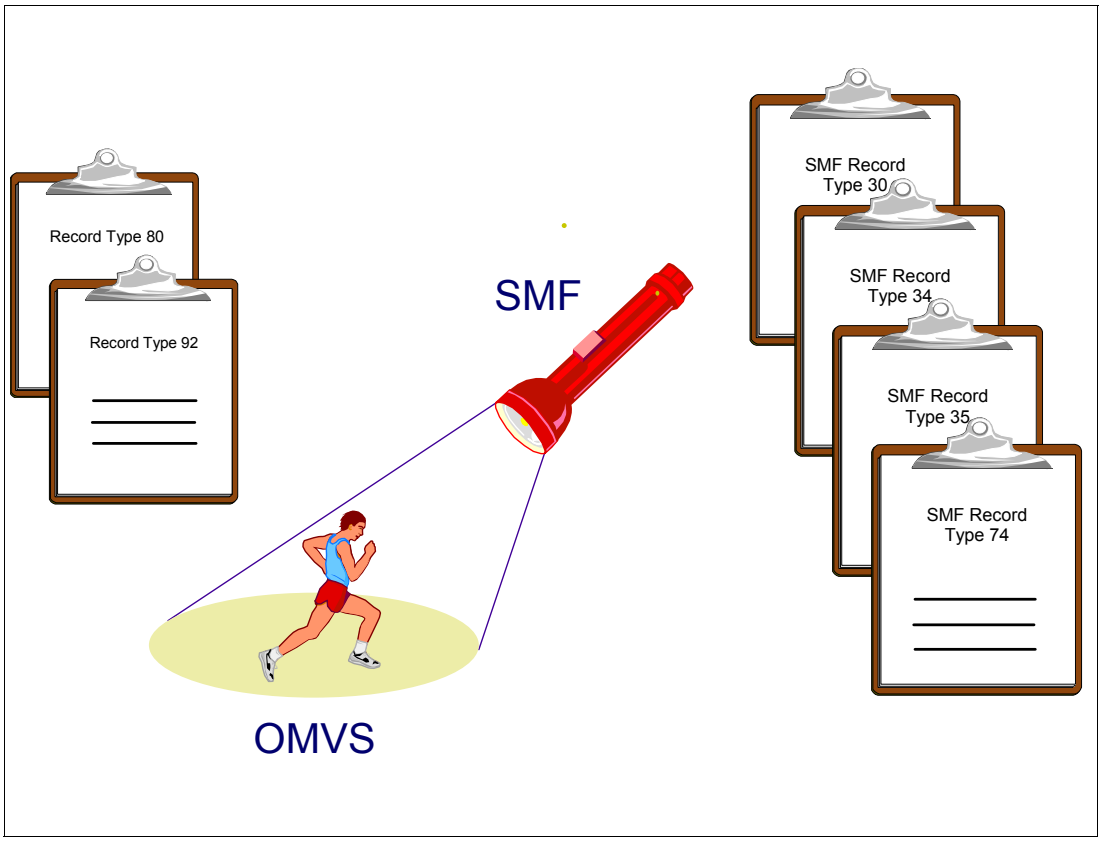


Figure 16-28 SMF recording of z/OS UNIX processing

SMF records for z/OS UNIX

You can use SMF to report on activity from a user application, to report activity on a job and jobstep basis, and to report the activity of mounted file systems and files.

SMF records are written to store information about the following activities for z/OS UNIX.

SMF record type 30

SMF record type 30 reports activity on a job and jobstep basis. Though file system activity is included in the EXCP count for the address space, the process section in the record breaks down the EXCP count into the following categories:

- Directory reads
- Reads and writes to regular files
- Reads and writes to pipes
- Reads and writes to character special files
- Reads and writes to network sockets

SMF record type 34 and 35

When a new address space is created for a fork or spawn, SMF cuts a type 34 record. When the process ends, SMF cuts a type 35 record. A type 34 record is defined as TSO/E logon and a type 35 record is defined as TSO/E logoff. If these records are not active in the

environment, no further actions are necessary. If they are active for TSO/E accounting, it is necessary (recommended) to suppress these records for UNIX processes. To suppress type 34 and type 35 records, add the following to the SMFPRMxx PARMLIB member:

```
SYS(TYPE(34,35)) SUBSYS(OMVS,NOTYPE(34,35))
```

SMF record type 74

SMF record type 74, subtype 3, reports kernel activity.

SMF record type 80

SMF record type 80 includes an extended length relocate section.

SMF record type 92

SMF record type 92 reports the activity of mounted file systems and files. I/O activity data from an entire mounted file system is provided only when the file system is unmounted. However, these records are useful because they provide information on the total space available in the file system and the total space currently used. This provides an indication of when it is time to increase the size of a mountable file system.

16.29 SMF 92 subtype 14 enhancement

- SMF 92 recording enhancement - z/OS V1R9
  - A subtype 14 record is added to the SMF 92 records for reporting file and directory deletion and renaming
  - SMF 92 recording now monitors the deletion and renaming of USS files and directories using subtype 14 records
- Monitor when files are deleted or renamed
- Monitor when directories are deleted or renamed
- More accurate monitoring of resources

In a shared file system mode, recording occurs on the user's system where the command was issued

Figure 16-29 SMF 92 subtype 14 recording enhancement

Subtype 14 enhancement

To gather information about the deleting of files and directories, you can set up monitoring for SMF type 92 records with subtype 14.

When files or directories are deleted, you will receive information about the time the file or directory was deleted or renamed, in addition to its file type, serial number, and unique device number. In a shared file system, the recording occurs on the user's system where the command was issued.

Subtype 14 records with SMF Type 92 recordithe following information:

- Time of the delete or rename
- Rename indicator
- File type
- File serial number
- File serial number of parent
- Unique device number for file or directory
- File system name
- File or directory name and length
- New name and length

