

S/390 PartnerWorld for Developers, ITS0/EFS Project

EFS Systems on a Linux Base: Additional Topics

LVM and raw disks

TCP/IP approaches

Advanced topics



Bill Ogden
Michael Ryan

Redbooks



International Technical Support Organization

EFS Systems on a Linux Base: Additional Topics

October 2003

Archived

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

Archived

First Edition (October 2003)

This edition was based on FLEX-ES Release 6.2.9 and the z/OS Application Development package for z/OS 1.4.

© Copyright International Business Machines Corporation 2003. 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	vii
Trademarks	viii
 Preface	ix
The authors	ix
Become a published author	ix
Comments welcome	x
 Chapter 1. Operational details	1
1.1 Using Red Hat 9.0	1
1.1.1 Linux service	2
1.2 64-bit operation	2
1.3 Terminal Solicitor connection from Linux desktop	3
1.4 CD recording under Linux	3
1.5 ThinkPad Docking Station	4
1.6 S/390 identification	5
1.7 Using a second disk in the Ultrabay	6
1.7.1 Disk layout (AD system with two hard disks)	7
1.8 Cloning ThinkPad hard disks	8
1.9 Multiple FLEX-ES instances	8
1.10 Display PSW and registers	12
1.11 Verify CKD disk	12
1.12 Restarting the MVS console	12
1.13 More about x3270 parameters	13
1.14 Booting from the Ultrabay	15
 Chapter 2. Tuning considerations	17
2.1 Basic memory usage	17
2.2 The vmstat command	18
2.2.1 Importance of Linux swapping	19
2.3 Other monitoring tools	20
2.4 Disk caches	20
2.5 How many emulated volumes	22
2.6 Tuning cachesize	22
2.7 Memory tuning	23
2.8 Maximum allocations	25
 Chapter 3. Networking	27
3.1 Sharing an adapter	27
3.1.1 Multiple adapters	27
3.2 Basic TSO networking	28
3.2.1 Typical, basic configuration	29
3.3 Networking limitations	32
3.4 FLEX-ES OSA channels	33
3.5 LAN device types	33
3.6 Using a local router	34
3.6.1 The problem	34
3.6.2 One solution	35
3.6.3 Router administration	37

3.7 NFS files	42
3.8 Using address E40	42
3.9 Remote resources	43
3.9.1 Working example	44
3.9.2 Practical operation	47
3.9.3 Common problems	48
3.9.4 Performance	48
3.9.5 Comments	48
3.10 Operating FLEX-ES remotely	49
3.10.1 Sample configuration	49
3.10.2 IPL with single remote MVS console	50
3.10.3 Exposures	51
3.10.4 Remote operation with two z/OS consoles	52
3.11 SNA adapter number	54
Chapter 4. Raw disk devices	57
4.1 Background	57
4.1.1 Raw device	58
4.2 Basic implementation	58
4.2.1 Creating an LVM partition	59
4.2.2 Initial LVM creation	60
4.2.3 Raw devices	61
4.2.4 Loading the AD system	63
4.2.5 FLEX-ES resource definitions	63
4.2.6 Typical FLEX-ES startup	64
4.3 Production implementation	66
Chapter 5. S/390 volume distribution	69
5.1 S/390 processing environment	70
5.2 Linux or UnixWare processing environments	70
5.2.1 Standard dd command	71
5.3 Practical distribution creation	71
5.3.1 Raw devices	72
5.3.2 Standard Linux files	72
5.3.3 Compression	73
5.4 A test	73
5.5 Backup and restore considerations	74
5.5.1 Using ftp for backups	75
Chapter 6. Printers and readers	77
6.1 JES2 customization	77
6.2 FLEX-ES printing options	78
6.3 Background	78
6.4 Using a PC printer	79
6.4.1 Print flow	79
6.4.2 Implementation	80
6.5 Emulated card reader	83
Chapter 7. FLEX-ES definitions/operation	85
7.1 System definitions	85
7.2 Resource definitions	88
7.2.1 Emulated control unit types	88
7.2.2 Emulated device types	88
7.2.3 Typical resource definitions	88

7.2.4 Cloned devices	93
7.2.5 Compiled files	93
7.3 Common rules	94
7.4 The resadm command	94
7.5 CLI commands	95
Chapter 8. z/OS base sysplex on z/VM	99
8.1 Test environment	99
8.2 Scope	100
8.2.1 Who should read this paper	100
8.2.2 What a base sysplex is	100
8.2.3 What a Parallel Sysplex® is	101
8.2.4 Why run a base sysplex	101
8.3 FLEX-ES system definitions	101
8.4 FLEX-ES resource definitions	103
8.5 z/VM guest ZOS1	106
8.5.1 Working allegiance	109
8.5.2 z/VM emulated CTCs	109
8.6 z/VM guest ZOS2	109
8.7 System IPL	111
8.8 Preparing z/OS for sysplex operation	112
8.9 DASD volumes	112
8.10 System and sysplex naming	113
8.11 Setting the system names at IPL	114
8.11.1 Other IPL parameters	115
8.11.2 PROCLIB changes	119
8.12 Couple data set preparation	120
8.13 VTAM customization	123
8.14 Setting up OMVS	124
8.15 Starting the sysplex	127
8.16 Operating the sysplex	127
8.17 Tuning	128
8.18 Conclusion	129
8.19 Notices	130
Chapter 9. Installing Linux for S/390	131
9.1 FLEX-ES and Linux definitions	132
9.2 Other preparations	133
9.3 Shell scripts for FLEX-ES	134
9.4 Start installation	135
9.5 Main installation steps	137
9.6 First use	141
9.7 Routine use	142
9.8 Left undone	143
Chapter 10. Tape usage	145
10.1 SCSI tape setup	145
10.1.1 Linux tape commands	146
10.1.2 Device characteristics	146
10.2 FLEX-ES FakeTape on z/OS	147
10.3 Tape resource options	148
10.4 Using a 4mm tape drive	149
10.5 Using an Overland T490E	149
10.6 Using the scsitfake program	150

Chapter 11. SNA over Ethernet	153
11.1 Network lab	153
11.2 Selecting and initializing the adapters	154
11.3 A word about MACADDRS	155
11.4 Selecting VTAM configuration information	156
11.5 Developing the adapter and device matrix	156
11.6 Defining the FLEX-ES resources	157
11.7 MVS device definitions	158
11.8 Adding an XCA device	164
11.9 Defining the SSCPs, the subareas, and the network	167
11.10 CDRM major nodes	169
11.11 Defining the VTAM XCA major nodes	170
11.11.1 Defining the SAPADDRS	170
11.11.2 XCA major node definition for zos1	171
11.11.3 XCA major node definition for z370	172
11.11.4 XCA major node definition for z232	172
11.11.5 XCA major node definition for zсна	173
11.12 Bringing up a link	174
11.13 TSO cross-domain logon	175
11.14 RTP activation	175
11.15 JES2 network job entry	176
11.15.1 NJE verification	179
11.16 FLEX-ES resource definitions	181
11.16.1 The base sysplex	181
Chapter 12. FAQ	197
Related publications	201
IBM Redbooks	201
Other publications	201
Online resources	201
How to get IBM Redbooks	201
Help from IBM	202
Index	203

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. 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 IBM's application programming interfaces.

Trademarks

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

eServer™
Redbooks (logo) ™
AFP™
CICS®
DB2®
DFS™
ECKD™
ESCON®
IBM®
IMS™
IPDS™
MVST™
Netfinity®
NUMA-Q®

OS/2®
OS/390®
Parallel Sysplex®
PartnerWorld®
PR/SM™
PR/SM™
RACF®
Redbooks™
S/390®
Sequent®
ServeRAID™
Sysplex Timer®
System/390®
ThinkPad®

Ultrabay™
VM/ESA®
VSE/ESA™
VTAM®
WebSphere®
xSeries™
z/Architecture™
z/OS™
z/VM™
zSeries™
Lotus Trademarks begin here:
Notes®

The following terms are trademarks of other companies:

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

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

C-bus is a trademark of Corollary, 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.

SET, SET Secure Electronic Transaction, and the SET Logo are trademarks owned by SET Secure Electronic Transaction LLC.

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

Preface

This IBM® Redbook describes additional or advanced techniques for using FLEX-ES (a product of Fundamental Software, Inc., Fremont, California) with z/OS™ and various z/OS packages. Basic installation and use of FLEX-ES is described in *EFS on a Linux base; Getting Started*, SG24-7007. Both books are intended primarily for members of the IBM S/390® PartnerWorld® for Developers (PWD) organization and for internal IBM users of the ITSO/EFS package.

The authors

Bill Ogden is a retired IBM Senior Technical Staff Member, still working on favorite projects with the International Technical Support Organization, Poughkeepsie Center. He specializes in small S/390 and z/OS systems, writes extensively, and teaches ITSO workshops relating to these areas. He is the originator of the ITSO/EFS package for internal IBM use. Bill has been with the ITSO since 1978.

Michael Ryan, Fundamental Software, Inc., brings considerable expertise to FSI. Before joining FSI, he worked as an MVT, SVS, and MVS™ systems programmer and MVS support center specialist. He has developed software products for z/OS and for Microsoft Windows, OS/2®, Linux, and UnixWare. He has managed both product development and field support organizations and now develops advanced enablement projects and provides second-level FLEX-ES, Linux, and UnixWare support for FSI.

We especially thank the following people for their contributions to this project:

Gary Ehemann, Fundamental Software, Inc., who helped us understand many additional aspects of FLEX-ES and Linux.

Scott Carter, Fundamental Software, Inc., who is a proofreader and corrector of errors.

Mike MacIsaac, IBM S/390 Linux group in Poughkeepsie, who help with the SuSE installation.

Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll team with IBM technical professionals, Business Partners and/or customers.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

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 Redbooks™ to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

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

ibm.com/redbooks

- ▶ Send your comments in an Internet note to:

redbook@us.ibm.com

- ▶ Mail your comments to:

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

Operational details

This chapter discusses several common details of EFS usage, in no particular order. There are several references to the *Getting Started* publication throughout this document. The full reference is *EFS Systems on a Linux Base: Getting Started*, IBM order number SG24-7007.

1.1 Using Red Hat 9.0

The *Getting Started* publication said that you must always rebuild the Linux kernel if you want to use FLEX-ES with the Red Hat 9.0 release. This was correct for FLEX-ES 6.2.10, which was the current version when that publication was written.

Subsequent FLEX-ES releases (such as 6.2.14, the current version at the time of writing) do not always require kernel rebuilds when used with Red Hat 9.0. The kernel rebuild is still required for SMP usage, or if you have an unusual LAN adapter, or if you have ServeRAID™ modules, or if you have updated LVM modules. This portion of the *Getting Started* text still applies.

Red Hat 9.0 is especially relevant if you have a PC with USB 2.0 ports. Red Hat 8.0 (the most commonly used Linux base for FLEX-ES at the time of writing) does not work with USB 2.0 ports unless you apply maintenance and rebuild it. IBM T40 ThinkPads are a common example of a PC with USB 2.0 ports. If you have a T40, for example, we suggest that you use Red Hat 9.0. Furthermore, we suggest that you do not install Red Hat (or other) maintenance on the 9.0 system unless you know that such maintenance will not affect FLEX-ES installation or operation.¹

¹ Subsequent maintenance, if it involves a new kernel, may cause the kernel to be out of synchronization with the kernel source. This creates problems for FLEX-ES installation and operation. If you (directly or via a maintenance package) install a new kernel, you *must* install matching kernel source and reinstall **msgmgr**.

1.1.1 Linux service

We have generally recommended that you do not install Linux service. This is a simple answer for a complex issue and this recommendation may not always be correct. A more complex answer is:

- ▶ If you rebuilt the Linux kernel using the kernel source provided by FSI, then you should not accept or install any kernel updates. Rebuilding the kernel with the FSI-provided kernel source is *required* for SMP operation. *You* must determine if any maintenance packages you are considering will alter the kernel (or the kernel source).
- ▶ Otherwise, you may install updates provided that you always keep the kernel source on your system synchronized with the kernel you are using. If these change, you must reinstall the FLEX-ES `msgmgr` module.
- ▶ This implies that you investigate the updates and maintenance provided by Red Hat, read their *up2date* offerings, and so forth, to determine what maintenance may affect the kernel and to verify that the kernel source will be synchronized after the update.
- ▶ If you do accept a new kernel as part of an update, *you* may need to verify that the appropriate functions and modules were included in that kernel. This information is not conveyed by a kernel level number.

If you are comfortable with these requirements or if there is maintenance that you *must* have (such as an urgent security fix), then you can work with Red Hat (or other) maintenance and update materials. If you are not comfortable with these requirements (or do not have the time to keep up to date with the status of the available fixes) we suggest you do not accept or apply Linux maintenance.

1.2 64-bit operation

FLEX-ES release 6.2.0 (and later releases) provides operation at Architectural Level Set 3 (ALS3). This is commonly known as 64-bit operation. You must have an appropriate FLEX-ES license key file to use this mode. There are separate controls in the license key file for the release level and whether ALS3 operation may be used.

Version 6.2.0 (and later) can operate in ALS2 (31-bit) or ALS3 (64-bit) mode (assuming that this option is enabled in your license key file). This is controlled by the *instset* specification in the FLEX-ES definition file:

```
instset(esa)      # 31-bit operation
instset(z)        # zArchitecture == ALS3
```

After changing a FLEX-ES definition file, you must recompile the file with the `cfcomp` command and restart any S/390 system image that is using that definition. Remember that z/OS automatically exploits 64-bit mode if it detects the presence of zArchitecture. The *instset(z)* parameter indicates the use of zArchitecture.

Performance implications

The current FLEX-ES system is implemented as a 32-bit application, running on 32-bit Intel processors. Emulating zArchitecture 64-bit operation is considerably slower than emulating S/390 or zSeries 31/32-bit operation. This performance impact is quite minor for typical z/OS usage because, even when running in ALS3 mode, only a small portion of z/OS operation actually uses 64-bit instructions and registers.

The situation is quite different when running Linux for zSeries. This system uses 64-bit instructions almost 100 percent of the time.² This means that Linux for zSeries (64-bit operation) is considerably slower than Linux for S/390 (31/32-bit operation) on EFS systems.

1.3 Terminal Solicitor connection from Linux desktop

The examples in the redbooks usually involve a shell script that starts specific x3270 sessions on the desktop. For example (in a shell script):

```
.....  
x3270 -model 3 -keymap pc -port tn3270 localhost:L701 &  
.....
```

This line starts an x3270 session connected to the device with terminal name L701 in your FLEX-ES resources. Instead (or in addition) you can start x3270 session(s) that are connected to the FLEX-ES Terminal Solicitor. From a Linux command line window enter:

```
$ xhost +localhost                (You need this command only once)  
$ x3270 -model 3 -keymap pc -port tn3270 localhost &
```

From the Terminal Solicitor you can select any unused 3270 session defined in your FLEX-ES resources. Only defined 3270 devices associated with a *started instance* are shown by the Terminal Solicitor. After starting the resource manager (with **resadm**) but before you start an instance (probably using a shell script such as sh14A), there will be no sessions available to the Terminal Solicitor.

Once you start an instance (by running sh14A, for example), a Terminal Solicitor screen will show the unassigned 3270 sessions associated with that instance. These are shown before (and after) you IPL the S/390 operating system. If you want to connect to the MVS master console via the Terminal Solicitor, you would do this before you IPL. Connections to a VTAM® terminal (for TSO, perhaps) could be performed either before IPL (where you would obtain a blank screen until VTAM starts) or after IPL.

Be careful with the **xhost** command. Do not use the form **xhost +** unless you understand the security implications.

1.4 CD recording under Linux

While not directly related to EFS or FLEX-ES,³ creating CDs under Linux can be useful in many cases. This is often described in general Linux documentation, so the discussion here is very brief. One method for *burning* a CD involves two Linux commands: **mkisofs** and **cdrecord**. We found both these commands on the second CD of Red Hat Linux 8.0 and installed them with the **rpm** command. Together, they required about one megabyte of disk space. We used the commands with a ThinkPad® CD-RW/DVD drive.

Linux must treat the CD-RW unit as a pseudo-SCSI device in order to use these commands. If the CD-RW unit is present (in the Ultrabay™) when Linux is installed, the parameter "hdc=ide-scsi" is automatically added to the Linux boot parameters. This parameter causes Linux to treat devices in the Ultrabay (/dev/hdc) as pseudo-SCSI devices.⁴ This parameter is discussed in the *Getting Started* publication. We assume you have this parameter present if you are using a ThinkPad and want to burn CDs.

² This is not completely true because 31/32-bit applications can be run under Linux for zSeries.

³ No FLEX-ES emulated devices support CD output.

⁴ If this parameter is present, a hard disk in the Ultrabay slot cannot be used in the normal manner. We usually have two boot options on our ThinkPad system; one has this parameter and the other does not.

To create a CD, in a very simple implementation, we simply move all the files we want to include on the output CD into the same directory, such as /tmp/burn, and proceed as follows:

```
# su                                (Need to be root)
# cdrecord -scanbus                 (Do we see the CD-RW unit?)
# mkdir /tmp/burn
# cp /fake/222222 /tmp/burn/222222 (Move desired files to this directory)
# cp ..... /tmp/burn.....        (Copy the files we need)
# ls -al /tmp/burn/*               (Verify the desired files are present)
# mkisofs -R /tmp/burn/* | cdrecord -v fs=6m dev=0,0 -
```

We must be *root* to use these commands. The **cdrecord -scanbus** looks for SCSI or pseudo-SCSI devices. If this fails, you should verify that you have a CD-RW unit installed and (if using the Ultrabay of a ThinkPad) have the “hdc=ide-scsi” parameter in your boot parameters. If it still fails, and if you rebuilt your Linux kernel, you may be missing the needed SCSI options in the kernel.⁵

We arbitrarily created a directory (/tmp/burn) and copied the files we wanted to burn on the CD into this directory. To keep the **mkisofs** command simple, we used simple (single-component) names for the files. The **mkisofs** command is piped to the **cdrecord** command. Do not forget the last “minus sign” in the **cdrecord** command.

We found that burning CDs was useful for distributing tape volumes in FakeTape format and for backing up 3390 volumes that had been compressed with **gzip** or **tar**. See “S/390 volume distribution” on page 69 for a discussion of CD usage for backup and distribution.

Several users have reported using **xcdroast**, which is also provided with Red Hat Linux. This is a more sophisticated interface and automatically handles some of the more obscure **mkisofs** parameters.

1.5 ThinkPad Docking Station

Note: The IBM Web pages indicate that IBM does not support the 2631 Docking Station or the 2877 Docking Station for Linux. Based on this, FSI does not support them when using FLEX-ES under Linux. The following informal discussion relates our experiences, but does not imply that the docking stations are supported for Linux.

We installed a docking station (IBM 2631) with our ThinkPad T23 machine. It included the following functions:

- ▶ An AC power supply is built into the docking station. When the ThinkPad is in the docking station, it is being charged.
- ▶ An additional Ultrabay is present and can be used at the same time as the Ultrabay in the ThinkPad.
- ▶ Two additional PCMCIA slots are present and can be used at the same time as the two PCMCIA slots in the ThinkPad. (Our normal EFS usage did not involve any PCMCIA adapters and we did not try these additional slots.)
- ▶ All of the ThinkPad ports are replicated on the back of the docking station. This does not increase the number of ports, but it is a convenience if many external devices must be connected or disconnected when you move the ThinkPad. (Separate keyboard and mouse connectors are provided, so a “Y” cable is not needed if an external keyboard and mouse are used with the docking station.)

⁵ You could rebuild your kernel again, or simply boot an older (standard Red Hat) kernel and use it temporarily.

- A half-size PCI adapter slot is present.

We can make the following general statements about the docking station in the context of our EFS systems:

- The additional Ultrabay slot is very convenient. This allows us to have two hard disks installed (one internally in the ThinkPad, one in the ThinkPad Ultrabay) and have the CD-ROM drive in the docking station Ultrabay slot.
- The docking station contains a half-size PCI slot. We installed a SCSI adapter (PCI Fast/Wide Ultra SCSI Adapter, IBM part number 02K3454) and connected a SCSI tape drive (formerly used on a P/390 system). The next time we booted Linux (Red Hat 8.0), we were given a choice to configure the adapter, ignore it, or do nothing.

We configured it and used it with FLEX-ES as /dev/sg0. (See “Tape usage” on page 145 for more details about using SCSI tape drives.)

When we removed the ThinkPad from the docking station and booted Linux, we were offered an option to deconfigure the adapter (since it was still in the docking station and no longer connected to the ThinkPad), ignore the change, or do nothing. We said to ignore the change and did not receive the same message when we booted again.

We noticed that the FLEX-ES dongle could not be recognized when connected to the top USB connector on our docking station. It worked with both USB connectors on the ThinkPad. This may have been a problem with our docking station.

Again, note that this Docking Station usage with Linux is not supported by IBM or FSI.

1.6 S/390 identification

The S/390 instruction STIDP stores an 8-byte field:

```

0APLSSSSIIII0000
| | | | | +---> the four zeros are constants
| | | | | +-----> machine type (see notes below)
| | | | | +-----> arbitrary number set by manufacturer (serial number)
| | | | | +-----> LPAR number
| | | | | +-----> processor number
+-----> the hex digits 0A are normally constants

```

The processor type field for an EFS system has one of these values:

```

1245 is the processor type for selected FLEX-ES systems
1247 is the processor type for a system obtained through PWD
1246 is the processor type for other FLEX-ES systems

```

The SSSS field is normally set to a unique value for each license and functions as a CPU serial number. The LPAR number defaults to zero, but you can set it through use of the `lparnum(n)` statement in the FLEX-ES system definition for each instance. This is used to provide unique identifiers when running multiple instances of FLEX-ES and where unique STIDP identifiers facilitates resource sharing. An example would be the VSE lockfile mechanism.

The processor number corresponds to one of the processor numbers in your system definitions. For a single processor, this is typically zero. A CLI instruction is available to display the CPU ID data:

```
flexes> display cpuid
```

These fields are set by the FLEX-ES license key and, with the exception of the processor number and LPAR number, cannot be changed.

1.7 Using a second disk in the Ultrabay

We obtained a second hard disk drive along with the mounting tray (IBM part number 08K6068) needed to use it in the Ultrabay of our T23 ThinkPad. We removed the CD-ROM drive and installed the second hard disk while the ThinkPad was turned off. We turned power on and let Linux boot. We then determined the Linux identity of the second drive:

```
# cat /proc/partitions
major minor #blocks name (... more data ....)
 3      0      nnnnn hda
 3      1      nnnnn hda1
 3      2      nnnnn hda2
..      ..      ....
22      0 31253040 hdc 0 0 0 0 0 0 0 0 0 0
```

The first hard disk (internal in the ThinkPad) is `/dev/hda`; looking at the above listing, we see a second hard disk as `/dev/hdc`.⁶ This drive (as expected for a new disk) had no partitions. (This will not work if you booted with the `hdc=ide-scsi` parameter; if you have this parameter, Linux will not see the hard disk in the Ultrabay. See the *Getting Started* publication for more information about this parameter.)

We can create whatever file systems we want on the new disk. In our case, we simply wanted a single file system (using all the space on the disk) and we wanted it mounted as `/s391`. We noted that IBM, like most other manufacturers, uses decimal numbers to describe disk capacity (1M = 1,000,000) while Linux utilities typically use power-of-two numbers (1M = 1,048,576). Using Linux numbers, the capacity of the 32 GB disk we used was about 29 GB.

We partitioned the disk with `fdisk` and formatted it:

```
# /sbin/fdisk /dev/hdc
n                                (fdisk option to add a partition)
Command action: p              (A primary partition)
Partition number: 1
First cylinder (1-66144): 1     (Default = 1)
Last cylinder (1-66144): 66144  (Make maximum size)
p                                (Print the partition table)
  Device Boot   Start   End   Blocks    Id  System
  /dev/hdc1     1     66144  31252998   83  Linux native
(if the ID is not 83 = Linux, use the t option to change it)
t                                (fdisk option to change partition ID)
Partition number: 1
Hex code: 83                    (83 is Linux)
p                                (fdisk option to list partitions)
w                                (fdisk option to write partition table and end)
# mkdir /s391                   (Create a mount point, if not already done)
# /sbin/mke2fs /dev/hdc1        (Format a file system)
# mount /dev/hdc1 /s391
# chown flexes:flexes /s391
```

The initial `fdisk` display indicates the units of measurement it uses for “cylinders.” This varies for different disk versions, so you need to do your allocations using the appropriate sizes for your disk.

⁶ As best we know, the Ultrabay is always `/dev/hdc` on a ThinkPad.

We used the new disk space to confirm that it worked, and then edited `/etc/fstab` to cause the new file systems to be automatically mounted at boot time. (There are no unwanted effects if the second hard disk is not present when Linux is booted; the file systems on that disk are simply not mounted.) We added a line at the end of `fstab`:

```
# vi /etc/fstab
LABEL=/          /          ext2 defaults 1 1
LABEL=/boot      /boot      ext2 defaults 1 2
/dev/fd0         /mnt/floppy auto noauto,owner 0 0
LABEL=/s390      /s390      ext2 defaults 0 0
none            /proc      proc defaults 0 0
none            /dev/pts   devpts gid=5,mode=620 0 0
/dev/hda7        swap       swap defaults 0 0
/dev/hdc1        /s391      ext2 defaults 0 0
```

Your disk sizes and partition numbers will probably differ from this example, but the general process should be similar.

Instead of modifying `fstab`, you could issue a `mount` command for the file systems on the second hard disk after booting Linux. This is easy if you create a shell script to do the mount. We created `/usr/flexes/rundir/mountit` with the following line:

```
mount /dev/hdc1 /s391
```

After booting Linux (and after switching to the `rundir` directory) we issued the command `sh mountit` if we had a second hard disk installed in the Ultrabay. (In this example we have only one file system on the second disk, but you can easily extend this method to mount any number of file systems.)

If you use LVM, we suggest it would be better to create a second volume group for the second hard drive. This permits you to use the system without the second hard disk present. If you do this, you may need to use the `vgscan` command when the second hard disk is again present.

1.7.1 Disk layout (AD system with two hard disks)

There are many ways to use the second disk. In our case, we wanted a minimum IPLable system on the internal disk, and we placed the lesser-used S/390 volumes on the second HDD.

Internal Hard Disk

```
Linux root
Linux swap partition
/s390 partition
S4RES1  3390-3 IPL volume
S4RES2  3390-3 more system libraries
OS39M1  3390-3 z/OS VSAM data sets, spool, paging, and so forth
S4USS1  3390-3 UNIX System Services HFS files
WORK02  3390-1 a local work volume
```

Second Hard Disk

```
/s391 partition
WORK01  3390-1 another local work volume
WORK03  3390-1 another local work volume
S4DIS1  3390-3 system DLIBs
S4DIS2  3390-3 system DLIBs
S4DIS3  3390-3 system DLIBs
S4DIS4  3390-3 system DLIBs
S4DB21  3390-3 DB2
S4CIC1  3390-2 CICS
S4IMS1  3390-2 IMS
```

```
S4WAS1    3390-3 WebSphere
S4WAS2    3390-3 more WebSphere
```

This provides us with an IPLable system (including our local work data sets) even if the second disk is not installed.

We created two FLEX-ES definition files and two FLEX-ES startup shell scripts: One for operation with only the internal hard disk and one for operation with both hard disks.

1.8 Cloning ThinkPad hard disks

Our operational ThinkPad hard disk often had dual boot, with both Microsoft Windows and Linux available to boot. The Windows partition had a number of products installed and the Linux partitions had FLEX-ES and z/OS installed. Our most common disk environment used a 60 GB ThinkPad disk with the following partitions:

- ▶ 13 GB for Windows
- ▶ 3 GB for Linux (root and all files except /boot)
- ▶ 80 MB for /boot (part of Linux)
- ▶ 512 MB for a Linux swap partition
- ▶ 40 GB for /s390 (a Linux file system where we installed z/OS volumes)

We found it convenient to copy the complete disk, including the master boot record (with the partition table). We placed a new hard disk, exactly the same size as the internal disk, in the UltraBay slot. We then booted Linux, logged in as *root*, and used the command:

```
# dd if=/dev/hda of=/dev/hdc bs=1M
```

In this command, the internal ThinkPad hard disk is our input device (/dev/hda), the hard disk in the Ultrabay slot is our output device (/dev/hdc), and we copy using 1 megabyte blocks. The complete 60 GB disk copy took an hour. We could then move the new drive (from the Ultrabay) to the internal disk position and boot from it. This was the easiest way we found to obtain a *complete* backup of our system.

Using this technique, the source disk is a running Linux system (which is performing the **dd** command.) The target system will have a “point in time” copy of a running system. When it is booted, it will appear that it is a reboot of a system that crashed. On the first boot, it will use **fsck** to recover disk metadata. If you then shut down Linux properly, subsequent boots will be clean.

1.9 Multiple FLEX-ES instances

You can define multiple S/390s by defining multiple FLEX-ES *system* definitions. You can run multiple S/390 systems, one at a time, by simply creating multiple shell scripts that name the appropriate *syscf* file in the **flexes** command. You can run multiple S/390 systems at the same time (*multiple instances*) if you have enough server memory and have defined your FLEX-ES resources appropriately.

Only one resource definition (*rescf* file) can be active on a server, so it must define all the resources needed by all the S/390 instances. Each S/390 instance must be started with a unique system definition (*syscf* file). You would normally start a separate CLI window, with its *flexes* prompt, for each instance. Some care is needed to use the right *flexes* windows when controlling the S/390 instances. The CLI command **set prompt** can be used to change the *flexes* prompt to something more meaningful for each instance.

The resource definitions and the multiple system definitions can be in separate files or all in a single file. The file or files must be compiled by the **cfcomp** command to produce the required syscf files and the rescf file.

The publication *NUMA-Q® Enabled for S/390: Technical Introduction*, SG24-6215, describes the simultaneous operation of fully-defined z/OS, VM/ESA®, and VSE/ESA™ instances. A NUMA-Q system (also named x/Series 430 EFS) is larger than a typical EFS system, but the details for running multiple FLEX-ES instances are the same.

Both the **gnome** and **kde** desktops (in Linux) provide multiple screen (or “panel”) sessions. There are icons for four panels at the bottom of the default gnome screen, for example. In a sense, using the multiple panels makes your Linux desktop larger. This function is very useful if you run multiple S/390 instances. Each instance will probably have one or two x3270 screens and a Linux command line (probably with the *flexes>* prompt) associated with it. Even two S/390 instances makes a very full desktop if all the windows are in the default panel. It is easy to place the x3270 and command windows for each S/390 instance in a different desktop panel. We find this much easier to use, with less confusion as to which window we are using at any instant.

Simple example of two instances

This example uses the AD CD-ROM z/OS 1.4 system for a S/390 instance and the stand-alone (single volume) z/OS 1.4 system that is distributed with the AD system as the second instance. We use minimal definitions in order to make the basic principles clearer.

Our systems are illustrated in Figure 1-1 on page 10. We have two emulated S/390 processors, S14Y and S14Z. Each has a 3174 control unit with local 3270 terminals. Note that the two 3174 control units use the same addresses, starting at 700. (We did this because both the systems we intend to IPL have system consoles defined at address 700.)

One system has a 3172 control unit (at address E20) for external LAN connections. (The emulated 3174s also use LAN connections for their 3270 terminals, but this is transparent to the S/390 instances.)

A single 3390 control unit, with five 3390 volumes, is shared by both systems in a typical shared DASD arrangement.⁷ We will IPL the S14Y system from address A80 (which is the IPL volume for the AD system) and the S14Z system from address A84 (which contains the stand-alone z/OS volume).

⁷ Fortunately, both the AD system and the stand-alone system specify shared DASD in their IODF entries for these 3390 addresses.

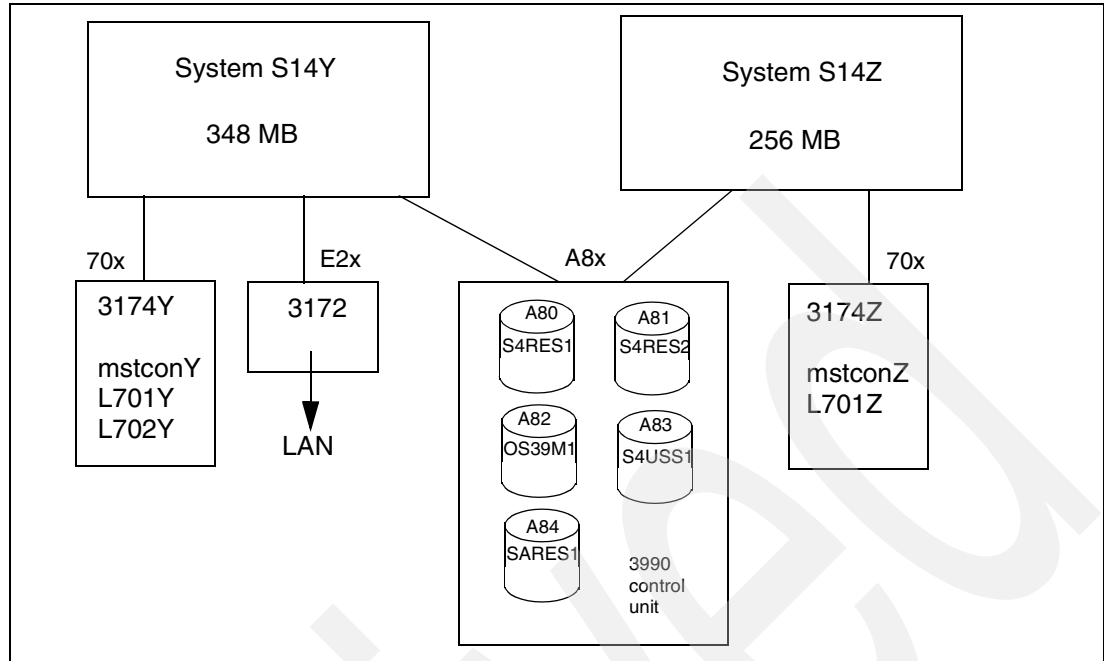


Figure 1-1 Two system instances

We elected to place all the FLEX-ES definitions in a single file named defYZ, as follows:

```

system S14Y:
memsize(393216)
cachesize(4096)
instset(z)
cpu(0)
channel(0) local
channel(1) local
channel(2) local
cu devad(0xA80,5) path(2) resource(CU3990)
cu devad(0x700,3) path(0) resource(CU3174Y)
cu devad(0xE20,2) path(1) resource(CU3172)
end S14Y

system S14Z:
memsize(262144)
cachesize(4096)
instset(z)
cpu(0)
channel(0) local
channel(2) local
cu devad(0xA80,5) path(2) resource(CU3990)
cu devad(0x700,2) path(0) resource(CU3174Z)
end S14Z

resources R14YZ:
CU3990: cu 3990
interface local(2)
device(00) 3390-3 /s390/S4RES1
device(01) 3390-3 /s390/S4RES2
device(02) 3390-3 /s390/OS39M1
device(03) 3390-3 /s390/S4USS1
device(04) 3390-3 /s390/SARES1
end CU3990

```

```

CU3174Y: cu 3174
interface local(1)
device(00) 3278 mstconY
device(01) 3278 L701Y
device(02) 3278 L702Y
end CU3174Y

CU3174Z: cu 3174
interface local(1)
device(00) 3278 mstconZ
device(01) 3278 L701Z
end CU3174Z

CU3172: cu 3172
interface local(1)
options 'ipaddress=192.168.0.111,adapternumber=0'
device(00) 3172 eth0
device(01) 3172 OFFLINE
end CU3172

end R14YZ

```

This definition file contains two system definitions and one resource definition. We intend to use a ThinkPad with 1 GB memory. The memsizes we selected (384 MB and 256 MB) will fit easily in the 1 GB system. Notice the interface definition for the 3990 control unit. It specifies *local(2)* because there are two channel interfaces (from the two S/390 instances) to this control unit. We compiled the definitions with the **cfcomp** command; this produced S14Y.syscf, S14Z.syscf, and R14YZ.rescf.

A very important concept is that DASD *control units* can be shared. Emulated volumes are connected to only one control unit. *Never* connect an emulated volume (/s390/S4RES1, for example) to multiple control units.

We created two shell scripts (arbitrarily named shY and shZ) to help us start our two S/390s. File shY contained:

```

flexes S14Y.syscf
x3270 -model 3 -keyboard pc -port tn3270 localhost:mstconY &
x3270 -model 3 -keyboard pc -port tn3270 localhost:L701Y &
flexesccli localhost S14Y

```

File shZ contained:

```

flexes S14Z.syscf
x3270 -model 3 -keyboard pc -port tn3270 localhost:mstconZ &
x3270 -model 3 -keyboard pc -port tn3270 localhost:L701Z &
flexesccli localhost S14Z

```

Each shell script starts its appropriate S/390 instance (S14Y or S14Z), starts two local 3270 sessions and then starts the interactive **flexesccli** program for that S/390 instance.

When we start our systems we will have a very busy Linux desktop unless we use separate desktop panels for the different instances. If we do not, the default desktop panel will have four 3270 windows, two flexesccli windows, and a root window that we use for **resadm** commands.

We started our system this way:

```

Linux command window 1: (in the default desktop panel)
$ su                                     (Must be root for resadm commands)

```

```

# cd rundir                                (We keep all our files here)
# resadm -s R14YZ.rescf                     (Start resource manager)
Linux command window 2: (in the default desktop panel)
$ cd rundir
$ sh shY                                    (Start system S14Y)
flexes> s prompt flexesY>                 (Set new prompt to avoid confusion)
flexesY> ip1 a80 0a8200                    (Appropriate IPL parameters)
Linux command window 3: (in a different desktop panel)
$ cd rundir
$ sh shZ                                    (Start system S14Z)
flexes> s prompt flexesZ>                 (Set different prompt)
flexesZ> ip1 a84 0a84sa                    (Appropriate IPL parameters)

```

The 3270 sessions (on the Linux desktop) have the FLEX-ES session name in the top lines. This helps prevent confusion about which session is associated with which instance. (We placed Y and Z letters in the 3270 session names for this purpose.)

1.10 Display PSW and registers

You may sometimes need to display the S/390 PSW. You can do this from the *flexes* prompt provided by the CLI program:

```

flexes> d psw                             (displays psw and next instruction, if possible)
flexes> d g                               (displays general purpose registers)

```

1.11 Verify CKD disk

The **ckdchk** command can be used to verify the internal format of an emulated S/390 CKD disk:

```
# ckdchk -a /s391/WORK01                    (Use your correct file name, of course)
```

The **-a** flag indicates that a full range of checks should be performed. Error messages from this utility should be taken seriously. However, the following message might be expected from volumes containing VM minidisks, or that once were used as minidisks:

```

$ ckdchk -a /s390/OS39M1
FSIDU166 [cyl = 99 head = 14 rec = 1] Record cylinder number on "OS39M1"
does not match home address cylinder number (hacyl: 0x0063 rec cyl: 0x0d70)

```

We also get these messages for some of the z/OS AD volumes, due to the use of VM systems during a step used when building new AD releases.

If you are using raw device interfaces, the command would be similar to:

```
$ ckdchk -a /dev/raw/A82
```

where this node has been linked to the appropriate logical volume by a previous **raw** command.

1.12 Restarting the MVS console

You may inadvertently disconnect your z/OS console. This is especially easy to do with the x3270 emulator. If z/OS has no other full-function operator console available, it will attempt to use the *system console* (also known as the *hardware console* or the *HMC console*). On a S/390 or zSeries™ machine, this is a limited-function console window available through the

Support Element (SE) or Hardware Management Console (HMC). It is *limited* because it works in a pseudo-typewriter manner. z/OS will attempt to use it only for critical messages. Nevertheless, messages can flood it and recovery can be problematical.

With FLEX-ES, the indications of this situation are z/OS operator output messages appearing on the window with the *flexes>* prompt. You must press Enter (in this window) after each z/OS operator message to restore the *flexes>* prompt.

The first recovery step is to reconnect a 3270 session to the proper address. We assume this is address 700 in this discussion and that it has the device name *mstcon* in the FLEX-ES resource definitions. You can reconnect to this session from the Linux desktop by opening a command window and using a command such as:

```
$ x3270 -model 3 -keymap pc -port tn3270 localhost:mstcon &
```

Alternatively, you could connect to the mstcon session through the FLEX-ES Terminal Solicitor. (If FLEX-ES detected that the emulated 3270 session was dropped, the console should be in the list of available sessions in the Terminal Solicitor.)