16.30 RMF reporting

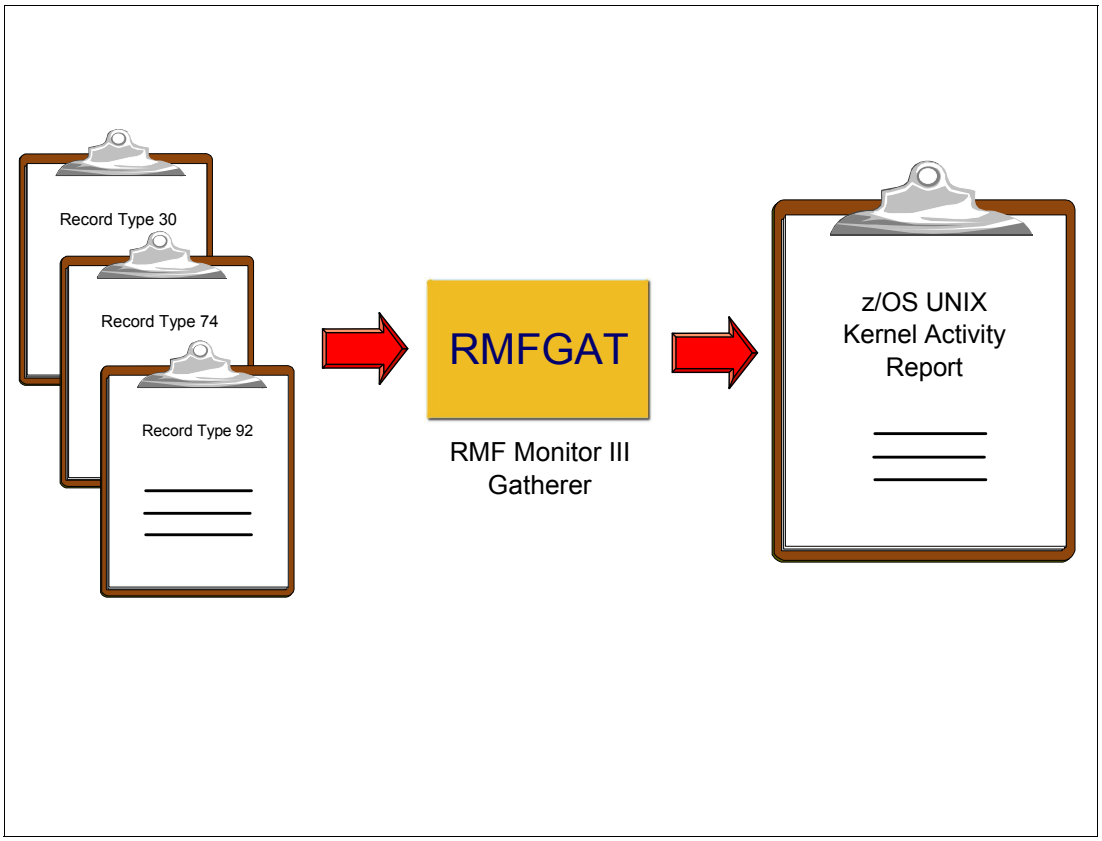


Figure 16-30 RMF reporting of z/OS UNIX processing

RMF reports for z/OS UNIX

The Resource Measurement Facility (RMF) provides z/OS UNIX report information using existing records as described previously.

RMF invokes the Monitor III procedure RMFGAT to obtain z/OS UNIX data. The RMFGAT started task must be associated with a user ID that has an OMVS segment. The following RACF command is usable to give RMFGAT a UID and to designate the root directory as its home directory:

```
ADDUSER RMFGAT DFLTGRP(<OMVSGROUP>) OMVS(UID(4711) HOME('/'))
```

Gathering options for z/OS UNIX are not included in the default PARMLIB member for RMF Monitor I. z/OS UNIX data is gathered by Monitor III, and not by Monitor I.

The Monitor III data gatherer collects z/OS UNIX data for input to the RMF post processor. This data can be used to create a z/OS UNIX kernel activity report.

16.31 RMF Monitor III support for zFS

- Monitor III zFS reports data in z/OS V1R7 on:
  - zFS response time/wait times
  - zFS cache activity
  - zFS activity/capacity by aggregate
  - zFS activity/capacity by filesystem
- Data helps to control the zFS environment for:
  - Cache sizes
  - I/O balancing
  - Capacity control for zFS aggregates

Figure 16-31 RMF Monitor III support with z/OS V1R7

RMF Monitor III support

Two new Monitor III reports provide overview and performance information about the zFS activity:

- The zFS Summary Report helps to control the zFS environment and the I/O balancing.
- The zFS Activity Report measures zFS activity on the basis of single file systems, for example, a file system's performance and DASD utilization.

RMF data reporting

To use the zFS UNIX file system to its full capacity, it is necessary to apply appropriate tuning options. The zFS performance especially depends on a suitable tailoring of its cache sizes to reduce I/O rates and path lengths. The performance can also be improved by adapting available disk space.

Therefore, this support provides a summary of zFS activity, response times and DASD statistics on the current system and thus helps to control and tune the zFS environment. For example, you can use the HIT% values in the Cache Activity section as indication whether the current cache sizes are sufficient.

16.32 zFS access to file systems

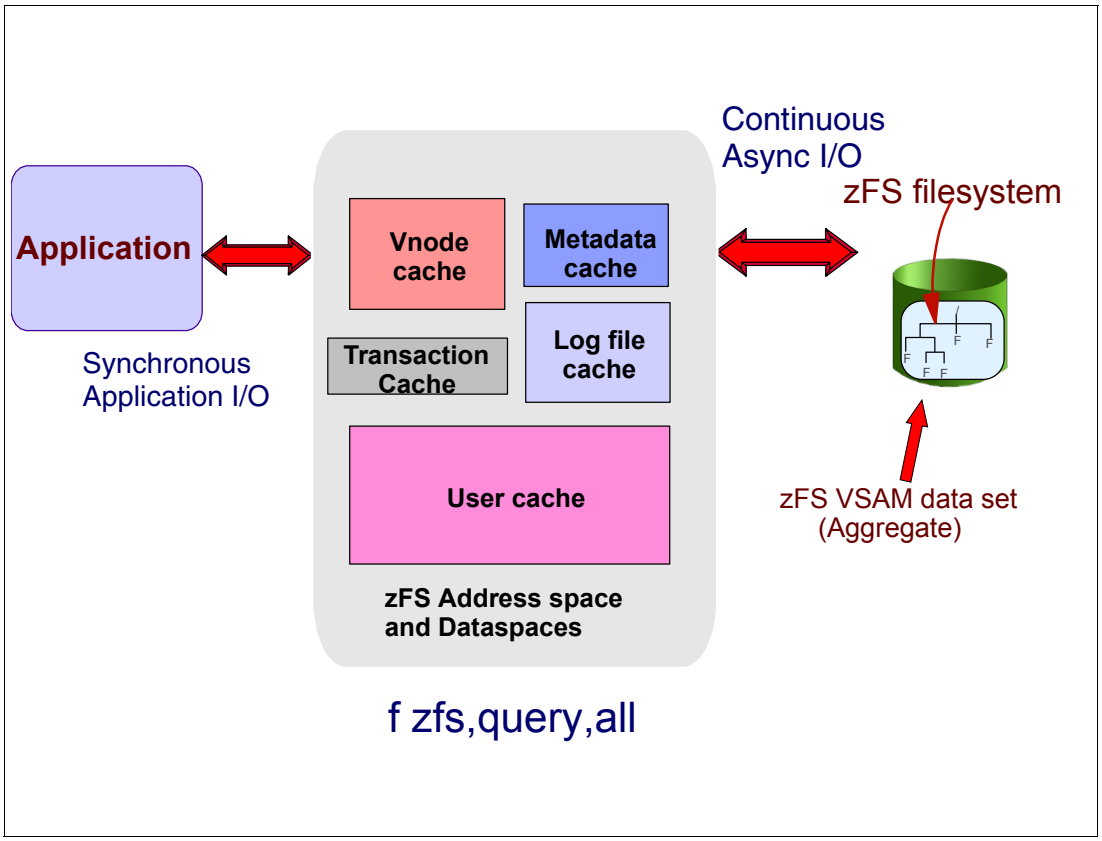


Figure 16-32 zFS cache and access to file data in the aggregates

zFS and access to cached data

zFS achieves its performance by caching file data and thus avoiding access to the file data by accessing the DASD volumes. The metadata cache is used to contain all file system metadata, which includes all directory contents, file status information (such as atime, mtime, size, permission bits, and so on), file system structures, and it also caches data for files smaller than 7K. Essentially, zFS stores a file by using one of the following three methods:

Inline If the file is smaller than 52 bytes, its data is stored in the structure that contains the status information for the file.