Once a 3270 emulator session is connected to address 700, you must then reactivate z/OS console operation on the session. If your only operational console is the *system console* (at the *flexes>* prompt), you must first activate it for z/OS command input. (Remember to press Enter to obtain the *flexes>* prompt after any z/OS messages are displayed.) Use the command **hwc** to send a message to z/OS through this interface:

```
flexes> clear messages                                (If needed to clear z/OS flood)
flexes> hwc V CN(*),ACTIVATE
```

The **ACTIVATE** command is critical. You cannot issue any other z/OS operator command until this command is accepted. Remember to press Enter after z/OS display messages. Once the **ACTIVATE** is functional, try to vary the z/OS console address online:

```
flexes> hwc V 700,ONLINE
```

If this does not work, try the following:

```
flexes> hwc V 700,OFFLINE                                (If no good, try FORCE)
flexes> hwc V 700,OFFLINE,FORCE                          (Reply to various messages)
flexes> hwc V 700,ONLINE
```

Once device 700 is online, issue these commands:

```
flexes> hwc V 700,CONSOLE
flexes> hwc V 700,MSTCONS
```

When the normal console is active again, you can prevent additional use of the system console by:

```
flexes> hwc V CH(*),DEACTIVATE
```

If you have another z/OS console (or if you have a TSO session where you can enter operator commands through SDSF or some other method), you can issue appropriate commands through this interface, of course, instead of using the **hwc** interface.

1.13 More about x3270 parameters

Many of our sample shell scripts contain the following lines:

```
xmodmap -e 'keysym Alt_L = Alt_L Meta_L'
xset fp+ /usr/flexes/fonts
xset fp rehash
```

These lines are included primarily for compatibility with the UnixWare version of FLEX-ES. Unless you have an unusual keyboard or unusual key requirements, they are not needed for Linux-based FLEX-ES. You can safely omit them from your startup shell scripts.

A number of z/OS functions, such as the first messages after IPLing, appear in a dark blue color. This color is difficult to read on many screens. You can change this color as follows:

```
# cd /usr/lib/X11/app-defaults
# cp X3270 X3270old           (Keep a spare copy)
# vi X3270
.....
X3270.colorScheme.default: \
  black blue red pink \      (Change blue --> white)
  green turquoise yellow white \
  black blue3 orange purple \
  paleGreen paleTurquoise2 grey white \
  white black dimGrey \
  4 2 1 15
```

Find this section in the file and make the indicated change. Instead of *white* you might use *orange* or one of the other colors. It is the plain *blue* that is so difficult to read. You will need to use **:w!** to save your changes because this is a read-only file.

The *Getting Started* publication suggested making several key changes to this file. In particular, it suggested changing the large Enter key on the PC keyboard to mean *NewLine* and the right-hand Ctrl key to mean *Enter*.

Here is a slightly larger set of key changes you might want to consider. These changes are in the same file as the color table:

```
...
x3270.keymap.pc: \
  Meta<Key>d:      Redraw()\n\
  Alt<Key>d:       Redraw()\n\
  Meta<Key>Return: Newline()\n\
  Alt<Key>Return:  Newline()\n\
  <Key>Return:     Newline()\n\      (Change this line, as shown)
  !Shift<Key>Tab:  BackTab()\n\
  ...
  <Key>BackSpace:  BackSpace() Delete()\n\      (Change this line, as shown)
  ...
  Alt<Key>Home:    Clear()\n\
  <Key>KP_Enter:   Enter()\n\      (Add this line)
  <Key>Control_R:  Enter()\n\      (Add this line)
  <Key>Control_L:  Reset()\n\      (Add this line)
  <Key>Pause:      Clear()\n\      (Add this line)
  <Key>End:        EraseEOF()\n\      (Add this line)
  <Key>:           Default()
x3270.keymap.pc84: \
```

The effect of the original table plus these changes provides the following:

Function	Key(s) (Multiple ways to do many of these functions)
Clear	Pause; Alt c; Alt Home
Enter	Control_R; Keypad Enter
PA1	Alt 1
PA2	Alt 2
Newline	Return; Alt Return
Reset	Control_L; Alt r
Insert	Insert; Alt i
Delete	Delete

Backspace	Backspace (deletes characters as it backspaces)
F13	Shift F1; Alt F1
EraseEOF	End; Alt f
EraseInput	Alt n
SysReq	Alt s
lpr	Print

Individual Linux users (using x3270, for example) can have their own keymap files. You can do this if you wish. We did not do this because we wanted to have the same key mapping for all x3270 sessions.

1.14 Booting from the Ultrabay

We built a Linux (and FLEX-ES) system on the internal disk of a T23 ThinkPad. This was a simple Linux installation and no dual boot function was installed. We also had another hard disk containing Windows.

We placed the Windows hard disk in the internal ThinkPad drive and placed the Linux disk in the Ultrabay. We verified that Windows still worked. We then changed the ThinkPad BIOS to boot from the Ultrabay disk (if present). If no hard disk is present in the Ultrabay, it boots from the internal (Windows) disk.

To change the BIOS, we followed this path:

```
F1 (during power up) --> Startup --> Boot --> Hard Drive
-Hard Drive
  IC25T060ATCS05-0-(PM)      (The 60 GB Windows disk)
  IC25T048ATDA05-0-(SM)      (The 48 GB Linux disk)
Select the first disk; F5 to switch order; F10 to save and exit
```

When Linux was booted this way, the swap partition was unusable but the remaining functions worked correctly. Our `/etc/fstab` contents were as follows:

LABEL=/	/	ext2	defaults	1 1
LABEL=/boot	/boot	ext2	defaults	1 2
none	/dev/pts	devpts	gid=5,mode=620	0 0
none	/proc	proc	defaults	0 0
none	/dev/shm	tmpfs	defaults	0 0
LABEL=/s390	/s390	ext2	defaults	1 2
/dev/hda5	swap	swap	defaults	0 0

We edited `/etc/fstab` and changed the swap entry to `/dev/hdc5` (instead of `hda5`) and rebooted. This produced a clean boot, with a working swap partition. (You may have `ext3` instead of the `ext2` file system types shown in the example.)

Of course, if we move the Linux disk back to the internal drive bay, we will need to change `fstab` again. This will be necessary if we need the Ultrabay for another use while running Linux. (The best example is that the Ultrabay is needed for a CD-ROM drive. If you have another way of attaching a CD-ROM drive and diskette drive, you might never need to move the Linux hard disk back to the internal drive.)

A small inconvenience is that Windows insists on discovering the FSI dongle every time it is booted. You can simply unplug the dongle provided you remember to replace it when you want to boot Linux and run FLEX-ES.

Another inconvenience is that if you boot without the hard disk in the Ultrabay, the BIOS automatically resets the internal drive as first in the boot list. When you reinstall the Linux

disk in the Ultrabay you must change the BIOS settings again. Other ThinkPad models may work differently.

Tuning considerations

This chapter discusses several aspects of tuning a FLEX-ES system. Most of the traditional S/390 tuning techniques are still valid, but a number of new considerations are also involved.

Many of the tuning possibilities involve the use of memory, real and virtual. This can be a somewhat complex area and we suggest you read this complete chapter before beginning any tuning steps.

2.1 Basic memory usage

The *memsize+essize+cachesize*11+DASD cache* total value (in your FLEX-ES definitions), when translated to bytes of storage, approximates the amount of Linux virtual storage needed to run an instance of S/390 emulation. If you emulate two S/390 systems (at the same time), you will need to add the values for each of the two emulated systems.⁸ You can emulate more S/390 instances, but each one will require more memory.

A key principle is that the Linux system *should never be forced to page* when running S/390 emulation.⁹ Linux might perform considerable paging when getting started, but once a S/390 instance is started and the system *working set* is established, the Linux paging rate should be zero. (S/390 operating systems running in the emulated S/390 might have substantial paging rates; this is not the point at issue here.) This almost always means that the server (ThinkPad or xSeries™) real memory *must be considerably larger* than the emulated real memory of all the S/390 instances that are active at any one time.

In principle, the Linux memory needed to emulate a S/390 is “just” virtual memory to Linux and might exist in a much smaller real memory used by Linux. In practice, this does not work. Your server should have enough memory so that all the S/390 requirements (*memsize + essize + cachesize*11 + DASD cache*) fit in your *real* server memory, without requiring Linux paging. If you assume 100 MB for Linux, FLEX-ES programs, TCP/IP operation, and so forth, then a 512 MB ThinkPad might be used as follows:

Rough guess for Linux, FLEX-ES, etc

100 MB

⁸ Some of the DASD cache memory may be shared among multiple emulated S/390s.

⁹ We use the term paging instead of swapping. Swapping is the traditional Linux term, however, this has a different implication in traditional MVS systems.

S/390 memsize	256 MB
S/390 essize	64 MB
FLEX-ES instruction cache * 11	22 MB
default DASD cache for 10 volumes (approx)	10 MB

	452 MB

This leaves a reasonable margin in a 512 MB system.¹⁰ The 100 MB for Linux is just a guess. The *working sets* (as seen by Linux) of the emulated S/390 memory (central and expanded), the FLEX-ES instruction cache, and most of the DASD cache will normally be their full sizes. The working set of Linux itself and its many system processes are much harder to determine.

We do not have a definite formula for computing the largest S/390 memory you can emulate in a specified real memory size. As a rough starting point, we suggest you can make your emulated S/390 memory equal to the PC real memory size minus 256 MB.¹¹ That is, reserve 256 MB of real memory for Linux, FLEX-ES modules, FLEX-ES caches, and other Linux processes. See “Maximum allocations” on page 25 for a discussion of maximum sizes.

If you have a small memory machine (less than 512 MB), you might consider running Linux without X windows. That is, use simple terminal interfaces instead of **gnome** or **kde**. You can run FLEX-ES in this environment, although you cannot use x3270. Your 3270 emulator sessions would need to be on other machines.

Do not confuse Linux paging (swapping) with z/OS or z/VM™ paging. Linux paging is to be avoided absolutely, while z/OS or z/VM paging can be tolerated up to the performance limit of your disk subsystem.

2.2 The vmstat command

The traditional UNIX command for monitoring swapping¹² rates, **sar**, is not available in base Linux distributions. The **vmstat** command can be used instead. For example, the command **vmstat 10 2** would run **vmstat** with 10 seconds between reports and quit after 2 reports.

```
# vmstat 10 2
procs          memory  swap          io      system          cpu
 r  b  w  swpd  free  buff  cache  si  so  bi  bo  in  cs  us  sy  id
 0  0  0    0 329156 28768 70156  0  0   4   1 105 196  1  0  99
 0  0  0    0 329156 28768 70156  0  0   0   0 105 196  1  0  99
```

You should refer to your Linux documentation or **man** listings for more complete information about **vmstat**. Very briefly, the key fields are documented as:

- Procs
 - r: Number of processes waiting for CPU time
 - b: Number of processes in uninterruptible sleep
 - w: Number of processes swapped out but otherwise runnable
- Memory
 - swpd: The amount of virtual memory used (in the swap file) (kB)

¹⁰ A substantial margin is important if you use simple Linux files for emulated DASD volumes. To work reasonably well, this mode assumes that sufficient free Linux memory is available to act as an effective disk cache. We do not have measurements for this, but suggest that at least 50 MB free Linux memory should be available for this purpose.

¹¹ Again, this statement assumes you are using simple Linux files for DASD emulation. If you use raw devices (as you should do in any production environment), you should need much less reserve memory for Linux because all the emulated DASD caches are under your direct control and allocation.

¹² In S/390 terms, we would say *paging* rates. In older contexts, *swapping* has a different meaning than *paging*. In modern Linux systems, *swapping* appears to have exactly the same meaning as *paging*.

- free: The amount of idle memory (kB)
- buff: The amount of memory used as buffers (kB)
- Swap
 - si: Amount of memory swapped in from disk (kB/s)
 - so: Amount of memory swapped out to disk (kB/s)
- IO
 - bi: Blocks received from a block device (blocks/s)
 - bo: Blocks sent to a block device (blocks/s)
- System
 - in: The number of interrupts per second (including clock interrupts)
 - cs: Number of context switches per second
- CPU
 - us: User time (percent of CPU time)
 - sy: System time (percent of CPU time)
 - id: Idle time (percent of CPU time)

We are particularly interested in swap (paging) rates. The unit of measurement is kilobytes per second.

The Linux swapping rate is meaningful (for our discussion) only in a steady-state condition with a typical S/390 workload. Linux swapping while booting, or while starting FLEX-ES, is not relevant. Unusual S/390 work, such as CLPA processing or very unusual disk access patterns (affecting disk caches), might temporarily drive Linux into swapping. This is not good, but can probably be tolerated for short periods.

There are several GUI monitors that include swapping (paging) data, and the **top** command also displays such data. Remember that these commands also consume real memory (while they are running) and so distort their own measurements. In a large memory system (1 GB or more) this distortion is probably not important.

2.2.1 Importance of Linux swapping

Why is the Linux swapping rate so important? A reasonable analogy is CICS® paging in a z/OS system. A system with many TSO users might have sustained paging rates of hundreds of pages per second (on a larger S/390) with no ill effects, but CICS on the same system would require a paging rate close to zero. The problem is that the whole address space (CICS, for example) is placed in *wait* when a page fault occurs. Placing CICS¹³ in *wait* causes all the CICS users serviced by that address space to *wait* while the page fault is resolved. A page fault in a TSO user address space causes only that one user to wait.

FLEX-ES operation is close to the CICS analogy. A Linux page fault in a key FLEX-ES process may cause the whole emulated S/390 instance to wait until the Linux page fault is resolved.

Remember that a page fault in z/OS has a very different effect than a page fault in Linux. For one thing, the S/390 page fault is seen only by the FLEX-ES emulation program—it is an emulated page fault. It is handled, by z/OS, as a S/390 page fault. If it occurs in a TSO user address space or a batch address space, it affects only that address space. If it occurs in a CICS address space, it affects all the users of that CICS. This is business as usual for z/OS.

¹³ Modern CICS systems ameliorate this situation in various ways; the description here should be regarded as conceptual.

The key message is that you should adjust your FLEX-ES system parameters (emulated S/390 memory, disk caches, instruction cache) to avoid Linux swapping. Defining a smaller emulated S/390 memory size may increase z/OS paging. Of course, it would be nice to avoid any paging, but z/OS paging is much less damaging than Linux swapping and your trade-offs should always be in this direction. You can juggle disk cache versus instruction cache versus S/390 memory allocations for your best performance. Simply be careful not to push Linux into swapping.¹⁴

2.3 Other monitoring tools

There are many Linux monitoring tools available. In addition to `vmstat`, the `top` command and the GUI display available through `gnome System Tools->System Monitor` can be interesting. Most of the information displayed by these (such as memory allocated to various processes) should be fairly static during FLEX-ES operation. However, PC utilization will vary with the S/390 workload and with Linux background processing. This is nicely displayed by the `System Monitor` graphics.

The `top` command is started from a command line window; the window where you started `resadm` may be convenient for this. Enter `q` to exit from `top`.

2.4 Disk caches

If you are using simple Linux files for your emulated S/390 volumes (as described in the *Getting Started* publication), on a lightly-loaded small system, you can probably ignore disk cache tuning. Linux will automatically use free memory as a disk cache and (if there is sufficient free memory) this is often good enough for a small, non-production FLEX-ES system. (There are negative effects for this mode of disk usage; these are described in section 2.1.5 of the *Getting Started* publication.)

If you are using the raw disk interface for emulated disk volumes (described in “Raw disk devices” on page 57), we *strongly* recommend that you work with the disk cache parameters described here. These can have *major* effects on the performance of your system. Any larger FLEX-ES system, especially those considered as production systems, should use the raw interfaces.

FLEX-ES automatically caches 15 tracks of data for each emulated 3390 or 3380. You can adjust this in three ways:

- ▶ Specify a different number of tracks to cache for a particular emulated drive.
- ▶ Specify a different number of tracks to cache at the control unit level; excess tracks (above those needed for the specified or default cache for each device on the control unit) will *float*, as needed, among all the devices on the control unit.
- ▶ Use the `writethroughcache` parameter to force a different operation of the cache (on a device level). The default operation uses a *writeback* cache technique.

Here is an example that uses all three options:

```
(resource definitions)
....
c3990A: cu 3390
interface local(1)
options 'trackcachesize=150'
```

¹⁴ Again, we stress that “Linux paging” refers to steady-state operation after S/390 emulation is started. Linux booting or FLEX-ES startup may cause Linux paging and we are not concerned with this temporary effect.

```

device(00) 3390-3 /usr/flexes/links/A3s1
device(01) 3390-3 /usr/flexes/links/B3s1 devopt 'trackcachesize=5'
device(02) 3390-3 /usr/flexes/links/C3s1 devopt 'trackcachesize=45'
device(03) 3390-3 /usr/flexes/links/D3s1 devopt 'trackcachesize=30,writethroughcache'
device(04) 3390-1 OFFLINE devopt 'trackcachesize=3'
end c3990A

```

This is a bit complex. The five devices defined will ask for $(15 + 5 + 45 + 30 + 3 =) 98$ tracks of cache. (Device (00) does not specify a cache size and defaults to 15 tracks.) The control unit definition specifies 150 tracks of cache. This is $(150 - 98 =) 52$ more tracks than needed by individual device caches, and the 52 tracks will be a *floating* cache. The floating cache is managed by internal FLEX-ES logic. Each 3390 track is about 57 KB, so the 150 tracks of cache will require about 8.8 MB of server storage.

Cache is normally allocated for an offline device, since you might perform a FLEX-ES **mount** command to use the device. If you are certain you will not use the device you can specify a cache of three tracks. Three tracks is the minimum allowed cache size; the smallest recommended number is five. (The minimum track cache size was stated as zero in earlier redbooks and earlier FLEX-ES documentation; this was incorrect.)

If you specify a control unit cache size of less than the sum of the individual device caches, the specified control unit cache size is ignored.

FLEX-ES defaults to *writeback* cache operation. This allows the S/390 disk write channel operation to complete when the data is in the FLEX-ES cache. A *writethrough* operation means that the S/390 channel operation for a disk write is not complete until the FLEX-ES write operation is complete. In either case, a copy of the data is retained in the FLEX-ES cache for possible future use.

The meaning of these statements depends on which disk interface you are using:

- ▶ If you are using simple Linux files for emulated disks, then FLEX-ES considers a disk write complete when Linux accepts the data to be written. At this point the data is in Linux disk buffers and will be written to disk at an indeterminate time in the future. This largely defeats the purpose of the *writethrough* mode. We suggest that there is little purpose in using this mode when using Linux files for emulated volumes.
- ▶ If you use raw devices for emulated disks, then FLEX-ES actually controls the disk I/O operations.¹⁵ A *writethrough* cache provides considerably lower performance than a *writeback* cache, but it provides higher integrity when using raw disk interfaces. It might be considered for a S/390 volume containing DB2® log data, for example.

In *writethrough* mode, FLEX-ES writes the disk data as soon as possible (and the S/390 I/O operation is not reported complete until the data is actually written). In *writeback* mode, FLEX-ES writes data from its disk caches when the data on a track has not been modified for about 10 seconds.

Do not misuse *writethrough* mode. It should be rarely needed.

If you have enough server memory, you can specify large disk caches for better overall system performance. There is obviously room for considerable tuning here, by manipulating cache sizes at the device and control unit level. You can use the **d ckdchestats cuu** command to monitor cache effectiveness:

```

flexes> d ckdchestats A80
ADDRESS  READS  WRITES  CACHE HITS  DEDICATED LINES  LINES USED
0a80     2880   182    1811 (65%)         15         15 (97%)
0a81     2880   182    1811 (70%)         15         15 (93%)
0a82     2880   182    1811 (91%)         15         15 (100%)

```

¹⁵ This statement ignores the effects of RAID adapter caches and buffers and disk drive buffers.

The command can specify any device address on the control unit. The *dedicated lines* column indicates the number of dedicated (non-floating) cache tracks. A **clear ckdccachestats cuu** command can be used to reset the statistics. The *lines used* column includes any floating cache being used.

You can display the definition of an emulated disk with the **d devstate cuu** command:

```
flexes> d devstate A87
Filename: /s390/OS39HA State: OPEN, READY
Options: trackcachesize=30
```

2.5 How many emulated volumes

On a small EFS system all your S/390 DASD volumes are emulated on a small number of “real” disks—in the most extreme case, a single disk on a ThinkPad. Therefore, why should you bother to create many emulated volumes? Why not simply place all the S/390 data sets on the smallest possible number of emulated 3390 volumes? This approach reduces the amount of real disk space needed and may serve well in many cases.

However, remember that each emulated volume automatically has 15 tracks of cache associated with it. (This number can be changed through FLEX-ES definitions.) This cache can be exceptionally effective in certain cases, such as the JES2 checkpoint data set or coupling data sets for a basic sysplex.

Access to the physical disks being used (such as a single, relatively slow ThinkPad drive) can be substantially delayed by task(s) that access the disk in patterns that are not helped by the caches. If critical data sets are on the same emulated volumes, access to those data sets is also delayed. The most common case involves the JES2 checkpoint data set. This is readily apparent because JES2 complains (via MVS operator messages) whenever there is a delay in accessing the checkpoint data set. Placing the checkpoint data set on a separate volume may resolve the problem via the separate cache for the separate volume. This costs memory, but memory on a small EFS system may be more available than disk performance.

Another factor is z/OS handling of multiple I/O requests to a volume that is shared (or marked as shared in the IODF). IOS schedules only one I/O at a time for such volumes.¹⁶ Whether or not this impacts your operation depends on your disk access patterns and the effectiveness of your cache. In a busy system this single-thread I/O can cause bottlenecks; for example, a request to the JES2 checkpoint (which is probably in cache) might occur just after a request to the same volume for a track that is not found in the cache. In this case, the JES2 checkpoint operation must wait for real disk I/O.

There is no easy answer for this type of tuning. In the case of a ThinkPad system used for personal education (and probably using Linux files instead of raw devices for emulated disks), probably no action is needed. In a complex environment, perhaps running a basic sysplex, disk volume layout (and the number of volumes) is likely to be an important tuning element.

2.6 Tuning cachesize

The FLEX-ES *cachesize* parameter (in the system definition section) specifies the number of S/390 bytes that should be reflected in the FLEX-ES instruction cache. FLEX-ES, in effect, compiles S/390 instructions into Intel instructions in order to execute them. This involves overhead that can be reduced by saving the compiled instructions. This is the purpose of the instruction cache. The underlying structure is complex and proprietary to the FLEX-ES

¹⁶ FLEX-ES does not emulate PAV hardware at this time.

product. The instruction cache requires an average of 11 bytes for each S/390 instruction byte cached. A S/390 LA instruction, for example, requires 44 bytes in the cache.

You can monitor the effectiveness of the instruction cache with the **d cachestats** command:

```
flexes> d cachestats
Cache hits:      20465153/212253141  96%
Cache misses:    760188/212253141   4%    <== monitor this number
...
```

The number of *cache misses* in this report is the critical information. If this number is above about 4 percent, you should increase your *cachesize* parameter. (But never increase it to the point where Linux starts paging!) If the number is considerably less than 4 percent, you might make better use of your server memory by increasing the defined S/390 memory size, defining more S/390 expanded memory, or increasing disk cache sizes. The CLI command **clear cachestats** will reset the statistics. The counters overflow, and you will probably need to reset the statistics before monitoring them.

FSI has not documented the meanings of the other statistics in this report.

Prior to FLEX-ES Release 6.2.12, the **d cachestats** command, by default, displays information for processor zero. If you have more than one processor enabled for S/390 emulation and want to display statistics for more than processor zero, you need to first issue a **set cpu** command:

```
flexes> set cpu 0 1                (display information for two processors)
flexes> d cachestats
```

The set cpu command

The CLI command **set cpu**, used in the previous example, affects more than the display of cache statistics. It affects most of the CLI commands that could be directed to multiple emulated S/390 CPUs (when they exist). For example, if you have a FLEX-ES instance with three emulated S/390 CPUs and enter:

```
flexes> set cpu 2
flexes> ip1 A80 0A92CS
```

then emulated S/390 CPU number 2 would be IPLed instead of the default CPU number zero. For z/OS this makes no practical difference. For z/VM, the IPLed CPU is used as the *Master Processor*.

If you want to display the general registers in CPU 0 and 1, you could:

```
flexes> set cpu 0 1
flexes> d g
```

If you want to see the registers only in CPU 1, you could **set cpu 1** and issue the appropriate display command.

2.7 Memory tuning

There are several ways you can tune FLEX-ES. Adding more processors, or faster processors, or more memory, or faster disks, or other hardware changes can be considered a form of tuning. Within a given hardware system (with a fixed number of processors licensed

for FLEX-ES), there are also some tuning parameters. Most of these are related to memory usage.

You can control key factors in memory allocation. Some of these include:

- ▶ How much memory is defined for S/390 memory?
- ▶ How much memory is defined for S/390 expanded storage? (If you are running z/OS with 64-bit architecture, you should have no expanded storage defined.)
- ▶ How much memory is allocated for disk caches? This can be a very important tuning factor.
- ▶ How much memory is defined for the FLEX-ES instruction cache? This cache is relatively small in terms of total system memory, and tuning usually consists of allocating enough memory to keep cache misses under 4 percent.

In all cases, you must manage memory so that there is no Linux paging during FLEX-ES operation. This primary goal must always be remembered throughout any other tuning.

Disk caches

If you use normal *Linux files* for emulated disks, then the normal Linux disk cache mechanism becomes involved. This mechanism will use any free memory as a disk cache. The size of this cache varies over time. The attractive factor is that it is automatic. Also, in this case the sizes of the FLEX-ES disk caches are not an important factor. However, using Linux files for emulated DASD has several bad effects that are described in Chapter 2 of the *Getting Started* book. It produces erratic disk performance (due to unmanaged interactions with the FLEX-ES disk caches), involves more processor overhead, and has no tuning controls.

If you use *raw devices* for emulated disks (instead of normal Linux files), the Linux disk cache mechanism is not used. Disk I/O is performed directly from the FLEX-ES disk caches. The FLEX-ES disk caches are described in “Disk caches” on page 20. The size of the FLEX-ES disk caches becomes an important tuning factor when the raw device interfaces are used.

Talking in terms of 3390 devices, each emulated volume has a cache of 15 tracks by default. This works, but is certainly not oriented for best performance. You can define larger cache sizes. You can define the cache size for each volume, and you can define additional cache for each emulated disk control unit. Cache defined at the control unit level is used, as required, for all the volumes within that control unit. As an example, a larger, highly-optimized FLEX-ES system might have 1500 tracks of cache for an emulated 3990 control unit. (This corresponds to about 87 MB of PC storage.)

You can juggle your cache allocations to favor certain volumes or control units. You might want two volumes containing your DB2 data base to have 500 tracks of cache each, but some seldom-used TSO volumes might have the default 15-track size. If you have no need to tune at the volume level, we suggest you assign cache at the control unit level.

We cannot tell you how to tune your system. If you have a simple system (perhaps the AD CD-ROM system on a ThinkPad, using a single emulated 3990 control unit) and you are using raw device interfaces, we suggest you start with 500 tracks cache for the control unit. This will use less than 32 MB storage and will produce a considerably faster system than one with only the default cache sizes.

If you are not using raw device interfaces (that is, you are using normal Linux files for your emulated disk volumes) you probably should *not* allocate extra FLEX-ES disk caches. These will subtract from the memory available for Linux disk caches, but will not reduce the processing overhead associated with Linux disk caches.

Expanded storage

How much expanded storage, if any, should you define? We cannot provide a general answer for this question. Logic might indicate that storage is better used as central storage instead of expanded storage, but there is long-established folklore that OS/390® works better if some expanded storage exists. (This is often related to “paging” of TSO swap sets into expanded storage, but many other elements enter the discussion.) Also, some DB2 advocates recommend the use of expanded storage.

A simple z/OS system, such as the AD system, works quite well without any expanded storage. Long-term directions for zArchitecture will have no expanded storage. Most of the exercises and tests we do for this IBM Redbook series do not use expanded storage. z/OS running in “64-bit mode” does not use expanded storage.

We have seen cases where OS/390 performance (under FLEX-ES) was substantially improved by reducing expanded storage and increasing disk cache sizes. (This comment applies only when you are using raw disk devices, as should be the case for production systems.)

2.8 Maximum allocations

There are four large memory “users” in a typical FLEX-ES implementation:

- ▶ S/390 central storage.
- ▶ S/390 expanded storage (if used).
- ▶ FLEX-ES disk caches (which typically involve large amounts of memory if you use raw device interfaces).
- ▶ Linux free memory. Some cushion is needed to avoid Linux paging. Also, if you use simple Linux files for emulated volumes (and if you do not specify large FLEX-ES disk caches) then this memory functions as an automatic disk cache (not visible to FLEX-ES) that has a substantial impact on performance.

We cannot directly control the amount of Linux free memory and we ignore this element in the following discussion.

The theoretical maximum sizes are:

- ▶ 2 GB for S/390 central storage
- ▶ 16 Terabytes for S/390 expanded storage
- ▶ Approximately 478 MB (8191 tracks) cache for each emulated DASD control unit

However, Linux considerations reduce these numbers. Unfortunately, there is no simple formula for workable maximum sizes. Also, this is an area that is sensitive to the level of the Linux kernel being used and may substantially change with future Linux kernels. The following discussion involves virtual memory sizes and parameters. You must always remember the most important tuning rule: *do not let Linux page during S/390 emulation!* Your real memory size must be large enough (or your virtual memory assignments small enough) to prevent Linux paging.

FLEX-ES obtains a separate shared segment (of virtual memory) for each of the following: central storage, expanded storage, and disk cache for each emulated DASD control unit. The maximum size of a shared segment is affected by the following:

- ▶ The value of Linux variable *shmmax*. This sets the size of the largest allowed shared segment request. The largest effective value for 32-bit Linux is 2 GB and FLEX-ES automatically sets this value. (Larger values can be set but are ineffective.)

- ▶ The current Linux restriction that a process can have no more than 2 GB of shared memory.¹⁷
- ▶ The value of Linux variable *shmall*, which is the maximum permitted sum of the sizes of all shared segments in the system. You can display the value of *shmall* with the command `cat /proc/sys/kernel/shmall`. This value was 2097152 in our small RH8.0 systems.¹⁸
- ▶ The shared segment used for an instance of S/390 central storage is also used for several small FLEX-ES work areas, a stack, and some Linux control blocks. This reduces the size available for central storage to something less than the theoretical maximum of 2 GB.

The result is that, with current FLEX-ES and Linux releases, the largest possible value for `memsize()` (with typical trackcache and no expanded storage) for a single instance is less than 2000 MB. Using storage for the following purposes will reduce the largest possible value for `memsize()`: additional trackcache, expanded storage, or large amounts of processor cache and/or trace buffers.¹⁹ This limit applies to each FLEX-ES instance running concurrently. (The average EFS owner probably runs only a single FLEX-ES instance at any given time, but experimental work may involve several instances.) If you run multiple instances, each is limited to about 1900 MB.

For a production system (which is assumed to use raw devices for emulated DASD), the primary balance that is needed is between processor memory (`memsize + essize`) and memory for disk caches. We suggest that a typical, middle-range system might have between 32 MB and 128 MB assigned to disk caches, leaving approximately 1800 MB as the largest possible S/390 memory size. This memory could be split between central storage and expanded storage in any way appropriate for the user.

If you attempt to define the largest possible system, a little experimentation may be necessary to find the exact maximum size possible.

Real memory

The above discussion involved virtual memory. If you wish to run the largest possible S/390 instance, we suggest you will need 3 GB real memory. This assumes that you follow our advice to permit no Linux paging during S/390 emulation. This implies that real memory is available to back up all virtual memory used by FLEX-ES plus some additional memory for Linux processes. More real memory will not improve S/390 emulation performance or increase the maximum size of a S/390 instance.

If you run multiple instances, each with maximum memory allocation, you will need an additional 2 GB of real memory for each instance.

Comment

Do not assume that defining the largest possible central storage size (or central storage + expanded storage) will produce the best performance. Disk cache sizes (and these can be tuned in different ways for multiple emulated control units) are a key factor. You may need to spend some time and thought tuning a FLEX-ES system for best performance.

¹⁷ This is not exactly true. A process can *map* a maximum of 2 GB at any instant. For practical purposes with FLEX-ES, this creates a 2 GB limitation.

¹⁸ This is expressed in 1K units and does not create a serious limitation.

¹⁹ As an experiment, we were able to start a FLEX-ES instance with 1984 MB of central storage. This was with a minimal trackcache. However, we suggest 1900 MB is a good planning number for the maximum central storage. Remember that expanded storage is directly subtracted from the maximum central storage size. For example, defining `essize(512)` will reduce the maximum possible `memsize()` by at least 512 MB.

Networking

This chapter contains topics related to S/390 networking and to Linux networking.

3.1 Sharing an adapter

FLEX-ES under Linux can share a single LAN adapter between Linux and emulated S/390 interfaces. You can use multiple IP stacks and multiple SNA stacks on the same adapter at the same time. This differs from FLEX-ES on UnixWare and from P/390-based emulation.

For multiple IP stacks, each stack must have a different IP address. For practical purposes, they will probably be on the same subnet. A typical EFS system will have TCP/IP active in both Linux (for Terminal Solicitor connections, among other uses) and z/OS. Each will have its own IP address, but both can use the same physical LAN interface.²⁰

If you have multiple SNA stacks (a less common situation), each stack must have its own Service Access Point (SAP) number.

3.1.1 Multiple adapters

There is no requirement to share a LAN adapter between Linux and FLEX-ES use. You can have multiple LAN adapters on your server and devote some of them solely to FLEX-ES use. To do this, the adapter must be *open* (with a proper IP address such as 10.0.0.1) *before* FLEX-ES can use it for an emulated control unit. This means you must define all adapters to Linux and give them IP addresses.

Do you need multiple adapters? For routine levels of TCP/IP usage we suggest you do not need separate adapters for FLEX-ES usage. In a normal FLEX-ES system, the only routine Linux usage should be for Terminal Solicitor connections and these usually involve small amounts of data.

²⁰ It is best to use the devopt 'ipaddress=ddd.ddd.ddd.ddd' parameter in your FLEX-ES definition. This reduces the overhead for the S/390 IP stack handling. However, if you are using z/VM with Linux guests and are using the *proxyarp* function of z/VM TCP/IP, you must omit this parameter.

You can use multiple adapters for your emulated S/390. There are many ways you might arrange this. A simple example is:

```
...
cu devad(0xE20,2) path(1) resource(CU3172A)
cu devad(0xE22,2) path(1) resource(CU3172B)
...
CU3172A: cu 3172
    interface local(1)
    options 'ipaddress=192.168.0.120'
    device(00) eth1                                #LAN adapter A
    device(01) OFFLINE
end CU3172A

CU3172B: cu 3172
    interface local(1)
    options 'ipaddress=9.12.6.120'
    device(00) eth2                                #LAN adapter B
    device(01) OFFLINE
end CU3172B

...
(In the TCPIP PROFILE for z/OS)
...
DEVICE LCS1 LCS 0E20
DEVICE LCS2 LCS 0E22
LINK ETH0 ETHERNET 0 LCS1
LINK ETH1 ETHERNET 0 LCS2
HOME 192.168.0.120 ETH0
HOME 9.12.6.120 ETH1
START LCS1
START LCS2
```

This example might use two Ethernet adapters connected to different LANs, with completely unrelated subnets.

3.2 Basic TSO networking

This section is introductory in nature and addresses common questions about z/OS usage with FLEX-ES.

z/OS, running under FLEX-ES, is a multiuser system—just as z/OS is a multiuser system when running on a “real” S/390. In z/OS terms, *multiuser* normally means multiple TSO users. Other subsystems and applications (including, for example, CICS, IMS™, WebSphere®, and HTTPD) handle multiple users, but do so largely by application programming. Multiple concurrent TSO users, just like multiple concurrent batch jobs, are a fundamental part of z/OS. Connecting multiple TSO users to z/OS (under FLEX-ES) can be considered *basic networking*.

TSO users can connect to a “real” S/390 in the following ways:

- ▶ **Local, non-SNA 3270s.** This is the most basic connection and does not involve LANs or conventional networks. Each 3270 terminal is connected (by coax) to a 3174 control unit²¹ (which typically handles up to 32 terminals). Each terminal has a device address. VTAM handles the terminal I/O, but the VTAM definitions are simple and the distributed VTAM definitions usually work without changes. This was the most common method for connecting TSO users on smaller, earlier mainframes. This form of connection is still used

²¹ This could also be a device that emulates a 3174 control unit, such as an IBM 2074 Console Controller.

to connect terminals for systems programmers because this connection requires little or no customization.

- ▶ SNA 3270 connections. These could be via coax, LANs, or SDLC connections, or through links to other VTAM or NCP nodes. This was (and still is) a common environment for larger networks, although many existing installations are moving to TCP/IP bases. There are many variations of this, with SNA control units and with directly-attached SNA client 3270 emulators.
- ▶ TN3270 TCP/IP connections to z/OS. These are LAN connections using TN3270 clients.

All of these connection modes work with z/OS under FLEX-ES and there are additional options:

- ▶ Channel-attached S/390 devices²² (such as 3174s, 3172s, 3745s, and similar control units) work just as they would on a “real” system. User connections and procedures would be the same as on a “real” system and we do not further describe this option.
- ▶ Local, non-SNA 3270s can be emulated using the FLEX-ES Terminal Solicitor.
- ▶ LAN SNA and TCP/IP connections can be made through FLEX-ES emulated control units, and end-user terminals work as they would on a “real” system.

3.2.1 Typical, basic configuration

Most of the FLEX-ES definition examples in this series of redbooks provide a basic network environment suitable for multiple TSO users and for applications such as CICS, HTTPD, and WebSphere. The exact purpose and use of the definitions may be a little confusing and this section describes the usage in more detail.

Our FLEX-ES definition examples include the following lines:

```
...
cu devad(0x700,3) path(0) resource(CU3174)
cu devad(0xE20,2) path(1) resource(CU3172)
...
...
CU3174: cu 3174                                # local, non-SNA 3174 control unit
interface local(1)
device(00) 3278 mstcon                          # address 700
device(01) 3278 L701                            # address 701
device(02) 3278 L702                            # address 702
end CU3174

CU3172: cu 3172                                # TCP/IP 3172 control unit
interface local(1)
options 'ipaddress=192.168.0.111,adapternumber=0'
device(00) 3172 eth0                            # address E20
device(01) 3172 OFFLINE                        # address E21
end CU3172
...
```

Local 3174 connections

The system being emulated is shown in Figure 3-1 on page 30. We can add more terminal connections to the 3174 by simply adding more lines in the FLEX-ES definitions. We usually show only three lines to keep the listings shorter. Of course, this assumes that your z/OS has enough local 3270 devices defined (in its IODF) starting at address 700 to match your definitions. Our examples assume the z/OS console is defined (in PARMLIB) at address 700

²² At the time of writing, these are not yet available for the Linux version of FLEX-ES. This discussion assumes that such channels will be available in the future.

and that the VTAM definitions have local 3270s defined starting at address 701. (The IODFs for recent AD CD-ROM systems have 64 local 3270s defined starting at 700. VTAM has 31 terminals defined starting at 701. TSO has only ten VTAM APPLs defined; you can easily add more.)

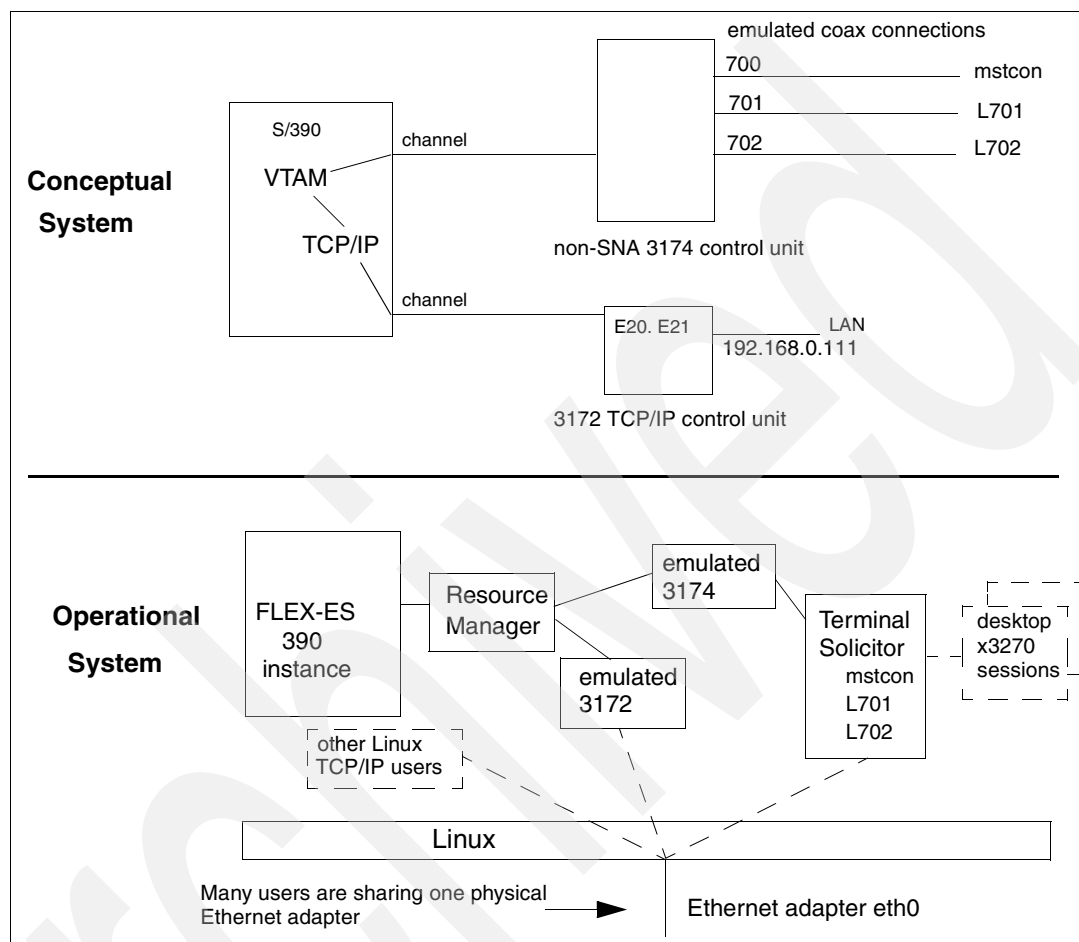


Figure 3-1 Emulated configuration

The terminal names shown in the figure (mstcon, L701) are arbitrary names and have no counterpart on a “real” S/390. On a real S/390, we would connect a coax 3270 terminal to one of the coax ports on the 3174 control unit. On the FLEX-ES emulation we use a TN3270 session (using x3270, PCOM, or any of a large number of TN3270 products) connected to the FLEX-ES Terminal Solicitor. The S/390 sees all connections through the Terminal Solicitor as channel-attached non-SNA 3270 terminals. It does not see that these are actually TN3270 TCP/IP LAN connections.