Fragmented If the file is less than 7K, it is stored in blocks on disk that could be shared with other files, hence multiple files are stored in the same physical disk block. Physical disk blocks are always 8K in size.

Blocked Files larger than 7K are stored in multiple blocks, blocked files are only stored in the user file cache, and all I/O is performed directly to or from user file cache buffers.

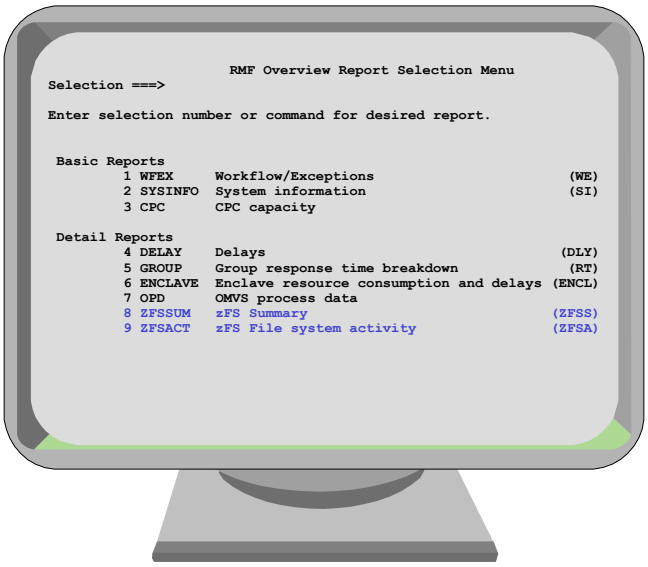
Since inline files are stored in the status block, files that are stored on disk by using the inline method are stored in the metadata and hence are cached in the metadata cache (and also in the user file cache). Since the metadata cache is the only component that knows about multiple files sharing the same disk blocks, small fragmented files are stored in the metadata cache (and also in the user file cache) and I/O is performed directly to or from the metadata cache for these small user files.

Metadata cache

The metadata cache is stored in the primary address space and its default size is 32M. Since the metadata cache only contains metadata and small files, it normally does not need to be nearly as large as the user file cache. The operator F ZFS,QUERY,ALL command output shows statistics for the metadata cache including the cache hit ratio and I/O rates from the metadata cache.

An optional metadata backing cache can be specified that extends the size of the metadata cache. It resides in a data space and increases the amount of metadata that can be kept in memory. It may improve the performance of workloads that require large amounts of metadata.

16.33 RMF Overview Report Selection Menu



- RMF Overview Report Selection Menu (Detail reports)
- Command interface
  - **ZFSSUM** or **ZFSS**:
    - zFS summary report
  - **ZFSACT** or **ZFSA**:
    - zFS activity report
- Gatherer options:
  - Option **NOZFS** | **ZFS**

Figure 16-33 RMF selection menu for zFS reports

RMF reports for zFS

The zFS Activity Report measures zFS activity on the basis of single file systems. With this information, you can monitor DASD performance to ensure that there are no volumes or channels working near the limit of their capacity (space and workload).

zFS Activity Report

To request the zFS Activity Report, select 1 from the Primary Menu, then 9 from the Overview Report Selection Menu (shown in Figure 16-33), or enter one of the following commands:

- ZFSACT
- ZFSA

In addition, you can navigate to this report through cursor-sensitive control from the ZFSSUM report to display zFS activity for a specific aggregate.

zFS Summary Report

To request the zFS Summary Report, select 1 from the Primary Menu, then 8 from the Overview Report Selection Menu (shown in Figure 7 in topic 2.5) or enter one of the following commands:

- ZFSSUM
- ZFSS

16.34 zFS Summary Report

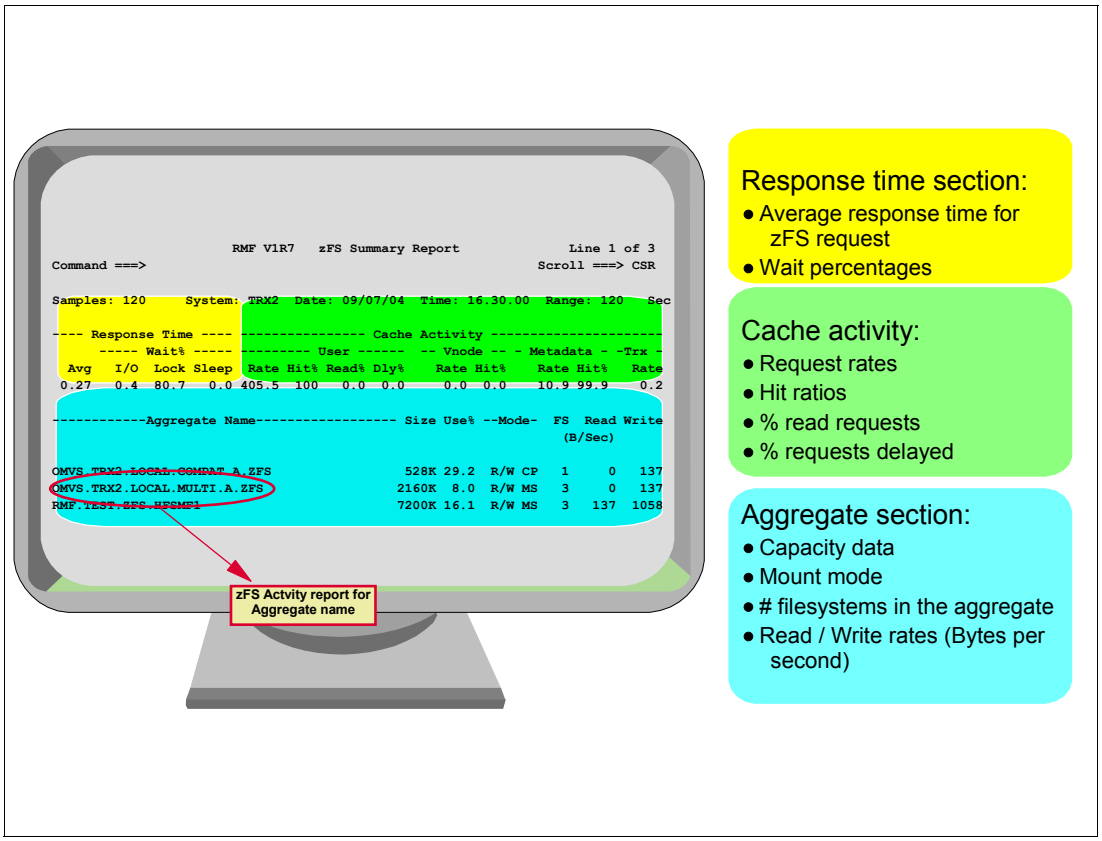


Figure 16-34 zFS Summary Report

zFS Summary Report

To use the zFS UNIX file system to its full capacity, it is necessary to apply appropriate tuning options. The zFS performance especially depends on a suitable tailoring of its cache sizes to reduce I/O rates and path lengths. The performance can also be improved by adapting available disk space. Therefore, this report provides a summary of zFS activity, response times, and DASD statistics on the current system and thus helps to control and tune the zFS environment. For example, you can use the HIT% values in the Cache Activity section as indication whether the current cache sizes are sufficient.

Response Time section

The Response Time section provides summary data for the overall response time for zFS requests and breaks down the response time into percentages for various delay reasons. This information can help to discover bottlenecks, for example with I/O operations.

Cache Activity section

The Cache Activity section provides an overview of zFS cache activity of the four main cache types (user file, vnode, metadata, and transaction caches).

Aggregate Name section

The Aggregate Name section provides measurements about zFS activity related to single zFS aggregates. With this information you can check whether the disk space allocations for the aggregates are appropriate.

16.35 zFS Summary I/O details by type

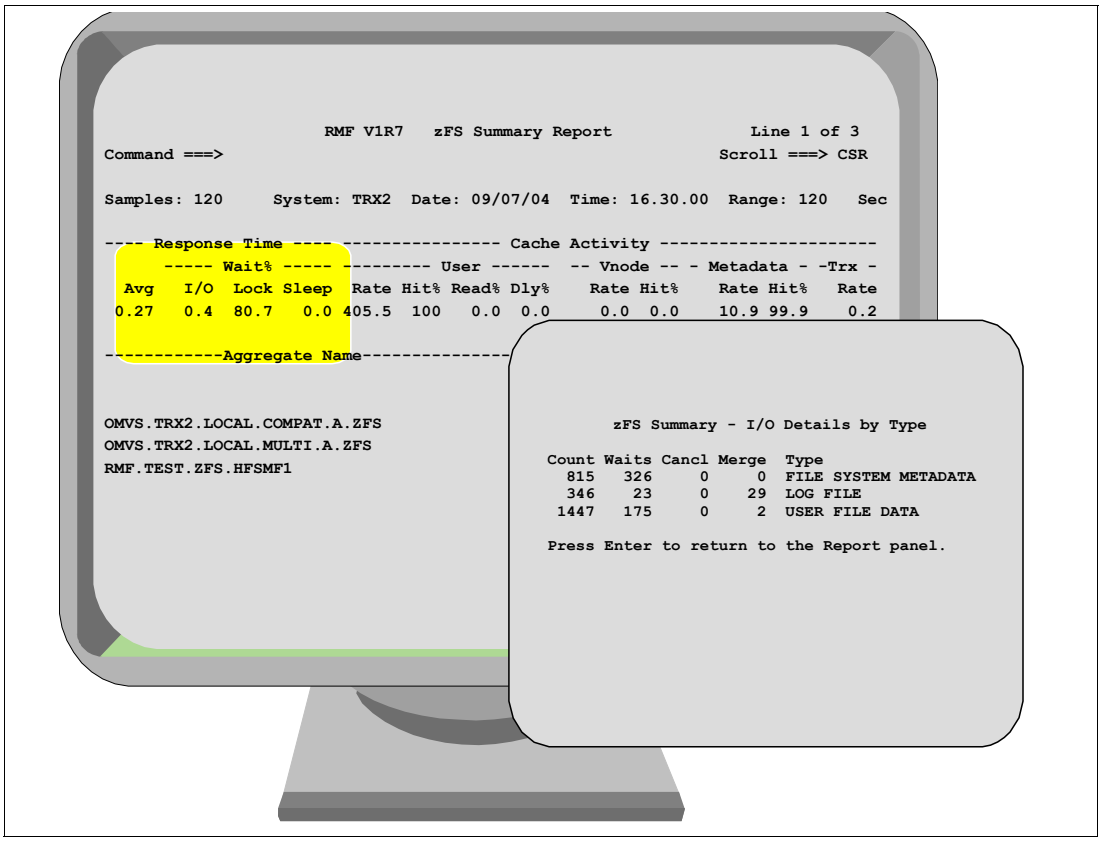


Figure 16-35 Summary Report details by type

Response Time

Avg Average time in milliseconds required for the completion of the zFS requests during the reporting interval.

The following Wait percentages are reported:

I/O Percentage of time that zFS requests had to wait for I/O completion.

Lock Percentage of time that zFS requests had to wait for locks.

Sleep Percentage of time that zFS requests had to wait for events.

Cache Activity

The user file cache is for caching regular user files that are larger than 7K. The measured statistics have the following meanings:

Rate Total number of read and write requests per second made to the user file cache.

Hit% Percentage of read and write requests to the user file cache that completed without accessing the DASDs.

Read% Percentage of read requests, based on the sum of read and write requests.

Dly% Percentage of delayed requests, with the following events counted as delays:

- **Read wait:** A read request must wait for a pending I/O operation.
- **Write wait:** A write request must wait because of a pending I/O operation.

Write faulted: A write request to a file in the user file cache needs to perform a read operation from DASD before writing, because the required page of that file is currently not in the cache.

Vnode

The vnode cache is used to hold virtual inodes. An inode is a data structure related to a file in the file system, holding information about the file's user and group ownership, access mode and type. The measured statistics have the following meanings:

Rate Number of read and write requests per second made to the vnode cache.

Hit% Percentage of read and write requests to the vnode cache that completed without accessing the DASDs.

Metadata

The metadata cache is used for file system metadata and for files smaller than 7K. It resides in the primary z/FS address space. An optional metadata backing cache, which resides in a data space, can be used as an extension to the metadata cache. The measured statistics have the following meanings:

Rate Number of read and write requests per second made to the metadata cache

Hit% Percentage of read and write requests to the metadata cache that completed without accessing the DASDs

Trx

The transaction cache is used for caching data structures that change metadata. There is one measured statistics:

Rate Number of transactions per second that started in the transaction cache

Aggregate Activity section

This section contains the following:

- Name of the zFS aggregate, that is, the name of the VSAM Linear Data Set (VSAM LDS)
- Size of the aggregate
- Percentage of used space in the aggregate
- Read data transfer rate in bytes/second for the aggregate
- Write data transfer rate in bytes/second for the aggregate

Note: From any value in the Wait% -I/O field, you can reach the I/O Details by Type panel.

I/O Details by Type report

The zFS Summary - I/O Details by Type report displays a breakdown of I/O requests into the following types: I/O for file system metadata, I/O for log data, and I/O for user file data.