The Terminal Solicitor accepts TN3270 connections through Linux TCP/IP port 24. If you point your TN3270 client to the Linux IP address (port 24) and if the FLEX-ES resource manager is started and if a S/390 instance is started (but not necessarily IPLed), you should receive the Terminal Solicitor welcome screen. On this screen you can select any of the named 3270 sessions.

Our examples use a shell script to start two x3270 sessions on the Linux desktop. This can be viewed as a shortcut through the Terminal Solicitor. With a little more typing, you could instead manually start two x3270 sessions (on the Linux desktop), connect them to the Terminal Solicitor, and select the appropriate terminals (which are mstcon and L701 in our

examples). We use the two sessions for a z/OS console and a TSO terminal. You can start more sessions or fewer sessions on the desktop. You do not need any sessions on the desktop (but you need a z/OS console *somewhere* before you IPL).

This description of 3270 connections, up to this point, does not involve z/OS TCP/IP. All the connections described thus far are seen by the S/390 as channel-attached 3270s. z/OS TCP/IP need not be running to use these terminals; only Linux TCP/IP is involved. These terminals are useful only for 3270 emulation. You cannot, for example, use them for an FTP connection to z/OS.

3172 LAN connections

You can define a 3172 (or one of several other FLEX-ES emulated control units) for LAN connections to the S/390. This can be used for z/OS TCP/IP functions. Other types of emulated LAN control units can be used for SNA connections; however, this discussion is limited to TCP/IP connections.

As with a real TCP/IP 3172 LAN port, there are two addresses (an even/odd pair) involved. FLEX-ES requires that they be defined as shown in the examples. The first device should be associated with the Linux network adapter (eth0 in the example). The second device must be offline. The IP address in the definition is the z/OS IP address (which must be different than the Linux IP address).²³ z/OS TCP/IP is not a DHCP client; it cannot be assigned a dynamic IP address. You must know your z/OS TCP/IP IP address in order to customize your z/OS TCP/IP profile parameters. The same IP address goes in the FLEX-ES 3172 definition.

You need appropriate z/OS definitions (IOCP, TCP/IP profile, resolver, VTAM, USS) to fully utilize a TCP/IP connection to z/OS. The complete setup is beyond the scope of this discussion.

User connections

The basic system just described has a single physical LAN interface and this has two IP addresses:

- ▶ One for Linux (192.168.0.110 in many of our examples)
- ▶ One for z/OS TCP/IP (192.168.0.111 in the examples)

All user connections (except for x3270 sessions on the Linux desktop) are through this interface. A user has many connection options; he can:

- ▶ **telnet** to Linux (port 23), although such usage does not involve FLEX-ES. (Linux ftp and other common applications are also available at this IP address.)
- ▶ **TN3270** to Linux (port 24) to connect to the FLEX-ES Terminal Solicitor. This allows connection to the emulated S/390 as a local, channel-attached 3270 terminal. Assuming z/OS and VTAM are properly configured, this will produce a VTAM logo and permit the user to log onto TSO or other VTAM applications.
- ▶ **TN3270** to z/OS TCP/IP (192.168.0.111, port 23), assuming z/OS TCP/IP is properly configured. This will produce a VTAM-like logo and permit the user to log onto TSO or other VTAM applications.
- ▶ **ftp** to z/OS TCP/IP (at 192.168.0.111), assuming TCP/IP and the FTP application are properly defined.
- ▶ **telnet** to z/OS TCP/IP, assuming TCP/IP and USS services are properly defined. (In the AD system, this uses port 1023.)
- ▶ **http** to z/OS HTTPD (port 80), assuming it is configured and started.

²³ Due to the general workings of IP networking, both addresses should normally be on the same IP subnet.

- Use whatever other z/OS TCP/IP applications are available.

All these options can be used simultaneously, by multiple users, assuming all the required user IDs, profiles, definitions, and applications are correct. A very simple example of this is shown in Figure 3-2.

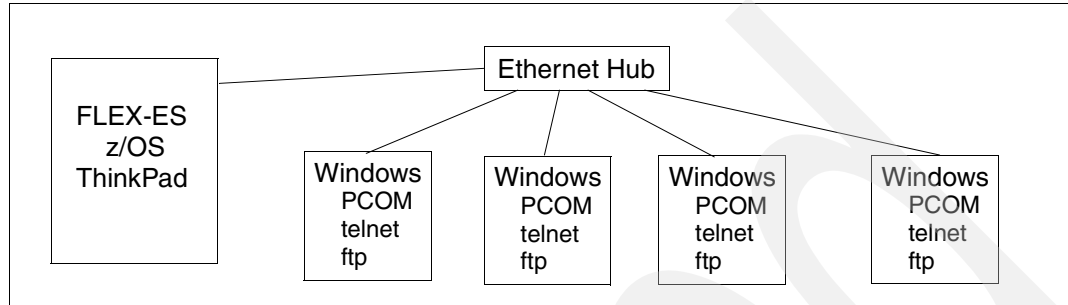


Figure 3-2 Multiple users

In a specific classroom case, the instructor had FLEX-ES and z/OS on a ThinkPad. Two x3270 sessions (including the z/OS console) on the ThinkPad were projected on a screen for the class to watch. Each of 15 students also had a ThinkPad, connected as shown. The students used PCOM (the primary IBM TN3270 product) to connect to z/OS. (Some connected via the Terminal Solicitor, some connected through z/OS TCP/IP. There is no practical difference to the end user.) Some class exercises required **telnet** sessions to USS; other exercises required **ftp** interactions with z/OS.

In this case, there were no external LAN connections. The private IP addresses (192.168.xxx.xxx) used in the examples in these redbooks were also used by the class. The only special setup required was to assign fixed addresses (in the 192.168.0.xxx range) to the Windows systems. This could have been avoided by using a DHCP-enabled router instead of a simple Ethernet hub.

FLEX-ES can work in much more complex network environments than illustrated in this example. (See “SNA over Ethernet” on page 153 for examples of more complex networking.) However, it is important for the FLEX-ES system owner to thoroughly understand simple FLEX-ES networking before starting implementation of something more complicated.

3.3 Networking limitations

FLEX-ES, at this time, does not support OSA-Express adapter²⁴ functions; that is, OSA Express channels or adapters are not emulated. Also, HiperSockets are not emulated. HiperSockets are normally used as an internal LAN between LPARs, and this environment is not relevant under FLEX-ES.

While FLEX-ES does not emulate hipersocket function, z/VM supports an equivalent function and this can be used when running z/VM under FLEX-ES. Therefore, anyone needing hipersocket functionality may have it by running under z/VM with its *guest lan* support. Be certain to have the PTF for APAR VM63180 and its pre-req VM63061 applied if running z/VM 4.3 or 4.2.

OSA-Express channels, especially when used with QDIO interfaces, provide increased LAN capacity and function offloading for zSeries machines. The client system, at the “other end” of

²⁴ While not exactly correct, the terms “OSA-Express adapter” and “OSA-Express channel” are commonly regarded as interchangeable terms.

the LAN, is not aware of the exact nature of the LAN adapter on the host. Emulated FLEX-ES LAN control units provide the same client connectivity as OSA-1 channels.

3.4 FLEX-ES OSA channels

Recent FLEX-ES releases have added a *localosa* channel type. Usage might be something like this:

```
channel (0) local
channel (1) localosa
channel (2) local
cu devad(0xE40,2) path(1) resource(CUOSA)
...
CUOSA: CU osaTR      #token ring TCP/IP
interface(1)
device(00) osa tr0
device(01) OFFLINE
end CUOSA
```

The localosa channel resolves definition problems within z/OS. This is discussed briefly in “Defining the FLEX-ES resources” on page 157.

3.5 LAN device types

FLEX-ES emulates several LAN device types, including 3172, OSA, and XCA. These are very similar (and all three are implemented by the same module). The key functions and differences are these:

CU type	Device Type,	Parameter	Usage	Notes
3174	3278, etc	name	Local 3270s	1
3172	3172	ethx	TCP/IP Ethernet	2
3172TR	3172	trx	TCP/IP token ring	
osa	osa	ethx	OSA TCP/IP Ethernet	
osaTR	osa	trx	OSA TCP/IP token ring	
osasna	osasna	ethx	OSA SNA Ethernet	
osasnaTR	osasna	trx	OSA SNA token ring	
xca	xca	ethx	XCA TCP/IP Ethernet	
xcaTR	xca	trx	XCA TCP/IP token ring	
xcasna	xcasna	ethx	XCA SNA Ethernet	
xcasnaTR	xcasna	trx	XCA SNA token ring	

The following notes may help with the above listing:

1. Logically, this line should not be in this table because an emulated 3174 is not a LAN control unit. We include it because it is a physical user of the LAN on a FLEX-ES machine. The S/390 operating system does not see an emulated 3174 as a LAN; it sees only a number of local, channel-attached, non-SNA 3270 terminals. However, this is implemented through TCP/IP on Linux. TN3270 client sessions (on the EFS machine or on other clients connected over the LAN) are transformed by FLEX-ES to appear as local, channel-attached 3270s.
2. This is the most commonly used CU and device for using S/390 TCP/IP functions. Most of the IBM Redbook examples use this emulated control unit for z/OS TCP/IP operation over Ethernet.

In addition to these device types, FLEX-ES emulates a number of other types that were typically used with mid-range VSE/ESA and older VM systems. These are detailed in FSI documentation (*FSIMM310: Resource Language Reference*).

When would you select these various types?

- ▶ The choice between Ethernet and token ring version is obvious.
- ▶ A 3172 Ethernet device requires explicit multicast IP addresses in the FLEX-ES definitions whereas osa and xca types can have a multicast address set by the host (z/OS, for example). Using the osa or xca definition and setting multicast addresses from the S/390 host TCP/IP specifications reduces the possibility of mis-coding in the FLEX-ES specifications requiring a disruptive resource refresh to correct it.
- ▶ If multicasting is not involved, there is no advantage to osa or xca over 3172 for TCP/IP usage. (Older software may support a 3172 better than an OSA adapter. The opposite may be true for newer software.)

3.6 Using a local router

A small, local router can be used to solve a number of problems when using a smaller FLEX-ES system. The following discussion is in terms of ThinkPad usage, but can also apply to a PC server.

3.6.1 The problem

A FLEX-ES system running z/OS usually has two TCP/IP systems: one for Linux and one for z/OS. Typically, these both use the same physical network adapter but should be viewed as two separate functions, using two separate IP addresses.

A user often wants to connect his FLEX-ES system (based on Linux and running z/OS) to an existing LAN. Most existing LANs assume that connected systems are DHCP clients. That is, the user plugs his system (probably a laptop, but possibly any PC) into the LAN connector. The DHCP client in the system will find a DHCP server on the network and the server will assign a temporary IP address to the system. This works well for Windows and Linux. It does not work for z/OS because z/OS is not a DHCP client. (z/OS can be a DHCP *server*, but that is not relevant in this discussion.) Furthermore, at least two IP addresses are normally needed by the machine running FLEX-ES (one for Linux and one for z/OS).

Most larger LAN installations are reluctant to assign permanent IP addresses, such as are needed by z/OS. Also, permanent IP addresses are not portable. For practical purposes, an IP address can be used only on a LAN that includes the subnet appropriate for the IP address. (This is not a problem with DHCP-assigned addresses, because they are used only for the duration of a system connection and are always appropriate for the LAN subnet.)

Our problem can be stated this way:

- ▶ We want to connect to an existing LAN that uses DHCP. (IBM internal LANs are an example of this situation.)
- ▶ We want connectivity from z/OS (on our FLEX-ES system) to the LAN.
- ▶ z/OS requires a fixed IP address.

We can note that exactly the same situation exists if we want to connect to many broadband services through a cable modem or a DSL modem.²⁵

3.6.2 One solution

We purchased a Netgear *Cable/DSL ProSafe Firewall/Print Server* Model FR114P unit. The unit cost approximately US\$110 at the time of writing, although the prices of such units are dropping rapidly. This is a small router that can be used with 10/100 Mbps Ethernet LANs. The *Cable/DSL* portion of the product name can be misleading. Cable and DSL “modems” provide an Ethernet connection²⁶ as the computer interface. This interface can be used as a normal Ethernet connection to a PC.²⁷ This particular unit allows 10 or 100 Mbps connections to the external LAN (which might be a cable or DSL modem). Other similar units are available from other vendors, but this was the only one we found (at the time of writing) that provided 100 Mbps connections to the external LAN. All the Netgear documentation refers to the external LAN as the WAN. The unit has one WAN Ethernet connector and four local LAN Ethernet connectors, as shown in Figure 3-3.

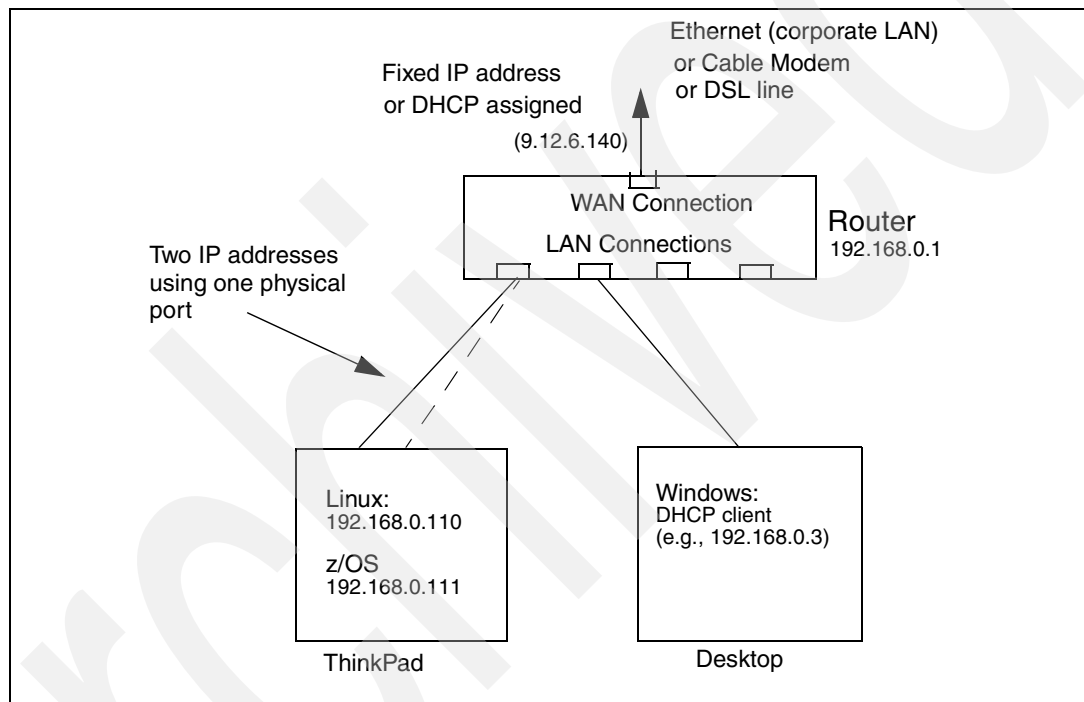


Figure 3-3 Small router overview

It is important to understand the terminology of the router. In this terminology, the WAN is the external network and the LAN is the local network created by the router. In the situation we describe, the WAN is probably a corporate LAN (but it could be a DSL or cable modem). In this chapter, we use the term *external LAN* instead of *WAN*, and *local LAN* instead of *LAN*.

The key element of the solution is the Network Address Translation (NAT) provided by the router. It automatically translates the local LAN addresses (the three addresses shown in Figure 3-3 in the 192.168.0.x range) to the single external IP address (shown as 9.12.6.140 in the figure). The external world sees only a single system at 9.12.6.140. The router maintains *state information* that keeps track of which local system (for example, z/OS at 192.168.0.111)

²⁵ This statement assumes the cable or DSL interface does not provide a Network Address Translation (NAT) function. The situation is more complex when NAT is provided by the “modem” and we do not address this in this discussion.

²⁶ These are all RJ-45 connectors.

²⁷ This usage is not documented by Netgear and may not be supported by them. We can only state that we have used it as described here and it works quite well.

is working with which external IP addresses and automatically routes packets between them. The external LAN never sees the 192.168.0.x addresses.

What produced the IP addresses in the figure? By default, the router itself is 192.168.0.1, as seen from the local LAN.²⁸ You can change this, but there is usually no reason to change it.²⁹ The 9.12.6.140 address (in the figure) was assigned to the router by a DHCP server on the external LAN. We assigned the fixed IP addresses 192.168.0.110 (Linux)³⁰ and 192.168.0.111 (z/OS); they were arbitrary choices on subnet 192.168.0. The 192.168.0.3 address (Windows) was assigned by a DHCP server that is included in the router.

The router includes these functions:

- ▶ It is a DHCP *client* on the external LAN. It will find a DHCP server on the external LAN and request an IP address. (You can disable the DHCP client function and assign a fixed external IP address if you wish.)
- ▶ It is a DHCP *server* for the local LAN. By default, it serves addresses in the 192.168.0.2 to 192.168.0.51 range. (You can disable the DHCP server if you want to assign fixed addresses to all your local systems.)
- ▶ You can have a mixture of DHCP and fixed addresses for your local systems, as shown in the figure.
- ▶ We normally use a netmask of 255.255.255.0. This means the local LAN network address is 192.168.0 and the local host addresses are 1 through 254. (Local host addresses 0 and 255 are reserved for special uses in most IP implementations.)
- ▶ All of the systems can be active at the same time, working with various external network servers and systems.
- ▶ Our router has only four local LAN connections. Larger routers are available. Also, if necessary, we could use one local port to link to a larger hub with many more connections.

Limitations

Small routers of this type work well for clients (on the local LAN side) that routinely use DHCP to acquire an IP address and a Domain Name Server (DNS server) address. The router provides these functions.

z/OS expects a *hard coded* IP address for a DNS name server. If you provide this (and if the name server is accessible), then domain names can be used from the z/OS interfaces.

Linux works well with DHCP, although we elected to use fixed IP addresses for our Linux systems. (Either way works well on the local LAN side of these small routers.) Our Linux could automatically obtain a DNS address from the external network only if we used DHCP to obtain the IP address. We wanted to use a static IP address for Linux, so we hard coded the same DNS server IP address that we used for z/OS.³¹

²⁸ Other similar routers may be different. For example, our DSL “modem” (which includes a NAT function) defaults to address 192.168.254.254 for itself on its local LAN.

²⁹ IP addresses in the range 192.168.0.0 to 192.168.255.255 (and also 176.16.0.0 to 176.31.255.255, and the whole 10.x.x.x range) are reserved as local addresses. These addresses are never used by devices connected directly to the Internet. Most Internet routers will not forward such addresses. Notice that our small router uses 192.168.0.x addresses only on the *local LAN* side.

³⁰ Linux can act as a DHCP client to automatically obtain an IP address. We elected to assign a fixed address.

³¹ We used a static address for Linux because we never bothered to create a local DNS server or to build useful /etc/hosts files. We addressed the various local systems by their numeric IP addresses and this is much easier if these are fixed addresses.

3.6.3 Router administration

This router is managed through a browser interface. In our case, we pointed a Linux Web browser (connected to one of the local LAN ports) to address 192.168.0.1 and received a logon page from the Web server that is built into the router. The default logon ID for our router is *admin* and the password is *password*. (You can change these after you log into the router the first time.)

A number of screens are available to control DHCP functions, firewall filtering, logging, and so forth. The firewall and logging functions are interesting but are not discussed here. Controls are also available to assign *service* addresses. This requires some understanding.

Suppose an external host, somewhere on the external LAN, attempts to connect to telnet at address 9.12.6.140 in our example. This is the address of the router. The router has three systems connected: Linux, z/OS, and Windows. All of these potentially have telnet servers. What should the router do? There is an administrative screen (via the browser connected to the router) where you can set relevant controls. For example, you might specify that an incoming request for telnet should go to the system at 192.168.0.110. If you have not provided sufficient information for the router to handle this, it would simply discard the request.

Outgoing requests are not a problem. If our Linux system attempts to telnet to a host at 9.12.3.15, the request is not ambiguous and is sent to that system. The telnet responses back to the telnet client (in Linux) are automatically handled because the router remembers which of our local hosts originated the request.

The router offers a Dynamic DNS function (DDNS) that we did not attempt to use. DDNS assumes that we have assigned system and domain names properly, and this requires some coordination with whoever manages local system names.

IP addressing plan

We made the following plan for IP addresses in our local LAN. This is an arbitrary plan, but we found that having a specific plan makes implementation easier. This plan allows for a variety of guest systems to be attached to our small router. (Since there are only four ports, we assumed that different systems would be connected at different times.) Our addressing plan is:

IP Address	Use.....
192.168.0.1	This is the router itself
192.168.0.2-51	Assigned by the DHCP server in the router
192.168.0.52-99	We will manually assign these to any guest systems that need it
192.168.0.100	Linux in our T20 ThinkPad
192.168.0.101	z/OS in our T20 ThinkPad
192.168.0.110	Linux in our T23 ThinkPad
192.168.0.111	z/OS in our T23 ThinkPad
192.168.0.112	Linux for S/390 in our T23 ThinkPad
192.168.0.120	Linux in our xServer system
192.168.0.121	z/OS #1 in our xServer system
192.168.0.122	z/OS #2 in our xServer system
192.168.0.123	Linux for S/390 in our xServer system
192.168.0.124	z/VM in our xServer system
192.168.0.190	OS/2 (in P/390 system #1) (Used as a PCOM client)

Specific configuration (router)

We logged into our router (by pointing the Linux browser in our T23 ThinkPad to 192.168.0.1) and entered user ID *admin* and password *password*. Using the screens presented by the browser, we set the following:

Basic Settings

Does your Internet provider require a Login:

☒ **No**

☐ Yes

Internet IP address:

☒ **Get dynamically from ISP**

☐ Use Static IP address

IP address:

IP Subnet mask:

Gateway IP address:

Domain Name Server (DNS) Address

☒ **Get dynamically from ISP**

☐ Use these DNS Servers

Primary DNS:

Secondary DNS:

Router MAC address

☒ Use default address

☐ Use this computer's MAC address

☐ Use this MAC address:

LAN IP Setup

☒ **Use router as DHCP Server**

Starting IP address: 192.168.0.2

Ending IP address: 192.168.0.51

Our key settings are shown in bold type. We specified that the router should obtain its external IP address from a DHCP server on the network. We specified that the router should dynamically obtain the address of a DNS server, so it could pass this to any local client who requests it. Finally, we specified that the router should act as a DHCP server for any local system that requests this service.

Specific configuration (Linux)

Within **gnome**, we selected the *Start* function and then selected *System Settings* from the menu. Within *System Settings*, we selected *Network*. This provides a window something like that shown in Figure 3-4. (The format and navigation of these **gnome** windows seems to change with Red Hat releases, so your system may be slightly different.)

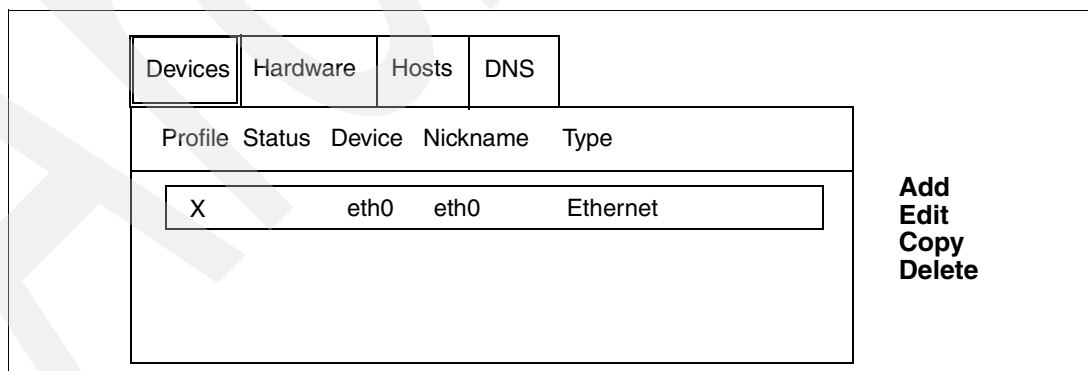


Figure 3-4 Initial network configuration window

Within this window, we selected the *eth0* device line and then selected **Edit**. This produced an Ethernet device window that included the following options:

☒ Activate device when computer starts

☐ Allow all users to enable and disable the device

☐ Automatically obtain IP address settings with DHCP

DHCP Settings:

```

        Hostname (optional) _____
        Automatically obtain DNS information from provider
X  Statically set IP address
    Address: 192.168.0.110
    Subnet mask: 255.255.255.0
    Default Gateway: 192.168.0.1

```

Our selections and the values we entered are shown in bold type. We then selected the DNS tab in the configuration window and set the following:

```

Hostname: t23
Primary DNS: 9.12.6.7

```

Your specific details will differ, of course. Please do not attempt to use the ITSO name server at the address shown.

TCP/IP profile (z/OS)

We used the z/OS 1.4 AD CD-ROM release for this work. The relevant TCP/IP profile is in the TCPIP.PROFILE.TCPIP data set. Our settings (with all the comment lines removed, for brevity) were as follows:

```

ARPAGE 20
GLOBALCONFIG NOTCPIPSTATISTICS
IPCONFIG DATAGRAMFWD VARSUBNETTING SYSPLEXROUTING
SOMAXCONN 10
TCPCONFIG TCPSENBFRSIZE 16K TCPCVBUFRSIZE 16K SENDGARBAGE FALSE
TCPCONFIG RESTRICTLOWPORTS
UDPCONFIG RESTRICTLOWPORTS
DEVICE LCS1 LCS E20 AUTORESTART
LINK ETH1 ETHERNET 0 LCS1
HOME
    192.168.0.121 ETH1
BEGINRoutes
;   Destination      Subnet Mask   First Hop   Link Packet Size
ROUTE 192.168.0.0 255.255.255.0 = ETH1 MTU 1500
ROUTE DEFAULT           192.168.0.1 ETH1 MTU DEFAULTSIZE
ENDRoutes
AUTOLOG 5
    FTPD JOBNAME FTPD1 ; FTP Server
    PORTMAP JOBNAME PORTMAP1 ; USS Portmap Server (SUN 4.0)
ENDAUTOLOG
PORT
    7 UDP MISC SERV ; Miscellaneous Server - echo
    7 TCP MISC SERV ; Miscellaneous Server - echo
    9 UDP MISC SERV ; Miscellaneous Server - discard
    9 TCP MISC SERV ; Miscellaneous Server - discard
    19 UDP MISC SERV ; Miscellaneous Server - chargen
    19 TCP MISC SERV ; Miscellaneous Server - chargen
    20 TCP * NOAUTOLOG ; FTP Server
; 20 TCP * NOAUTOLOG SAF FTPDATA ; FTP Server
    21 TCP FTPD1 ; FTP Server
; 21 TCP FTPD2 BIND FEC0:0:0:1:0009:0067:0115:0066 ; FTP IPv6
    23 TCP INTCLIEN ; Telnet 3270 Server
; 23 TCP INETD1 BIND 9.67.113.3 ; z/OS UNIX Telnet server
    25 TCP SMTP ; SMTP Server
    53 TCP NAMED ; Domain Name Server
    53 UDP NAMED ; Domain Name Server
    111 TCP PORTMAP ; Portmap Server (SUN 3.9)
    111 UDP PORTMAP ; Portmap Server (SUN 3.9)
; 111 TCP PORTMAP1 ; Unix Portmap Server (SUN 4.0)

```

```

; 111 UDP PORTMAP1      ; Unix Portmap Server (SUN 4.0)
123 UDP SNTPD           ; Simple Network Time Protocol Se
135 UDP LLBD            ; NCS Location Broker
161 UDP OSNMPD          ; SNMP Agent
162 UDP SNMPQE          ; SNMP Query Engine
389 TCP LDAPSrv         ; LDAP Server
443 TCP HTTPS           ; http protocol over TLS/SSL
443 UDP HTTPS           ; http protocol over TLS/SSL
512 TCP RXSERVE         ; Remote Execution Server
514 TCP RXSERVE         ; Remote Execution Server
; 512 TCP * SAF OREXECd  ; z/OS UNIX Remote Execution Serv
; 514 TCP * SAF ORSHLLD  ; z/OS UNIX Remote Shell Server
515 TCP LPSERVE         ; LPD Server
520 UDP OROUTED         ; OROUTED Server
580 UDP NCPROUT         ; NCPROUTE Server
750 TCP MVSkerb         ; Kerberos
750 UDP MVSkerb         ; Kerberos
751 TCP ADM@SRV         ; Kerberos Admin Server
751 UDP ADM@SRV         ; Kerberos Admin Server
1933 TCP ILMTSRVR       ; IBM LM MT Agent
1934 TCP ILMTSRVR       ; IBM LM Appl Agent
3000 TCP CICSTCP        ; CICS Socket
3389 TCP MSYSLDAP       ; LDAP Server for Msys

```

SACONFIG ENABLED COMMUNITY public AGENT 161

TelnetParms

Port 23

TELNETDEVICE 3278-3-E NSX32703 ; 32x80

TELNETDEVICE 3279-3-E NSX32703 ; 32x80

TELNETDEVICE 3278-4-E NSX32704 ; 48x80

TELNETDEVICE 3279-4-E NSX32704 ; 48x80

TELNETDEVICE 3278-5-E NSX32705 ; 132x27

TELNETDEVICE 3279-5-E NSX32705 ; 132x27

LUSESSIONPEND

MSG07

CodePage ISO8859-1 IBM-1047 ; Linemode ASCII, EBCDIC code pa

Inactive 0 ; Let connections stay around

PrtInactive 0 ; Let connections stay around

TimeMark 600

ScanInterval 120

; SMFinit std

; SMFterm std

WLMClusterName

TN3270E

EndWLMClusterName

EndTelnetParms

BeginVTAM

Port 23 ; 992

DEFAULTLUS

SCOTCP01..SCOTCP30

ENDDEFAULTLUS

LINEMODEAPPL TSO ; Send all line-mode terminals to TSO

ALLOWAPPL TSO* DISCONNECTABLE ; Allow all users access to TSO

USSTCP USSN

ALLOWAPPL * ; Allow all applications

EndVTAM

START LCS1

This profile uses almost all the existing (non-comment) lines in the distributed TCPIP.PROFILE.TCPIP and has customized a specific IP address and routing information. We elected to use the BEGINRoute parameters instead of the older GATEWAY parameters to

specify static routes. The key information is in the following sections: DEVICE, LINK, HOME, and BEGINRoutes. Note the DEFAULT route points to the router.

The following is the TCPIP.TCPIP.DATA file we used (with comment lines removed). Please do not attempt to use the name server listed (9.12.6.7); you need to find your own name server.

```
TCPIPJOBNAME TCPIP
HOSTNAME NF
DOMAINORIGIN ITSOEFS.ITSO.POUGHKEEPSIE.COM
DATASETPREFIX TCPIP
NSINTERADDR 9.12.6.7
RESOLVEVIA UDP
RESOLVERTIMEOUT 10
RESOLVERUDPRETRIES 1
ALWAYSUTO NO
```

With these parameters, we could access all external Internet sites from z/OS and the existence of the router was generally transparent for simple uses.

Specific configuration (Windows)

Within Windows³² we used the path *My Computer -> Control Panel -> Network & Dial Up -> Local Area Connection*. Click *Properties* in this panel. This should produce a list of the protocols being supported on your LAN connection. Select *TCP/IP* (in the list of protocols). Then click on *Properties*. This should produce a screen where the following parameters can be set:

```
X Obtain IP address automatically
  Use the following IP address
    IP Address:
    Subnet mask:
    Default gateway:
X Obtain DNS server address automatically
  Use the following DNS server address
    Preferred DNS server:
    Alternate DNS server:
```

While Windows is running, you can use an **ipconfig** command to display the IP address it is using and an **nslookup** command to display the DNS server address. (Use an **exit** command to terminate the **nslookup** command.)

Router service requests

The settings described above permitted users to access Internet locations from our local Linux, z/OS, and Windows systems. However, external users could not access our systems. Remember that the only IP address visible externally is that of the router itself (9.12.6.140 in our example). If an external host attempts to connect to ftp, for example, at this IP address, our router would not know how to handle the ftp request and would discard it.

By using a local browser connection to the administrative functions in the router, we could configure it to pass inbound service requests to our local systems. For example, we could specify that an inbound request for ftp (TCP ports 20 and 21) should always be given to the local system at address 192.168.0.111 (z/OS in our example). A certain amount of TCP/IP knowledge is needed to set up these definitions, and a little experimentation can be useful. Most routers will have similar functions, although the administrative interfaces may differ.

³² This description is for Microsoft Windows 2000. The process is similar on most recent Microsoft Windows systems.

With our router, there was a browser panel to define *rules*. These rules connected specific *services* (ftp, telnet, finger, http, and so forth)³³ to specific local IP addresses. Rules exist for inbound and outbound services. By default, all local systems (Linux, z/OS, and Windows in our example) were permitted to use all outbound services. By default, no local system is connected to any inbound service. An inbound service (such as an ftp request) should be connected to only one local system (such as z/OS, in our example).

In addition to connecting service requests (inbound and outbound), the rules definition could also establish operational time windows, specific external LAN addresses to accept (or deny), enable local logging, and so forth. Collectively, these establish a firewall function.

Many TCP/IP applications open randomly-selected ports after being started. For some of these we needed to assign a large range of port numbers to the system handling the applications (z/OS). We defined a new *service* (using an arbitrary name) with a large range of ports associated with it. We then made a new *rule* to connect this service to the IP address for z/OS.

Because these administrative interfaces differ for various brands and models of routers, we will not attempt to document them here.

3.7 NFS files

Can we use remote files, mounted through NFS, as emulated FLEX-ES files? As FakeTape files?

We cannot recommend using files mounted through NFS as normal FLEX-ES files. The exposure is that NFS response can be very erratic, depending on network traffic. z/OS has many timers running to verify that I/O is operating as expected. The Missing Interrupt Handler (MIH) is only part of the timing controls. NFS response may be outside the acceptable range of these timers. (A mild symptom you might see even during normal z/OS startup using local files are messages about *JES2 waiting for checkpoint data set*.)

Nevertheless, some users have reported success with NFS-mounted files for FakeTape. However, they indicated that a 100 Mbps LAN should be used, either dedicated or with a switch (instead of a simple hub) that effectively produced a dedicated link.

3.8 Using address E40

The AD CD-ROM system has used roughly the same addresses (device numbers) for every release. Addresses E20-E23 and E40-E43 are generated (in the IODF) as CTC devices. CTC (channel-to-channel) device descriptions are used by many devices, including many types of LAN interfaces. It is common to use E20/E21 for a TCP/IP LAN interface. A user looking for another set of addresses for another TCP/IP LAN interface might try E40/E41 and discover it does not work, with a resulting set of obscure error messages.

The problem is that a VTAM definition (which is automatically started in the AD system) attempts to use address E40 for an XCA connection. VTAM acquires the address (assuming a device is defined for use at the address) before TCP/IP has a chance to allocate it.

³³ Another Web page allowed us to configure *custom* services if none of the standard services matched our requirements. We needed to configure custom services to allow external users to connect to WebSphere running on z/OS, for example.

You can use addresses E22/E23 or E42/E43 for your second LAN interface. Alternatively, you could remove member names XCAE40E and XCAE40R from the list in the ADCD VTAMLST member ATCCON00. Both these XCA members attempt to use address E40.

(TCP/IP LAN interfaces for LCS/3172/OSA devices require two device addresses, and they must be an even/odd pair. This is why we refer to a LAN interface as E22/E23, for example.)

3.9 Remote resources

FLEX-ES can use TCP/IP links to other instances of FLEX-ES on remote servers. You could, for example, run only the FLEX-ES *resource manager* on a remote server that has a large disk configuration. With the proper system definitions (locally) and resources definitions (locally and remotely), a FLEX-ES instance could access the remote disks (as emulated S/390 devices) as readily as local disks. When used with a fast LAN, the performance can be quite acceptable. Usage is not limited to emulated disks; for example, a SCSI tape drive on a remote system could be used, or emulated channel-to-channel connections. A FLEX-ES license (and dongle) for each system is required.

Setup for this type of operation can be recognized by the keyword *network* in channel definitions and in cu definitions. FLEX-ES automatically uses TCP/IP port 555 for communication between multiple *resource managers*. You must ensure that your firewall or other security controls do not block this port.

We do not routinely use remote resources for our IBM Redbook projects, and we have not written much about it. However, FLEX-ES remote resources is potentially a very powerful and useful function in appropriate situations.

A conceptual network is shown in Figure 3-5. In this example, a single instance of FLEX-ES is executing (in system B in the center of the figure). It is using remote resources on the other two systems; disk resources on system A and a tape drive on system C. The FLEX-ES resource managers in all three systems communicate with each other. The emulated S/390 is unaware that some of the emulated devices are on remote systems.

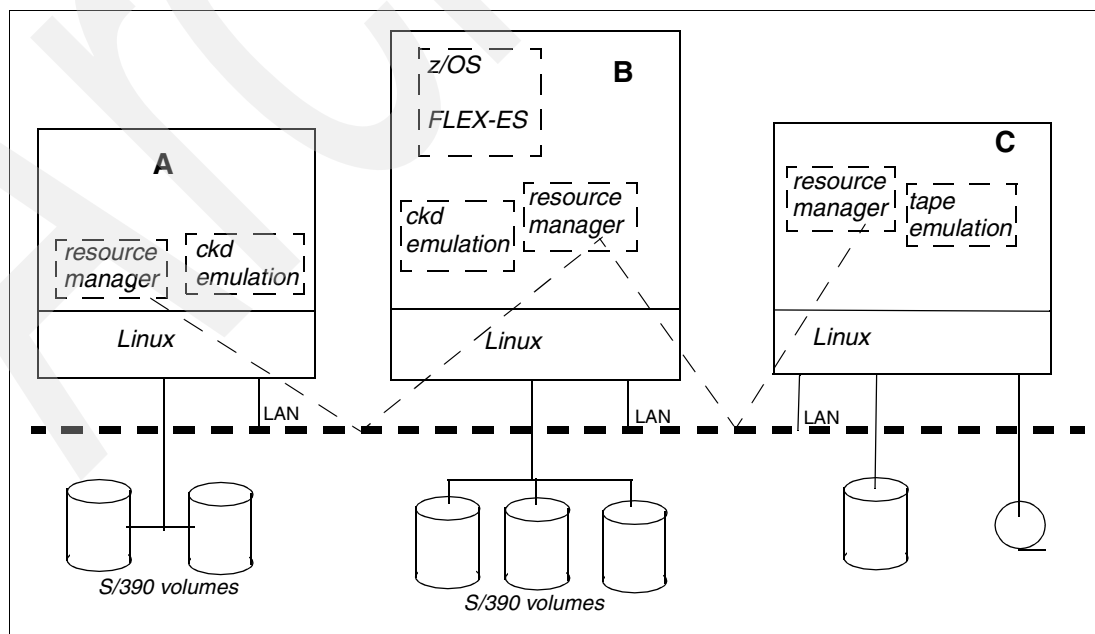


Figure 3-5 Conceptual view of distributed EFS system

3.9.1 Working example

We created a small working example of remote resources. This is illustrated in Figure 3-6 on page 45. If you are interested in using remote resources we suggest you study this example closely. It illustrates most of the principles involved in larger, more complex remote resource operations.

In this example, we have two systems: t23 (a T23 ThinkPad) and nf (a Netfinity®) with the Linux IP addresses shown in the illustration. (None of the setup we illustrate involves z/OS TCP/IP. All of the discussion is about Linux TCP/IP.) We have an emulated 3990 control unit on each system. System t23 has four disk volumes containing a z/OS system. System nf has a single-volume z/OS system and two local work volumes.³⁴

The goal is to have both 3990 control units online to both systems. This is a simple shared-DASD configuration whose conceptual operation should be familiar to z/OS users. The z/OS systems we are using are not configured to allow multiple machines to IPL the same SYSRES volume. We needed two separate z/OS systems on different volumes. The IPL volume for one of the z/OS systems is S4RES1 and the other IPL volume is SARES1.

We used **resadm** on each system to start the appropriate resource manager. We then started a S/390 instance on each system (with an appropriate shell script). We IPLed from A80 on system t23 and IPLed from A90 on system nf. When z/OS was ready, we displayed the online DASD (**d u,dasd,online**) and all seven volumes were present on both systems. We could, for example, use ISPF on system t23 to edit a file on volume WORK02.

Please note that we did not have GRS active. Indeed, we did not have the emulated CTC links defined that would be needed to run GRS. There was no ENQ protection of data sets across the two systems, so we were careful not to attempt to edit the same data set at the same time on both systems.

³⁴ The t23 system happens to be the AD CD-ROM 1.4s release and the nf system is the “recovery system” from CD 13 of the AD CD-ROM 1.4 release.

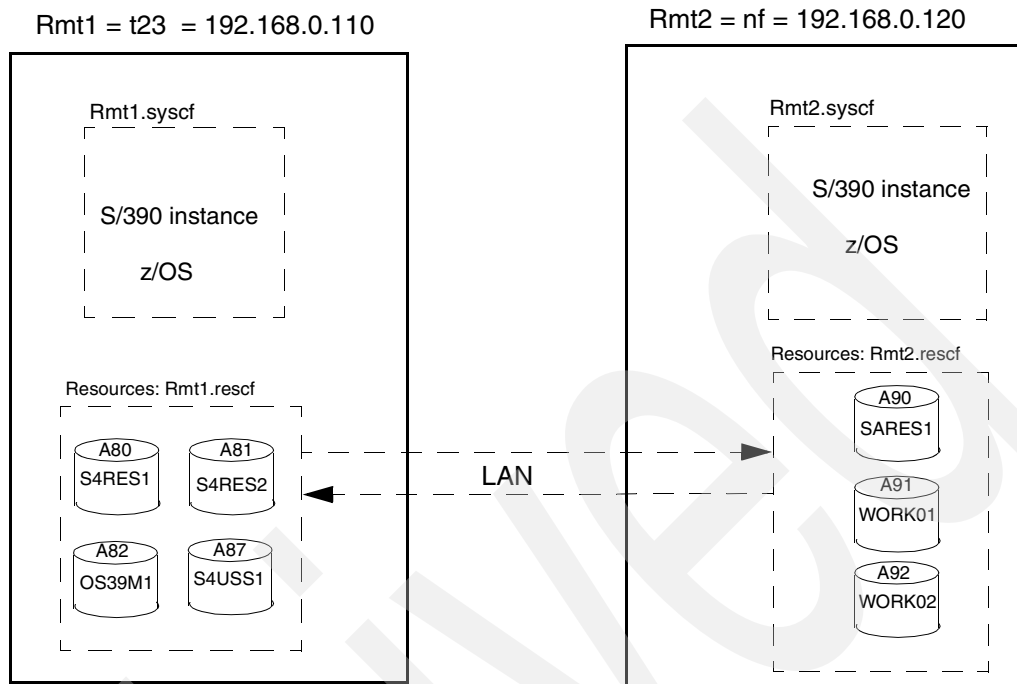


Figure 3-6 Small example of remote resources

Shared resource definitions

We used the following FLEX-ES definitions for the two systems:

```
=====System t23 (Rmt1)=====
system Rmt1:
  memsize(524288)
  cachesize(4096)
  instset(z)
  cpu(0)
  channel(0) local
  channel(1) local
  channel(2) local
  channel(3) network
  cu devad(0xA80,8) path(2) resource(CU3990A)
  cu devad(0xA90,3) path(3) resource(192.168.0.120:CU3990B)
  cu devad(0x700,3) path(0) resource(CU3174)
  cu devad(0xE20,2) path(1) resource(CU3172)
end Rmt1

resources Rmt1:
  nodes: anynode end nodes

CU3990A: cu 3990
  interface local(1)
  interface network(1)
  options 'trackcachesize=500,ssid=0010'
  device(00) 3390-3 /dev/raw/A80
  device(01) 3390-3 /dev/raw/A81
```

```

device(02) 3390-3 /dev/raw/A82
device(03) 3390-3 OFFLINE
device(04) 3390-3 OFFLINE
device(05) 3390-3 OFFLINE
device(06) 3390-3 OFFLINE
device(07) 3390-3 /dev/raw/A87
end CU3990A

CU3174: cu 3174
interface local(1)
device(00) 3278 mstcon
device(01) 3278 L701
device(02) 3278 L702
end CU3174

CU3172: cu 3172
interface local(1)
options 'ipaddress=192.168.0.111,adapternumber=0'
device(00) 3172 eth0
device(01) 3172 OFFLINE
end CU3172
end Rmt1

=====System nf (Rmt2)=====
system Rmt2:
memsize(524288)
cachesize(4048)
instset(z)
cpu(0)
channel(0) local
channel(1) local
channel(2) local
channel(3) network
cu devad(0xA80,8) path(3) resource(192.168.0.110:CU3990A)
cu devad(0xA90,3) path(2) resource(CU3990B)
cu devad(0x700,3) path(0) resource(CU3174)
end Rmt2

resources Rmt2:
nodes: anynode end nodes

CU3990B: cu 3990
interface local(1)
interface network(1)
options 'ssid=0011'
device(00) 3390-3 /s390/SARES1
device(01) 3390-1 /s390/WORK01
device(02) 3390-1 /s390/WORK02
end CU3990B

CU3174: cu 3174
interface local(1)
device(00) 3278 mstcon
device(01) 3278 M701
device(02) 3278 M702
end CU3174

end Rmt2

```

You might note the following elements in these definitions that are relevant to shared resource operation:

- ▶ A network channel is defined for each system. This channel provides the path required for a control unit connected to remote resources.
- ▶ A control unit is defined, for each system, with the resource name in the format *system:name*. For example:

```
cu devad(0xA80,8) path(3) resource(192.168.0.110:CU3990A)
```

We could use *t23* instead of *192.168.0.110* in this definition. (We have *t23* defined in our */etc/hosts* file.) In this example, the FLEX-ES resource definitions in the system at address 192.168.0.110 must have an appropriate control unit defined with name CU3990A.

- ▶ A new resource element was defined on each system:

```
nodes:
  anynode
end nodes
```

In this definition *anynode* is a keyword with the obvious meaning; it allows connection to these resources from any external FLEX-ES system. Instead of this keyword you can list the names (or IP addresses) of specific remote nodes that you want to allow to connect to your resources.

- ▶ An additional interface [*interface network(1)*] is defined for the two 3990 control units that are to be shared.
- ▶ We assigned specific and different *ssid* values to the two 3990 control units. Otherwise they will have the same default *ssid* value and this can cause errors. By default, ascending *ssid* values are assigned to local emulated 3990 control units. This causes conflicts when remote resources are used.

You may notice that *t23* used raw devices for its emulated DASD volumes and *nf* used simple Linux files. There is no requirement for this, of course, and it simply reflects how *t23* and *nf* were set up before we started to define the remote resource example.

3.9.2 Practical operation

Referring to our small example, we need to start the resource managers in both systems (in any order). This is done with normal **resadm -s** commands. At this point we could verify that the remote links were operational. On system *t23*, we entered:

```
# resadm -h 192.168.0.120 -n (Directed at system nf)
FSIRA200: Node: t23 IP Address: 127.0.0.1 Flags: LOCAL SECURE
FSIRA200: Node: nf IP Address: 192.168.0.120 Flags: LOCAL SECURE
FSIRA200: Node: 192.168.0.110 IP Address: 192.168.0.110 Flags: REMOTE SECURE
# resadm -h 192.168.0.120 -r (Directed at system nf)
FSIRA250: Resource: CPU Flags: READY Type: CPU Port 33619 PID: 1528
FSIRA250: Resource: CU3990B Flags: READY Type: CU Port 33620 PID: 1529
FSIRA250: Resource: CU3174 Flags: READY Type: CU Port 336239 PID: 1530
FSIRA250: Resource: NETCU Flags: READY Type: NETCU Port 33624 PID: 1531
FSIRA250: Resource: TS3270 Flags: READY Type: TS3270 Port 33627 PID: 1532
```

The resource listing (from the second command) is from the *other* system. You can verify this by moving to the other system (*nf*, in our example) and entering the command **resadm -r**.

If these results are not available from **resadm -h** commands, there is no point in starting the S/390 instances on the systems. No remote resources are available if the **resadm -h** commands do not show connections to the other system(s).

After IPLing and running S4RES1 on *t23* and SARES1 on *nf* for a while, we shut down both z/OS systems. We then IPLed SARES1 from *t23*. That is, we IPLed from a volume on the *other* system. The IPL process took longer than usual (approximately twice as long) but, once started, TSO response was about normal. We expect that performance of a heavily-loaded system could be substantially reduced by using remote disk volumes.

3.9.3 Common problems

Be certain you have TCP/IP connectivity between your systems. Using **ping** commands is a good way to verify this.

The most common error message when attempting to use remote resources is this:

```
# resadm -h 192.168.0.120 -n          (Same result when the -r option is used)
resadm -- error: unable to receive message
```

This appears to be a general-purpose error message. In our case, we received this message for almost a day while we tried changing many parameters. We finally discovered that the unit count (in a *cu* statement for a remote control unit) on one system did not match the unit count for that resource on the other system.

This error message should probably be regarded as saying, "Something related to remote resources is wrong with your FLEX-ES definitions." As is true with many aspects of FLEX-ES operation, all your definitions and setup must be *exactly* correct before FLEX-ES runs as expected.

Be certain you do not have a firewall that might hinder operation. With Red Hat 8.0, you can do this by *System Settings* -> *Security Level* and selecting *No Firewall*. We did not attempt to customize firewall operation to permit FLEX-ES remote resource connections.

3.9.4 Performance

We used a private 100 Mbps Ethernet LAN. That is, we had a small router with only the two FLEX-ES machines connected on the local side. In general, response is good in this environment for light, casual use of remote devices. We cannot recommend use of remote resources over a general Internet connection. We expect that serious use of remote FLEX-ES resources would use a private LAN. Remember that z/OS has many timers running and a delay of a few seconds in some I/O responses can trigger error recovery functions.

If you have multiple machines on your private LAN, and if the LAN tends to be busy, then a *switch* instead of a simple *hub* can provide considerably better performance. Significant use of DASD through remote resources can create heavy LAN traffic. A switch provides, in effect, a private point-to-point connection between two LAN users. A switch costs more than a simple Ethernet hub, but is well worth the investment in the proper circumstances.

Other uses of remote resources, such as for a tape drive or printer, tend to be much lighter LAN users and a simple hub may be effective.

3.9.5 Comments

Using remote resources can be very effective under some circumstances. For example, if you already have multiple FLEX-ES systems, you can share expensive devices such as SCSI-attached 3480/90 tape drives or a high-performance parallel-port printer. Some operational effort is needed to ensure that only one system at a time has a tape drive or printer online. However, setting up the remote resource definitions and documenting the

correct operational procedures is easier than physically uncabling and moving the devices when they are needed on another system.³⁵

We have not attempted to address the security issues of LAN traffic for remote resources. For performance reasons, we expect any serious use of remote resources to be on a private LAN where, we assume, security is not an issue.

3.10 Operating FLEX-ES remotely

You can operate FLEX-ES and z/OS remotely. It is very easy to do and no special setup is required except for establishing a remote z/OS operator console. The key elements are these:

- ▶ Obtaining a Linux command-line session on the target server. You need this to use **resadm** (as root), then (under user ID *flexes*) to start whatever shell script you have defined for operation, and to enter CLI commands (such as **ip1**).
- ▶ Obtaining 3270 sessions. This is easiest to do using the Terminal Solicitor that is part of FLEX-ES.
- ▶ Obtaining access to a 3270 session that is defined as a z/OS operator console. This can be done through the Terminal Solicitor (provided that the 3270 session is not started in the shell script used to start the S/390 instance).

There are many variations possible, but operation might be something like this:

- ▶ From a remote system, **telnet** to your server and log in as *flexes*. Then **su** to *root*. Enter your normal **resadm** startup command, such as **resadm -s R14A.rescf**.
- ▶ Exit from root (that is, return to user ID *flexes*) in the telnet session. Execute your normal FLEX-ES startup shell script, for example, **sh sh14A**. This should return the normal *flexes* prompt. You must do this before you can see the 3270 sessions in the Terminal Solicitor.
- ▶ Start a TN3270 client session, directed to the IP address of your EFS Linux server. It should display the Terminal Solicitor menu. Select the terminal session in the Terminal Solicitor that you have defined as a z/OS operator console. Be certain to start the operator TN3270 session before IPLing z/OS.
- ▶ Enter your normal **ip1** command at the *flexes* prompt.

Again, there are many variations on this theme. You might prefer to have two telnet sessions, with one operating as *root* and the other as *flexes*.

3.10.1 Sample configuration

Assume we have the configuration shown in Figure 3-7. We have a remote system (perhaps at home) and want to start and operate FLEX-ES and z/OS on a server located elsewhere (perhaps at your office). (You would not have the IP addresses shown in the example in this case, but that is a different issue.) The data volume used for the remote telnet and TN3270 sessions is low and these might be handled over a multiuser network. (There is also a security issue involved; there are several ways to address this, such as the use of **ssh**.)

³⁵ Yes, this mode of operation has been observed in several installations.

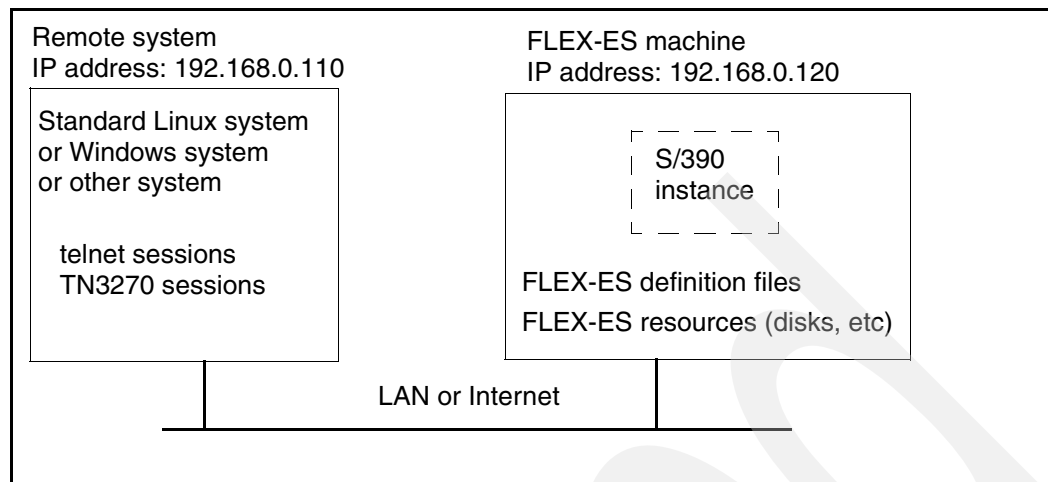


Figure 3-7 Simple remote operation

3.10.2 IPL with single remote MVS console

This is the most straightforward case. We will start our FLEX-ES from a remote computer and have the MVS master console appear on that remote computer. This remote system can be Linux or Windows or any other system than can provide **telnet** and TN3270 sessions. We use Linux for our examples, but realize that use of a Windows system would be more common.

Our FLEX-ES definitions are the same ones we have used for many examples. We have 3270 devices defined at addresses 700, 701, and so forth. We know that our z/OS system looks for a console at address 700. Part of our FLEX-ES definitions appear as:

```

....
cu devad(0x700,3) path(0) resource(CU3174)
.....
CU3174: cu 3174
interface local(1)
device(00) 3278 mstcon
device(01) 3278 L701
device(02) 3278 L702
end CU3174
  
```

Assume the definitions, when compiled, produced RC.rescf (resource file) and RC.syscf (system file). This is the shell script we used for starting the system. Notice that this shell script does not start any x3270 sessions. We placed this script in file RC0sh:

```

flexes RC.syscf
flexescli localhost RC
  
```

We assume our FLEX-ES machine is booted (that is, Linux is running) but no FLEX-ES functions are running. From another (remote) system, we **telnet** to the FLEX-ES machine:

```

$ telnet 192.168.0.120           (Working on our remote system)
  
```

We then logged into Linux as *flexes* and entered the following commands:

```

Remember: the following commands are in a telnet window (on the remote system)
and are being executed on the FLEX-ES server (via the telnet session)
$ su                               (Switch to root on the server)
# cd rundir                        (Assume our files are in rundir)
# resadm -s RC.rescf              (Start resource manager on server)
  
```

```

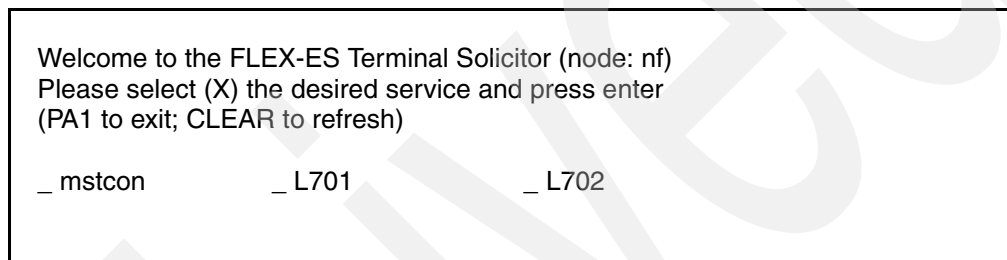
# resadm -r          (Verify it worked)
# exit              (Leave root)
$ cd rundir          (If not already there)
$ sh RC0sh           (Start shell script)
flexes>

```

At this point we need one or two TN3270 sessions (for the z/OS console and possibly for a TSO session). We want to connect them to the FLEX-ES Terminal Solicitor on the FLEX-ES server. This means a connection to port 24 on the IP address used by Linux on the server. How this is done varies with the operating system and TN3270 product on the remote machine. Using PCOM under a version of Windows is probably the most common situation. We used Linux on our remote system and used **x3270** as the TN3270 emulator:

```
$ x3270 -model 3 -keymap pc -port tn3270 192.168.0.120 & (Executed on remote system)
```

This produced a connection to the FLEX-ES Terminal Solicitor, which appeared (on the x3270 session) something like that shown in Figure 3-8.



```

Welcome to the FLEX-ES Terminal Solicitor (node: nf)
Please select (X) the desired service and press enter
(PA1 to exit; CLEAR to refresh)

_ mstcon      _ L701      _ L702

```

Figure 3-8 Terminal Solicitor

We selected (with an X) the session named **mstcon** (because we knew this was assigned to the 3270 device at address 700 in our FLEX-ES definitions). This provided a blank 3270 screen because we had not yet IPLed our z/OS system.

Returning to the window with the *flexes>* prompt in the telnet session on the remote system:

```
flexes> ip1 0A80 0A8200
```

We entered our normal IPL command. After a few seconds z/OS NIP messages appeared on the screen and the IPL process proceeded as usual. We later started another x3270 session to the Terminal Solicitor, selected L701, and started a TSO session.

Alternative method

If your remote system provides X windows support, you can include **x3270** commands in your shell script. The x3270 windows will then be started (via X windows functions) on your remote system. You would need to issue a previous **xhost +** command (or a more restrictive **xhost** command) earlier. We did not use this method because it is less general than using a remote TN3270 client that connects via the Terminal Solicitor.

3.10.3 Exposures

There are obvious security exposures to remote FLEX-ES operation, especially over the Internet. Our examples use telnet sessions, as simply as possible, to make the principles clearer. A secure alternative is use of **ssh**, the secure shell, instead of **telnet**, if your remote system provides this. This document is not about **ssh**; you can find ample documentation for it in UNIX and Linux sources. We note that, if you use **x3270** in your shell script to start an x3270 session on your remote desktop, that x3270 session will benefit from the encrypted ssh link.

There is a second exposure and this involves session or connection drops and timeouts. z/OS does not like to lose its last console. Where there is only one console, this is the last console. If the last console is lost, z/OS will attempt to use the “hardware console”. With FLEX-ES this produces output messages through the command line interpreter (that is, a window with the *flexes>* prompt). If the z/OS console (a TN3270 session) *and* the link needed to obtain a flexesci window are lost, z/OS might not recover (depending on exactly what is happening in z/OS at the time).³⁶

You can partly avoid this exposure by always starting a z/OS console on the FLEX-ES server, in addition to another z/OS console at your remote location. If your remote session fails, z/OS continues to run with its other console. You can then connect (or reconnect) a TSO session and use the ISPF LOG option to issue commands from your remote machine.³⁷ The next example illustrates the use of two consoles.

3.10.4 Remote operation with two z/OS consoles

Recent versions of the AD CD-ROM system have two MVS consoles defined. These definitions are in the CONSOL00 member of PARMLIB and look something like this:

```
CONSOLE
  DEVNUM(700)
  ALTERNATE(908)
  AREA(10)
  AUTH(MASTER)
  (more specifications for this console)
CONSOLE
  DEVNUM(908)
  ALTERNATE(700)
  (more specifications for this console)
```

This defines two consoles at addresses 700 and 908. Both will be MVS consoles *if present at IPL*. Either can be varied online as a console later, provided it is not used by VTAM. Later console additions involving one of these addresses (after IPL) require operator commands. Only addresses in the active CONSOLxx PARMLIB member can become consoles. The easiest way to handle additional MVS consoles is to have them online when the system is IPLed.

There is only one NIP console during IPL.³⁸ The NIP console receives the first few messages during system startup. There can be multiple NIP consoles defined in the system, but only the first available one is used. For the AD system we use for our examples, address 700 (if online at IPL) is the NIP console and the z/OS console. If 3270s at addresses 700 and 908 are both online at IPL, only 700 will be the NIP console but both will be z/OS consoles.

The NIP console might not be needed during a routine IPL. It is needed (that is, the operator must reply to messages there) if there are duplicate volsers online, or if the monoplex operation needs to be initialized, or if something else is wrong early in the IPL process. If you

³⁶ FLEX-ES can always send messages to the S/390 hardware console because the FLEX-ES console (which includes S/390hardware console emulation) is a virtual console. The **flexesci** command provides a connection to this virtual console. In this case, we assume the remote operator has lost the link to the FLEX-ES machine and can access neither 3270 interfaces or **flexesci** interfaces.

³⁷ You might even restart your failed z/OS console by using commands through ISPF. See “Restarting the MVS console” on page 12 for more about this.

³⁸ This is the Nucleus Initialization Program (NIP) function, which has control for a short time after the IPL process starts. The z/OS consoles we mention are also known as MVS consoles or MCS (Multiple Console Support) consoles. If your normal console is both the NIP and z/OS console (as is the case in almost all the examples in this EFS series of redbooks), you will see the console screen change format after the first few messages. This is the point where the ownership of the console goes from NIP to MCS. At that point any additional MCS consoles become active.

want to start and operate your z/OS system remotely, it is safer to arrange for your remote console to be both the NIP console and a z/OS console. For our example, we do this by using the 3270 at address 700 as the remote console.

We changed our FLEX-ES definitions to include the second z/OS console:

```
...
cu devad(0x700,3) path(0) resource(CU3174)
cu devad(0x908,1) path(0) resource(CU3174B)
...

CU3174: cu 3174
interface local(1)
device(00) 3278 mstcon
device(01) 3278 L701      #You could define more terminals (L703, L704, etc) for
device(02) 3278 L702      #additional TSO users, of course
end CU3174

CU3174B: cu 3174
interface local(1)
device(00) 3278 mstcon2
end CU3174B
...
```

Using these definitions the terminal named mstcon will be at address 700 and the terminal named mstcon2 will be at address 908. We want mstcon (address 700) to appear on our remote PC so we have access to NIP and z/OS messages. We want mstcon2 to appear in a window on the FLEX-ES server desktop so that a full z/OS console will survive if the remote PC connection is lost.

We modified our shell script slightly, to automatically start an x3270 session (on the Server system) connected to mstcon2:

```
flexes RC.syscf
x3270 -display nf:0.0 -model 3 -keymap pc -port tn3270 nf:mstcon2 &
flexesccli localhost RC
```

We then followed exactly the same startup process described in “IPL with single remote MVS console” on page 50 (using our modified shell script). This produced the “normal” z/OS console (including NIP) on the remote PC and started another z/OS console on the FLEX-ES server.

Recovery exercise

We tested the environment just described by activating FLEX-ES remotely and IPLing z/OS remotely. Both z/OS consoles worked as expected: one on the remote system (device address 700) and one on the FLEX-ES server desktop. We then disconnected the Ethernet cable on the remote system.

The remaining z/OS console (on the FLEX-ES server) continued to operate. After several minutes the following messages appeared on the remaining console (on the server) and were repeated every six seconds:

```
IOS075E 0700,RECURRING MIH CONDITION FOR THIS DEVICE
IOS076E 0700,00,*MASTER*,CLEAR SUBCHANNEL INTERRUPT MISSING
```

These recurring messages could eventually fill the spool volumes, but this would take a considerable time. We suspect that the nature of these error messages may depend on the state of the remote console at the exact instant it was disconnected.

We reconnected the LAN cable to the remote system and started another x3270 session connected to the FLEX-ES Terminal Solicitor. The missing console, named mstcon in our examples, appeared in the Terminal Solicitor list of available sessions. (This means that FLEX-ES detected that the terminal was disconnected. This might not always happen, depending on the exact state when the session failure occurred.)

We selected mstcon in the Terminal Solicitor and received a blank 3270 screen. This is because, even though the 3270 device at 700 was connected to the S/390 again, z/OS does not automatically restart console activity after a console failure. We started another x3270 session on the remote system, connected to the Terminal Solicitor, and selected one of the other terminal sessions (L701). This is a VTAM terminal and it presented the VTAM logo screen. We logged onto TSO, started ISPF, and used the LOG interface to issue the commands:

```
V 700,OFFLINE
V 700,ONLINE
V 700,CONSOLE
```

This restarted console activity on our remote z/OS console.

An alternate console recovery method is:

- ▶ Reconnect the 3270 console (probably via the Terminal Solicitor) and press Enter to create a pending I/O operation. (If an error is displayed, use the 3270 Reset key and press Enter again.)
- ▶ From the *flexes*> prompt, enter **ext** to create an external interrupt. You may need to do this twice.

Comments

The two-console operation just described is a bit complex and you may need to experiment a little to make it work in your environment. Also, console recovery is not always exactly the same process and may require some experience to work through it. It may not always work.

Should you attempt to use this two-console operation? Most EFS users will probably not need it. However, if you run serious work with an unattended system, you might consider it. We suggest that the described technique is most workable if you routinely have a z/OS console visible at your remote location and if someone checks it periodically. This implies that you would be aware of a link failure within a few hours (at the most) and could take actions to reestablish your remote z/OS console.

There are other ways to produce remote windows from Linux, using X windows functions, and you may find other methods more attractive. The setup described here is only one of many ways to implement remote consoles and operation.

3.11 SNA adapter number

SNA XCA and OSA control units (both token ring and Ethernet) assign adapter numbers to each LAN interface. These numbers are in the range 1 through 255. Zero is not a valid adapter number. The equivalent FLEX-ES definition might be:

```
CU3172X: cu xcasna
options 'adapternumber=1'           # Zero is not a valid choice
interface local(1)
device(00) scasna eth0
end CU3172X
```

The adapter number has no relation to a physical slot in the server or to the Linux device name. It relates this FLEX-ES definition to a PORT definition in VTAM.

Archived

Archived

Raw disk devices

The FLEX-ES design is optimized for the use of *raw disk devices* for emulated S/390 DASD. The *Getting Started* publication does not address this and uses standard Linux files for emulated S/390 DASD. This was done for two reasons:

- ▶ Using standard Linux files for emulated DASD is simpler to understand and implement, especially for those with little Linux or UNIX experience.
- ▶ Many users of the *Getting Started* publication install FLEX-ES on IBM ThinkPads. The penalty for using standard Linux files appears slight where no serious S/390 workloads are encountered and ample unused real memory is available for Linux disk caches.

Any production FLEX-ES system should be implemented using raw disk devices for emulated DASD. Even ThinkPad users, after gaining experience and confidence using FLEX-ES, should consider using raw disk devices.

4.1 Background

A standard Linux file is typically in an *ext2* or *ext3* file system.³⁹ A file system usually exists in a disk partition (or might encompass the whole disk). The file system provides inodes and directory functions. When a Linux application (such as FLEX-ES) reads or writes a standard Linux file, the I/O routines associated with the file system provide many services, including:

- ▶ Directory processing, to find the file within the file system, to manage inodes and space within the file system, and to keep timestamps
- ▶ Blocking of data to fit the sector and block structure of the hardware and file system
- ▶ Using idle processor memory to cache disk data
- ▶ Flushing the cache to disk at appropriate times

These functions provide convenience and performance for typical applications. However, carefully written, highly optimized applications can be designed to manage their own I/O

³⁹ There are other file system types that are also considered standard, but these are the most common examples.

functions. Major data base products are a typical example and FLEX-ES is another example. Such applications often include these characteristics:

- ▶ They tend to work with a small number of large files
- ▶ They work with blocks of data that map onto hardware disk sectors
- ▶ They contain their own disk cache management

When such applications use standard files, there is unnecessary processing overhead (application processing and file system processing, with duplication of some work), and the application disk cache functions may conflict with the file system disk cache functions. The operation of two separate disk cache functions is quite apparent when FLEX-ES is used with standard Linux files for emulated DASD volumes.

The S/390 DASD emulation functions of FLEX-ES provide a disk cache at the emulated volume and control unit level. The use of raw disk devices eliminates unneeded file system processing overhead and allows the application (FLEX-ES, in our case) to completely control its disk cache functions. This reduces processor overhead and removes erratic disk response caused by two disk cache mechanisms (FLEX-ES and the Linux file system) that are not aware of each other.

4.1.1 Raw device

What is a *raw disk device*? In its basic form, it is a disk partition with no internal structure. That is, it is a partition with no file system; there are no inodes, no directories, no files in the partition. The whole partition can be used as *one* simple file by Linux.

For FLEX-ES, this means that a disk partition is needed for each emulated S/390 DASD volume that is accessed as a raw disk device. (Each emulated 3390, for example, is a *single* Linux file. A raw partition can be used as a *single* Linux file.⁴⁰) A considerable number of disk partitions may be required for a typical z/OS installation, one for each emulated disk volume.

Linux disk partitions are typically created with the **fdisk** utility, or something similar. This has several serious limitations:

- ▶ This command is dangerous. It is all too easy to destroy all the contents of a disk with a simple error using this command.
- ▶ It is difficult or impossible to make minor adjustments after the initial partitioning of a disk.
- ▶ The maximum number of partitions possible is small.

An alternative to “real” disk partitions (created with **fdisk**) is the use of a Logical Volume Manager (LVM) to create Logical Volumes. For most purposes, a logical volume can be treated as a partition. The current LVM implementation in Linux allows up to 256 logical volumes. This directly translates to a maximum of 256 emulated S/390 disk volumes.⁴¹

4.2 Basic implementation

It is possible to have all the Linux file systems (except the one containing the boot files) in logical volumes. We describe a somewhat simpler arrangement. When installing Linux (as described in Chapter 3 of the *Getting Started* publication), you can partition your disk as

⁴⁰ This statement should not be taken too literally; specific coding techniques are needed to make this work.

⁴¹ Very few FLEX-ES systems require more than this. It is possible to have some emulated volumes on raw disks and other emulated volumes as standard Linux files, although this would not be recommended unless more than 256 emulated disk volumes are needed.

shown in Figure 4-1. If you create an LVM partition during basic Linux installation, be certain to note the identifier (such as /dev/hda6).

When planning to use raw disk devices, Linux installation is done as described in the *Getting Started* publication. The only difference is in partitioning your disk. The general goal is illustrated in Figure 4-1. You can create the LVM partition during Linux installation (probably using Disk Druid), or you can create it later using **fdisk**.

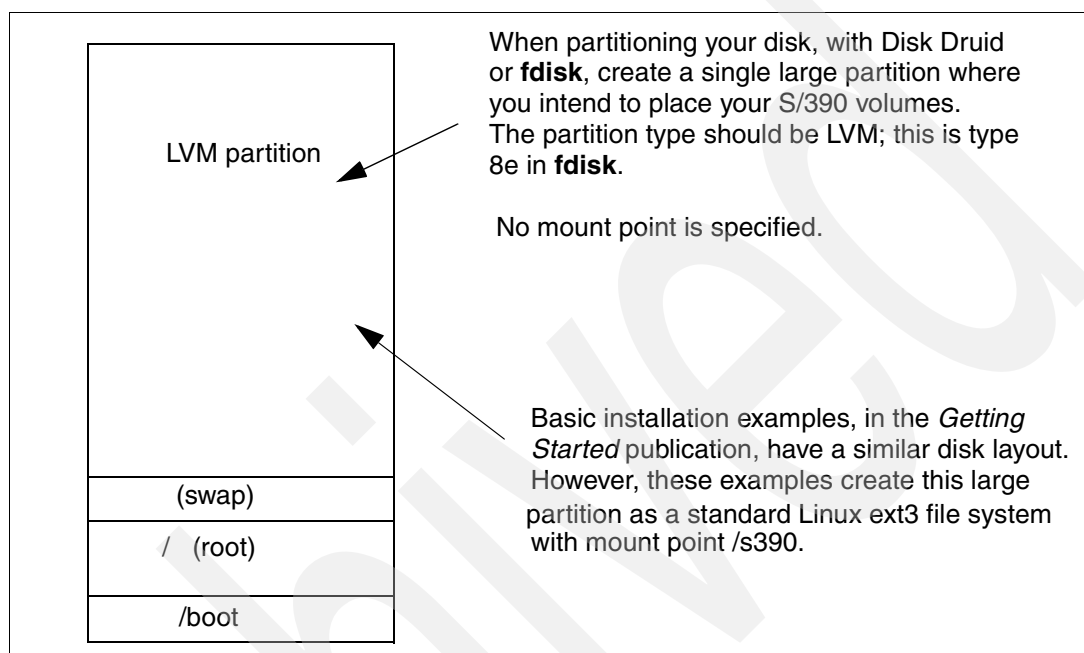


Figure 4-1 Basic disk arrangements for LVM

After installing Linux, FLEX-ES installation (which installs the FLEX-ES modules in /usr) is the same as described in *Getting Started*, and it can be done before or after your LVM customization.

4.2.1 Creating an LVM partition

If you did not create an LVM partition during Linux installation, but left a large unallocated disk area instead, you can create the LVM partition later. The following is an example of this:

```
# su
# /sbin/fdisk /dev/hda
m
l
n
p
First cylinder (2584-6201, default 2584):
(Enter)
Using default 2584
Last cylinder or +size or +sizeM or +sizeK (2584-6201, default 6201):
(Enter)
Using default value 6201
p
...
...
Start   End     Blocks  Id
/dev/hda6 2584   6201   27352048 83
...
```

(Must be root to use fdisk)
(Assume we want to work on hda)
(Display fdisk menu)
(Display partition types. Find LVM)
(Create new partition)
(Create primary partition)
(Your cylinder numbers will differ)
(List partitions)

```

t                                     (Edit a partition description)
Partition number (1-6): 6
Hex code (type L to list codes): 8e      (Type 8e is Linux LVM)
Changed system type of partition 6 to 8e (Linux LVM)
w                                     (Write partition table and exit)
#
(reboot Linux to make changes effective)

```

Be very careful using **fdisk**. It is best to do this while installing Linux or very shortly thereafter. (The reason being that there is less work lost if you make a mistake and need to reinstall everything!)

LVM options

LVM can do many wonderful things. It can combine multiple physical disks into a single logical disk, for example. Some LVMs can perform mirroring and various RAID functions. It can safely resize partitions. It can add new disks to expand the space in an existing logical disk. In our simple case, we use none of these functions. We use LVM in one “real” partition of our disk solely to create multiple logical volumes. You can use all the available LVM functions if you wish (and if you have some experience planning such things). If you are new to LVM, we suggest you follow our method first. You can expand upon it later.

Naming conventions

Both LVM and raw devices provide additional levels of name redirection. This can become confusing unless you adopt naming conventions that are intuitive to you. Logical volume names and raw device names (two separate items) can use any name you like. You can name logical volumes AAA, bbb, x or anything you like within the scope of normal Linux file names.

We decided to name logical volumes to match the volser of the S/390 volume we intend to install in that logical volume. It is easy to rename logical volumes, so you can change names to match new S/390 system releases. We decided to name raw disk devices with the S/390 address (device number) where we intend to use the device. For example, users of the AD CD-ROM system often place the IPL volume at address A80 and we use A80 as the raw device name.

This naming is arbitrary. Our use of upper case names is arbitrary. You can use any naming scheme you devise, but we suggest you think about it first. It is easy to become confused by too many names and links, even in a small system.

4.2.2 Initial LVM creation

Assuming we have a disk partitioned as described above, with `/dev/hda6` being an LVM partition, we proceed as follows:

```

# su                                     (Must be root to do this work)
# /sbin/pvcreate /dev/hda6
pvcreate -- physical volume "/dev/hda6" successfully created
# /sbin/vgcreate vg390 /dev/hda6          (The name "vg390" is arbitrary)
vgcreate -- INFO: using default physical extent size 4 MB
vgcreate -- INFO: maximum logical volume size is 255.99 Gigabytes
vgcreate -- doing automatic backup of volume group "vg390"
vgcreate -- volume group "vg390" successfully created and activated

```

Our goal is to create four 3390-3 volumes. The eventual volsers that we plan to install are S4RES1, S4RES2, OS39M1, and S4USS1.

```

# /sbin/lvcreate -L 2792M -n S4RES1 vg390 (Create logical volume, size 2792 MB)
lvcreate -- doing automatic backup of "vg390"

```

```

lvcreate -- logical volume "/dev/vg390/S4RES1" successfully created
# /sbin/lvcreate -L 2792M -n S4RES2 vg390
# /sbin/lvcreate -L 2792M -n OS39M1 vg390
# /sbin/lvcreate -L 2792M -n S4USS1 vg390
# /sbin/lvscan
lvscan -- ACTIVE      "/dev/vg390/S4RES1" [2792 MB]
lvscan -- ACTIVE      "/dev/vg390/S4RES2" [2792 MB]
lvscan -- ACTIVE      "/dev/vg390/OS39M1" [2792 MB]
lvscan -- ACTIVE      "/dev/vg390/S4USS1" [2972 MB]
lvscan -- 4 logical volumes with 11168 MB total in 1 volume group
lvscan -- 4 active logical volumes
# chown flexes:flexes /dev/vg390/S4RES1 (Userid "flexes" must own logical volumes)
# chown flexes:flexes /dev/vg390/S4RES2
# chown flexes:flexes /dev/vg390/OS39M1
# chown flexes:flexes /dev/vg390/S4USS1

```

LVM has an operational characteristic we must mention. When some releases of Linux are booted, LVM is inactive. In these cases, it must be activated before it can be used. The **vgcreate** command in the above sequence activated it. We need to introduce a command after each boot (but not the **vgcreate** command) to activate LVM before we use it. This was the case with RH8.0. Later Linux releases may activate LVM automatically. If this happens, you can remove the **vgchange** command used in the shell script described later.⁴²

The appropriate logical volume sizes are:⁴³

3390-1	933 MB
3390-2	1862 MB
3390-3	2792 MB
3390-9	8382 MB

These numbers are from the *FSIMM100: Planning Guide*. When discussing both memory and disk sizes, FSI uses 1 MB = 1024 KB = 1048576. A logical volume may be larger than the required emulated disk size.

4.2.3 Raw devices

Red Hat Linux defines raw device nodes in /dev/raw. If you look there (with **ls /dev/raw**) you will find 128 nodes (representing character devices) named raw1, raw2, raw3, and so forth. We could use any of these for our raw devices. Instead, we will create our own names, such as A80, A81, and so forth. The sole reason for not using the existing raw nodes is that we want more meaningful names.

Linux has a maximum of 256 raw device nodes. Red Hat Linux already has 128 of these defined using the names raw1, raw2, and so forth.⁴⁴ This leaves us with another 128 that we can define. (Red Hat's previously defined 128 names may be deleted if desired.) We decided to define additional raw device nodes that we may need for other emulated DASD volumes we may add later. These nodes take very little space. The syntax we need is this:

```

mknod /dev/raw/name c 162 nnn
|      |           |  |      +---minor number; must be unique; max 255
|      |           |  |      +-----major number; 162 is for raw devices
|      |           |  +-----c means a character device; required here
|      +-----Linux uses /dev/raw/xxxx for raw device names

```

⁴² If LVM is already activated and you attempt to activate it again, you will receive a message about "LVM already activated" and no harm is done.

⁴³ These sizes are for single slices, or the LVM equivalent, as described in this redbooks series. The total size when multiple slices are used for a single volume might be slightly larger.

⁴⁴ As far as we know, these defined raw nodes are not used for anything in the simple Linux implementations described in these redbooks.