Count Total number of I/O requests of the indicated type

Waits Number of zFS requests waiting for an I/O completion of the indicated I/O type

Cancl Number of cancelled zFS requests during an I/O request of the indicated type, for example, a user tried to delete a file during a pending I/O to this file's metadata

Merge Number of merges of two I/O requests into a single request because of better performance

Type Type of the I/O request (I/O for metadata, log data or user file data)

16.36 User and vnode cache detail

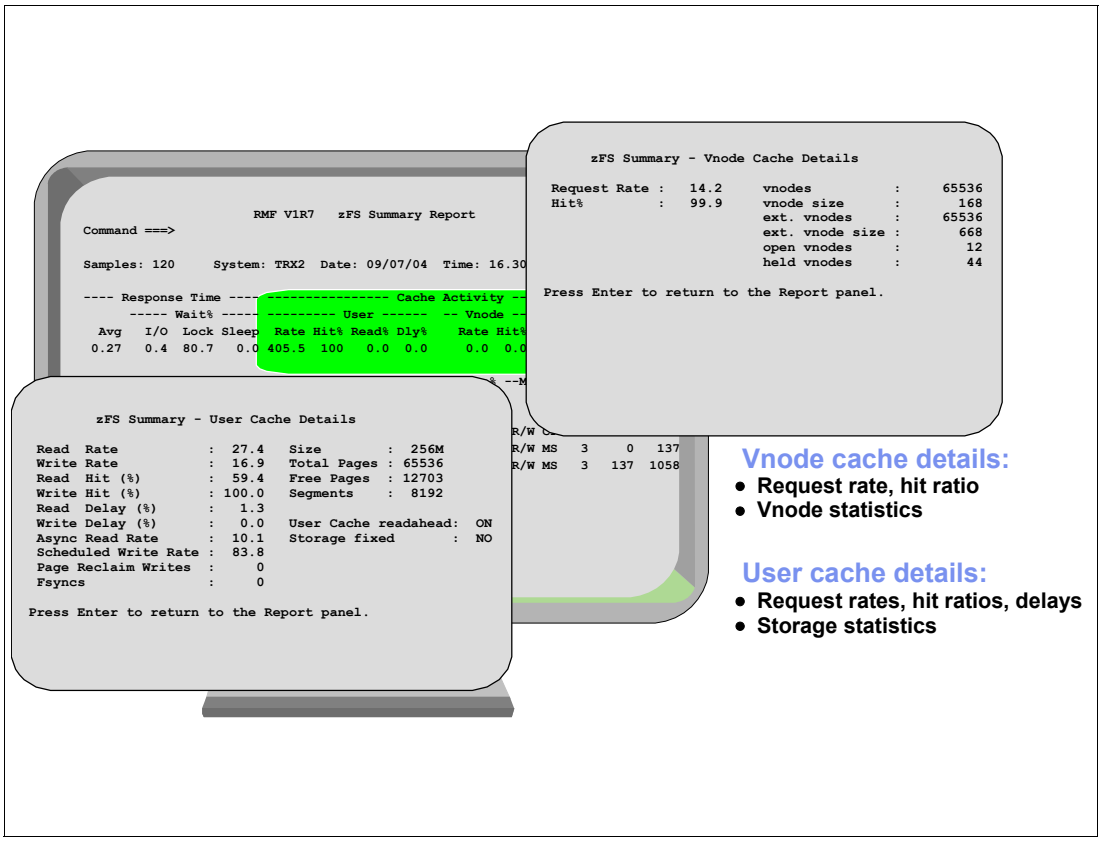


Figure 16-36 User cache and vnode cache details

User and vnode cache detail

The user file cache is for caching regular user files that are larger than 7K. The zFS Summary - User Cache Details report displays the following details of the user file cache activity.

User file cache

Read Rate Number of read requests per second made to the user file cache

Write Rate Number of write requests per second made to the user file cache

Read Hit (%) Percentage of read requests to the user file cache that completed without accessing the DASD

Write Hit (%) Percentage of write requests to the user file cache that completed without accessing the DASD

Read Delay (%) Percentage of delayed read requests to the user file cache. A read request is delayed if it must wait for pending I/O, for example, because the file is in a pending read state due to asynchronous read ahead from DASD to the user file cache.

Write Delay (%) Percentage of delayed write requests to the user file cache

The following reasons are counted as write request delays:

Write wait Write must wait for pending I/O.

Write faulted Write to a file needs to perform a read from DASD. If a write-only updates a part of a file's page, and this page is not in the user file cache, then the page must be read from DASD before the new data is written to the cache.

Async Read Rate Number of read-aheads per second.

Scheduled Write Rate Number of scheduled writes per second.

Page Reclaim Writes Total number of page reclaim writes. A page reclaim write action writes one segment of a file from the user file cache to DASD. Page reclaim writes are performed to reclaim space in the user file cache. If page reclaim writes occur too often in relation to the write rate, then the user file cache may be too small.

Fsyncs Total number of requests for file synchronization (fsync) between user file cache and DASD.

Size Total size of the user file cache.

Total Pages Total number of pages in the user file cache.

Free Pages Total number of free pages in the user file cache.

Segments Total number of allocated segments in the user file cache.

User Cache readahead Shows if the zFS parameter user_cache_read is on or off. This parameter specifies whether zFS performs read-ahead for sequential access.

Storage fixed Shows whether the size of the user file cache storage is fixed. If the zFS parameter user_cache_size is set to “fixed”, then zFS reserves real storage for use by zFS only. The fixed option helps to improve performance during data access and can be applied if you have enough real memory available.

Vnode cache detail

The vnode cache is used to hold virtual inodes. An inode is a data structure related to a file in the file system, holding information about the file's user and group ownership, access mode and type. The zFS Summary- Vnode Cache Details report displays the following details of the vnode cache activity.

Request Rate Number of requests per second made to the vnode cache.

Hit (%) Percentage of requests to the vnode data that found the target vnode data structures in the vnode cache. High hit rates indicate a favorable zFS environment, because each miss involves initialization of vnode data structures in the vnode cache.

Vnodes Number of currently allocated vnodes in the vnode cache. If more vnodes are requested than are currently available, then zFS dynamically allocates more vnodes.

Vnode Size Size of a vnode data structure in bytes.

Ext. Vnodes Number of extended vnodes.

Ext. Vnode Size Size of an extended vnode data structure in bytes.

Open Vnodes Number of currently open vnodes.

Held Vnodes Number of vnodes currently held in zFS by USS.

16.37 DFSMSdss dump and restore for zFS file systems

- Current method for backup - restore
  - 1. Quiesce the aggregate (this drains any activity and suspends any new requests)
  - 2. Backup the aggregate (and all the file systems)
  - 3. Unquiesce the aggregate (allow zFS activity to continue)

Figure 16-37 APARs to automatically quiesce and copy aggregates

Dump and restore aggregates

This support allows a quiesce of zFS data sets during logical dump and logical copy operations. This allows the user to perform logical dump and logical copy of mounted zFS data sets, without having to first quiesce or unmount the zFS.

The quiesce ability allows you to dump a zFS data set while it is in use, as long as you run the dump job on the same system that the zFS data set is currently mounted on.

Note: When this became available, the dump or copy operation had to be executed on the same system on which the zFS is mounted.

Any existing zFS data sets must be unmounted or detached and remounted or reattached in order to activate the DFSMSdss support for those zFS data sets.

16.38 UNQUIESCE command

- Previously, if a zFS aggregate was quiesced and the job failed to unquiesce it, you would need to use OMVS - zfsadm unquiesce command
  - Not possible to unquiesce from operator console
  - A command to issue an unquiesce
    - F ZFS,UNQUIESCE,ROGERS.HARRY.ZFS
    - Must be issued from owning system
- This command works as follows:
  - It must be issued from the owning system
  - It does not forward requests to other members of the sysplex

Figure 16-38 UNQUIESCE command with z/OS V1R7

UNQUIESCE operator command

This operator command enables you to query internal ZFS counters and values. They are displayed on the system log. It also allows you to initiate or gather debugging information. The ZFS PFS must be running to use this command.

The command syntax must be issued from the owning system and is as follows for aggregate ROGERS.HARRY.ZFS:

```
F ZFS,UNQUIESCE,ROGERS.HARRY.ZFS
```

The UNQUIESCE option causes a quiesced aggregate to become unquiesced. Only locally attached aggregates can be unquiesced using the F UNQUIESCE command. You must issue this command on the system that owns the aggregate.

List the system that owns the aggregate

Use the z/OS UNIX **zfsadm lsaggr** command to determine which system owns the aggregate.

```
PAUL @ SC65:/>zfsadm lsaggr
IOEZ00106I A total of 4 aggregates are attached
TRAUNER.ROLAND.ZFS                              SC63      R/W
ZFSFR.ZFSG.ZFS                                  SC65      R/W
OMVS.D8F202S.HFS                                SC64      R/W
ROGERS.HARRY.ZFS                                SC70      R/W QUIESCE
```

16.39 zFS recovery support

- zFS is a logging file system
  - It logs metadata updates
  - On a system crash, the log is replayed to bring the file system to a consistent state
- I/O requests are started immediately (asynchronously) so on a system crash, most data is already on disk
- Salvager utility - provides aggregate recovery
  - **-recoveronly** - Recover the specified aggregate by replay of log of metadata changes
  - **-verifyonly** - Verify the aggregate structure to determine if it contains any inconsistencies and report
  - **-salvageonly -** Salvage the aggregate and attempt to repair any inconsistencies it finds

Figure 16-39 zFS recovery utilities

zFS Salvager utility

Every zFS aggregate contains a log file used to record transactions describing changes to the file system structure. This log file is, by default, 1% of the aggregate size but is tailorable by the administrator on the **`ioeagfmt`** command. Usually, 1% is sufficient for most aggregates. Especially large aggregates might need less than 1% while very small aggregates might need more than 1% if a high degree of parallel update activity occurs for the aggregate.

The zFS utility scans an aggregate and reports inconsistencies. Aggregates can be verified, recovered (that is, the log is replayed), or salvaged (that is, the aggregate is repaired). This utility is known as the Salvager.

This utility is not normally needed. If a system failure occurs, the aggregate log is replayed automatically the next time the aggregate is attached (or for compatibility mode aggregates, the next time the file system is mounted). This normally brings the aggregate (and all the file systems) back to a consistent state.

Salvager utility options

The **`ioeagslv`** utility invokes the Salvager on the zFS aggregate specified with the **`-aggregate`** option. Following a system restart, the Salvager employs the zFS file system log mechanism to return consistency to a file system by running recovery on the aggregate on which the file system resides. Recovery means replaying of the log on the aggregate; the log records all changes made to metadata as a result of operations such as file creation and deletion. If problems are detected in the basic structure of the aggregate, if the log mechanism is

damaged, or if the storage medium of the aggregate is suspect, the **ioeagslv** utility must be used to verify or repair the structure of the aggregate.

The primary options you specify when using the utility are:

```
ioeagslv -aggregate name [-recoveronly] [-verifyonly | -salvageonly]
```

-recoveronly Directs the Salvager to recover the specified aggregate. The Salvager replays the log of metadata changes that resides on the aggregate.

-verifyonly Directs the Salvager to verify the specified aggregate. The Salvager examines the structure of the aggregate to determine if it contains any inconsistencies, reporting any that it finds.

-salvageonly Directs the Salvager to salvage the specified aggregate. The Salvager attempts to repair any inconsistencies it finds on the aggregate.

16.40 zFS aggregate corruption

- If zFS determines a disk is corrupt it issues an abend
  - A PMR should be reported
- If a disk corruption occurs: Run the ioeagslv program
  - On all affected file systems
  - **IOEAGSLV VERIFYONLY** - Checks a file system to:
    - Make sure it is intact
    - Issue messages if the file system is not intact
  - If not intact you can run: **IOEAGSLV SALVAGEONLY**
    - This puts the file system in an inactive state
  - Since there was disk corruption files may be missing
    - IOEAGSLV can fix the metadata for corrupted files
    - Possible for missing files depending on the corruption of the lost data

Figure 16-40 Using the ioeagslv program

Recovering from aggregate corruption

Use the utility's **`-recoveronly`**, **`-verifyonly`**, and **`-salvageonly`** options to indicate the operations the Salvager is to perform on the specified aggregate.

The following rule summarizes the interaction of these options: The salvage command runs recovery on an aggregate and attempts to repair it unless one of the three salvage options is specified; if one of these options is specified, you must explicitly request any operation you want the Salvager to perform on the aggregate.

Specify the -recoveronly option

Use this option to run recovery on the aggregate without attempting to find or repair any inconsistencies found on it. Recovery is the replaying of the log on the aggregate. Use this option to quickly return consistency to an aggregate that does not need to be salvaged; this represents the normal production use of the Salvager. Unless the contents of the log or the physical structure of the aggregate are damaged, replaying the log is an effective guarantee of a file system's integrity.

Specify the -verifyonly option

Use this option to determine whether the structure of the aggregate contains any inconsistencies without running recovery or attempting to repair any inconsistencies found on the aggregate. Use this option to assess the extent of the damage to an aggregate. The Salvager makes no modifications to an aggregate during verification. Note that it is normal for the Salvager to find errors when it verifies an aggregate that has not been recovered; the

presence of an unrecovered log on an aggregate makes the findings of the Salvager, positive or negative, of dubious worth.

Specify the -recoveronly and -verifyonly options

Use these options to run recovery on the aggregate and then analyze its structure without attempting to repair any inconsistencies found on it. Use these options if you believe replaying the log can return consistency to the aggregate, but you want to verify the consistency of the aggregate after recovery is run. Recovering an aggregate and then verifying its structure represents a cautious application of the Salvager.

Specify the -salvageonly option

Use this option to attempt to repair any inconsistencies found in the structure of the aggregate without first running recovery on it. Use this option if you believe the log is damaged or replaying the log does not return consistency to the aggregate and may in fact further damage it. In most cases, you do not salvage an aggregate without first recovering it.

Omit the -recoveronly, -verifyonly, and -salvageonly options

Do this to run recovery on the aggregate and then attempt to repair any inconsistencies found in the structure of the aggregate. Because recovery eliminates inconsistencies in an undamaged file system, an aggregate is typically recovered before it is salvaged. In general, it is good first to recover and then to salvage an aggregate if a system goes down or experiences a hardware failure.

Note: Omit these three options if you believe the log should be replayed before attempts are made to repair any inconsistencies found on the aggregate. Omitting the three options is equivalent to specifying the -recoveronly and -salvageonly options.

Authorization required

If only the **`-verifyonly`** option is included, the issuer needs only READ authority for the specified VSAM LDS (aggregate).

If the **`-recoveronly`** or **`-salvageonly`** option is included, or if all three of these options are omitted, the issuer must have ALTER authority for the specified VSAM LDS.

In addition, the user must be uid 0 or have READ authority to the SUPERUSER.FILESYS.PFSCTL profile in the z/OS UNIXPRIV class.