+-----command

Minor numbers range from 0 through 255. Red Hat Linux already used the first 128 of these for its predefined raw nodes, so we must start with number 129. We proceed as follows:

```
# mknod /dev/raw/A80 c 162 129
# mknod /dev/raw/A81 c 162 130
# mknod /dev/raw/A82 c 162 131
# mknod /dev/raw/A83 c 162 132
# mknod /dev/raw/A84 c 162 133
# mknod /dev/raw/A85 c 162 134
# mknod /dev/raw/A86 c 162 135
# mknod /dev/raw/A87 c 162 136
# mknod /dev/raw/A88 c 162 137
# mknod /dev/raw/A89 c 162 138
# mknod /dev/raw/A8A c 162 139
# mknod /dev/raw/A8B c 162 140
# mknod /dev/raw/A8C c 162 141
# mknod /dev/raw/A8D c 162 142
# mknod /dev/raw/A8E c 162 143
# mknod /dev/raw/A8F c 162 144
# mknod /dev/raw/A90 c 162 145
# mknod /dev/raw/A91 c 162 146
# mknod /dev/raw/A92 c 162 147
# mknod /dev/raw/A93 c 162 148
# mknod /dev/raw/A94 c 162 149
# chown flexes:flexes /dev/raw/A*
```

User ID *flexes* must own the raw device nodes, and **chown** is used to transfer ownership to *flexes*. Again, we mention that our names are arbitrary. The following example uses only four of these nodes. We show the extra node definitions merely to illustrate what can be done. Alternatively, we could issue new **mknod** commands whenever we needed new nodes.

Shell script

We must link a logical volume to a raw device before we can use it for FLEX-ES. The command to do this is **raw**. A characteristic of Linux is that these raw device links are temporary; they are lost when you reboot a system. For this reason, we place them in a shell script that we can run before starting FLEX-ES. We also mentioned that, for some Red Hat releases, LVM must be activated after booting. We can place the necessary commands in the same shell script. Both the **raw** commands and the LVM commands must be run under *root*. We created a shell script in `/usr/flexes/rundir/shLV` containing the following:

```
/sbin/vgscan
/sbin/vgck
/sbin/vgchange -ay vg390
raw /dev/raw/A80 /dev/vg390/S4RES1
raw /dev/raw/A81 /dev/vg390/S4RES2
raw /dev/raw/A82 /dev/vg390/OS39M1
raw /dev/raw/A87 /dev/vg390/S4USS1
echo 'Raw devices defined'
```

We already have a shell script that we use when starting a FLEX-ES instance. We cannot combine the two shell scripts because one (`shLV`) must be run as *root* and the other (`shR14Araw` in our examples) must be run as *flexes*.

You can rerun this script multiple times, perhaps after changing it. This may produce warnings about the volume group being already activated, but this seems to do no harm.⁴⁵ If your Linux activates LVM automatically, you can remove the **vchange** command from the script.

A more sophisticated shell script that does both the **mknod** and **raw** commands automatically is shown in “Production implementation” on page 66. We have stepped through the individual commands in this discussion to make the flow clearer. We use the steps shown here for the systems used while producing various IBM Redbooks.

4.2.4 Loading the AD system

Many FLEX-ES users start with the z/OS AD CD-ROM system. Loading this system is described in the *Getting Started* publication. Loading it with raw devices is almost the same as loading into standard Linux files. Before starting this, you must have the logical volumes and raw device links usable. This is done by running the shell script just described.

The key command for installing the current AD CD-ROM systems is:

```
$ unzip -p /mnt/cdrom/zos14/s4res1.zip | ckdconvaws - /dev/raw/A80 3390-3
```

This command is missing the **-r** option (for **ckdconvaws**) that would be used if loading a standard Linux file. Note the space before and after the “-” in the **ckdconvaws** command. The “-” indicates that **ckdconvaws** is to read *stdin*, which is supplied through the pipeline from **unzip**. The loading function would be repeated for each CD that is being loaded, using the appropriate file and raw node names.

4.2.5 FLEX-ES resource definitions

The FLEX-ES definitions must match the raw device names. We copied an existing defR14A file (described in the *Getting Started* publication) and named the new file defR14Araw. In the new definition file we changed the system name to S14Araw and the resource name to R14Araw. This avoids conflicts with existing file names. Our new definition file contains:

```
system S14Araw:
memsize(524288)
cachesize(2048)
instset(z)
cpu(0)
channel(0) local
channel(1) local
channel(2) local
cu devad(0xA80,8) path(2) resource(CU3990)
cu devad(0xA90,2) path(2) resource(CU3991)
cu devad(0x700,3) path(0) resource(CU3174)
cu devad(0xE20,2) path(1) resource(CU3172)
end S14Araw
```

```
resources R14Araw:
CU3990: cu 3990
interface local(1)
options 'trackcachesize=500'
device(00) 3390-3 /dev/raw/A80
device(01) 3390-3 /dev/raw/A81
device(02) 3390-3 /dev/raw/A82
device(03) 3390-3 OFFLINE
device(04) 3390-3 OFFLINE
device(05) 3390-3 OFFLINE
device(06) 3390-3 OFFLINE
device(07) 3390-3 /dev/raw/A87
end CU3990
```

⁴⁵ However, do not change and rerun the script (or enter **raw** commands) to reassign LVM devices to different raw devices while FLEX-ES is running. This could produce unexpected results.

```

CU3991: cu 3990
interface local(1)
device(00) 3390-1 OFFLINE
device(01) 3390-1 OFFLINE
end CU3991

CU3174: cu 3174
interface local(1)
device(00) 3278 mstcon
device(01) 3278 L701
device(02) 3278 L702
end CU3174

CU3172: cu 3172
interface local(1)
options 'ipaddress=192.168.0.111,adapternumber=0'
device(00) 3172 eth0
device(01) 3172 OFFLINE
end CU3172

end R14Araw

```

The only required changes are the Linux names associated with the DASD devices, but we also added a floating cache of 500 tracks. We then made a new shell script to start a FLEX-ES instance; a new script is needed because of the name changes. We created shR14Araw containing:

```

flexes S14Araw.syscf
xmod -e 'keysym Alt_L Meta_L'
xset fp+ /usr/flexes/fonts
xset fp rehash
x3270 -model 3 -keymap pc -port tn3270 localhost:mstcon &
x3270 -model 3 -keymap pc -port tn3270 localhost:L701 &
flexesccli localhost S14Araw

```

Some of these names, such as R14Araw, are awkward to type and are used only to emphasize the relationship with files described in earlier redbooks. You could use shorter, easier names.

4.2.6 Typical FLEX-ES startup

We normally start FLEX-ES operation using two Linux windows. We added one additional step to our startup to invoke the shell script with the LVM and raw commands:

```

log into Linux as flexes
Window 1:
$ su                                (Must be root for "raw" and "resadm")
# cd rundir                         (Our working files are here)
# sh shLV                           (Execute the script with LVM and raw commands)

vgscan--reading all physical volumes (This may take a while...)
vgscan--found inactive volume group "vg390"
vgscan--"/etc/lvmtab" and "/etc/lvmtab.d" successfully created
vgscan--WARNING: This program does not do a VGDA backup ....
vgck --VGDA of "vg390" in lvmtab is consistent
vgck --VGDA of "vg390" on physical volume is consistent
vgchange--volume group "vg390" successfully activated
/dev/raw/raw129: bound to major 58, minor 0
/dev/raw/raw130: bound to major 58, minor 1
/dev/raw/raw131: bound to major 58, minor 2

```

```

/dev/raw/raw136: bound to major 58, minor 3
Raw devices defined
# resadm -s R14Araw.rescf          (Start FLEX-ES resource manager)
    (wait about 5 seconds)
# resadm -r                        (Verify the resources are ready)
Window 2:
$ cd rundir
$ sh shR14Araw
flexes> ip1 a80 0a82CS
flexes>

```

Other useful commands

When using raw devices, command programs provided with FLEX-ES use the raw file names in the commands. For example:

```
$ ckdfmt -n /dev/raw/A90 3390-1
```

Notice that the `-r` flag is missing from the `ckdfmt` command shown here. The `-r` flag indicates that the named file should be created if it does not already exist. However, the raw file (node) must exist before you can use these commands, so this flag is not needed. Before using these commands you must link the raw node with a logical volume; this is done with a `raw` command, as shown in “Shell script” on page 62.

Other useful commands include:

```

# /sbin/lvrename /dev/vg390/S4RES1 /dev/vg390/Z5RES1  (Rename a logical volume)

# rm /dev/raw/A94                                     (Delete a raw node)

# cat /proc/devices                                   (List major device numbers)

# /sbin/vgdisplay vg390                               (Volume group statistics)
---Volume Group---
VG name      vg390
...
Alloc PE/Size    3072 / 12 GB
Free PE/Size     3604 / 14.08 GB
...

# /sbin/lvscan                                         (Logical volume data)
lvscan -- ACTIVE  "/dev/vg390/S4RES1" [2792 MB]
lvscan -- ACTIVE  "/dev/vg390/S4RES2" [2792 MB]
...

# /sbin/vgcfgbackup                                   (Back up key LVM data)
# /sbin/lvremove /dev/vg390/S4RES1                    (Delete a logical volume)

```

The `vgdisplay` command shows the amount of used and free space in the volume group. The `lvscan` command shows the size of each logical volume; this is useful if you forget which volume is a 3390-3 or a 3390-1, for example.

You can verify the volser of a volume associated with a particular raw device. The volume must be formatted with a volser, of course:

```

$ ckddump /dev/raw/A80 0 0 | grep VOL1
00000000 E5D6D3F1                                *VOL1
00000000 E5D6D3F1 E2F4D9C5 E2F14000 00000101    *VOL1S4RES1

```

In this example the volser is S4RES1. The first argument for `ckddump` is the raw device node name. You must have previously used a `raw` command to link this node to a logical volume or partition.

If you are not certain what is mounted at a S/390 device address, you can use the command:

```
flexes> d mount A80 (Or whatever address interests you)
/dev/raw/A80
```

The CLI command **d mount <addr>** is very useful to verify a FLEX-ES definition for a running system.

4.3 Production implementation

Production FLEX-ES systems, as typically delivered by FSI business partners, often start the resource manager automatically by placing **resadm** commands in S90FLEXES files in `/etc/rc.d/rc2.d`, `rc3.d`, and `rc5.d`.⁴⁶

We can also place shell commands in the S90FLEXES files to create the necessary raw links. These could be hard coded, such as listed in “Shell script” on page 62. If we establish a different naming convention for logical volume names, we can create a more general set of shell commands in which we do not need to list each logical volume. The following example assumes that logical volume names for 3390 and 3380 volumes begin with the character 3, and that logical volume names for 9345 volumes begin with the character 9. This extract from an S90FLEXES shell script was provided by Gary Ehemann:

```
<snip>
#
# add chown/chgrp lines for raw disk slices here.
#-----following LVM logic inserted by Gary Ehemann-----
# make sure LVM volume groups we need are started
/sbin/vgscan
/sbin/vgck
/sbin/vgchange -ay s390vg
# first remove existing /dev/raw/<volumenames> nodes
for name in /dev/raw/[39]*
do
    eval "rm -f $name"
done
# now create new /dev/raw nodes with minor numbers starting
# at 129.
i=128
for name in /dev/s390vg/[39]*
do
    i=$((i+1))
    # make the /dev/raw/<volumename> device
    eval "mknod /dev/raw/'basename $name' c 162 $i"
    # make sure flexes owns it
    eval "chown flexes:flexes /dev/raw/'basename $name' "
    # equate the raw device to the block device
    eval "raw /dev/raw/'basename $name' $name"
done
# make sure flexes owns the block devices, too.
chown flexes:flexes /dev/s390vg/[39]*
<end-snip>
```

This example uses slightly different naming conventions than we used earlier. The logical volume group is `s390vg` instead of `vg390`, for example. You are not intended to use this code exactly as shown. It is intended to illustrate an approach that may be useful. Note that it removes and recreates the raw device names at each Linux boot when S90FLEXES is executed during the boot process. It also produces raw device node names that are

⁴⁶ These files are actually links to a single file, `/etc/init.d/FLEXES`.

coordinated with the logical volume names. This can reduce confusion about names in a larger system. For example, `/dev/raw/33903_z43res` is automatically linked to `/dev/s390vg/33903_z43res`.

We have not discussed examples of production rc scripts for several reasons:

- ▶ FSI business partners each have their own naming conventions and styles for FLEX-ES installation.
- ▶ Design and installation of significant code in Linux rc scripts is, perhaps, not a good exercise for new Linux users.
- ▶ Experienced Linux users will undoubtedly have their own styles for designing and coding such scripts. If you are in this category, we suggest you study the example carefully before creating your own version.

Archived

S/390 volume distribution

S/390 software usable on a FLEX-ES system can be packaged in a number of ways, with various goals and options, including:

- ▶ Physical packaging:
 - Tape (4mm, 3480/90, DLT), depending on the user's available hardware
 - CD
 - DVD
 - Network deliverable (probably FTP)
- ▶ Operating environment needed to process distribution:
 - S/390
 - Linux or UnixWare (without emulated S/390 active)
- ▶ Smallest unit of use:
 - Whole S/390 volume
 - Individual S/390 data sets (using an ADRDSSU dump, for example)
- ▶ Installation target:
 - Simple Linux or UnixWare file
 - Raw device or slices
- ▶ Emulated disk format:
 - FLEX-ES emulated disks
 - AWSCKD emulated disks
- ▶ Compression technique, if used:
 - tar compression
 - gzip compression
 - PC zip/unzip compression

It would be nice, of course, to have all distributions in several of these formats. This is unlikely to happen for practical reasons. It would be nice to have distributions in a form that is most widely usable. This requires some thought.

The most common current technique, for ITSO/EFS users and Partner World for Developer FLEX-ES users, is CD-ROMs with files in AWSCKD format compressed with a PC zip program. This format is used for compatibility with previous systems, but the requirement for such compatibility will end soon and this format is unlikely to be used after that.

5.1 S/390 processing environment

This environment means that the distribution is processed by a S/390 program. This implies either:

- ▶ An IPLable distribution, such as a distribution that starts with a stand-alone S/390 restore program.
- ▶ A usable S/390 operating system already is operational and will be used to install the new distribution. For example, an existing z/OS could be used to install a new ServerPac release.

Distributions to be processed in a S/390 environment are almost always in tape format. This includes:

- ▶ Tape volumes, assuming the intended users have compatible tape drives on their FLEX-ES systems.
- ▶ Emulated tape volumes in FakeTape, AWSTAPE, or OMA/2 format. The emulated tape volumes can be delivered by CD, DVD, FTP, or tape (although this is unlikely). The file containing the emulated tape can be used from a Linux/UnixWare disk or a mounted CD or DVD.

The emulated tape format might be compressed on the distribution media and would need to be uncompressed before use. Typical compression is with the **gzip** or **tar** (with compression option) programs. Compression is solely to reduce the size of the file, typically so that it will fit on a CD.

A volume backup, created by the z/OS ADRDSSU utility for example, might be considered a distribution. This format has a strong advantage in that individual data sets can be restored. It has the disadvantage that a working S/390 environment is needed to restore anything from it. An ADRDSSU backup (or a z/VM DDR backup) can be directed to an emulated tape drive (typically in FakeTape format). The result is a single Linux file that contains the complete emulated tape (which might have many internal files and tape marks). This Linux file can be compressed (if desired), written to CD, or FTPed to another machine.

This is a convenient distribution method for an application or data when the existence of an operational S/390 system can be assumed. The distribution need not be a volume dump. It can be any form that is written to tape (which we assume to be an emulated tape device, using FakeTape). It could be a PDS in unloaded format, for example. The emulated tape can have standard volume labels and be handled just as it would be with a “real” tape drive. See “Tape usage” on page 145 for more information about FakeTape operation.

It is also a convenient format because it is independent of whether simple Linux files or raw devices are used for emulated DASD volumes.

5.2 Linux or UnixWare processing environments

This environment means that the distribution is processed by a Linux or UnixWare program. This might be a standard utility (such as **tar**) or one of the utilities provided with FLEX-ES. A key factor is that there are two FLEX-ES disk formats involved:

- ▶ Simple Linux files. All commands, including **tar**, can be used with these.
- ▶ Raw devices (or multiple slices in the FLEX-ES UnixWare environment⁴⁷). The standard utilities, such as **tar** and similar Linux or UnixWare programs, are not compatible with these.

A discussion of simple Linux files versus raw devices appears in Chapter 2 of the *Getting Started* publication and you should read that for a more complete discussion. For the present discussion, we will simply note that the raw device interfaces are recommended for larger, serious, and production FLEX-ES systems, while the use of simple Linux files is typically confined to smaller, non-production, lightly-loaded, educational/demonstration systems.

FLEX-ES provides three utilities that are relevant for distributions: **ckdconvaws**, **ckdbackup**, and **ckdrestore**. These are unique in that they work with standard Linux files *and* raw devices. We do not discuss **ckdconvaws** further because it is probably not appropriate as a future distribution technique.

The **ckdbackup** and **ckdrestore** commands are complimentary, of course. They function with all the disk formats that are used for FLEX-ES emulated DASD:

- ▶ Simple Linux (or UnixWare) files, as described in the *Getting Started* publication
- ▶ A raw device that is pointing to a block device, as described in “Raw disk devices” on page 57
- ▶ A block device, such as a logical volume or “real” partition
- ▶ Raw disk slices, as used with the UnixWare version of FLEX-ES

We recommend using these commands for S/390 volume distributions and suggest that organizations using other techniques consider switching to these commands in the future.

5.2.1 Standard dd command

The standard Linux/UnixWare **dd** command can be used to transfer raw device files to or from standard files. This is described at length in FLEX-ES document *FSIMN020: Reloading from Emulated DASD*. This can be a useful technique within a system using raw devices, but has limitations when multiple slices are involved (a concern with UnixWare-based FLEX-ES systems).

We cannot recommend use of **dd** as a standard *distribution* tool. (A special case involves FBA devices. These are distributed and installed with **dd** commands.)

5.3 Practical distribution creation

The following scenarios use **ckdbackup** and **ckdrestore** as the basis for creating and installing emulated S/390 volumes. We assume that compression is needed in order to fit the distribution file on a CD. The examples work with one S/390 volume. A distribution with multiple volumes would simply repeat the process for each volume.

This discussion does not address how to create the original volume on your FLEX-ES system. We assume that, however you created it, you now want to package it for distribution. Our examples use the S4RES1 volume from a recent AD CD-ROM release as an example,⁴⁸ but this could be any emulated CKD DASD volume.

⁴⁷ This series of redbooks does not generally discuss the UnixWare version of FLEX-ES. However, it is particularly relevant in the discussion of distribution formats and is mentioned where needed.

⁴⁸ Perhaps we modified the contents of the volume and want to distribute a new version of it.

5.3.1 Raw devices

To create a CD suitable for distribution we used the following general procedure. We prefer to place all the files intended for burning a CD in one directory and we created /tmp/burn for this purpose:

```
$ mkdir /tmp/burn
```

 (Arbitrary name and location)

We wanted to create a CD copy of our S4RES1 volume, which is on logical volume S4RES1 and is associated with /dev/raw/A80. (See “Raw devices” on page 61 for more information about this assignment.) We used **ckdbackup** and **gzip** to create a suitable CD copy in the burn directory:

```
# sh shLV
```

 (You must have your raw node linked to the appropriate Logical Volume before starting. Use whatever shell script or raw commands are appropriate.)


```
$ ckdbackup /dev/raw/A80 | gzip -c > /tmp/burn/S4RES1.gz
```

We added **gz** to the output file name to indicate that the file has been compressed by **gzip**. Using the CD-RW unit in our system, we wrote a CD containing only this file:

```
$ su
```

 (Must be root to burn CD)

```
# cdrecord -scanbus
```

 (Can we see the CD writer?)

```
# ls -al /tmp/burn/*
```

 (Confirm file is present)

```
# mkisofs -R /tmp/burn/* | cdrecord -v fs=6m dev=0,0 -
```

 (Burn CD)

```
# rm /tmp/burn/S4RES1.gz
```

 (Clean up if you are short on space)

The CD that is produced can be restored (using **ckdrestore**) to a simple Linux file or to a raw device. You could use this CD in the restore example in “Standard Linux files” on page 72.

To install (restore) this CD, we can use:

```
$ mount /dev/cdrom /mnt/cdrom
```

 (If automount does not do it for you)

```
$ ls -al /mnt/cdrom
```

 (Verify file name on CD)

```
$ gunzip -c /mnt/cdrom/S4RES1.gz | ckdrestore - /dev/raw/A80
```

If the target location is a raw device, the raw device must exist (and be large enough) but no prior formatting of the raw device is necessary. We mounted the CD and used **gunzip** piped to **ckdrestore** to install the volume. There is no need to restore it to /dev/raw/A80, of course; it could be restored to any suitable raw device (or standard Linux file).

Be careful to enter the **ckdrestore** command correctly. Note the “minus sign” that is separated by blanks.

5.3.2 Standard Linux files

[This text is almost identical to the text in the last section. Please excuse the repetition. We think it makes the meaning and comparison clearer.]

To create a CD suitable for distribution we used the following general procedure. We prefer to place all the files intended for burning a CD in one directory and we created /tmp/burn for this purpose:

```
$ mkdir /tmp/burn
```

 (Arbitrary name and location)

We wanted to create a CD copy of our S4RES1 volume, which is located in /s390/S4RES1. We used **ckdbackup** and **gzip** to create a suitable CD copy in the burn directory:

```
$ ckdbackup /s390/S4RES1 | gzip -c > /tmp/burn/S4RES1.gz
```

Using the CD-RW unit in our system, we wrote a CD containing only this file:

```
$ su                                (Must be root to burn CD)
# cdrecord -scanbus                (Can we see the CD writer?)
# ls -al /tmp/burn/*               (Confirm file is present)
# mkisofs -R /tmp/burn/* | cdrecord -v fs=6m dev=0,0 - (Burn CD)
# rm /tmp/burn/S4RES1.gz           (Clean up if you are short on space)
```

The CD that is produced can be restored (using **ckdrestore**) to a simple Linux file or to a raw device. You could use this CD in the restore example in “Raw devices” on page 72.

To install (restore) this CD, we can use:

```
$ mkdir /s390/restore              (Make a test directory)
$ ckdfmt -r -n /s390/restore/S4RES1 3390-3
$ mount /dev/cdrom /mnt/cdrom      (If automount does not do it for you)
$ ls -al /mnt/cdrom                (Verify file name on CD)
$ gunzip -c /mnt/cdrom/S4RES1.gz | ckdrestore - /s390/restore/S4RES1
```

If the target location is a standard Linux file, the **ckdrestore** command *requires* that the target file (/s390/restore/S4RES1 in the example) *must* exist (and be the correct size) before the **ckdrestore** command is executed. We arbitrarily decided to install the volume in /s390/restore (because we did not want it to conflict with another S4RES1 file in /s390). We used **ckdfmt** to create a 3390-3 volume in the desired directory. We then mounted the CD and used **gunzip** piped to **ckdrestore** to install the volume.

Be careful to enter the **ckdrestore** command correctly. Note the “minus sign” that is surrounded by blanks.

We then used **cmp /s390/S4RES1 /s390/restore/S4RES1** to verify that the newly-installed volume was exactly identical to the original volume. We would not routinely do this, of course, but we wanted to verify the operation the first time we tried it.

5.3.3 Compression

The **gzip** program offers various levels of compression, expressed as a digit from 1 through 9. The value 1 provides the fastest operation, but with the least compression. The value 9 provides the best compression but with the slowest operation. The default value is 6.

We found the level 9 compression took as much as three times longer than level 6 compression. For example, one of our volumes needed 8 minutes at level 6 and over 25 minutes at level 9. The additional compression often produced output that was a few percent smaller than the default level 6 compression.

Based on this, you may or may not want to take the extra time for the additional compression. If you are burning a CD (unless you are just barely over the allowable size for a CD) it may not be meaningful. If you intend to FTP the files many times, the extra time spent in compression will be appreciated by the FTP users.

There is no time penalty when you **gunzip** a file with the additional compression.

5.4 A test

This section should be considered optional reading. It exercises your understanding of basic FLEX-ES use of raw devices.

We decided to verify that a **ckdbbackup** file made from a standard Linux file could be restored correctly to a raw device. We used the CD created as described in “Standard Linux files” on page 72. We used the raw device setup as described in “Basic implementation” on page 58. We already had our original S4RES1 volume on /dev/raw/A80 and we did not want to lose it.

After running our shLV script (to activate LVM and establish our four basic 3390 volumes) we created a new logical volume large enough for a 3390-3:

```
# /sbin/lvcreate -L 2792M -n S4RES1b vg390
```

and associated it with a raw device (A83) that we had already created but had not used:

```
# raw /dev/raw/A83 /dev/vg390/S4RES1b
```

We then restored the CD to this raw device:

```
$ gunzip -c /mnt/cdrom/S4RES1.gz | ckcrestore - /dev/raw/A83
```

We modified our FLEX-ES definitions to use this raw device at address A80:

```
.....
cu devad(0xA80,8) path(2) resource(CU3990)

.....
resources R14Araw:
CU3990: cu 3990
interface local(1)
options 'trackcachesize=500'
device(00) 3390-3 /dev/raw/A83      #This line changed
device(01) 3390-3 /dev/raw/A81
device(02) 3390-3 /dev/raw/A82
device(03) 3390-3 OFFLINE
device(04) 3390-3 OFFLINE
device(05) 3390-3 OFFLINE
device(06) 3390-3 OFFLINE
device(07) 3390-3 /dev/raw/A87
end CU3990
```

This end effect is that logical volume S4RES1b, associated with /dev/raw/A83, will be addressed as A80 when z/OS is IPLed. It is addressed as A80 because it is first in the emulated control unit that is addressed beginning with A80. We are using the newly-installed S4RES1 instead of the already-installed S4RES1 (associated with /dev/raw/A80 in our shLV script) when we IPL.

We later restored the FLEX-ES definitions to the original state and issued the following command to remove our test logical volume:

```
# /sbin/lvremove /dev/vg390/S4RES1b
```

All the **raw** commands are lost when Linux is shut down, so we did nothing to undo the **raw** command in this exercise.

5.5 Backup and restore considerations

Backup and restore operations are closely related to the creation and installation of distributions, as just described. A “distribution” might be intended for public use, while a “backup” is probably for a single system, but the operational techniques can be the same.

A choice must be made between S/390 utility functions (such as ADRDSSU or DDR) and server functions (such as **tar** or **ckdbbackup**). The S/390 utilities have the advantage that

individual data sets can be easily restored, while the server functions have the advantage of being simpler and faster. In either case, the result is typically a Linux file. This file might contain FakeTape output from ADRDSSU, or it might be the result of a **ckdbbackup** operation. The Linux file might be compressed (probably with **gzip**) simply to save space or to fit on a CD.

The most basic difference between a *backup* and a *distribution* is where the files are stored. A distribution is typically prepared on the system containing the original data and the original distribution files might be left there indefinitely. A backup, in most cases, needs to be moved away from the system that created the backup. One purpose of a backup is disaster recovery. If a hard disk fails and that disk contains both the operational volumes and their backups, the backups have not served their purpose.

A volume backup might be stored on the original system, but on a separate disk (assuming these are not both in the same RAID array). This would survive a hard disk failure, but would not survive a more general server disaster.

If your server has a high-performance tape drive (which is generally anything other than a 4mm drive), you might consider ADRDSSU backups directly to tape. This is the same process that is used in most “real” S/390 installations. However, you might find other alternatives (involving **ckdbbackup** and/or FakeTape) to be considerably faster.

5.5.1 Using ftp for backups

One way to back up S/390 volumes on an EFS system is to **ftp** the Linux file containing the backup to another system. (The backup file can be ADRDSSU to FakeTape, or a **ckdbbackup** file—the principle is the same.) Using a 100 Mbps LAN connection, especially on a private or local LAN, performance can be quite good. These files are large, by any standard, and ftp’ing compressed files probably makes more sense. We created the following shell script:

```
Shell script /usr/flexes/rundir/buZ4RES1c

VOL=Z4RES1
ckdbbackup /s390/$VOL | gzip -c > /tmp/$VOL.gz
ftp -niv <<EOF
  open nf                                (nf is another Linux machine)
  user ogden passwd
  bin
  put /tmp/$VOL.gz $VOL.gz
EOF
rm /tmp/$VOL.gz
```

This script automates backing up volume Z4RES1 on our t23 system. The script uses **ckdbbackup** piped to **gzip** to create a file in the /tmp directory. It then FTPs this file to our nf system (another Linux machine). We have set up a standard user ID on the nf machine (user *ogden*) and the FTPed file will be placed in /home/ogden/Z4RES1.gz. (That is, our script places the FTPed file in the home directory of the target user. This is the easiest method to work around any access restrictions on the target system.) The /tmp file (on the first machine) is then deleted.

We then executed this function:

```
$ cd /usr/flexes/rundir
$ sh buZ4RES1c
FSIDB133 Max head = 14, cyl = 3343, blks = 57
FSIDB150 Cylinder 3342 completed in 46 milliseconds
FSIDB190 CKD Backup Completed
Connected to nf (192.168.0.120)
```

```
220 nf FTP server (Version wu-2.6.2-8) ready.
331 Password required for ogden
230 User flexes logged in. Access restrictions apply.
200 Type set to I.
local: /tmp/Z4RES1.gz remote: Z4RES1.gz
226 Transfer complete
384962480 bytes sent in 40.6 seconds (9.3e+03 Kbytes/s)
221-You have transferred 384962480 bytes in 1 files.
221-Total traffic for this session was 384962908 bytes in 1 transfers.
221-Thank you for using the FTP server on nf
221-Goodbye.
```

The **gz** function took about 6 minutes and the **ftp** function took less than one minute in this example. We used a private 100 Mbps LAN connection, with no other active systems on it. The target system (our xSeries EFS machine) was otherwise idle during this time. It would have been faster to **ftp** the volume than to compress it first, but we wanted the compression offered by **gzip**.

We could create a script file for each S/390 volume.⁴⁹ Once created these are very convenient to use and automatically take backups and place them on another system for safety. The compressed backups (for 3390-3 volumes) tend to be in the range of 350 MB to about 600 MB, depending on the contents of the volume.

We could, of course, take ADRDSSU backups (to FakeTape files) and then **gzip** and **ftp** these files. This might be the best solution, although it would require the most processing and disk space.

There are clearly several ways to approach backup and restore for an EFS system. We cannot recommend any as the *right* way, and you will probably use a combination of techniques. We strongly suggest that you take time to understand the options available, try whatever is appropriate for you, and establish a routine for taking backups.

⁴⁹ If you have a little experience writing shell scripts, you could change the script to pass the volser to be processed (assuming your naming conventions relate the volser to the Linux file).

Printers and readers

Printed output is usually a requirement for any serious S/390 operation. FLEX-ES emulation and typical PC printers can provide an expensive method for obtaining typical S/390 printed output. There are very few card readers still in use anywhere, but an emulated card reader can provide an interesting link between the base Linux system underlying FLEX-ES and an emulated S/390.

6.1 JES2 customization

The printer and card reader techniques discussed in this chapter assume you have a printer and card reader defined in your z/OS and in JES2. Depending on how your z/OS system was customized, these elements may or may not exist. For this discussion, we assume you are using a recent release of the AD CD-ROM system. Recent releases have a 2540 card reader defined at address 00C and 1403-N1 printers defined at addresses 00E and 00F.⁵⁰

These AD systems do not have JES2 definitions for the reader and printers. We added definitions for the reader and one printer by editing member JES2PARM in PARMLIB. (In the release we used, the effective PARMLIB for this member was ADCD.ZOSV14S.PARMLIB.) You can edit the JES2 definitions while JES2 is running; changes will be effective when you restart JES2 or re-IPL.⁵¹

JES2 is forgiving about errors in its parameter definitions, but you should take care when adding lines to the definitions. The beginning and ending of comment blocks in the JES2PARM used with the AD system is not always immediately apparent. The most common error is to place your changes such that you remove an initial or end comment delimiter.

We located a printer definition in JES2PARM that was *commented out* and added our definitions after this point:

```
000624 /*PRT(1)      WS=(W,R,Q,PMD,LIM/F,T,C,P),          */
000625 /*              UNIT=000F,CLASS=A,FCB=EX01,DRAIN      */
..... PRT(1)  WS=(W,R,Q,PMD,LIM/F,T,C,P),UNIT=00E,CLASS=C
..... RDR(1)  UNIT=00C,START=YES
```

⁵⁰ By *defined*, we mean it is present in the IODF.

⁵¹ A JES2 warm start will accept the changes described here.

We arbitrarily decided to use SYSOUT class C for this printer. Class C is not used for anything else in the AD system.

You can add JES2 definitions dynamically, but we prefer to modify JES2PARM to make a permanent change.

6.2 FLEX-ES printing options

There are many ways to handle printing with FLEX-ES. As an overview, we might list these categories of printers:

- ▶ Channel-attached printers. These could be line printers or AFP™ printers, attached on parallel channels. Users already having these devices are likely to connect them to channel adapters and continue using this method of printing.
- ▶ Network-attached printers connected (via TCP/IP) to z/OS. Some of these printers can handle IPDS™ (AFP) data. Recent additions to z/OS printing software allow network-attached printers to be used for almost all z/OS printing requirements.
- ▶ Locally-attached Server printers. A PC laser printer is the most common example. These are inexpensive, capable of high-quality output, and the more robust models can handle substantial amounts of printing.
- ▶ Network-attached printers connected to the Server. With the proper setup, these could be the same printers that are network-connected with z/OS.

Not included in this list are typical ink-jet printers, as found with many PCs, although they might be used in some circumstances.

Printing can be a complex topic, and we make no attempt to address the whole topic here. Instead, we selected one method and style we found useful and we describe our implementation. A little background information is needed to explain why we devised this particular implementation.

6.3 Background

Basic z/OS printing is closely related to the hardware available on the original S/360 machines. The most common printer available at that time was the IBM 1403 printer. It printed lines that were a maximum of 120 characters long (or 132 characters long with an additional feature) and was normally set up to print 6 lines/inch on fan-fold paper that was 11 inches long. This meant a full page held 66 lines. In practice, many programs counted their output lines and *skipped to a new page* after 60 or 61 lines.

Much of the utility software with the system, such as JCL processors, assemblers, compilers, system report generators, and so forth were designed to fit these pages. That is, they printed lines of up to 120 characters (or sometimes 132 characters) with about 60 lines per page. This convention is still with us today. Current z/OS compilers and utilities default to these characteristics.

Later hardware replaced the line printers (such as the 1403) with laser printers (often known as AFP printers). These were devices such as the IBM 3800, 3820, 3825, 3900, and so forth. With proper programming, these printers can produce sophisticated output using many fonts and graphic components. However, the system utilities (compilers, for example) continued to produce listings in “1403 format.” Software for the AFP printers can accept this “1403 format”

data and list it. These listings are typically two-sided, landscape mode, and contain up to 66 lines of 132 characters on each page.

Although “real” 1403 printers are historical items, there are a variety of uses for pseudo-1403 devices. z/OS and JES2 still support 1403 printers. FLEX-ES can emulate 1403 printers. This emulation makes several assumptions. It assumes that the 1403 is always in 6 lines/inch mode and always has the same carriage control tape (with a channel 1 punch in line 3). By default, the FLEX-ES 1403 emulator writes output to Linux that *always* has 66 lines per page. That is, the device emulation program converts page ejects to the appropriate number of blank lines.⁵² The output is also converted to ASCII using a fixed translation table that appears to correspond to a TN print chain, and an NL character is added to every line.⁵³

The Linux output produced by the 1403 emulation can be printed directly on a typical PC printer. However, the output is not very attractive. PC printers usually default to a 10-pitch fixed-font character set. Output that is 120 (or 132) characters wide is too long for a printer line and is discarded or wrapped to the next line.

6.4 Using a PC printer

Our goal was to use a common PC laser printer and have utility output appear in about the same format that it would appear on an AFP printer. That is, in landscape mode with up to 132 characters per line, and up to 66 lines per page. If the printer provides duplex printing (both sides of the page), then this function would be used.

We elected to use a very simple-minded solution. You could use the same, or enhance it in a number of ways. Our solution is outlined in Figure 6-1.

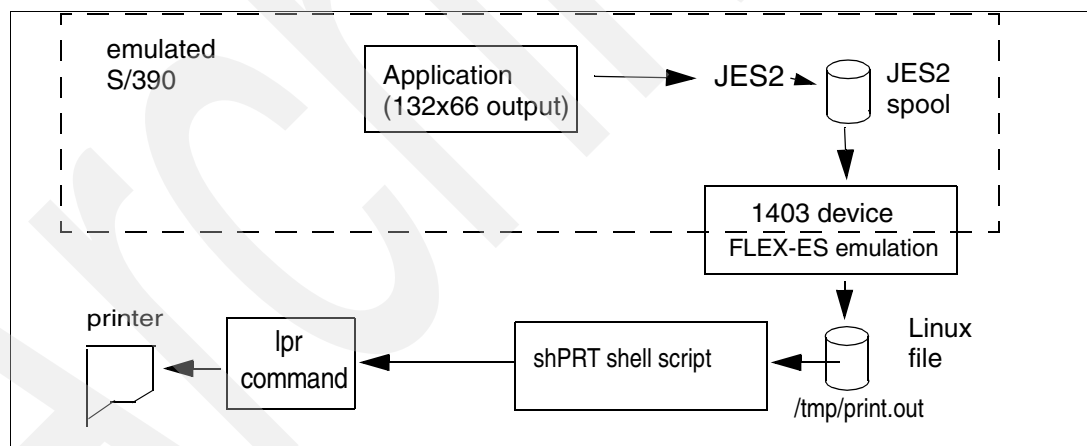


Figure 6-1 Design for local printing

6.4.1 Print flow

In this figure, a S/390 application generates SYSOUT in normal line mode. The application might be the COBOL compiler. JES2 spools the data and eventually prints it. In our case, we defined PRT1 (a JES2 printer) as a 1403 at address 00E. The FLEX-ES 1403 emulation program accepts the data, replaces page skips with the appropriate number of blank lines

⁵² You can use the *useformfeeds* option in your FLEX-ES definitions to change this. With this option FLEX-ES will generate a form feed character (instead of blank lines) to advance to the next page.

⁵³ The devopt operand “onlcr” can be used in the FLEX-ES resource definition for the printer to obtain CR+LF characters instead of the NL character at the end of each line.

and converts the data to ASCII. In our FLEX-ES resources definition, we specified a fixed Linux file name, */tmp/print.out*, as the destination for the 1403 at address 00E.

The 1403 emulation program accepts whatever data is written to the 1403 and adds it to the Linux output file. There may be many z/OS jobs in the file, with the separator pages generated by JES2. The emulation program cannot detect z/OS job boundaries, although some crude filtering is possible based on the content of JES2 separator pages. This output file will grow indefinitely until you clear it.

We created the **shPRT** shell script. This shell script sends printer control characters and the print data to the **lpr** command. The **lpr** command, in turn, sends the data to a Linux print queue where it is handled by the print daemon. The result is output that looks very much like line-mode output from a 38xx printer.

6.4.2 Implementation

There are a number of steps required to implement this process. These include:

- ▶ Connect an appropriate printer to the server and Linux
- ▶ Define a Linux raw print queue
- ▶ Create the shPRT shell script
- ▶ Define a 1403 printer in the FLEX-ES resources
- ▶ Define a 1403 printer for JES2, if not already done
- ▶ Understand the operational steps needed to use this process

Printer and print queue

We simply connected our Lexmark Optra L printer (with the duplex feature) to the parallel port on our EFS system and rebooted the system. If the printer had not been connected before, Linux offers to configure it for you. After doing this, we then used a Red Hat tool (under **gnome** in RH 8.0 it is found with **System Settings-->Printing**) to configure a print queue. In this tool:

- ▶ Select **New** (followed by **forward** to move through the panels).
- ▶ Select a *local* printer and define the queue name as *rawprint* (this is an arbitrary name).
- ▶ The device is */dev/lp0* (and our Linux automatically detected the Lexmark printer).
- ▶ Select the print driver as *Raw Print Queue*. (Do not select any of the specific printer names.)
- ▶ Select the *Apply* option. This should restart the **lpd** daemon.
- ▶ Exit from the printer configuration tool.

Shell script

We created our shPRT shell script in the */usr/flexes/rundir* directory where we keep all our FLEX-ES operational files. The shell script contains:

```
CTL=""033\105\033\050\163\060\160\061\066\056\066\067\150\070\056\166
    \060\163\060\142\124\033\046\154\061\157\061\163\065\056\064\143
    \055\061\060\060\060\132"
CTL2=""014\033\105"
(/bin/echo -ne $CTL; cat /tmp/print.out; /bin/echo -ne $CTL2) | lpr -P rawprint
echo 'Sent to print queue - Remember to "load 00E" '
```

The CTL and CTL2 constants are printer control characters, written in octal.⁵⁴ The octal format was the most convenient for use in a shell script. If you have good script writing skills you can do this many different ways. The particular control characters shown here are for the

⁵⁴ CTL is shown as three lines here, but it is actually created as one long line containing 38 octal constants.