16.41 Debugging data sets

- IOEFSPRM file - or - IOEPRMxx PARMLIB data sets
  - msg_output_dsn
    - Output messages that come from the ZFS PFS
  - trace_dsn
    - Contains the output of an operator F ZFS,TRACE,PRINT command or the trace output if the ZFS PFS abends
  - With sysplex sharing - can use &SYSNAME for different data sets on each member
  - Specifying IOEPRMxx and IOEFSPRM in a sysplex

```
msg_output_dsn=HLQ.&SYSNAME..ZFS.MSGOUT
trace_dsn=HLQ.&SYSNAME..ZFS.TRACEOUT
```

Figure 16-41 Debugging data sets

Debugging data sets

Depending on whether the IOEFSPRM file is used or the IOEPRMxx PARMLIB member for the zFS parameters, you can define debugging data sets.

msg_output_dsn option

This data set, if specified, contains any output messages that come from the ZFS PFS. This message output data set is only used for zFS initialization messages. This may be helpful for debugging since this data set can be sent to IBM service if needed. The msg_output_dsn is optional. If it is not specified, ZFS PFS messages go only to the system log.

The data set should be preallocated as a sequential data set with a RECFM=VB and LRECL=248 and should be large enough to contain all ZFS PFS initialization messages between restarts. The space used depends on how many zFS initialization messages are issued. A suggested primary allocation is 2 cylinders with a secondary allocation of 2 cylinders. The F ZFS,QUERY command output is written to the system log only.

trace_dsn

This data set contains the output of any operator F ZFS,TRACE,PRINT commands or the trace output if the ZFS PFS abends. Each trace output creates a member in the PDSE. Traces that come from the ZFS PFS kernel have member names of ZFSKNT*nn*. *nn* starts with 01 and increments for each trace output. *nn* is reset to 01 when ZFS is started (or restarted).

Sysplex sharing mode

The msg_output_dsn specification or the trace_dsn specification can be shared across systems in a sysplex if you use system symbols to differentiate the data set names you are using in the IOEFSPRM file or the IOEPRMxx PARMLIB member. For example:

```
msg_output_dsn=HLQ.&SYSNAME..ZFS.MSGOUT
trace_dsn=HLQ.&SYSNAME..ZFS.TRACEOUT
```

IOEFSPRM and IOEPRMxx in a sysplex

It is possible to have multiple IOEPRMxx members and it is also possible to have IOEPRMxx members that are shared among all members of the sysplex, and another IOEPRMxx member that contains options that are specific to a particular sysplex member.

When the IOEFSPRM is specified in the IOEZPRM DD statement of the ZFS PROC, there can only be one IOEFSPRM file for each member of a sysplex.

16.42 zFS hang detection

- If a zFS hang occurs, it usually takes awhile before it is detected
  - The longer it takes before the hang is detected
    - The longer some applications are hanging
    - The less likely the information needed to diagnose the problem is still available
- z/OS V1R8 enhancement
  - Issue a message when hangs are detected
    - IOEZ00524I zFS has a potentially hanging thread
  - New command to find hanging threads
    - F ZFS,QUERY,THREADS
- Use F ZFS,HANGBREAK
  - Attempts to break the hang condition

Figure 16-42 zFS hang detection processing

zFS hang detection

The zFS hang detector, intended for use by IBM Service, monitors the current location of the various tasks processing in zFS. At a set interval, the hang detector thread wakes up and scans the current user requests that have been called into zFS. The hang detector processes this list of tasks and notes various pieces of information that allow it to determine the location of the task. When the hang detector determines that a task has remained in the same location for a predefined period of time, it flags the task as a potential hang and generates message IOEZ00524I or IOEZ00547I to the console. If on a subsequent iteration the hang detector recognizes that this task has finally progressed, it will DOM the message (remove it from the console). If the message is removed by zFS, it means that there was no hang.

Messages IOEZ00524I or IOEZ00547I only indicate a potential hang. Further review of the situation is necessary to determine if a hang condition really exists.

Hang detection monitoring

Perform the following steps when a hang condition occurs:

1. Continually monitor for the following messages:

   ```
   IOEZ00524I zFS has a potentially hanging thread.
   IOEZ00547I zFS has a potentially hanging XCF request.
   ```

   Messages IOEZ00524I and IOEZ00547I are also issued and cleared when slowdown occurs—this is not an indication of a hang, but that things are progressing slowly due to stressful workload or some other issue. Message IOEZ00547I (hanging XCF request) can

indicate an XCF issue. To start investigating, issue D OMVS,W to check the state of sysplex messages/waiters.

2. Issue the F ZFS,QUERY,THREADS command to determine whether any zFS threads are hanging and why.
3. Enter the D A,ZFS command to determine the zFS ASID.
4. Enter F ZFS,QUERY,THREADS at one to two minute intervals for six minutes.
5. Interrogate the output for any user tasks (tasks that do not show the zFS ASID) that are repeatedly in the same state during the time you issued F ZFS,QUERY,THREADS. If there is a hang, this user task will persist unchanged over the course of this time span. If the information is different each time, there is no hang.
6. Verify that no zFS aggregates are in the QUIESCED state by checking their status using the **zfsadm lsfs** or **zfsadm aggrinfo** command. For example, quiesced aggregates display as follows:

   ```
   DCESVPI:/home/susvpi/> zfsadm lsaggr
   IOEZ00106I A total of 1 aggregates are attached
   SUSVPI.HIGHRISK.TEST                            DCESVPI   R/W QUIESCE
   DCESVPI:/home/susvpi/> zfsadm aggrinfo
   IOEZ00370I A total of 1 aggregates are attached.
   SUSVPI.HIGHRISK.TEST (R/W COMP QUIESCED): 35582 K free out of total 36000
   DCESVPI:/home/susvpi/>
   ```

   Resolve the QUIESCED state, continuing to determine if there is a real hang condition. The hang condition message can remain on the console for up to a minute after the aggregate is unquiesced.
7. Check if any user tasks are hung. User tasks will not have the same address space identifier (ASID) as the zFS address space. One or more threads consistently at the same location might indicate a hang (for example, Recov, TCB, ASID Stack, Routine, and State). The threads in the zFS address space with the zFS ASID (for example, xcf_server) are usually waiting for work. It is normal for the routine these threads are waiting in to have the same name as the entry routine.
8. If you are sure there is a valid hang condition and not a slowdown, IBM Support will need dumps of zFS, OMVS and the OMVS data spaces for problem resolution. Obtain and save SYSLOG and dumps of zFS, OMVS and the OMVS data spaces using JOBNAME=(OMVS,ZFS),DSPNAME=('OMVS'.*) in your reply to the DUMP command. If you are running in a sysplex and zFS is running on other systems in the sysplex, dump all the systems in the sysplex where zFS is running, dumping zFS, OMVS and OMVS data spaces. The following is an example of the DUMP command:

   ```
   DUMP COMM=(zfs hang)
   R x,JOBNAME=(OMVS,ZFS),SDATA=(RGN,LPA,SQA,LSQA,PSA,CSA,GRSQ,TRT,SUM),
   DSPNAME=('OMVS'.*),END
   ```
9. If you know which task is hung, issue the CANCEL command to clear the job.
10. As a last resort to recover from what seems to be a hang, issue the F ZFS,HANGBREAK command to break the hang condition. F ZFS,HANGBREAK posts any threads that zFS suspects to be in a hang with an error and can cause abends and dumps to occur, which you can ignore. When you issue F ZFS,HANGBREAK, the hang message can remain on the console for up to one minute. If you question the hang condition or if F ZFS,HANGBREAK does not seem to have resolved the situation, contact IBM Support and provide the dump and SYSLOG information.

16.43 zFS hang detection

- If hangs are detected, the operator can use:
  - F ZFS,HANGBREAK command to attempt to break the hang condition
  - Command posts any threads that zFS suspects to be in a hang with an error
    - Can cause abends and dumps, which you can ignore
    - Hang message can remain on the console for up to one minute
  - With no resolution to situation
    - Contact IBM Support and provide the dump and SYSLOG information

Figure 16-43 zFS hang detection

Hangbreak processing

Hangbreak causes zFS to post with a failure any requests in zFS that are waiting and are suspected of being hung by the hang detector. It will not break threads working on administration tasks. This might break and resolve the hang condition. This should only be used if you suspect that there is a hang involving zFS. The **`modify zfs,query,threads`** operator command can be used to determine if one or more requester threads remain in the same wait over several queries. If this command does not successfully break the hang, you will need to stop or cancel zFS. If you suspect that zFS is in an infinite loop, you will need to cancel zFS.

zFS hang detection

Enter the **`MODIFY ZFS,HANGBREAK`** command to attempt to break the hang condition. This command posts any threads that zFS suspects are in a hang condition with an error and can cause abends and dumps to occur, which you can ignore. After entering this command, the hang message can remain on the console for up to one minute. When the command completes, it issues message IOEZ00025I. However, IOEZ00025I does not mean that the system cleared the hang. Enter **`F ZFS,QUERY,THREADS`** to check the output for indication that the hang is clear. It is possible that the **`MODIFY ZFS,HANGBREAK`** command can clear the current hang condition only to encounter yet another hang. You might have to enter this command several times. If users are hung in the file system, forcefully unmount the file system by entering the **`MODIFY ZFS,ABORT`** command. Contact IBM Support and provide all the dumps and SYSLOG information.

16.44 z/OS UNIX Internet information

http://www.ibm.com/servers/eserver/zseries/zos/unix/bpxa1tun.html

- Performance tips
  - General performance tuning guidelines
  - Use of virtual storage in the HFS
  - Tuning ported UNIX applications
- Other resources
  - Java performance considerations
  - XML Toolkit performance considerations
  - WebSphere Troubleshooter - see the hints and tips
  - Web server tuning: hints and tips
  - z/OS UNIX performance tools

Figure 16-44 Where to find information about z/OS UNIX on the Internet

z/OS UNIX Internet information

Figure 16-44 shows the Web address for information about performance considerations in various areas of z/OS UNIX.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks

For information on ordering these publications, see “How to get IBM Redbooks” on page 826. Note that some of the documents referenced here may be available in softcopy only.

- *z/OS Version 1 Release 2 Implementation*, SG24-6235
- *z/OS Version 1 Release 3 and 4 Implementation*, SG24-6581
- *z/OS Version 1 Release 5 Implementation*, SG24-6326
- *z/OS Version 1 Release 6 Implementation*, SG24-6377
- *z/OS Version 1 Release 7 Implementation*, SG24-6755
- *z/OS Version 1 Release 8 Implementation*, SG24-7265
- *z/OS UNIX System Services z/OS Version 1 Release 7 Implementation*, SG24-7035
- *z/OS Distributed File Service zSeries File System z/OS V1R11 Implementation*, SG24-6580-04

Other publications

These publications are also relevant as further information sources:

- *z/OS UNIX System Services Planning*, GA22-7800
- *z/OZ UNIX System Services User's Guide*, SA22-7801
- *z/OS UNIX System Services Command Reference*, SA22-7802
- *z/OS MVS System Commands*, SA22-7627
- *z/OS UNIX System Services Programming: Assembler Callable Services Reference*, SA22-7803
- *z/OS Using REXX and z/OS UNIX System Services*, SA22-7806
- *z/OS UNIX System Services Messages and Codes*, SA22-7807
- *z/OS UNIX System Services File System Interface Reference*, SA22-7808
- *z/OS MVS System Codes,* SA22-7626
- *z/OS Security Server RACF System Programmer's Guide*, SA22-7681
- *z/OS Security Server RACF Security Administrator's Guide*, SA22-7683-11
- *TSM Using the Backup-Archive Clients*, SH26-4105
- *TSM Installing the Clients*, SH26-4102
- *z/OS Distributed File Service zSeries File System Administration*, SC24-5989
- *z/OS Communications Server: IP Configuration Reference*, SC31-8776

© Copyright IBM Corp. 2011. All rights reserved.

Online resources

These Web sites and URLs are also relevant as further information sources:

- z/OS UNIX Tools and Toys

  http://www.ibm.com/servers/eserver/zseries/zos/unix/bpxa1tun.html

- The ShopzSeries Web address is:

  http://www.ibm.com/software/shopzseries

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Redbooks

ABCs of z/OS System Programming: Volume 9

(1.5” spine)
1.5”<-> 1.998”
789 <->1051 pages

ABCs of z/OS System Programming: Volume 9

z/OS UNIX, TCP/IP installation

zSeries file system, z/OS UNIX security

Shell and programming tools

The ABCs of z/OS System Programming is a 13-volume collection that provides an introduction to the z/OS operating system and the hardware architecture. Whether you are a beginner or an experienced system programmer, the ABCs collection provides the information that you need to start your research into z/OS and related subjects. If you would like to become more familiar with z/OS in your current environment, or if you are evaluating platforms to consolidate your e-business applications, the ABCs collection will serve as a powerful technical tool.

The contents of the volumes are as follows:

Volume 1: Introduction to z/OS and storage concepts, TSO/E, ISPF, JCL, SDSF, and z/OS delivery and installation

Volume 2: z/OS implementation and daily maintenance, defining subsystems, JES2 and JES3, LPA, LNKLST, authorized libraries, SMP/E, Language Environment

Volume 3: Introduction to DFSMS, data set basics storage management hardware and software, catalogs, and DFSMStvs

Volume 4: Communication Server, TCP/IP, and VTAM

Volume 5: Base and Parallel Sysplex, System Logger, Resource Recovery Services (RRS), global resource serialization (GRS), z/OS system operations, automatic restart management (ARM), Geographically Dispersed Parallel Sysplex (GDPS)

Volume 6: Introduction to security, RACF, Digital certificates and PKI, Kerberos, cryptography and z990 integrated cryptography, zSeries firewall technologies, LDAP, and Enterprise identity mapping (EIM)

Volume 7: Printing in a z/OS environment, Infoprint Server and Infoprint Central

Volume 8: An introduction to z/OS problem diagnosis

Volume 9: z/OS UNIX System Services

Volume 10: Introduction to z/Architecture, zSeries processor design, zSeries connectivity, LPAR concepts, HCD, and HMC

Volume 11: Capacity planning, performance management, WLM, RMF, and SMF

Volume 12: WLM

Volume 13: JES3

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-6989-05 ISBN 0738435309