The z/OS system we use (an AD CD-ROM version) already has a 1403 printer defined at address 00E, so no action was needed for this step. (At this level, a 1403 definition consists of an appropriate definition in the IODF.)

JES2 definitions

We added a printer definition to JES2, as described in “JES2 customization” on page 77, and re-IPLed z/OS. (There are various ways to do the same thing without the re-IPL.) We verified that the printer was online (**d u,,,00E,1**) and then issued a JES2 command to start it (**\$SPRT1**). Use a **\$SPRT1** command as the response to requests to mount forms and so forth.

Operational technique

We then ran jobs that sent output to SYSOUT=C. For example,

```
//OGDEN1 JOB 1,OGDEN,MSGCLASS=C
// EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=*
//SYSIN DD DUMMY
//SYSUT1 DD DSN=ADCD.ZOSV14S.PARMLIB(IEASYS00),DISP=SHR
//SYSUT2 DD SYSOUT=*
```

After submitting this job, you should notice z/OS console messages about jobs sent to PRT1. FLEX-ES emulation immediately sends these jobs to the emulated printer. As described here, this is file /tmp/print.out in Linux. Additional output (from multiple jobs) from z/OS is simply added to the file.

At some point you can stop the JES2 printer (**\$PPRT1**) and prepare to print the accumulated output under Linux. (You do not need to stop the JES2 printer. When you take the emulated 1403 printer offline with a FLEX-ES command, the printer will appear as NOT READY to JES2.) You should *unload* the Linux print file before taking the next steps. This disconnects the output file (/tmp/print.out) from the device.⁵⁵

```
flexes> unload 00E
```

You can then run the shell script, working from any Linux terminal window.

```
$ cd /usr/flexes/rundir          (Directory containing the shell script)
$ sh shPRT                      (Execute the shell script)
```

The printer should begin printing output. Notice that the output includes all the job separator pages produced by JES2. When it finishes, you should use the FLEX-ES **load** command to reconnect the output file to the device.

```
$ load 00E
```

This use of the **load** command truncates (deletes all the contents) of /tmp/print.out. You should do this even if you did not **unload** the printer, otherwise you will print the same data again (plus any new data) the next time you run **shPRT**. If you stopped the JES2 printer, you need to start it again (**\$SPRT1**).

Comments

As mentioned earlier, this is a very basic implementation of a printing process that produces acceptable output on a local PC laser printer. The simple shell script makes no effort to begin a new z/OS job on a new page (instead of the “back” of a page of the previous job). If JES2 is configured to produce at least two separator pages at the beginning and ending of a job, this should not be a problem.

⁵⁵ Actually, you do not need to do any of this if you are certain that no additional output from JES2 will be sent to the 1403 while you are performing the print operations in Linux. In a small EFS system, this may be the best method.

6.5 Emulated card reader

You can create z/OS jobs while working in Linux, using a Linux editor and working in ASCII. You can send the jobs (in the form of a JCL *deck*) to z/OS (or another S/390 operating system) through an emulated card reader. There is not much need for this with z/OS, but an equivalent operation for VSE/ESA might be more useful.

The easiest way to do this is to use the **hotdrdr.sh** shell script provided with FLEX-ES. It automates the more detailed steps needed to do the function, including ASCII to EBCDIC conversion and a number of FLEX-ES CLI commands.

There are several steps involved in setting this up:

- ▶ The operating system must have a card reader defined. (The IODF for the AD system has a 2540R device defined at address 00C.)
- ▶ The reader must be defined in JES2. This is described in “JES2 customization” on page 77. The definitions in this example create a hot reader, one that starts automatically when cards are placed in it.
- ▶ Copy **hotdrdr.sh** to your normal FLEX-ES working directory.
- ▶ Create a holding directory for use by the **hotdrdr.sh** script. Do not place any permanent files in this directory.
- ▶ The proper FLEX-ES definitions must be created.
- ▶ Start **hotdrdr.sh** as part of a normal FLEX-ES instance.
- ▶ You can create your jobs (JCL) in any convenient Linux location, using the ASCII editor of your choice.
- ▶ Copy your JCL jobs to the holding directory to submit the job to z/OS. The file is automatically deleted from the holding directory after the job is submitted.

Initial setup

We used these commands to create a holding directory and to copy the shell script to our rundir:

```
$ cp /usr/flexes/examples/hotdrdr.sh /usr/flexes/rundir
$ mkdir /usr/flexes/rundir/reader
```

Do not use this new directory for your files. The **hotdrdr.sh** script uses it as a holding area for jobs (files) being sent to the S/390 card reader. These files are automatically deleted after they are sent.

There is no requirement for the holding directory to be in rundir. It can be placed anywhere in Linux (provided user ID *flexes* has read/write permissions there). We placed it in rundir simply to keep all our locally-defined FLEX-ES materials together. We used the name *reader* to remind us of its purpose; you can use any name.

FLEX-ES definitions

We added the reader definition to the control unit we earlier used for the emulated printer. The addresses for this control unit start at 00C (for the reader). There is no device defined for z/OS at address 00D, so we created a dummy punch as a placeholder in the resource definition. The printer is then defined to occupy address 00E.

```
system S14A:
...
channel(1) local
cu devad(0x00C,3) path(1) resource(CU2821)
```

```

...
end S14A

resources R14A:
...
CU2821: cu 2821
interface local(1)
device(00) 2540R OFFLINE          # Address 00C
device(01) 2540P OFFLINE          # Address 00D
device(02) 1403 /tmp/print.out devopt "onlcr" # Address 00E
end CU2821
...
end R14A

```

We recompiled the FLEX-ES definition with **cfcomp** after making these changes.

Shell script

We added a line to our standard shell script:

```

flexes S14A.syscf
/usr/flexes/rundir/hotdr.sh S14A 00C /usr/flexes/rundir/reader &
x3270 -model 3 -keymap pc -port tn3270 localhost:mstcon &
x3270 -model 3 -keymap pc -port tn3270 localhost:L701 &
flexesccli localhost S14A

```

This line calls **hotdr.sh** with three arguments: the instance name (S14A), the address of the card reader (00C), and the name of the directory used to submit jobs (/usr/flexes/rundir/reader). The **hotdr.sh** script runs for the duration of the S/390 instance. It checks the holding directory every five seconds and submits any file it finds there.

Operation

We created several z/OS jobs in files in /tmp. (You can place the jobs anywhere in Linux; we did not want to clutter our rundir with these jobs, so we placed them in /tmp. If you want to keep the jobs, you should create a permanent directory for them.) The jobs are normal JCL, following all the JCL conventions. You can use any Linux editor; we used **gedit** instead of **vi**.

To submit a job, simply copy it to the holding directory:

```
$ cp /tmp/myJOB1 /usr/flexes/rundir/reader
```

After a few seconds you should see the job start in z/OS. Any user who has *write* permission to the holding directory can submit jobs.

FLEX-ES definitions/operation

This chapter is intended as a brief tutorial for FLEX-ES definitions and commands. You must refer to the full FLEX-ES documentation for complete details.

This chapter reviews a selected subset of the FLEX-ES parameters needed to define emulated S/390 systems and their resources. The formal FLEX-ES documentation should be reviewed for additional information, more emulated devices, and many other parameters. The last section also discusses basic CLI commands that can be used at the *flexes* prompt.

The illustrations in this section are only for discussion and are not intended to represent a working system.

FLEX-ES definitions must be *exactly* correct or they will not work. Take special care with colons, parentheses, and hex numbers.

7.1 System definitions

System definitions are usually something like this:

```

system S14A:
memsize(262144)           # K Central Storage = 256 MB
cachesize(4096)           #
essize(64)                # M Expanded Storage = 64 MB
instset(esa)              # Use "z" for ALS3 (64-bit) mode
tracesize(512)
feature lpar               # Normally only for VM
cpu(0) dedicated          # Single Pentium, dedicated
channel(0) local
channel(1) local
channel(2) local
channel(3) local
cu devad(0x00C,3) path(0) resource(R14A2821) # 2450R, 2540P, 1403
cu devad(0x560,1) path(0) resource(R14A3480) # tape drive (4mm)
cu devad(0x700,16) path(1) resource(R14A3174) # 3270 terminals
cu devad(0xA80,12) path(2) resource(R14A3990) # 3390 disks
cu devad(0xE20,2) path(3) resource(R14A3088) # CTCs for TCP/IP
end S14A

```

The first line (system S14A) provides a name for the system being defined. The FLEX-ES compiler will create a file named, for this example, *S14A.syscf*.

memsize always expresses kilobytes of storage. The largest accepted value is 2097152, which would define a 2 GB system.⁵⁶ The size specified will be the size of central storage in the emulated S/390. There is no direct relation to the memory size on the server, although we strongly recommend that memsize be such that no paging is required by the underlying Linux system. Although FLEX-ES can emulate an ALS3 (64-bit) system, you cannot emulate a system with more than 2 GB real memory. For zArchitecture⁵⁷ the storage size must be an even multiple of 1 MB; this means the figure for memsize must be an even multiple of 1024.

The parameter **essize** specifies expanded storage size and expresses megabytes of storage. Furthermore, the number specified must be an even multiple of 16. If essize is not specified, then no expanded storage is emulated. A specification of 64 means 64 MB of expanded storage. Remember that expanded storage is not used by z/OS in ALS3 (64-bit) mode.

The **instset** parameter should always be specified as *esa* or *z* for the systems we are discussing. An *esa* system emulates a 31-bit S/390. A *z* system emulates a zSeries machine that can switch into 64-bit mode. This is the only FLEX-ES definition parameter involved in moving to 64-bit emulation.

The **cachesize** parameter specifies the amount of storage to be used by FLEX-ES to cache dynamically compiled S/390 instructions. This parameter expresses kilobytes of S/390 storage. (The server storage required averages 11 times the specified S/390 storage value.) Practical values are in the range 1024 to 8192. We suggest you do not use values outside the recommended range unless you are clearly in an experimental mood. A specification of 2048 means 22 MB of server storage will be used by FLEX-ES for a S/390 instruction cache.

The **tracesize** parameter specifies the number of entries in a FLEX-ES trace table. We suggest 512 as a reasonable number. The FLEX-ES instruction trace function runs only when started. There is no processor overhead caused by defining a tracesize parameter. This is a somewhat imprecise parameter because the size of each entry in the trace table is not well defined. In any event, the total memory used (with the recommended value) is fairly trivial.

There is also a processor *event trace* that uses a permanently defined buffer. This event trace is always active and is not associated with the **tracesize** parameter.

The **feature lpar** statement causes the LPAR bit to be set in the *scpinfo* response. If VM detects this bit, it will not use *active wait*. (*Active wait* means that VM loops waiting for work.) This feature is most meaningful if you are using VM and not using a dedicated processor. This is the only purpose of the **lpar** statement; no other PR/SM™ functions are emulated.

The **cpu** parameters indicate the number of emulated S/390 CPUs and assign identification numbers to each one. Multiple **cpu** parameters are used to specify multiple emulated S/390 CPUs within a single emulated S/390. For example:

```
cpu(0)
cpu(1)
cpu(2)
```

as part of a system definition would specify an emulated S/390 with three CPUs. The CPU numbers (which are defined S/390 architecture elements) are 0, 1, and 2. *The number of CPUs specified may not exceed the number of server processors enabled for S/390 emulation.* For a ThinkPad EFS system, this will be one. (The single CPU may be given any number in

⁵⁶ In practice, you cannot define a system this large at this time. See “Maximum allocations” on page 25 for more information.

⁵⁷ Also known as ALS3 or z mode.

the range 0 - 7, but there is no reason to use anything other than 0.) For a dual-processor xServer, for example, you might have one or two CPUs defined, depending on how many processors are specified in your FLEX-ES license.

The **dedicated** keyword on a CPU definition statement means that the underlying processor will be used *only* for S/390 emulation; it will not be used for Linux work. This may provide a performance boost. This option cannot be used with a single-processor system because only one processor is available for both S/390 emulation and other Linux work; this option is relevant only to SMP systems. All processors enabled for S/390 emulation should be dedicated or none should be dedicated; that is, having some processors used for S/390 emulation as *dedicated* and some as not dedicated does not work well.

For example, if you have a two-processor system and you have a FLEX-ES license for a single processor, you would probably use the *dedicated* option for that processor. The “other” processor in your system would then do all the I/O and background Linux work. If you have a two-processor FLEX-ES license for the same system, you would not use the *dedicated* option. You cannot dedicate both processors because there would be no processor left for I/O and other Linux work. You would not dedicate one processor because this would create a mixed situation (two processors doing S/390 emulation and only one *dedicated*) that is not recommended.

If you run multiple S/390 concurrent instances, you would probably not use the *dedicated* option. It would dedicate a processor (or processors) solely to the S/390 instance that contained the *dedicated* option.

The **channel** parameters specify emulated channels and assign channel numbers. The following example defines two channels with the channel numbers 3 and 1:

```
channel(3) local
channel(1) local
```

The channel numbers must be unique, but need not be in sequence. The **local** keyword means the channels are emulated block multiplexor channels and are emulated using resources on your server.⁵⁸ The channel numbers (or *path numbers*) are somewhat arbitrary and do not necessarily mirror the channels you might have on a “real” S/390. The channel numbers you assign are *not* reflected in the device addresses (“device numbers”) used by z/OS.

The **cu** statement defines an emulated control unit. The **devad** parameter defines the beginning address (device number) and the number of (emulated) devices connected to the control unit. This address is the address seen by the operating system. For example, if you want to “IPL from A82,” then address A82 would appear in a devad parameter.⁵⁹ The **path** parameter specifies an emulated channel connected to the control unit. The **resource** parameter specifies the name of a section in a resource definition (described below) that defines the emulated *devices* connected to the control unit. Resource names, used in **cu** statements, must be unique.

The **cu** definition specifies a number of units for each cu. You *must* define this number of devices in the matching resource definition. For example:

```
cu devad(0xA80,12) path(2) resource(R14A3990)
```

⁵⁸ Other types of channels may be specified, such as byte multiplexor and channels using remote (via TCP/IP) resources. These are described in detail in the FSI documentation.

⁵⁹ For this example, we might have **devad(0xA80,8)** as the parameter. This defines 8 addresses, A80, A81, A82, and so forth. The A82 name is implied in this sequence.

specifies that 12 devices are attached to this control unit. The R14A3990 section in the resource definitions used with this system definition *must* contain 12 devices. Their addresses will be A80, A81, A82, and so forth.

An **end** statement is required for the system definition, and the parameter should specify the same name as the *system* statement.

7.2 Resource definitions

Resource definitions refer to actual devices or files and are defined separately from system definitions. When a resource definition file is compiled, it produces a rescf file. In practice, both a system definition and the corresponding resource definition might be in the same text file. Both must be compiled by the FLEX-ES **cfcomp** program to produce syscf and rescf files before they can be used.

7.2.1 Emulated control unit types

FLEX-ES emulates many control unit types (cutype parameters) and device types; the full documentation should be consulted for a complete list. The commonly used cutypes include:

3990	DASD control unit (for all 3380 and 3390 devices)
3480 and 3490	Tape control unit (for FakeTape and SCSI tape drives)
3174	Terminal control unit (for emulated locally-attached 3270s)
3172	LAN control unit (for Ethernet LANs)
3172TR	LAN control unit (for token ring LANs)
3215	Typewriter system console (mostly for VM and Linux for S/390)
2821	Unit record control unit (for emulated card reader/punch, printer)
ctc	Channel-to-channel control unit

7.2.2 Emulated device types

FLEX-ES emulates many device types, and these are documented in the full FLEX-ES materials. The following list includes commonly used devices for an EFS system running current S/390 operating systems:

3390-1	3390 model 1, with 1117 cylinders (1113 usable cylinders)
3390-2	3390 model 2, with 2230 cylinders (2226 usable cylinders)
3390-3	3390 model 3, with 3343 cylinders (3339 usable cylinders)
3390-9	3390 model 9, with 10038 cylinders (10017 usable cylinders)
3172	LCS device, for Ethernet connections
3172TR	LCS device, for token ring connections
3215	Typewriter console (not for OS/390 or z/OS)
3178	Most typical 3270 definition
3420	“Round” tape, emulated with SCSI device or FakeTape
3480	Emulated with SCSI device or FakeTape
3490	Emulated with SCSI device or FakeTape
3490-E	Emulated with SCSI device or FakeTape
1403	Printer
2540R	Card reader, emulated with UNIX files
CTC	Channel-to-channel adapter, emulated with UNIX sockets

7.2.3 Typical resource definitions

A simple set of resource definitions, corresponding to the system definitions above, might be:

```
resources R14A:
```

```

R14A2821: cu 2821
    interface local(1)
    device(00) 2540R OFFLINE
    device(01) 2540P OFFLINE
    device(02) 1403 OFFLINE
end R14A2821

R14A3480: cu 3480
    interface local(1)
    device(00) 3480 OFFLINE
end R14A3480

R14A3174: cu 3174
    interface local(1)
    device(00) 3278 mstcon
    device(01) 3278 payroll1
    device(02) 3278 develop2
    device(03) 3278 OFFLINE
    .....
    device(15) 3278 sysprogs
end R14A3174

R14A3990: cu 3990
    interface local(1)
    device(00) 3390-1 /s390/.....
    device(01) 3390-3 /s390/.....
    device(02) 3390-3 OFFLINE
    ....
    device(10) 3390-3 /s390/..... devopt 'writethroughcache,trackcachesize=50'
    device(11) 3390-1 /s390/.....
end R14A3990

R14A3088: cu 3172
    interface local(1)
    options 'ipaddress=9.12.17.211'
    device(00) 3172 eth0
    device(01) 3172 OFFLINE
end R14A3088

end R14A

```

A **resources** block must have a name; R14A is used in this example. Each set of resources in the block must also have a name. Each set name should match a name in a cu statement in the system definition. The general syntax for defining a resource set is:

```

resourcename: cu cutype
    parameters
end resourcename

```

You assign a resource name. It must follow the FLEX-ES rules for names. The name is followed by a colon and the keyword **cu**. The cutype must be one of the cutypes known to FLEX-ES. This is followed by whatever parameters are needed to specify the particular device(s) being defined. The keyword **end** is then followed by the resource name. There is no need to use the indentation pattern shown here; it is merely for improved readability.

A **memory** definition is needed only if your system contains an FSI parallel channel adapter and you are running under UnixWare. *Do not use this statement with a Linux-based system.*

In the following paragraphs, we show examples of commonly used device resource definitions and describe parameters used for each device type.

CKD disk resources

Here we discuss CKD disk resources.

```
R14A3990: cu 3990
interface local(1)
device(00) 3390-1 /s390/.....
device(01) 3390-3 /s390/.....
device(02) 3390-3 OFFLINE
.....
device(10) 3390-3 /s390/..... devopt 'writethroughcache,trackcachesize=50'
device(11) 3390-1 /s390/.....
end R14A3990
```

The **interface local(1)** parameter means that the defined devices are local (on the same server running FLEX-ES) and that there is one path (channel + control unit) pointing to these devices. Multiple paths are typically found when running multiple S/390 instances (that is, emulating multiple S/390 systems at the same time) and sharing control units among the multiple instances. Shared DASD is the most common example of this. FLEX-ES does not support multiple paths from one S/390 instance to a control unit.

The **device(00)** parameter describes the first device of this resource set. The index number is *decimal*; it must begin with 00 and be incremented by one for each additional device. If your device addressing scheme (at the z/OS level) requires addressing gaps, you must define dummy (**OFFLINE**) devices in the resource set. The addresses seen at the z/OS level are set by the *cu* statement (in the system definitions section) that points to this resource set.

The device type (**3390-1** and **3390-3** in this example) must be from the list of known FLEX-ES emulated device types. (These are listed in “Emulated device types” on page 88.) The next parameter, **/s390/xxxxx**, is the Linux file that contains the emulated 3390 volume if you are using simple Linux files for emulated volumes. If you are using raw device interfaces (for better performance), the syntax will look like **/dev/raw/xxx**.

The keyword **devopt** indicates that optional parameters follow. The **writethroughcache** parameter causes the FLEX-ES 3390 emulation program to consider a S/390 write operation complete when the data is actually written to disk. This is described in “Disk caches” on page 20.

The **trackcachesize=** parameter tells the emulation program to allocate the indicated number of track buffers (3390 tracks, in this example) for a cache. The default cache size is one emulated cylinder (15 tracks for a 3390) for each emulated S/390 disk volume. This can be an important tuning parameter if you use raw device interfaces. If you use simple Linux files for emulated volumes, the default trackcachesize is probably sufficient.

3270 terminal resources

Emulated 3270 terminals, which are actually TN3270 sessions (or x3270 sessions) to a FLEX-ES server running under Linux and appear to the S/390 as local, channel-attached, non-SNA terminals, are defined as follows:

```
R14A3174: cu 3174
interface local(1)
device(00) 3278 mstcon
device(01) 3278 term701
device(02) 3278 term702
device(03) 3278 OFFLINE
device(04) 3278 @9.12.17.210
.....
device(15) 3278 term70F
end R14A3174
```

The interface, local, device, and device index parameters have already been described. The **3174** cutype is a CU type emulated by FLEX-ES and device type **3278** is a device type emulated by FLEX-ES. Each emulated 3270 is connected through the FLEX-ES TN3270 server. If the 3278 keyword is followed by **@IP-address** (such as @9.12.17.210 in the example), the FLEX-ES TN3270 server waits for a TN3270 connection from this IP address. If the 3278 keyword is followed by the OFFLINE keyword, then a FLEX-ES **mount** command must be issued to assign either an IP address for the connection or a name for the Terminal Solicitor. If the 3278 keyword is followed by a name (such as *mstcon* or *term70F*), this name is added to the list of terminals available through the FLEX-ES **Terminal Solicitor**.

Typical mount commands, from a *flexes* prompt, might be:

```
flexes> mount 703 bills
flexes> mount 704 @9.12.17.211
```

These commands would add a new terminal name (*bills*) to the Terminal Solicitor screen and also enable a TN3270 connection from 9.12.17.211. Where do the addresses (703 and 704) come from? The *cu* statement in our example (“System definitions” on page 85) specified an addressing range with 16 addresses. The resource name in the *cu* statement matches the resource set name of the 3270 definitions, so these resources have addresses beginning with 700.

The same mount commands could be placed in a shell script used to start an emulated S/390. The shell statements would be:

```
$ echo 'mount 703 bills' | flexescli localhost S14A
$ echo 'mount 704 @9.12.17.211' | flexescli localhost S14A
```

Tape resources

The tape resources discussed here involve emulated S/390 tapes. If you channel-attach “real” S/390 tapes on an EFS machine, they would be defined differently, as channel devices. An emulated tape might be a SCSI-attached device, such as a 4mm drive, a DLT drive, or a SCSI 3480/3490 unit. A SCSI tape drive might be defined as:

```
R14A3480: cu 3480
         interface local(1)
         device(00) 3480 /dev/sg0
end R14A3480
```

You can define FakeTape devices; the definition might be:

```
R14A3480: cu 3480
         interface local(1)
         device(00) 3480 /tmp/tapes/222222          (Linux file to hold the tape volume)
end R14A3480
```

The most likely definition for FakeTape usage would be:

```
R14A3480: cu 3480
         interface local(1)
         device(00) 3480 OFFLINE
end R14A3480
```

The keyword 3480 specifies a standard FLEX-ES emulated device. The following parameter, */tmp/tapes/222222*, is the Linux file name to be used for the emulated tape volume. If the definition is OFFLINE, then a CLI **mount** command can be used to temporarily assign a Linux file to the drive. FLEX-ES supports 3420, 3480, 3490, and 3490-E emulation. You should use the appropriate type to match whatever is defined in z/OS.

Several specialized parameters can be used with tape resource definitions. These would be written like this:

```
device(00) 3480 /tmp/tapes/222222 devopt 'maxwritesize=200'
```

The options are placed (in single quotes) after a *devopt* keyword. The **maxwritesize** option is probably the most commonly used. It specifies, in megabytes, the maximum size of the emulated tape media. When the amount of data written approaches this size, an end-of-tape reflective strip is emulated and the operating system would normally write EOV labels. This is especially important with FakeTape, to prevent a runaway program from filling the Linux file system with tape output.

LAN resources

LAN resources are used to define z/OS TCP/IP or SNA LAN connections to the S/390. A definition for z/OS TCP/IP use might be:

```
R14A3088: cu 3172
  interface local(1)
  device(00) 3172 eth0
  device(01) 3172 OFFLINE
end R14A3088
```

The *cu* type 3172 is a defined FLEX-ES keyword and defines an LCS-type LAN connection to S/390. (On the S/390 side, there would normally be two CTC addresses (even/odd addresses) defined.) A LAN device for use by z/OS TCP/IP must be specified exactly as shown (except that the device name of the interface might be different). That is, it will be defined as two devices (00 and 01), with the second device OFFLINE. The IP address for this interface would be specified in the z/OS TCP/IP profile.

The device designation in this example, *eth0*, may appear strange to experienced UNIX users, who will want to write something like */dev/eth0*. However, *eth0* is the proper format.

An additional parameter may be required for a LAN definition. A “real” 3172 unit may have up to four adapters. These are numbered 0-3, and this number appears in the z/OS TCP/IP or VTAM specifications.⁶⁰ By default, FLEX-ES uses number zero. If you want a different number, you must include an **options** statement:

```
R14A3088: cu 3172
  options 'adapternumber=1'
  interface local(1)
  device(00) 3172 eth0
  device(01) 3172 OFFLINE
end R14A3088
```

This emulates the “real” 3172 adapter number 1. This number is used in the z/OS TCP/IP PROFILE definition. This number is not related to the actual LAN adapter number in your Linux. The actual PC LAN adapter that is used is specified by the device parameter, such as *eth0*, *eth1*, and so forth.

You can share a LAN adapter between Linux and z/OS TCP/IPs. In order to do this, you should normally specify the IP address to be used by z/OS, as a parameter in the resource definition.⁶¹ (You should specify the same address in your z/OS TCP/IP parameters.) For example:

```
R14A3088: cu 3172
  options 'ipaddress=9.12.17.211'
```

⁶⁰ The same situation exists with a P/390 or MP3000 emulated I/O LAN adapter. It is known as the *MPTS number* in these cases.

⁶¹ An exception exists when running multiple Linux guests under z/VM.

```

interface local(1)
  device(00) 3172 eth0
  device(01) 3172 OFFLINE
end R14A3088

```

Using this example, the single Ethernet adapter in our ThinkPad EFS machine appears as IP address 9.12.17.210 (if that was defined when we installed Linux) and as IP address 9.12.17.211 (when we start FLEX-ES). The single adapter responds to two different IP addresses.

7.2.4 Cloned devices

The FLEX-ES documentation includes an optional **cloned** parameter for many resource definitions. We did not use this parameter. This parameter applies to a limited set of devices—emulated printers and card readers are good examples—for which the resource definitions can be automatically replicated to serve multiple S/390 instances. This is meaningful only when multiple S/390 instances are used. It potentially saves a little time when creating resource definitions.

7.2.5 Compiled files

FLEX-ES system definitions and resource definitions are compiled (with the **cfcomp** command) to produce **syscf** and **rescf** files. You can have separate source files for system and resource definitions, or combine them into one file (like we do in this publication). You can have multiple system and/or resource definitions in a single file. For example:

```

system patti:
  system definitions
end patti

system billy:
  definitions for another system
end billy

system fran:
  definitions for yet another system
end fran

resources mixed:
  resource definitions
end mixed

```

If you compile this file with **cfcomp**, you will have four output files: **patti.syscf**, **billy.syscf**, **fran.syscf**, and **mixed.rescf**. The **syscf** files are potential operands for a **flexes** command, and the **rescf** file is a potential operand for the **resadm** command. We could produce exactly the same results by breaking this input file into four files and compiling each one separately.

Remember that you can have only one resource definition active (via the **resadm** command) on a server, but you can have multiple emulated S/390 systems active. Assuming we defined compatible systems and resources in this example, we could start the resources (**resadm -s mixed.rescf**) and then start three systems (**flexes patti.syscf**, **flexes billy.syscf**, **flexes fran.syscf**). For practical purposes, we would probably also want to start three interactive **flexescli** windows to control the three systems.

7.3 Common rules

FLEX-ES names, in general, may be up to 255 characters and use uppercase/lowercase letters, numeric digits, and a few special characters. These special characters are underscore, hyphen, dollar, at, and period. A name cannot consist of a valid decimal or hexadecimal number. The following are *not* valid names: 2, 123, 1403, 0X1, 7F, 7f, 1C2, and a30.

White space (usually blanks) may be included almost anywhere between words. Two special cases exist:

- ▶ No white space may exist in the parameters of an *options* statement or in a *devopt* parameter. For example:

```
device(00) 3390-1 /S390/VOLAs1 devopt 'trackcachesize = 10'
```

This is incorrect; there should not be spaces before or after the equal sign.

- ▶ If a device number range is used, there must be spaces before and after the hyphen. For example:

```
device(00 - 15) 3278 OFFLINE
```

This is correct because there is a space before and after the hyphen.

FLEX-ES *keywords* cannot also be used as *names* in system or resource sections.

Numbers (in system and resource definitions) are always decimal. If you want a hexadecimal number, you must indicate it. For example:

```
cu devad(0xA80,10) path(1) resource(xyz)
```

contains a hexadecimal number (0xA80) and a decimal number (10). CLI *commands* differ; they assume device addresses are hexadecimal.

You should not use duplicate names in any definitions.

7.4 The resadm command

The **resadm** command, used to start and control the resource manager, has a number of options that are frequently used. Using R14A.rescf as an example of a compiled resource definition, the options are:

# resadm -s R14A.rescf	(start the resource manager)
# resadm -k	(kill the resource manager)
# resadm -t cu:cu3990A	(terminate an individual resource)
# resadm -T	(terminate all resources)
# resadm -r	(list all local active resources)
# resadm [-h hostname] -r	(list remote active resources)
# resadm [-h hostname] -n	(list node names on remote resource manager)
# resadm -x R14A.rescf	(refresh the resource definitions)

A few notes about **resadm** may be useful:

- ▶ The -r and -n options may be used by anyone. The other options are available only to *root*.
- ▶ The -r option is the most frequently used option.
- ▶ For all except the -s option, the resource manager is assumed to be running when these commands are issued.
- ▶ The -n option is for use when multiple servers are cooperatively running FLEX-ES resources.

- ▶ You should terminate resources gracefully, with -t or -T options, instead of killing them (with -k or Linux commands).
- ▶ A common sequence is to -T (terminate all resources), then -x (refresh with a newly compiled resource file), and then re-IPL a S/390 operating system.
- ▶ You can -t (selectively terminate a resource) and then -x (refresh the resources file). This will refresh only resources that are not running at that time.
- ▶ The resource manager writes log information to the /var/log/messages file (via the Linux syslog logging facility).
- ▶ The resource manager automatically uses TCP/IP port 555 to communicate with other elements of FLEX-ES, whether there is a single server or there are multiple servers involved.

7.5 CLI commands

Every S/390 emulated system instance has an associated *main console* for FLEX-ES control commands. This is a virtual console that is not directly connected to a real terminal. The commands it processes are the *CLI commands*, some of which are described here. (CLI means Command Line Interface.) A program named **flexesccli** is used to communicate with the virtual main console. **flexesccli** works from a Linux command line, and you can start as many **flexesccli** instances as you like. The **flexesccli** program sends commands to a *main console* and returns the results.

In discussions we typically ignore the details of the **flexesccli** and *main console* interaction and describe operations in terms of commands to **flexesccli**. Also, for our basic EFS discussion, we usually assume that only a single S/390 is being emulated at any given time.

CLI commands can be entered two ways:

- ▶ You can start the **flexesccli** program in interactive mode and then issue CLI commands directly at the *flexes* prompt provided by the CLI program. All the sample startup scripts in the redbooks do this.
- ▶ You can use ECHO to pipe a command to **flexesccli**. In this case, it executes the command and terminates.

The syntax for the **flexesccli** command is:

```
$ flexesccli IPname systemname
$ flexesccli IPname systemname filename
```

If the **flexesccli** program is not in the current PATH, you would need to issue the full path name for it. The *IPname* can be an IP address, a host name that is resolved by DNS, or a local name (in /etc/hosts). The standard IPname *localhost* is used to reference the local IP loopback address. The *systemname* is the name of a syscf file used to start an emulated S/390. In our examples, this is S14A.syscf. (The syscf suffix is omitted for the **flexesccli** command.) The **flexesccli** program must always be directed to a specific S/390 instance (even if there is only one running).

An example of executing a single command through the CLI interface might be:

```
# echo 'ipl a80 0a8200' | flexesccli localhost S14A
# ... -----+--- S/390 system name (S14A.syscf)
               |   |   +----- local system (TCP/IP loopback)
               |   |   +----- the flexesccli command
               |   +----- the pipe operator
               +----- CLI command to be executed
```

The **flexesccli** program senses whether input is waiting for it in a pipe; if so, it executes the waiting command and quits. If started this way:

```
# flexesccli localhost S14A
flexes>
```

it issues a flexes prompt and waits for commands. It will run until it receives a **quit** or **shutdown** command.

If the **flexesccli** command includes a file name as the third operand, the CLI commands in that file are executed and then an interactive prompt is provided. For example, we might have file **CLlcmd** (in the current directory) containing:

```
mount 703 termX
mount 704 termY
mount 500 /usr/flexes/tapes/222222
ipl A80 0A8200
set prompt Rel1.5>
```

This could be used as follows (assuming the instance name is S14A):

```
$ flexesccli localhost S14A CLlcmd
(the commands in the CLlcmd file are executed, including the IPL)
REL1.5>                                     (Waiting for next CLI command)
```

There are a considerable number of CLI commands. Many of them are for low-level S/390 hands-on operation (such as instruction stepping, inspecting registers and memory, and so forth). These are described in detail in *FSIMM210: CLI Language Reference*, from FSI. We describe only a few of the higher-level commands here. We assume you are in **flexesccli** interactive mode for these examples:

flexes> halt	(stops current emulated S/390 CPUs)
flexes> go	(resumes operation, if not single step)
flexes> hwc	(enter text for system console)
flexes> ipl devaddr [parm]	(IPLs the first emulated CPU)
flexes> iplc devaddr [parm]	(clears state and memory first)
flexes> mount devaddr [filename OFFLINE]	(discussed later)
flexes> notready devaddr	(make emulated device not ready)
flexes> quit	(terminate flexesccli; also 'exit')
flexes> ready devaddr	(makes emulated device ready)
flexes> restart	(like the 'restart' button on S/390)
flexes> rewind devaddr	(only for FakeTape or SCSI drives)
flexes> shutdown	(gracefully terminates S/390)
flexes> unload devaddr	(unloads FakeTape or SCSI drive)
(like a 'mount devaddr OFFLINE' for other devices)	

Here are a few brief notes concerning CLI commands:

- ▶ Unlike resource definitions, CLI numeric parameters are assumed to be hexadecimal.
- ▶ There are many aliases for commands, not shown here (for example, **go** = **g** = **start**).
- ▶ The **mount** command is perhaps the most common CLI command. Do not confuse it with the Linux **mount** command. They are completely different commands that perform different functions.
 - A resource definition can define a device as **OFFLINE**. This means that the emulated device exists but is, in effect, turned off.
 - A CLI **mount** command can, while an emulated S/390 is running, dynamically “turn on” the device, using a specified file or name. Some examples are:

```
flexes> mount A90 /s390/WORK01          (assume A90 is a 3390)
flexes> mount 710 altcons                (assume 710 is a local 3270)
```

flexes> mount 560 /home/tape3

(a FakeTape file)

- The first example is similar to mounting a disk pack (if disk packs were available for 3390s). The named file (or symbolic link) should point to a properly formatted emulated 3390 volume.
- The second example causes the name *altcons* to be added to the list of connections available through the Terminal Solicitor.
- The third example, in effect, is a tape mount.

Archived

z/OS base sysplex on z/VM

This FLEX-ES Usage Note was contributed by Michael Ryan of Fundamental Software, Inc., Fremont, California.

This paper describes test results we obtained while defining and operating a z/OS base sysplex on z/VM using Fundamental Software's FLEX-ES System/390® and z/Architecture™ emulator.

Acknowledgements

David MacMillan of Fundamental Software helped greatly by proofreading this document and pointing out technical inconsistencies and/or errors where they occurred. Gary Ehemann of Fundamental Software supplied z/VM expertise that helped us sort out several difficult z/VM issues.

8.1 Test environment

The sysplex was built and tested on an IBM eSeries x235 server running FLEX-ES version 6.2.10 and UnixWare 7.1.2.⁶² The server had two 2.8GHz Xeon processors. Both server processors were enabled for mainframe emulation. The server had 6GB of RAM and 18 SCSI drives set up as two RAID5 arrays. One array of three drives contained UnixWare 7.1.2; a second array of 14 drives in an IBM EXP300 external enclosure was dedicated to z/VM 4.3 and z/OS 1.4. All drives were 36G, 15K RPM U320 drives. We used a ServeRAID 4H controller and defined one of the 18 drives as a hot spare.

The sysplex was built using the ADCD z/OS 1.4 distribution as a base. All members of the sysplex shared one set of system files (PARMLIB, PROCLIB, master catalog, NUCLEUS, LPALIB, and so forth). Each member had its own set of LOGREC, SMF, paging, and VIO journal files. We ran one of the guest systems (zos1) with 1024MB and the other (zos2) with 512MB with appropriate FLEX-ES track cache and instruction cache sizes without

⁶² This IBM Redbook generally discusses FLEX-ES running in a Linux environment. The base sysplex described in this chapter is in a UnixWare environment instead of a Linux environment. This makes very little difference to the material being presented here. A few parameters in the FLEX-ES definition files differ for the UnixWare environment. UnixWare 7.1.2 is also known as Open UNIX 8.

encountering any UnixWare 7.1.2 paging. We defined two virtual processors for each z/OS system.

Because each FLEX-ES instance maintains a separate time-of-day clock, the sysplex must be run in a single FLEX-ES instance in order to meet the conditions needed for z/OS to run in ETR (External Timer Reference) emulation mode. Each FLEX-ES instance is logically equivalent to a separate System/390 central processor complex (CPC). ETR emulation is required because the IBM 9037 Sysplex Timer® does not support FLEX-ES.

8.2 Scope

Definition of a base sysplex requires relatively sophisticated systems work such as system cloning, tuning, problem determination, and specifying z/OS parameters. This paper describes what we did and how it worked, but we cannot guarantee that our decisions will fit all circumstances.

For base sysplex to properly sequence timestamps, all z/OS members must either be run on the same CPC (if using System/390 or z/Series hardware) or be run on the same VM host (if using z/VM guest machines). Although a base sysplex can be brought up in separate FLEX-ES instances on one server, such a mode of operation is not currently supported by z/OS.

This paper is a test report; it cannot and does not guarantee that base sysplex will function as desired in all circumstances; in particular, we have not tested the base sysplex in any kind of production environment using production workloads.

A Base Sysplex configuration is not supported by Fundamental Software for production use.

8.2.1 Who should read this paper

This paper will be of interest to anyone who is investigating a base sysplex on FLEX-ES. The paper assumes familiarity with MVS systems programming, especially system initialization, configuration, and planning. This paper does not intend to cover all issues concerning sysplex definition; we assume the reader is able to perform MVS configuration. We do mention some of the mistakes we made along the way and how we corrected them. We further assume the reader is a user or prospective user of FLEX-ES and has a general understanding of the mainframe emulator and its functions.

8.2.2 What a base sysplex is

The base sysplex, as defined in *Parallel Sysplex Overview: Introducing Data Sharing and Parallelism in a Sysplex*, SA22-7661, is a more tightly-integrated version of a loosely-coupled (shared DASD, shared spool) multi-system configuration. In a base sysplex, multiple copies of MVS (interchangeably called images, systems, or members) may be started from one residence volume. The images of necessity share some DASD volumes but also may access private DASD. The members share one JES2 spool and associated checkpoint records. Any of the members may select work for processing. Work may be restricted to a specific system using SYSAFF or initiator classes. TSO and other subsystems may run on any or all of the images, and, since the members may share one RACF® database, users may log on to any image running VTAM and TCAS. The systems communicate using channel-to-channel (CTC) adapters. Inbound and outbound CTC connections are required from each system to every other system in the sysplex.

Global Resource Serialization (GRS) is used to extend ENQ processing to SYSTEMS (all-members) scope and optionally to convert reserve/release requests that normally serialize access to an entire volume to ENQ/DEQ requests that serialize specific data set(s). GRS also uses CTCs to communicate among the members.

8.2.3 What a Parallel Sysplex® is

A Parallel Sysplex looks a lot like a base sysplex, but uses an IBM Coupling Facility (CF) for signaling among systems. A parallel sysplex extends base sysplex functionality by storing and retrieving lists of data contained in coupling facility structures. CF structures are used to implement extended catalog facilities, workload balancing, subsystem functions (for subsystems like JES2, DB2, CICS, and IMS), and a common system log, among others. CF can be run only on selected IBM System/390 and z/Series processors.

8.2.4 Why run a base sysplex

Using a Base Sysplex on an EFS system offers advantages in the following areas:

- ▶ Ease of maintenance
- ▶ Isolation
- ▶ Throughput
- ▶ Single-control point
- ▶ Approximation of customer environments

Base sysplex offers the advantages of running more than one z/OS system without requiring the overhead and work of maintaining separate z/OS installations. Each member of the base sysplex may be initialized using the same residence volumes and master catalog. Only one set of target and distribution library data sets needs to be maintained.

The base sysplex offers the ability to isolate workloads from each other; for example, some programmers might use one image to develop and bench-test code while others use a second image for integration testing, and a third image runs other work. If one image fails or is damaged by testing, the other(s) may continue to run.

Global Resource Serialization (GRS) provides systems-scope ENQ/DEQ for members of the sysplex. GRS may be implemented without defining a base sysplex, but a sysplex cannot be defined without activating GRS. Under GRS, only the target data set is serialized--remaining data on a volume may be accessible to other jobs and users. Improvements in throughput are relative to a shared-spool, shared-DASD configuration that uses reserve/release to protect shared data. When large numbers of System/390 processors are available (four or more), our experience suggests that base sysplex may improve overall throughput.

The base sysplex allows an operator to monitor and control all systems from one console. Commands may be routed to specific systems or broadcast to all systems.

Many EFS systems are used by independent software vendors (ISVs); testing using a base sysplex may more closely approximate their customers' environments.

8.3 FLEX-ES system definitions

Our FLEX-ES system definition for running z/VM was:

```
system sys1:
feature lpar
lparnum(1)
memsize(0x1F0000)
```

```

esssize(0x320)
cachesize(0x2000)
tracesize(0x100)
instset(esa)
cpu(0)
cpu(1)
channel(0) localbyte
channel(1) local
channel(2) localosa
cu devad(0x0005,01) path(0) resource(ZVM.3211A)
cu devad(0x000c,04) path(0) resource(ZVM.2821A)
cu devad(0x0018,01) path(0) resource(ZVM.4245A)
cu devad(0x0020,32) path(1) resource(ZVM.3274A)
cu devad(0x0080,08) path(1) resource(ZVM.3274B)
cu devad(0x0180,16) path(1) resource(ZVM.3422A)
cu devad(0x0190,16) path(1) resource(ZVM.3490A)
cu devad(0x0200,32) path(1) resource(ZVM.3990A)
cu devad(0x0300,32) path(1) resource(ZVM.3990B)
cu devad(0x0A80,32) path(1) resource(ZVM.3990C)
cu devad(0x0e00,02) path(2) resource(ZVM.0SA1)
cu devad(0x0e10,02) path(2) resource(ZVM.0SA3)
cu devad(0x0e60,01) path(2) resource(ZVM.0SA2)
end sys1

```

We defined an Enterprise Systems Architecture (ESA) logical central processor complex (CPC) with two System/390 (31-bit) processors. We could have run the instance in z/Architecture (64-bit) mode by specifying *instset(z)* instead of *instset(esa)*. The xSeries 235 server has two Xeon processors; the FLEX-ES lab license allowed both to be enabled for mainframe emulation, and so we specified:

```

cpu(0)
cpu(1)

```

The numbers (0) and (1) are the logical System/390 CPU addresses. Both of the server's Xeon processors were used by FLEX-ES System/390 emulation, although neither was dedicated exclusively to emulation.⁶³

A z/VM function called active wait became a factor because the Xeon processors were not exclusively dedicated. Active wait kept the server's Xeon processors fully utilized; while this would not have been an issue for dedicated processors (because z/VM would have had exclusive control), it could have interfered with other S/390 or UnixWare 7.1.2 work. We specified feature *lpar* and *lparnum(1)* which kept z/VM from using active wait. For more information about feature *lpar*, see the FLEX-ES documents *FSIMM300: System Programmer's Guide* and *FSIMM320: Resource Language Tutorial*.

We used FLEX-ES *osa* and *osasna* control units for TCP/IP and SNA-over-Ethernet, respectively; we defined the required *localosa* channel for these device types.

⁶³ Although our xSeries server's Xeon processors could have been reserved exclusively for mainframe emulation with the dedicated keyword, at least one server processor had to remain non-dedicated and thus available to dispatch other UnixWare 7.1.2 processes (including other FLEX-ES processes). Since our server had two Xeon processors, we could have dedicated only one of them. Because CPUs in a single FLEX-ES instance should not be a mix of dedicated and non-dedicated types (for performance reasons), we did not use the dedicated keyword on either CPU definition.

8.4 FLEX-ES resource definitions

To run z/VM, we defined only a subset of the available memory, disk slices, network adapters, and other resources available on our xSeries server. Because we were defining only the resources needed to run one z/VM host, we named the resources zvm.

```
resources zvm:
#####
#Unit-Record
#####

ZVM.3211A: cu 3211
    interface local(1)
        device( 0x00 ) 3211 OFFLINE
end ZVM.3211A

ZVM.2821A: cu 2821
    interface local(1)
        device( 0x00 ) 2540R OFFLINE
        device( 0x01 ) 2540P OFFLINE
        device( 0x02 ) 1403-N1 OFFLINE
        device( 0x03 ) 1403-N1 OFFLINE
end ZVM.2821A

ZVM.4245A: cu 4245
    interface local(1)
        device( 0x00 ) 4245 OFFLINE
end ZVM.4245A

#####
#Tape
#####

ZVM.3490A: cu 3490
    interface local(1)
        device( 0x00 - 0x0f ) 3490-E OFFLINE
end ZVM.3490A

ZVM.3422A: cu 3422
    interface local(1)
        device( 0x00 - 0x0f ) 3422 OFFLINE
end ZVM.3422A

ZVM.3480A: cu 3480
    interface local(1)
        device( 0x00 - 0x0f ) 3480 OFFLINE
end ZVM.3480A

#####
#Displays
#####

ZVM.3274A: cu 3274
    interface local(1)
        device( 0x00 ) 3278 vm_console
        device( 0x01 ) 3278 zvm-1
        device( 0x02 ) 3278 zvm-2
        device( 0x03 ) 3278 zvm-3
        device( 0x04 ) 3278 zvm-4
        device( 0x05 ) 3278 zvm-5
```

```

device( 0x06 ) 3278 zvm-6
device( 0x07 ) 3278 zvm-7
device( 0x08 ) 3278 zvm-8
device( 0x09 ) 3278 zvm-9
device( 0x0a ) 3278 zvm-10
device( 0x0b ) 3278 zvm-11
device( 0x0c ) 3278 zvm-12
device( 0x0d ) 3278 zvm-13
device( 0x0e ) 3278 zvm-14
device( 0x0f ) 3278 zvm-15
device( 0x10 ) 3278 zvm-10
device( 0x11 ) 3278 zvm-11
device( 0x12 ) 3278 zvm-12
device( 0x13 ) 3278 zvm-13
device( 0x14 ) 3278 zvm-14
device( 0x15 ) 3278 zvm-15
device( 0x16 ) 3278 zvm-16
device( 0x17 ) 3278 zvm-17
device( 0x18 ) 3278 zvm-18
device( 0x19 ) 3278 zvm-19
device( 0x1a ) 3278 zvm-1a
device( 0x1b ) 3278 zvm-1b
device( 0x1c ) 3278 zvm-1c
device( 0x1d ) 3278 zvm-1d
device( 0x1e ) 3278 zvm-1e
device( 0x1f ) 3278 zvm-1f
end ZVM.3274A

ZVM.3274B: cu 3274 interface local(1)
device( 0x00 ) 3278 x080
device( 0x01 ) 3278 x081
device( 0x02 ) 3278 x082
device( 0x03 ) 3278 x083
device( 0x04 ) 3278 x084
device( 0x05 ) 3278 x085
device( 0x06 ) 3278 x086
device( 0x07 ) 3278 x087
end ZVM.3274B

#####
#Network
#####

ZVM.OSA1: cu osa options 'adapternumber=0'
interface local(1)
device(00) osa /dev/net2
device(01) osa OFFLINE
end ZVM.OSA1

ZVM.OSA2: cu osasna options 'adapternumber=1'
interface local(1)
device(00) osasna /dev/net2
end ZVM.OSA2

ZVM.OSA3: cu osa options 'adapternumber=0'
interface local(1)
device(00) osa /dev/net4
device(01) osa OFFLINE
end ZVM.OSA3

```

```

ZVM.XCA1: cu xcasna options 'adapternumber=1'
interface local(1)
device(00) xcasna /dev/net3
end ZVM.XCA1

ZVM.3172A: cu 3172 options 'adapternumber=0'
interface local(1)
device( 0x00 ) 3172 /dev/net5
device( 0x01 ) 3172 OFFLINE
end ZVM.3172A

#####
#DASD
#####

ZVM.3990A: cu 3990
interface local(1)
device( 0x00 ) 3390-3 /aos/systems/x370/430res
device( 0x01 ) 3390-3 /aos/systems/x370/430w01
device( 0x02 ) 3390-3 /aos/systems/x370/430w02
device( 0x03 ) 3390-3 /aos/systems/x370/430w03
device( 0x04 ) 3390-3 /aos/systems/x370/430w04
device( 0x05 ) 3390-3 /aos/systems/x370/sles8d
device( 0x06 ) 3390-3 /aos/systems/x370/dosres
device( 0x07 ) 3390-3 /aos/systems/x370/syswk1
device( 0x08 - 0x1f ) 3390-3 OFFLINE
end ZVM.3990A

ZVM.3990B: cu 3990 options 'trackcachesize=1800'
interface local(1)
device(0x00) 3390-3 /aos/systems/x370/zp4rs1
device(0x01) 3390-3 /aos/systems/x370/zp4rs2
device(0x02) 3390-3 /aos/systems/x370/zp4sys
device(0x03) 3390-3 /aos/systems/x370/zp4us1
device(0x04) 3390-3 /aos/systems/x370/zp4ck1
device(0x05) 3390-3 /aos/systems/x370/zp4ck2
device(0x06) 3390-3 /aos/systems/x370/zp4sp1
device(0x07) 3390-3 /aos/systems/x370/zp4sp2
device(0x08) 3390-3 /aos/systems/x370/zp4sms
device(0x09) 3390-3 /aos/systems/x370/zp4cds
device(0x0a) 3390-3 /aos/systems/x370/zp4log
device(0x0b) 3390-3 /aos/systems/x370/zp4pg0
device(0x0c) 3390-3 /aos/systems/x370/zp4pg1
device(0x0d) 3390-3 /aos/systems/x370/zp4pg2
device(0x0e) 3390-3 /aos/systems/x370/zp4wk1
device(0x0f) 3390-3 /aos/systems/x370/zp4wk2
device(0x10) 3390-3 /aos/systems/x370/zp4ds1
device(0x11) 3390-3 /aos/systems/x370/zp4ds2
device(0x12) 3390-3 /aos/systems/x370/zp4ds3
device(0x13) 3390-3 /aos/systems/x370/zp4ds4
device(0x14) 3390-3 /aos/systems/x370/zp4db2
device(0x15) 3390-3 /aos/systems/x370/zp4cic
device(0x16) 3390-3 /aos/systems/x370/zp4ims
device(0x17) 3390-3 /aos/systems/x370/af0lib
device(0x18) 3390-3 /aos/systems/x370/af0mnt
device(0x19) 3390-3 /aos/systems/x370/aotso1
device(0x1a) 3390-3 /aos/systems/x370/zp4wk3
device( 0x01b - 0x01f ) 3390-3 OFFLINE
end ZVM.3990B

```

```

ZVM.3990C: cu 3990 options 'trackcachesize=1800'
interface local(1)
device( 0x00 ) 3390-3 /aos/systems/x370/z4res1
device( 0x01 ) 3390-3 /aos/systems/x370/z4res2
device( 0x02 ) 3390-3 /aos/systems/x370/os39m1
device( 0x03 ) 3390-3 /aos/systems/x370/z4ckpt
device( 0x04 ) 3390-3 /aos/systems/x370/spool0
device( 0x05 ) 3390-3 /aos/systems/x370/z4pag0
device( 0x06 ) 3390-3 /aos/systems/x370/z4pag1
device( 0x07 ) 3390-3 /aos/systems/x370/z4pag2
device( 0x08 ) 3390-3 /aos/systems/x370/z4pag3
device( 0x09 ) 3390-3 /aos/systems/x370/z4dis1
device( 0x0a ) 3390-3 /aos/systems/x370/z4dis2
device( 0x0b ) 3390-3 /aos/systems/x370/z4dis3
device( 0x0c ) 3390-2 /aos/systems/x370/z4cic1
device( 0x0d ) 3390-2 /aos/systems/x370/z4ims1
device( 0x0e ) 3390-3 /aos/systems/x370/z4db21
device( 0x0f ) 3390-3 /aos/systems/x370/work01
device( 0x10 ) 3390-3 /aos/systems/x370/work02
device( 0x11 ) 3390-3 /aos/systems/x370/z4was1
device( 0x12 ) 3390-3 /aos/systems/x370/z4was2
device( 0x13 ) 3390-3 /aos/systems/x370/z4uss1
device( 0x14 ) 3390-3 /aos/systems/x370/sares1
device( 0x15 ) 3390-3 /aos/systems/x370/z4dis4
device( 0x16 ) 3390-3 /aos/systems/x370/z4plex
device( 0x17 - 0x1f ) 3390-3 OFFLINE
end ZVM.3990C

end zvm

```

On DASD control unit ZVM.3990A, we mounted the 3390-3 images needed to run z/VM. Control unit ZVM.3990B contained the z/OS sysplex volumes. Control unit ZVM.3990C contained a full z/OS 1.4 ADCD distribution, which we used as a driver system.

8.5 z/VM guest ZOS1

In order to share DASD between two or more z/VM guests, the volumes must be defined as z/VM minidisks. In our case, we used full-volume minidisks, specifying cylinder 000 through the end of the volume:

```
MDISK 300 3390 0000 END ZP4RS1 MWV
```

This statement defines a full-pack minidisk at virtual address 300; the device is a 3390 encompassing cylinders from 0000 to END. z/VM locates the volume by volume serial number (in this case, ZP4RS1). The letters MWV have the following meanings:

- ▶ MW = Multiple virtual machines may have write access to the minidisk
- ▶ V = CP will use its virtual reserve/release support in I/O operations for the minidisk

We defined two guest machines in the z/VM directory, zos1 and zos2. The directory entry for zos1 was:

```

USER ZOS1 ZOS1 1024M 1024M BG
MACHINE ESA 2
OPTION MAINTCCW LNKEXCLU
ACCOUNT 2 SYSTEMS
CPU 00 BASE NODEDICATE
CPU 01 NODEDICATE
IPL CMS

```

```

*
*E00/E01 =FLEX-ES EMULATED OSA-1 FOR ZOS TCP/IP
*
DEDICATE 0E00 0E00
DEDICATE 0E01 0E01
*
*E60 =FLEX-EX EMULATED OSASNA FOR SNA OVER ETHERNET
*
DEDICATE 0E60 0E60
*
*AD/CD ZOS 1.4 DRIVER SYSTEM (IPL A80 LOADPARM 0A8200)
*
DEDICATE 0A80 0A80
DEDICATE 0A81 0A81
DEDICATE 0A82 0A82
DEDICATE 0A83 0A83
DEDICATE 0A84 0A84
DEDICATE 0A85 0A85
DEDICATE 0A86 0A86
DEDICATE 0A87 0A87
DEDICATE 0A88 0A88
DEDICATE 0A89 0A89
DEDICATE 0A8A 0A8A
DEDICATE 0A8B 0A8B
DEDICATE 0A8C 0A8C
DEDICATE 0A8D 0A8D
DEDICATE 0A8E 0A8E
DEDICATE 0A8F 0A8F
DEDICATE 0A90 0A90
DEDICATE 0A91 0A91
DEDICATE 0A92 0A92
DEDICATE 0A93 0A93
DEDICATE 0A94 0A94
DEDICATE 0A95 0A95
DEDICATE 0A96 0A96
*
*Z/OS MASTER CONSOLE
*
CONSOLE FC01 3215
*
*TSO SESSIONS
*
SPECIAL 701 3270
SPECIAL 702 3270
SPECIAL 703 3270
SPECIAL 704 3270
SPECIAL 705 3270
SPECIAL 706 3270
SPECIAL 707 3270
SPECIAL 708 3270
*
*CTC'S FOR SYSPLEX SIGNALING
*
SPECIAL 1031 CTCA
SPECIAL 1032 CTCA
*
*UNIT RECORD GEAR
*
SPOOL 00C 2540 READER A
SPOOL 00D 2540 PUNCH B

```

```

SPOOL 00E 1403 A
SPOOL 00F 1403 A
*
*CMS A DISK
*
MDISK 191 3390 0201 010 430W03 MR ZOS ZOS ZOS
*
*Z/OS BASE SYSPLEX
*
MDISK 300 3390 0000 END ZP4RS1 MWV
MDISK 301 3390 0000 END ZP4RS2 MWV
MDISK 302 3390 0000 END ZP4SYS MWV
MDISK 303 3390 0000 END ZP4WK1 MWV
MDISK 304 3390 0000 END ZP4WK2 MWV
MDISK 305 3390 0000 END ZP4WK3 MWV
MDISK 306 3390 0000 END ZP4SMS MWV
DASDOPT WRKALLEG
MDISK 307 3390 0000 END ZP4CDS MWV
DASDOPT WRKALLEG
MDISK 308 3390 0000 END ZP4LOG MWV
DASDOPT WRKALLEG
MDISK 309 3390 0000 END ZP4US1 MWV
MDISK 30A 3390 0000 END ZP4PG0 MWV
MDISK 30B 3390 0000 END ZP4PG1 MWV
MDISK 30C 3390 0000 END ZP4PG2 MWV
MDISK 30D 3390 0000 END AF0LIB MWV
MDISK 30E 3390 0000 END AF0MNT MWV
MDISK 30F 3390 0000 END ZP4CK1 MWV
MDISK 310 3390 0000 END ZP4CK2 MWV
MDISK 311 3390 0000 END ZP4SP1 MWV
MDISK 312 3390 0000 END ZP4SP2 MWV
MDISK 313 3390 0000 END ZP4CIC MWV
MDISK 314 3390 0000 END ZP4DB2 MWV
MDISK 315 3390 0000 END ZP4IMS MWV
MDISK 316 3390 0000 END ZP4DS1 MWV
MDISK 317 3390 0000 END ZP4DS2 MWV
MDISK 318 3390 0000 END ZP4DS3 MWV
MDISK 319 3390 0000 END ZP4DS4 MWV
MDISK 31A 3390 0000 END AOTS01 MWV
*
*CMS MINI-DISKS
*
LINK MAINT 0190 0190 RR
LINK MAINT 019D 019D RR
LINK MAINT 019E 019E RR

```

A few items of note:

- We dedicated the volumes of the ADCD driver system to user *zos1* since we did not intend to share them with *zos2*. When we wanted to run the driver system, we logged onto the *zos1* guest machine and issued the command: **ip1 a80 loadparm 0a8200**.
- When we ran the driver system, we didn't bother to define a 3270 for the operator; we used the guest machine's console to emulate a hardware service processor console. See the **vinput** command in SC24-6008-03, *CP Command and Utility Reference*.
- We used full-pack MDISK statements for the sysplex so that we could share the volumes among more than one guest machine.

Note that we dedicated both the FLEX-ES emulated OSA-1 used for TCP/IP (0E00, 0E01) and the FLEX-ES *osasna* device used for the sysplex SNA-over-Ethernet connection (0E60):

```
*
*E00/E01 =FLEX-ES EMULATED OSA-1 FOR ZOS TCP/IP
*
DEDICATE 0E00 0E00
DEDICATE 0E01 0E01
*
*E60 =FLEX-EX EMULATED OSASNA FOR SNA OVER ETHERNET
*
DEDICATE 0E60 0E60
```

8.5.1 Working allegiance

Because we were sharing DASD between VM guests, we needed to set working allegiance (WRKALLEG) on for any volumes that contained any type of sysplex couple data set. This setting guarantees that if multiple virtual machines start an operation to one device, that machine's channel program runs to completion or terminates before the other machine's is allowed to start. Without this setting, CP may interleave CCWs from more than one virtual machine in a single channel program. This can interfere with proper operation of the couple data sets, particularly cross-system lock integrity. In our case, the couple data sets were all located on one of the following volumes: ZP4SMS, ZP4CDS, or ZP4LOG. We requested WRKALLEG for these three volumes by coding DASDOPT statements immediately following their MDISK definitions:

```
MDISK 306 3390 0000 END ZP4SMS MWV
DASDOPT WRKALLEG
MDISK 307 3390 0000 END ZP4CDS MWV
DASDOPT WRKALLEG
MDISK 308 3390 0000 END ZP4LOG MWV
DASDOPT WRKALLEG
```

8.5.2 z/VM emulated CTCs

In each virtual machine's directory entry, we defined z/VM virtual CTCs used by z/OS for sysplex signaling and Global Resource Serialization (GRS):

```
SPECIAL 1031 CTCA
SPECIAL 1032 CTCA
```

We then used a REXX exec to connect the virtual CTCs and to IPL the systems.

8.6 z/VM guest ZOS2

The directory entry for system zos2 was a little different, since we established links to zos1's minidisks:

```
USER ZOS2 ZOS2 512M 512M BG
MACHINE ESA 2
OPTION MAINTCCW LNKEXCLU
ACCOUNT 2 SYSTEMS
CPU 00 BASE NODEDICATE
CPU 01 NODEDICATE
IPL CMS
*
*FLEX-ES EMULATED OSA-1 FOR TCP/IP
*
```

```

DEDICATE 0E10 0E10
DEDICATE 0E11 0E11
*
*ZOS2 MVS MASTER CONSOLE
*
CONSOLE FC02 3215
*
*TSO SESSIONS
*
SPECIAL 701 3270
SPECIAL 702 3270
SPECIAL 703 3270
SPECIAL 704 3270
SPECIAL 705 3270
SPECIAL 706 3270
SPECIAL 707 3270
SPECIAL 708 3270
*
*CTCS FOR SYSPLEX SIGNALING
*
SPECIAL 1031 CTCA
SPECIAL 1032 CTCA
*
*UNIT RECORD
*
SPOOL 00C 2540 READER A
SPOOL 00D 2540 PUNCH B
SPOOL 00E 1403 A
SPOOL 00F 1403 A
*
*CMS A DISK
*
MDISK 0191 3390 1001 010 430W03 MR ZOS ZOS ZOS
*
*Z/OS BASE SYSPLEX -ZOS1 MINIDISKS
*
LINK ZOS1 0300 0300 MW
LINK ZOS1 0301 0301 MW
LINK ZOS1 0302 0302 MW
LINK ZOS1 0303 0303 MW
LINK ZOS1 0304 0304 MW
LINK ZOS1 0305 0305 MW
LINK ZOS1 0306 0306 MW
LINK ZOS1 0307 0307 MW
LINK ZOS1 0308 0308 MW
LINK ZOS1 0309 0309 MW
LINK ZOS1 030A 030A MW
LINK ZOS1 030B 030B MW
LINK ZOS1 030C 030C MW
LINK ZOS1 030D 030D MW
LINK ZOS1 030E 030E MW
LINK ZOS1 030F 030F MW
LINK ZOS1 0310 0310 MW
LINK ZOS1 0311 0311 MW
LINK ZOS1 0312 0312 MW
LINK ZOS1 0313 0313 MW
LINK ZOS1 0314 0314 MW
LINK ZOS1 0315 0315 MW
LINK ZOS1 0316 0316 MW
LINK ZOS1 0317 0317 MW

```

```

LINK ZOS1 0318 0318 MW
LINK ZOS1 0319 0319 MW
LINK ZOS1 031A 031A MW
*
*CMS MINIDISKS
*
LINK MAINT 0190 0190 RR
LINK MAINT 019D 019D RR
LINK MAINT 019E 019E RR

```

Devices E10 and E11 represent the FLEX-ES emulated OSA-1 that *zos2* used for TCP/IP.

8.7 System IPL

We elected to use the console defined in each directory entry as the MVS master console. For CMS usage, this console was defined as a 3215; we changed it to a 3278 after we were finished with CMS so that MVS could then use it. Our primary need for CMS was to start the IPL exec. To make the console change and to IPL each system, we created a small REXX exec that accomplished the following tasks:

- ▶ Established connections on the virtual channel-to-channel adapters for sysplex signaling.
- ▶ Detached the CMS minidisks so that there would be no conflict with devices defined to z/OS.
- ▶ Changed the console from 3215 to 3270 mode.
- ▶ IPLed the virtual machine with the proper LOADPARM.

The REXX code we used was:

```

/*ZOS1 IPL EXEC */
/*FILENAME =ZOS1 EXEC */
"CP SET PF12 RETRIEVE"
"CP SET RUN ON "
"CP COUPLE 1031 TO ZOS2 1031"
"CP COUPLE 1032 TO ZOS2 1032"
le='15'x /*CP LINE END CHARACTER */
iplstr='DETACH 190'||le||'DETACH 19E'||le||'DETACH 19D'||le||'TERM CONMODE
3270'||le||'IPL 300 LOADPARM 030200'
PARSE VALUE DIAG(8,iplstr) WITH STUFF

```

The line that begins with *iplstr* is broken into two lines in the example above and below, but is a single line in the REXX execs. It contains a set of CP commands separated by the CP line end character. These are passed to CP with a DIAG 8 function call. The results from the DIAG call are not useful and are assigned to variable *stuff*.

```

/*ZOS2 IPL EXEC */
/*FILENAME =ZOS2 EXEC */
"CP SET PF12 RETRIEVE"
"CP SET RUN ON "
"CP COUPLE 1031 TO ZOS1 1031"
"CP COUPLE 1032 TO ZOS1 1032"
le='15'x /*CP LINE END CHARACTER */
IPLSTR='DETACH 190'||LE||'DETACH 19E'||LE||'DETACH 19D'||LE||'TERM CONMODE
3270'||LE||'IPL 300 LOADPARM 030200'
PARSE VALUE DIAG(8,iplstr) WITH STUFF

```

The LOADPARM value identified the volume containing the MVS SYS1.IPLPARM data set (0302) and the LOADxx member to be used for the IPL. Our LOADPARM in both cases was 030200, which selected SYS1.LOADPARM(LOAD00) on device 0302.

We stored these execs on each guest machine's A disk; on z/VM guest machine zos1 we began the MVS IPL by typing:

```
exec zos1
```

Similarly, on z/VM guest zos2, we began the MVS IPL by typing:

```
exec zos2
```

8.8 Preparing z/OS for sysplex operation

We began with the standard z/OS 1.4 ADCD distribution. So that we could use the standard ADCD distribution as a driver system while we built the sysplex, we cloned and renamed all of the ADCD volumes, created a new master catalog, and copied and renamed the SMP DLIB and/or target zones for the base system and installed products. This let us place the sysplex volumes on shared DASD along with a standard ADCD distribution without volume name conflicts. Describing the process of cloning the volumes and reorganizing the catalogs is beyond the scope of this paper.

So as not to confuse our work with other customization, we defined two new data sets, ZPLEX.PARMLIB and ZPLEX.PROCLIB and put them in front of the standard ADCD concatenations. When we changed a member of PROCLIB to support the sysplex, we first copied it into ZPLEX.PROCLIB from USER.PROCLIB or ADCD.ZOSV1R4.PROCLIB, as appropriate. We followed the same procedure with ZPLEX.PARMLIB.

8.9 DASD volumes

We created the following volumes before cloning the system. Our new volumes were created in UnixWare 7.1.2 raw disk slices, initialized with the FLEX-ES utility **ckdfmt**, and then initialized under the driver system with ICKDSF.

Because we were in a lab setting that standardized all volume sizes, all of our volumes were 3390 model 3s, which wasted some disk space. In a more permanent installation, we would have defined the dedicated volumes as much smaller device types. By dedicating volumes to the JES2 checkpoint data sets, we prevented JES2 reserve/release from serializing other I/O. By dedicating a volume to the primary couple data set, we reduced IOS queue lengths for the device; neither of these actions addressed contention at the level of the server RAID array, but since we were using large arrays, we never found such contention to be a problem.

Volume	Use.....
ZP4RS1	Cloned from Z4RES1.
ZP4RS2	Cloned from Z4RES2.
ZP4SYS	Cloned from OS39M1.
ZP4CK1	Dedicated to SYS1.HASPCPKT.
ZP4CK2	Dedicated to SYS1.HASPCPK2.
ZP4SMS	ZPLEX.COMMDS and the alternate sysplex couple data set; primary or alternate for the following couple data sets: ARM, LOGR, OMVS, SFM and WLM.
ZP4CDS	Dedicated to the primary sysplex couple data set (SYS1.ZPLEX.CDS01).
ZP4LOG	Primary or alternate for the following couple data sets: ARM, LOGR, OMVS, SFM, and WLM.
ZP4PG0	System zos1's page data sets (common, local, PLPA).

ZP4PG1	System zos2's page data sets (common, local, PLPA).
ZP4WK1--ZP4WK3	Work packs.
ZP4US1	HFS mountable file systems; sysplex root and system roots for both zos1 and zos2.
ZP4SP1	JES2 spool.
ZP4SP2	JES2 spool.
ZP4DS1--ZP4DS4	Clones of Z4DIS1-Z4DIS4 (we also renamed the zone and CS data sets).
ZP4DB2	Clone of Z4DB21.
ZP4CIC	Clone of Z4CIC1.
ZP4IMS	Clone of Z4IMS1.
AOTS01	Storage volume (TSO user data sets).

8.10 System and sysplex naming

The ADCD distribution ships with a sysplex name of *ADCDPL* and a member name of *P390*. The SMF system id is *SYS1*. The introduction of more than one system to a sysplex requires using at least one new system name. The sysplex could have been built by keeping the *ADCDPL* sysplex name and using one of the other system names defined in *SYS1.SCDS* for the second (and subsequent) members. We would have only needed to set the SMF system id to a unique value for each member (since it becomes the JES2 system name by default).

We chose to create our own naming scheme, even though the ADCD-supplied names would have been fine. We named the sysplex *ZPLEX* and we set the member name, SMF system id, and JES2 node names to *zos1*, *zos2*, *zos3*, and so forth. In order to change the names, we first added the new names to *SYS1.SCDS* using the ISMF M.2 dialogs under ISPF. We added a new System Group Name, *ZPLEX*, and new system names (*zos0* through *zos3*). We added the other names (besides *zos1* and *zos2*) for other testing. The following illustrates the ISMF *system names* panel after these actions:

```

                                SCDS BASE ALTER                                Page 2 of 2
Command ==>

SCDS Name . : SYS1.SCDS
SCDS Status : VALID
Specify one of the following options . .      (1 Add, 2 Delete,3 Rename)

Specify System Name . . . . .                or Sys Group Name . .

New System/Sys Group Name . .                (For option 3,Rename)

System: P390      ZOS0      ZOS1      ZOS2      ZOS3
Sysgrp: ZPLEX

```

We created a new *COMMDS* and *ACDS* using the following job and then used the *SETSMS* operator command to activate the changed *SCDS* and the new *ACDS* and *COMMDS*.

```

//FLEXSMS JOB 0,FLEX,CLASS=Z,MSGCLASS=A
//S EXEC PGM=IDCAMS,REGION=50M
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
DEFINE CLUSTER( NAME( ZPLEX.ACDS ) LINEAR VOL(ZP4LOG) -
CYL( 2 2 ) SHAREOPTIONS( 3,3 ) ) -
DATA( NAME( ZPLEX.ACDS.DATA ) )

DEFINE CLUSTER( NAME( ZPLEX.COMMDS ) LINEAR VOL(ZP4LOG) -
CYL( 2 2 ) SHAREOPTIONS( 3,3 ) ) -
DATA( NAME( ZPLEX.COMMDS.DATA ) )
/*

```

```
//
```

We created ZPLEX.PARMLIB(IGDSMS00) so that the changed data sets would be activated on subsequent IPLs:

```
SMS ACDS(ZPLEX.ACDS)
COMMDS(ZPLEX.COMMDS)
RLSINIT(NO)
INTERVAL(15)
DINTERVAL(150)
REVERIFY(NO)
ACSDEFAULTS(NO)
TRACE(ON)
SIZE(128K)
TYPE(ALL)
JOBNAME(*)
ASID(*)
SELECT(ALL)
```

8.11 Setting the system names at IPL

Both zos1 and zos2 IPLed from the same volumes and used the same PARMLIB and PROCLIB data sets. System and sysplex names were set using system variables. The process began when we specified the LOADPARM parameter at IPL. We used a LOADPARM suffix of 00 for both systems.

SYS1.IPLPARM(LOAD00) contained:

```
*---+---1---+---2---+---3---+---4---+---5---+---6---+---7-
IODF    07 SYS1    ZOS      10
SYSCAT   ZP4SYS113CCATALOG.ZP4.MASTER
SYSPARM  00
IEASYM   00
PARMLIB  ZPLEX.PARMLIB                                ZP4RS1
PARMLIB  ADCD.ZOSV1R4.PARMLIB                          ZP4RS1
PARMLIB  SYS1.PARMLIB                                  ZP4RS1
NUCLEUS  1
SYSPLEX  ZPLEX
VMUSERID ZOS1
IEASYM   01
VMUSERID ZOS2
IEASYM   02
```

This member accomplishes many things. Ours defined the IODF (SYS1.IODF07), the master catalog (CATALOG.ZP4.MASTER), the PARMLIB concatenation, and the name of the sysplex (ZPLEX). It selected member IEASYS00 of PARMLIB for system parameters, which was found in data set ZPLEX.PARMLIB because that data set was first in the concatenation.

We used two VMUSERID filters to differentiate the systems. When the guest machine with z/VM user ID zos1 was initialized, the filter selected member IEASYM01 in ZPLEX.PARMLIB. When guest zos2 was initialized, the filter selected member IEASYM02 in ZPLEX.PARMLIB. See the description of the LOADnn member in SA22- 7592-03, *MVS Initialization and Tuning Reference*, for more information. The IEASYMnn members were then used to set the system variables that distinguished one MVS image from the other.

The line with *IEASYM 01* selected ZPLEX.PARMLIB(IEASYM01), where we set the system variables that distinguished that system:

```
SYSDEF SYSNAME(ZOS1)
```

```

SYSCONE(01)
SYMDEF(&UNIXVER='V1R4M0')
SYMDEF(&SYSR2.='ZP4RS2')
SYMDEF(&SYSSYS.='ZP4SYS')

```

SYSNAME named the system. We used the SYSCONE number when we needed a numeric index for the systems. UNIXVER was used to mount the proper HFS for UNIX System Services. SYSR2 was the second residence volume and SYSSYS named the volume containing the master catalog and SYS1.IPLPARM (this volume was similar to OS39M1 in the standard ADCD volume set).

Our IEASYM02 member contained:

```

SYSDEF SYSNAME(ZOS2)
SYSCONE(02)
SYMDEF(&UNIXVER='V1R4M0')
SYMDEF(&SYSR2.='ZP4RS2')
SYMDEF(&SYSSYS.='ZP4SYS')

```

We were thus able to set the system ID elsewhere in PARMLIB and PROCLIB using the &SYSNAME variable. For example, we set the SMF system ID and we give each system its own SMF data sets this way in SMFPRM00:

```

ACTIVE
DSNAME(ZPLEX.&SYSNAME..MAN1A,
      ZPLEX.&SYSNAME..MAN1B,
      ZPLEX.&SYSNAME..MAN1C,
      ZPLEX.&SYSNAME..MAN1D,
      ZPLEX.&SYSNAME..MAN1E,
      ZPLEX.&SYSNAME..MAN2A )

NOPROMPT
REC(PERM)
MAXDORM(3000)
STATUS(010000)
JWT(1200)
SID(&SYSNAME)
LISTDSN
SYS( NOTYPE(14:17,19,62:69),
    EXITS(IEFU83,IEFU84,IEFACTRT,IEFUSI,IEFUJI,IEFU29),
    NOINTERVAL,
    NODETAIL )
SUBSYS(STC,EXITS(IEFU29,IEFU83,IEFU84,IEFUJP,IEFUS0))

```

This scheme allowed us to have separate SMF data sets for each system, which was required if we wanted to collect SMF data. Note that JES2 used the SMF system ID as the MAS member name because no other name was given in JES2PARM.

8.11.1 Other IPL parameters

Several other PARMLIB members needed minor adjustments.

IEASYS00 member of PARMLIB

All systems were IPLed with the same, default, member of ZPLEX.PARMLIB:

```

CLOCK=00,
CMB=(UNITR,COMM,GRAPH,CHDR),
CMD=00,
CON=(00,NOJES3),
COUPLE=&SYSCONE,
CSA=(3000,40000),

```

CVIO,	
DIAG=00,	
DUMP=DASD,	
FIX=00,	
GRS=TRYJOIN,	
GRSCNF=00,	
GRSRNL=00,	
ILMMODE=NONE,	
IOS=00,	
LNKAUTH=LNKLST,	
LOGCLS=L,	
LOGLMT=999999,	
LOGREC=SYS1.&SYSNAME..LOGREC,	
LPA=00,	
MAXUSER=250,	
MLPA=00,	
MSTRJCL=00,	
OMVS=00,	
OPI=YES,	
PAGTOTL=(12),	
PAGE=(SYS1.&SYSNAME..PLPA.PAGE,	
SYS1.&SYSNAME..COMMON.PAGE,	
SYS1.&SYSNAME..LOCAL.PAGE,L),	
PAK=00,	
PROD=00,	
PROG=00,	
PLEXCFG=MULTISYSTEM,	
REAL=128,	ALLOWS 2 64K OR 1 128K JOB TO RUN V=R
RSU=0,	NO RECONFIG STORAGE UNITS
RSVNONR=100,	RESERVED ASVT ENTRIES
RSVSTRT=5,	RESERVED ASVT ENTRIES
SCH=00,	SELECT SCHED00
SMF=00,	SELECT SMFPRM00,SMF PARAMETERS
SMS=00,	
SQA=(15,64),	SQA SIZE APPROX 640K
SSN=00,	
SVC=00,	
VAL=00,	SELECT VATLST00
VIODSN=SYS1.&SYSNAME..STGINDEX,	VIO DS
VRREGN=64	DEFAULT REAL-STORAGE REGION SIZE

Most of the items in the parameter list applied equally to all images; some of them, however, were customized for sysplex operation or on a per-image basis:

- ▶ **COUPLE=&SYSCClone** selected one of the COUPLEnn members of PARMLIB, depending upon which system was initializing. The &SYSCClone variable was set by the selected IEASYMnn member of PARMLIB. When we IPLed as z/VM user zos1, member IEASYM01 set &SYSCClone=01, this parameter became COUPLE=01, and member COUPLE01 was selected for the IPL. When we IPLed with z/VM user ID zos2, then member IEASYM02 set &SYSCClone=02, this parameter became COUPLE=02, and member COUPLE02 was selected for the IPL. We needed to select different COUPLE members so that the PATHIN and PATHOUT statements that identify the CTC connections were correct for the initializing member.
- ▶ **GRS=TRYJOIN** told the Global Resource Serialization component to attempt to join a GRS ring at IPL if one already was running. If no GRS complex existed, then the IPLing system started the GRS complex. GRS is required for base sysplex. Systems communicated using the channel-to-channel adapters defined in the COUPLEnn member of PARMLIB.

- ▶ LOGREC=SYS1.&SYSNAME..LOGREC selected a different LOGREC data set for each IPLing system. We of course needed to pre-allocate and initialize the data sets. When system zos1 was IPLing, for example, the LOGREC data set selected was SYS1.ZOS1.LOGREC.
- ▶ The PAGE=parameter selected different page data sets for each system in the same manner.
- ▶ PLEXCFG=MULTISYSTEM specified that the initializing system wanted to belong to a sysplex composed of one or more MVS systems that all used the same sysplex couple data sets.
- ▶ VIODSN=SYS1.&SYSNAME..STGINDEX named a separate VIO journal data set for each system.

CLOCK00 member of PARMLIB

The SIMETRID statement informed z/OS that the sysplex member was initializing as a VM guest, as an LPAR, or as a combination of the two on a single CPC. In such a situation, MVS used the TOD clock of the CPC (here, the FLEX-ES instance's TOD clock) as the point of synchronization. Our one CLOCK00 member (used by all systems) looked like this:

```

OPERATOR NOPROMPT
TIMEZONE W.05.00.00
SIMETRID 00
ETRZONE NO
ETRDELTA 10

```

Each image that initialized with the same SIMETRID value satisfied the requirement for a timing facility. Note that we were not allowed to specify ETRZONE, ETRMODE, or OPERATOR PROMPT because we specified SIMETRID. We set our server clock to UTC+0 (GMT) and left it that way. The TIMEZONE statement corrected our TOD clock to Eastern Standard Time.

CONSOL00 member of PARMLIB

We defined a unique console for each MVS system. In member CONSOL00, we defined the address of the master console. We chose this method (and we set the console name as well) to make it more readily apparent which system we were talking to when using any of the master consoles. This is an extract in which we defined a master console:

```

CONSOLE
DEVNUM(FC&SYSCLONE.)
ALTERNATE(700 )
NAME(CONS&SYSCLONE.)
AREA(25)
AUTH(MASTER)
CON(N)
DEL(RD)
MFORM(S)
PFKTAB(PFKTAB1)
RNUM(24)
ROUTCODE(ALL)
RTME(1)
SEG(24)
UNIT(3270-X)

```

Thus, the master console address for zos1 became FC01 and the master console address for zos2 became FC02; their names were CONS01 and CONS02, respectively. The operator command area on zos1 appeared as follows:

```

IEE612I CN=CONS01 DEVNUM=FC01 SYS=ZOS1 CMDSYS=ZOS1
IEE163I MODE=RD

```

BPXPRM00 member of PARMLIB

This member mounts the HFS data sets for UNIX System Services. We discuss how we created and initialized the HFS data sets for each system in “Setting up OMVS” on page 124. Our mount statements for the system-specific file systems were:

```
ROOT      FILESYSTEM('HFS.&SYSPLEX..ROOT')
          TYPE(HFS)
          MODE(RDWR)

MOUNT      FILESYSTEM('HFS.&SYSNAME..HFS')
          TYPE(HFS)
          MODE(RDWR) NOAUTOMOVE
          MOUNTPoint('/&SYSNAME.')

MOUNT      FILESYSTEM('HFS.&UNIXVER..VERSION.HFS')
          TYPE(HFS)
          MODE(READ)
          MOUNTPoint('/$VERSION')

MOUNT      FILESYSTEM('HFS.&SYSNAME..DEV')
          TYPE(HFS)
          MODE(RDWR) NOAUTOMOVE
          MOUNTPoint('/&SYSNAME./dev')

MOUNT      FILESYSTEM('HFS.&SYSNAME..VAR')
          TYPE(HFS)
          MODE(RDWR) NOAUTOMOVE
          MOUNTPoint('/&SYSNAME./var')

MOUNT      FILESYSTEM('HFS.&SYSNAME..ETC')
          TYPE(HFS)
          MODE(RDWR) NOAUTOMOVE
          MOUNTPoint('/&SYSNAME./etc')

MOUNT      FILESYSTEM('HFS.&SYSNAME..TMP')
          TYPE(HFS)
          MODE(RDWR) NOAUTOMOVE
          MOUNTPoint('/&SYSNAME./tmp')

MOUNT      FILESYSTEM('HFS.USER.MAIL')
          TYPE(HFS)
          MODE(RDWR)
          MOUNTPoint('/usr/mail')

MOUNT      FILESYSTEM('HFS.USERS')
          TYPE(HFS)
          MODE(RDWR)
          MOUNTPoint('/u')
```

There was one sysplex root data set, HFS.ZPLEX.ROOT. It contained the mount points for each member's root file system. When zos1 initialized, its root file system (HFS.ZOS1.HFS) was mounted in the sysplex root at mount point /ZOS1 (which is actually specified in the sysplex root file system with a variable name, \$SYSNAME). The correct DEV, VAR, ETC, and TMP file systems (unique to each system) were thus mounted on the correct system root and appeared to users of that system as though they were mounted relative to /. Each shared file system was mounted on the same mount point name--and the result was that each system mounted the shared file systems in its own root.

COMMND00 member of PARMLIB

Several lines in this member were customized by using variables on a per-system basis:

```
COM='S JES2,PARM='WARM,NOREQ''
COM='S VLF,SUB=MSTR,NN=&SYSCONE'
COM='S NET'
COM='S VTAMAPPL'
COM='S DLF,SUB=MSTR'
COM='DD ADD,VOL=&SYSSYS'
COM='DD NAME=SYS1.&SYSNAME..DMP&SEQ'
COM='DD ALLOC=ACTIVE'
COM='S EZAZSSI,P=&SYSNAME'
```

IECIOS00 member of PARMLIB

We set the DASD missing interrupt threshold to ten minutes after seeing some contention problems with the default value of 15 seconds:

```
MIH DASD=10:00
MIH TAPE=20:00
MIH IOTDASD=00:00
```

JES2PARM member of PARMLIB

We did some customization that suited our own needs (initiators, class defaults, printers, and so forth), and we changed the parameters that name the spool volumes and that locate and name the checkpoint data sets.

Specific volume references

Because we had cloned the system and changed the volume names, we had to make changes to other PARMLIB and PROCLIB members that explicitly name volumes. These included LPALST00, PROG00, and VATLST00 in PARMLIB and DBSPROC, ISPFPROC, and OMVSPROC in PROCLIB.

MSTJCL00 member of PARMLIB

We put ZPLEX.PROCLIB at the beginning of the IEFPDSI concatenation. All we can say here is that extra caution is warranted; this is not a good place to have a JCL error--the system will not start. We always had the ADCD driver system available to fix any mistakes we made, however. Our MSTJCL00 member was:

```
//MSTJCL00 JOB MSGLEVEL=(1,1),TIME=1440
//          EXEC PGM=IEEMB860,DPRTY=(15,15)
//STCINRDR DD SYSOUT=(A,INTRDR)
//TSOINRDR DD SYSOUT=(A,INTRDR)
//IEFPDSI  DD DSN=ZPLEX.PROCLIB,DISP=SHR
//          DD DSN=ADCD.ZOSV1R4.PROCLIB,DISP=SHR
//          DD DSN=SYS1.PROCLIB,DISP=SHR
//SYSUADS  DD DSN=SYS1.UADS,DISP=SHR
//SYSLBC   DD DSN=SYS1.BROADCAST,DISP=SHR
```

8.11.2 PROCLIB changes

We needed relatively few changes to PROCLIB. We modified the JES2 procedure to include ZPLEX.PROCLIB at the beginning of the PROC00 concatenation. We modified the TSO procedures DBSPROC, ISPFPROC, and OMVSPROC because they reference many data sets by explicit volume serial number.

We modified the TCPIP, FTPD, NFSC, and NFSS procedures because we created a separate TCPIP profile data set for each system. For example, here is our procedure for TCPIP:

```
//TCPIP EXEC PGM=EZBTCPIP,REGION=0M,TIME=1440,
// PARM='&PARMS'
//STEPLIB DD DSN=ADCD.ZOSV1R4.VTAMLIB,DISP=SHR
//SYSPRINT DD SYSOUT=*,DCB=(RECFM=VB,LRECL=132,BLKSIZE=136)
//ALGPRINT DD SYSOUT=*,DCB=(RECFM=VB,LRECL=132,BLKSIZE=136)
//CFGPRINT DD SYSOUT=*,DCB=(RECFM=VB,LRECL=132,BLKSIZE=136)
//SYSOUT DD SYSOUT=*,DCB=(RECFM=VB,LRECL=132,BLKSIZE=136)
//CEEDUMP DD SYSOUT=*,DCB=(RECFM=VB,LRECL=132,BLKSIZE=136)
//SYSERROR DD SYSOUT=*
//PROFILE DD DISP=SHR,DSN=TCPIP.&SYSNAME..PROFILE.TCPIP
//SYSTCPD DD DSN=TCPIP.&SYSNAME..TCPIP.DATA,DISP=SHR
```

With this method, we selected TCPIP.ZOS1.PROFILE.TCPIP and TCPIP.ZOS1.TCPIP.DATA when zos1 was IPLed.

8.12 Couple data set preparation

A primary couple data set is required for the sysplex to operate; several other couple data sets must be defined for various purposes. Some are required and others are optional. We describe only the required ones.

Creating the couple data sets is described in detail in *MVS: Setting up a Sysplex*, SA22-7625-05. Each couple data set contains the name of the sysplex; for this reason, since we changed the name of our sysplex, we could not use the data sets in the ADCD distribution. The procedure to create new data sets was very straightforward, however.

Sysplex couple data set

To create the primary couple data set and a backup, we used this job:

```
//COUPLEDS JOB ('0'),'FLEX',CLASS=Z,MSGCLASS=A
//S1 EXEC PGM=IDCAMS,REGION=30M
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
DELETE SYS1.ZPLEX.CDS01
DELETE SYS1.ZPLEX.CDS02
SET MAXCC=0
/*
//S EXEC PGM=IXCL1DSU,REGION=30M
//STEPLIB DD DSN=SYS1.MIGLIB,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
DEFINEDS SYSPLEX(ZPLEX)
DSN(SYS1.ZPLEX.CDS01) VOLSER(ZP4CDS)
MAXSYSTEM(8)
CATALOG
DATA TYPE(SYSPLEX)
ITEM NAME(GROUP) NUMBER(32)
ITEM NAME(MEMBER) NUMBER(64)
ITEM NAME(GRS) NUMBER(1)
DEFINEDS SYSPLEX(ZPLEX)
DSN(SYS1.ZPLEX.CDS02) VOLSER(ZP4SMS)
MAXSYSTEM(8)
CATALOG
DATA TYPE(SYSPLEX)
ITEM NAME(GROUP) NUMBER(32)
```

```

ITEM NAME(MEMBER) NUMBER(64)
ITEM NAME(GRS)      NUMBER(1)

/*
//

```

Couple data sets are created with the IXCL1DSU utility as described in *MVS: Setting up a Sysplex*. A base sysplex may or may not use the ARM, SFM, and LOGR data sets to good advantage. The SYSPLEX, WLM, and OMVS data sets are mandatory for the system to function properly, however.

COUPLEnn members of PARMLIB

Couple data sets are cataloged in the master catalog and are specified in the COUPLEnn member of PARMLIB selected for a given IPL. Our COUPLEnn members were identical with the exception of the PATHIN and PATHOUT statements. In COUPLE01, used by zos1 (shown below), we specified PATHIN of 1032 and PATHOUT of 1031; in COUPLE02, we reversed the specification, coding PATHIN of 1031 and PATHOUT of 1032.

```

COUPLE SYSPLEX(&SYSPLEX.)
    INTERVAL(85)
    PCOUPLE(SYS1.&SYSPLEX..CDS01)
    ACOUPLE(SYS1.&SYSPLEX..CDS02)
DATA TYPE(LOGR)
    PCOUPLE(SYS1.&SYSPLEX..LOGR.CDS01)
    ACOUPLE(SYS1.&SYSPLEX..LOGR.CDS02)
DATA TYPE(BPXMCD)
    PCOUPLE(SYS1.&SYSPLEX..OMVS.CDS01)
    ACOUPLE(SYS1.&SYSPLEX..OMVS.CDS02)
DATA TYPE(WLM)
    PCOUPLE(SYS1.&SYSPLEX..WLM.CDS01)
    ACOUPLE(SYS1.&SYSPLEX..WLM.CDS02)
DATA TYPE(SFM)
    PCOUPLE(SYS1.&SYSPLEX..SFM.CDS01)
    ACOUPLE(SYS1.&SYSPLEX..SFM.CDS02)
DATA TYPE(ARM)
    PCOUPLE(SYS1.&SYSPLEX..ARM.CDS01)
    ACOUPLE(SYS1.&SYSPLEX..ARM.CDS02)
PATHIN  DEVICE(1032) MAXMSG(10240) RETRY(15)
PATHOUT DEVICE(1031) MAXMSG(10240) RETRY(15)
LOCALMSG MAXMSG(8192)

```

IBM recommends that each class of couple data sets have both a primary and an alternate, and that's how we did it. The SYSPLEX primary and alternate are used by XCF to store information about the systems in the sysplex, the members running in the sysplex, and status information. The OMVS couple data sets must be defined before UNIX System Services can properly share file systems among the members of a sysplex; without it, UNIX Systems Services sessions will fail. The WLM couple data set is used to store the Workload Manager service definitions. See *MVS Planning: Workload Management*, SA22-7602-05, for more information.

Note that the couple data set defined the inbound and outbound CTC connections. The systems automatically contacted each other on these connections; no other information was required beyond the device numbers.

OMVS couple data set

We created a new OMVS couple data set with this job:

```

//OMVSDS  JOB ('0'),FLEX,CLASS=Z,MSGCLASS=A
//S       EXEC PGM=IXCL1DSU,REGION=30M
//STEPLIB DD DSN=SYS1.MIGLIB,DISP=SHR

```

```

//SYSPRINT DD SYSOUT=*
//SYSIN DD *
  DEFINEDS SYSPLEX(ZPLEX)
    DSN(SYS1.ZPLEX.OMVS.CDS01) VOLSER(ZP4SMS)
    CATALOG
    MAXSYSTEM(8)
  DATA TYPE(BPXMCD)
    ITEM NAME(MOUNTS) NUMBER(500)
    ITEM NAME(AMTRULES) NUMBER(50)
  DEFINEDS SYSPLEX(ZPLEX)
    DSN(SYS1.ZPLEX.OMVS.CDS02) VOLSER(ZP4LOG)
    CATALOG
    MAXSYSTEM(8)
  DATA TYPE(BPXMCD)
    ITEM NAME(MOUNTS) NUMBER(500)
    ITEM NAME(AMTRULES) NUMBER(50)
/*
//

```

WLM couple data set

We created the WLM couple data set with this job:

```

//WLMD S JOB ('0'),FLEX,CLASS=Z,MSGCLASS=A
//S EXEC PGM=IXCL1DSU,REGION=30M
//STEPLIB DD DSN=SYS1.MIGLIB,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
  DEFINEDS SYSPLEX(ZPLEX)
    DSN(SYS1.ZPLEX.WLM.CDS01) VOLSER(ZP4LOG)
    CATALOG
  DATA TYPE(WLM)
    ITEM NAME(POLICY) NUMBER(10)
    ITEM NAME(WORKLOAD) NUMBER(35)
    ITEM NAME(SRVCLASS) NUMBER(30)
    ITEM NAME(SVDEFEXT) NUMBER(5)
    ITEM NAME(SVDCREXT) NUMBER(5)
    ITEM NAME(APPLENV) NUMBER(100)
    ITEM NAME(SVAEAEXT) NUMBER(5)
    ITEM NAME(SCHENV) NUMBER(100)
    ITEM NAME(SVSEAEXT) NUMBER(5)
  DEFINEDS SYSPLEX(ZPLEX)
    DSN(SYS1.ZPLEX.WLM.CDS02) VOLSER(ZP4SMS)
    MAXSYSTEM(8)
    CATALOG
  DATA TYPE(WLM)
    ITEM NAME(POLICY) NUMBER(10)
    ITEM NAME(WORKLOAD) NUMBER(35)
    ITEM NAME(SRVCLASS) NUMBER(30)
    ITEM NAME(SVDEFEXT) NUMBER(5)
    ITEM NAME(SVDCREXT) NUMBER(5)
    ITEM NAME(APPLENV) NUMBER(100)
    ITEM NAME(SVAEAEXT) NUMBER(5)
    ITEM NAME(SCHENV) NUMBER(100)
    ITEM NAME(SVSEAEXT) NUMBER(5)
/*

```

8.13 VTAM customization

We created separate VTAMLST data sets for each system, and we used the &SYSNAME variable to select ZPLEX.&SYSNAME..VTAMLST. Some customization was required for VTAM to operate properly in the configuration.

Although the topic of defining the network is beyond the scope of this discussion, we offer a few notes about how we configured VTAM. For TSO access to members of the sysplex, there were two possible approaches (both could be used simultaneously).

Locally-attached non-SNA terminals

The first method defined locally-attached, non-SNA 3278 terminals for each system. Within z/OS, we defined 31 non-SNA 3278s starting at address 0701; these were defined to VTAM as well as in the IODF.

We then defined a few of these in the z/VM user directory for each guest machine. We could have coded SPECIAL statements for all 31, had we chosen to do so.

```
SPECIAL 701 3270
SPECIAL 702 3270
SPECIAL 703 3270
SPECIAL 704 3270
```

To connect to a TSO session on *zos1*, we connected to one of the z/VM terminals defined in the FLEX-ES resource definitions and thus offered on the FLEX-ES terminal solicitor screen. These are the terminals marked *zvm-n*:

```
Welcome to the FLEX-ES Terminal Solicitor (node:x235)
Please select (X) the desired service and press enter
(PA1 to exit; CLEAR to refresh)
```

_ zvm-3	_ zvm-9	_ zvm-5	_ zvm-7
_ zvm-10	_ zvm-15	_ zvm-14	_ zvm-13
_ zvm-6	_ zvm-19	_ zvm-16	_ zvm-11
_ zvm-12	_ zvm-17	_ zvm-8	_ zvm-1f
_ zvm-1a	_ zvm-1e	_ zvm-18	_ zvm-1c
_ zvm-1d	_ zvm-1b	_ x082	_ x087
_ x083	_ x086	_ x084	_ x085

Once we had the z/VM logo, we used the VM **dial** command to connect to a TSO session:

```
dial zos1
```

This connected us to the next available terminal. Once we had the VTAM logo on the screen, we could log on to *zos1* by issuing a **logon** command, or we could request to logon to *zos2* as follows:

```
L zos2tso
```

We could similarly dial to *zos2* and log onto either system from there. The name *zos2tso* is the TSO APPLID we defined in VTAMLST.

RTP using CTCS

We also set up VTAM is to allow RTP (rapid-transit protocol) resources to be dynamically defined by VTAM as needed. For this to function, we had to modify the ATCSTR00 member of each system's VTAMLST to define unique values for SSCPID, HOSTSA, SSCPNAME, and HOSTPU. We needed a common NETID. Other members of VTAMLST were customized to provide unique IMS, CICS, DB2, and TSO APPLIDS; the local, non-SNA terminals also had to have unique names.

Once all that was done, we could use RTP to access any member of the sysplex from any other system on our network that was defined to the Cross Domain Resource Manager. Only system zos1 needed to be connected to the VTAM network and defined in the CDRM major nodes. See “SNA over Ethernet” on page 153 for a complete description of the VTAM parameters used for such a configuration. The following messages issued during an IPL of the sysplex confirmed that the two SSCP’s were correctly configured to use RTP:

```
ISG011I SYSTEM ZOS2 -JOINING GRS COMPLEX
ISG004I GRS COMPLEX JOINED BY SYSTEM ZOS2
IST1086I APPN CONNECTION FOR FLEX.ZOS2SSCP IS ACTIVE -TGN =21
IST093I ISTP0102 ACTIVE
IST1488I ACTIVATION OF RTP CNR00002 AS PASSIVE TO FLEX.ZOS2SSCP
IST1488I ACTIVATION OF RTP CNR00001 AS ACTIVE TO FLEX.ZOS2SSCP
IST1096I CP-CP SESSIONS WITH FLEX.ZOS2SSCP ACTIVATED
```

TSO logon restrictions

The sysplex enforces session limits sysplex-wide; that is, a TSO user may only log on to one system at a time.

8.14 Setting up OMVS

Setting up UNIX System Services for sysplex operation consists of several steps; we have already shown how we used system variables to mount the sysplex root, system root, and system-specific files systems in member BPXPRM00.

We created these files using DFSMS; we first cloned the ADCD volume Z4USS1 following the recommended procedure for dumping and restoring HFS files; we then duplicated and renamed the system-specific files. Finally, we created the new sysplex root (there is only one); we created a unique system root for each member of the sysplex.

Cloning the HFS volume

The ADCD distribution ships all Hierarchical File System (HFS) files on one volume, Z4USS1. To begin the process, we copied the files to a new volume, ZP4US1, using this job:

```
//COPYUS2 JOB 0,FLEX,CLASS=Z,MSGCLASS=A,TIME=720,REGION=30M
//DSS EXEC PGM=ADRDSSU
//SYSPRINT DD SYSOUT=*
//Z4USS1 DD VOL=SER=Z4USS1,UNIT=3390,DISP=SHR
//ZP4US1 DD VOL=SER=ZP4US1,UNIT=3390,DISP=SHR
//OUTDD1 DD DSN=&HFSDUMP,
// DCB=(BLKSIZE=32760),
// DISP=(NEW,DELETE,DELETE),
// SPACE=(CYL,(950,550)),
// UNIT=SYSALLDA
//SYSIN DD *
DUMP -
ALLDATA(*) -
ALLEXCP -
CANCELERROR -
COMPRESS -
DATASET( -
INCLUDE( HFS.*,HFS.** ) -
EXCLUDE( SYS1.**, SYS1.*, HFS.ADCPL.ROOT )) -
LOGINDDNAME(Z4USS1) -
OPTIMIZE(4) -
OUTDDNAME(OUTDD1) -
TOLERATE(ENQF) -
```

```

        WAIT(2,2)

SERIAL

RESTORE -
  DATASET( -
    INCLUDE(**) -
    INDDNAME(OUTDD1) -
    TOLERATE(ENQF) -
    OUTDDNAME(ZP4US1) -
    RECATALOG( CATALOG.ZP4.VZP4RS2 ) -
    RENAMEU( HFS.P390.* ,HFS.ZOS1.* ) -
    STORCLAS(DEFAULT)
  /*

```

In the process, the system name portion of the system-specific file systems was changed to ZOS1. Thus, HFS.P390.DEV became HFS.ZOS1.DEV, and so forth. We excluded the sysplex root (ADCDPL.ROOT) because we intended to create a new one.

We used a similar job to create a new set of system-specific files systems by copying the existing ones:

```

//CLONUSS  JOB 0,FLEX,CLASS=Z,MSGCLASS=A,TIME=720,REGION=30M
//DSS      EXEC PGM=ADDRSSU
//SYSPRINT DD SYSOUT=*
//ZP4US1   DD VOL=SER=ZP4US1,UNIT=3390,DISP=SHR
//OUTDD1   DD DSN=&HFSDUMP,
//          DCB=(BLKSIZE=32760),
//          DISP=(NEW,DELETE,DELETE),
//          SPACE=(CYL,(950,550)),
//          UNIT=SYSALLDA
//SYSIN    DD *
DUMP -
  ALLDATA(*) -
  ALLEXCP -
  CANCELERROR -
  COMPRESS -
  DATASET( -
    INCLUDE( HFS.ZOS1.* ) -
    LOGINDDNAME(ZP4US1) -
    OPTIMIZE(4) -
    OUTDDNAME(OUTDD1) -
    TOLERATE(ENQF) -
    WAIT(2,2)

SERIAL

RESTORE -
  DATASET( -
    INCLUDE(**) -
    INDDNAME(OUTDD1) -
    TOLERATE(ENQF) -
    OUTDDNAME(ZP4US1) -
    RECATALOG(CATALOG.ZP4.VZP4RS2) -
    RENAMEU( HFS.ZOS1.*, HFS.ZOS2.* ) -
    STORCLAS(DEFAULT)
  /*

```

Creating the sysplex root file system and new system-specific root file systems proved to be a little tricky. When we ran the jobs provided in SYS1.SAMPLIB, we got an error because the

/tmp directory could not be accessed. We worked out the following procedure to create the file systems.

First, we used the **ish** (interactive shell) program (a menu-driven interface into the HFS file systems) to create a new, empty file system of five cylinders in data set TEMPNAME. We then used **ish** to dismount all file systems. Once all the file systems were dismounted, we exited the interactive shell and returned to the TSO READY prompt.

We then used the TSO **mount** command to mount TEMPNAME at mount point / (root). We then created the /tmp directory using the TSO **mkdir** command:

```
READY
mount
IKJ56700A ENTER FILE SYSTEM -
tempname
IKJ56700A ENTER MOUNT POINT -
'/'
IKJ56700A ENTER TYPE -
hfs
READY
mkdir '/tmp '
READY
```

With our small file system mounted as root and with the /tmp directory defined, we tried the jobs again. This time they ran successfully. We found that the job that creates a system-specific root must be run on the system that will use it. To create the zos2 root, for example, we used /*JOBPARM SYSAFF ZOS2 to ensure that an initiator on zos2 ran the job. Once the three jobs were complete (one for the sysplex root, and one for each of the zos1 and zos2 system roots), we IPLed both systems and the proper mounts were made by member BPXPRM00 of PARMLIB.

This is the job we ran to create the one new sysplex root:

```
//BPXISYSR JOB 0,FLEX,CLASS=Z,MSGCLASS=A
//IDCAMS EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
DELETE HFS.ZPLEX.ROOT NONVSAM SCRATCH -
CATALOG(CATALOG.ZP4.MASTER )
SET MAXCC=0
/*
//IKJEFT1A EXEC PGM=IKJEFT1A,PARM='BPXISYS1',REGION=30M
/*
//ROOTSYSP DD DSN=HFS.ZPLEX.ROOT,
// DISP=(,CATLG),
// DSNTYPE=HFS,
// SPACE=(CYL,(2,0,1)),
// UNIT=3390,VOL=SER=ZP4US1
//SYSEXEC DD DSN=SYS1.SAMPLIB,DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD DUMMY
//
```

This is the job we ran once for each system-specific root file system:

```
//BPXISYS JOB 0,FLEX,CLASS=Z,MSGCLASS=A,REGION=30M
/*JOBPARM SYSAFF ZOS1
//SO EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
DELETE HFS.ZOS1.HFS NONVSAM SCRATCH
```

```

SET MAXCC=0
/*
//IKJEFT1A EXEC PGM=IKJEFT1A,PARM='BPXISYS2'
//HFSSYSTS DD DSN=HFS.ZOS1.HFS,
//          DISP=(,CATLG),
//          DSNTYPE=HFS,
//          SPACE=(CYL,(8,0,1)),
//          UNIT=3390,VOL=SER=ZP4US1
//SYSEXEC DD DSN=SYS1.SAMPLIB,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD DUMMY
//

```

8.15 Starting the sysplex

We set MFORM=(S) (using the **K S** command) so that we could see which system issued what messages. Once all systems cleared nucleus initialization, all messages appeared on all consoles, and we used only one of them from that point on.

Depending on the state of the systems at the last shutdown, they may all IPL without intervention or one or more of the systems may prompt the operator asking if the system should initialize or join the sysplex. We replied 'I' (for initialize) on only one system and we replied 'J' (for join) on all other systems.

All systems except the one that initialized the sysplex issued messages similar to this:

```

ISG003I SYSTEM ZOS1 ASSISTING IN GRS JOIN PROCESSING
ISG004I GRS COMPLEX JOINED BY SYSTEM ZOS2

```

8.16 Operating the sysplex

Each system's console displayed message traffic from all other systems in the sysplex. When we needed to issue a CP command, we pressed the PA1 key, which switched to the CP screen and issued a CP READ prompt. We subsequently used the CLEAR key to restore the z/OS console screen. When CP had a message that it wanted to display, it often switched screens for us. Again, a press of the CLEAR key restored the z/OS console.

The screen capture below of zos1's master console shows a mixture of messages from zos1 and zos2 (the latter was still in the IPL process):

```

- ZOS2 IEC368I - CATALOG INITIALIZATION C
- THE MULTILEVEL ALIAS FACILITY HAS BEEN INITIALIZED
- THE NUMBER OF LEVELS OF QUALIFICATION IS 1
ZOS2 IXC811I SYSTEM ZOS2 IS NOW ARM CAPABLE
- ZOS2 IVT5507I CSM PARMLIB INFORMATION FOUND IN MEMBER IVTPRM00
ZOS2 ADY012I THE FOLLOWING DAE OPTIONS ARE IN EFFECT:
START
SVCDUMP =NOTIFY(3,30)MATCH UPDATE SUPPRESSALL
SYSDUMP =MATCH UPDATE
RECORDS =400
DSN =SYS1.DAE
ZOS2 CNLS003I INITIALIZE SUCCESSFUL
ZOS2 ANTM6000I SNAPSHOT WORKING SPACE DATASETS BEING REFRESHED
ZOS2 ANTM6001I 0 SNAPSHOT WORKING SPACE DATASETS REFRESHED
- ZOS2 IXZ0001I CONNECTION TO JESXCF COMPONENT ESTABLISHED,
- GROUP SYSJES MEMBER ZOS2

```

```

ZOS2 COF511I DLF INITIALIZATION IS IN PROGRESS.
ZOS2 COF011I VLF INITIALIZATION IS IN PROGRESS.
ZOS2 COF525I DLF INITIALIZATION IS COMPLETE.
- ZOS2 IRRB001I (#)RACF SUBSYSTEM HRF7707 IS ACTIVE.
- ZOS2 IRRB002I (#)INITIALIZATION COMPLETE FOR RACF SUBSYSTEM.
ZOS2 COF025I VLF INITIALIZATION IS COMPLETE.
ZOS2 IEE855I DUMPDS COMMAND RESPONSE                                C
DUMPDS COMMAND SYS1.DUMP DATA SET STATUS
NAME PATTERN ACCEPTED:SYS1.&SYSNAME..DMP&SEQ
ZOS2 IEE855I DUMPDS COMMAND RESPONSE                                C
DUMPDS COMMAND SYS1.DUMP DATA SET STATUS
DASD VOLUMES ADDED:ZP4SYS
- ZOS1 d ts,1,1=z
ZOS1 IEE114I 14.57.45 2003.193 ACTIVITY 435                        C
JOBS M/S TS USERS SYSAS INITS ACTIVE/MAX VTAM OAS
00002 00010 00001 00030 00007 00001/00040 00010
FLEX OWT
ZOS2 IGD020I SMS IS NOW ACTIVE
ZOS2 CMP001I DFSMS COMPRESSION SERVICES AVAILABLE
00 ZOS2 ANTB8000I XRC INITIALIZATION STARTED
ZOS2 ANTB8001I XRC INITIALIZATION COMPLETED
IEE612I CN=CONS01 DEVNUM=FC01 SYS=ZOS1 CMDSYS=ZOS1
IEE163I MODE=RD

```

The systems operated as any other loosely coupled systems would operate with a few exceptions. Operator replies were honored on a sysplex-wide basis, and commands could be routed from one system to another with the MVS route command. For example, when we wanted to shut down the sysplex, we entered this command:

```
ro *all,s shutdown
```

Once we saw *ALL AVAILABLE FUNCTIONS COMPLETE* from each JES2 system, we then shut down JES2 with this command:

```
ro *all,$pjes2
```

When we saw JES2 TERMINATION COMPLETE from all JES2 systems, we could then press the PA1 key and reset the virtual machine and end our session (this needed to be done on each z/VM guest machine's console):

```
logoff
```

8.17 Tuning

We did a limited amount of system tuning, finally following the suggestion made in *MVS: Setting up a Sysplex*, SA22-7625-03, to dedicate volumes to the couple data sets. We also dedicated volumes to the JES2 checkpoint data sets--while it is possible under GRS to convert device reserves to global data set ENQs, common wisdom suggests JES2 should be allowed to use reserve/release. After looking at the reserve time JES2 accumulated on the checkpoint volume, we decided to let JES2 own it.

It made sense to group related data as closely as possible to reduce seeks and associated latency delays in the underlying RAID subsystem. We had some success, for example, by putting page data sets in contiguous space on adjacent emulated-DASD volumes. It therefore seemed that we should attempt to treat z/OS volumes as one large contiguous disk as much as possible.

Things proved not to be that simple, however. We found that putting anything on the same emulated DASD volume with the JES2 checkpoint data set(s) rendered any other data on

those volumes effectively inaccessible. Even in an emulated-DASD environment, some factors weigh in favor of dedicated volumes—with more volumes, there may be more possibilities for concurrent I/O. So the trade-off is as follows: more emulated DASD may degrade performance by spreading the data farther apart on the underlying server disk, but more emulated DASD may also improve performance by segregating processes with long reserve times and providing more concurrency.

Additional dedicated volumes and RNL changes

After we provided a dedicated checkpoint volume and two separate volumes containing couple data sets, performance improved dramatically. However, we still saw some occurrences of global resource or reserve contention. Up until this point, we had been using the default GRSRNLxx member of PARMLIB. We went through the Sysplex and GRS documentation again, and gleaned the following ideas:

- ▶ ***RNL EXCLUSION LIST.*** The exclusion list is used to convert global (sysplex-wide) ENQs into local ENQs. Because we named each system's LOGREC data set uniquely, we added their names to the exclusion list—to avoid the overhead of the systems-scope ENQ. We did the same for SMF files—although we eventually turned SMF recording off just to avoid its operational demands. We added the JES2 checkpoint data sets to the exclusion list—leaving them to use reserve/release on dedicated volumes.
- ▶ ***RNL INCLUSION LIST.*** The inclusion list is used to convert local ENQs into global ones. We elected to use one UADS and one BROADCAST data set for the entire sysplex—and so we included QNAMEs SYSIKJUA and SYSIKJBC on the RNL inclusion list along with SYSDSN. One QNAME, SYSDSN, protects all data sets in the sysplex.
- ▶ ***RNL CONVERSION LIST.*** The conversion list is used to convert specific reserves into global ENQs and releases into global DEQs. The default list is empty. Following IBM's advice, we added a few names:
 - IGDCDSXS - To help prevent deadlocks during VARY SMS processing.
 - SYSIGGV2 - This is probably the most important item in this list. This converts reserves against catalog volumes into global ENQ's, allowing access to other data sets on the volume while the catalog is reserved.
 - SYSZRACF - The RACF database.
- ▶ ***GRSCNFX CHANGES.*** Because we used virtual channel-to-channel adapters, we found it beneficial to reduce the RESMIL value from its default of 10 milliseconds to three milliseconds. The parameter is used to control the rate at which the systems pass information around the GRS ring--the system is guaranteed to wait at least three milliseconds before passing GRS token data on to the next system in the ring. We also set ACCELSYS to two—this greatly speeds global-ENQ processing when more than two systems are active. ACCELSYS sets the maximum number of systems that must acquiesce before a global-ENQ is granted. After we made these changes, we saw no further deadlocks. There is an excellent discussion of the GRS PARMLIB members in *MVS Planning: Global Resource Serialization*, SA22-7600-02.

8.18 Conclusion

Base sysplex on z/VM using FLEX-ES offers the potential to isolate workloads in a very tightly-coupled environment; major benefits—workload isolation, enhanced availability, and single-console operations—are possible.

For an xServer system, we recommend at least 4GB of RAM and a large RAID5 array dedicated to z/OS with at least nine drives in the array. Faster processors may increase the

amount of RAM or the number of RAID arrays needed to achieve the processor's full potential.

8.19 Notices

This paper cannot guarantee that the base sysplex will work under all workloads or configurations; we did not run any production workloads during our testing. The information in this document is provided *AS IS*. If you attempt to implement a base sysplex, you do so solely at your own risk.

Installing Linux for S/390

This chapter briefly describes a method for installing a basic SuSE SLES 8.0 system under FLEX-ES. There are many ways to do this and many ways to configure the resulting Linux for S/390. We describe one way that might be used as an initial introduction to this area.

Starting with a simple FLEX-ES system (our T23 ThinkPad) and the CDs for the SuSE release, we ran the complete installation process on this system. We were not connected to any external network. Our IP addresses, used throughout this installation, were:

192.168.0.110	Red Hat Linux, the base operating system on the ThinkPad
192.168.0.112	SuSE Linux for S/390 that we are installing

Our Red Hat IP address was already selected when we started. We arbitrarily selected 192.168.0.112 for SuSE Linux.

Our goal was to install SuSE (SLES 8.0) on a single 3390-3 drive. We partitioned this drive to have a separate swap partition; otherwise, everything was installed in the main (root) partition. You would probably not install a “real” Linux for S/390 this way, but this minimal installation provided a good learning experience.

Our most important advice for the installation steps we describe here is to work slowly. Recovery from an error or missed step can be complex and may result in starting all over again. The general flow of installation was this:

- ▶ Copy the installation RAMdisk image and initialization program image from SuSE CD#1 directly into S/390 memory and start the program.
- ▶ Provide installation setup information by conversing with this program through the S/390 “hardware console.” In FLEX-ES terms, this used the *flexes*> window and the CLI command **hwc**. The Red Hat system on our ThinkPad was identified as the FTP server for installing SuSE.
- ▶ The initial information starts an X server window on the Red Hat desktop, running the SuSE YaST setup tool. YaST requests more setup information. A side step, requiring a **telnet** connection to the running SuSE RAMdisk system, is needed to format and partition the target 3390 volume.
- ▶ YaST then loads basic SuSE modules onto the target 3390 volume.

- IPL from the 3390 volume. YaST resumes automatically and completes its system configuration.

9.1 FLEX-ES and Linux definitions

We decided to use a raw disk interface for the emulated 3390 volume where we will install SuSE Linux. There is no special requirement for this, other than the better performance of the raw disk interface. You could use a simple Linux file for the target 3390 volume and skip the LVM and raw interface setup if you prefer.

Raw device preparation

We based the raw interface setup on the LVM and volume group described in “Raw disk devices” on page 57. That is, we assumed LVM was set up in our Red Hat Linux and volume group `vg390` already existed. We did the following:

```
$ su                                (Need to be root)
# /sbin/vgchange -ay vg390          (Activate LVM and vg390)
# /sbin/vgdisplay vg390            (How much room left in volume group?)
...
Free PE/Size      3604 / 14.08 GB   (Lots of room available)
# /sbin/lvcreate -L 2792M -n LNX01A vg390 (Create new logical volume)
# chown flexes:flexes /dev/vg390/LNX01A (Must be owned by flexes)
# mknod /dev/raw/200 c 162 160
# chown flexes:flexes /dev/raw/200
```

We arbitrarily decided to name the logical volume `LNX01A`. The `mknod` command used major number 162 (required) and minor number 160 (arbitrary). This minor number was not used by Red Hat or by our setup described in “Raw devices” on page 61. The node name “200” was selected because we intend to use S/390 device address 200 for the 3390 drive. The node name is arbitrary, but fits with the naming convention described earlier. The `vgchange` command, to activate LVM, may not be required in Linux releases that activate LVM automatically.

Raw device shell script

We created file `shLVLNX` in our `/usr/flexes/rundir` directory, as follows:

```
/sbin/vgscan
/sbin/vgck
/sbin/vgchange -ay vg390
raw /dev/raw/200 /dev/vg390/LNX01A
echo 'Raw interface for 200 defined'
```

This shell script must be run (as root) after Red Hat Linux is booted and before the FLEX-ES instance is started. It activates LVM and binds (for the duration of this boot or until changed) the indicated logical volume to the indicated raw device node.

Create volume

After creating the shell script, we ran it and then we initialized⁶⁴ the logical volume:

```
$ su                                (Must be root to run this shell script)
# cd rundir                         (Location of shell script)
# sh shLVLNX
# exit                              (Leave root)
$ ckdFmt -n /dev/raw/200 3390-3
```

⁶⁴ If you prefer to use a simple Linux file for the volume, you can skip the “Raw device preparation” and “Raw device shell script” sections and initialize a volume with a command like `ckdFmt -n -r /s390/LNX01A 3390-3`.

This formats the logical volume into 3390 CKD form. It does not format it for Linux use.

FLEX-ES definitions

We created a S/390 definition with one disk drive and a LAN interface. No console definition is needed. The complete definition (in our file `/usr/flexes/rundir/SLinux`) was as follows:

```
system SLinux:
  memsize(262144)
  cachesize(4096)
  instset(esa)
  cpu(0)
  channel(1) local
  channel(2) local
  cu devad(0x200,1) path(1) resource(CU3990)
  cu devad(0x2E0,2) path(2) resource(CU3172)
end SLinux

resources SLinux:
CU3990: cu 3990
  interface local(1)
  options 'trackcachesize=500'
  device(00) 3990-3 /dev/raw/200 #Address 200
end CU3990
CU3172: cu 3172
  interface local(1)
  options 'ipaddress=192.168.0.112,adapternumber=0'
  device(00) 3172 eth0 #Address 2E0
  device(01) OFFLINE #Address 2E1
end CU3172
end SLinux
```

This creates a S/390 instance having 256 MB memory, one 3390 disk drive at addresses 200, and a 3172 LCS device at addresses 2E0/2E1. We compiled the definitions with **cfcomp**, creating `SLinux.syscf` and `SLinux.rescf`.

This is a rather basic FLEX-ES definition. For more information about similar definitions, see the *Getting Started* redbook and/or “FLEX-ES definitions/operation” on page 85.

9.2 Other preparations

We needed to ensure that an FTP server was available on our base Red Hat system. An easy check is:

```
$ ftp 192.168.0.110 (FTP to ourself)
```

The IP address 192.168.0.110 is the address of Red Hat Linux on our ThinkPad. That is, we want an FTP connection to our own machine. If the connection is refused, it may be that you do not have an FTP server installed. We used Red Hat 8.0 CD#3 and installed the `wu-ftp` package. (We had previously installed `ftp.0.17-15.i386.rpm` from CD#1, but this is only an FTP client.)

After installing `wu-ftp`, we enabled it. To do this, edit `/etc/xinetd.d/wu-ftp` and change the line containing `disable = yes` to `disable = no`. You must restart `xinetd` (with a `SIGHUP` or a reboot) to make this change effective. (The command `/etc/init.d/xinetd restart` could be used.)

We needed a user ID having the root directory as its home directory. Using the **gnome** functions **System Settings -> Users and Groups -> Add User** we created user *helper* with home directory / (that is, home directory *root*).

We needed to permit SuSE Linux to open an X window in Red Hat Linux later in the installation process. In a Red Hat window, we did this:

```
# xhost +
```

Special FLEX-ES function

Recent FLEX-ES releases provide the CLI command **iodelay**. The syntax is:

```
iodelay <address> <milliseconds>
```

This command is issued from the *flexes>* prompt. An example is **iodelay 200 50**. This causes the completion of every I/O operation to S/390 address 200 to be delayed by 50 milliseconds. The command **iodelay 200 0** would set the delay to zero, for normal operation.

There is a mismatch between the operation of an older *dasd* driver for Linux and the very fast I/O emulation of FLEX-ES. This is not a FLEX-ES error, but is a *dasd* driver design problem that was fixed in the March 2003 stream of fixes from IBM Developerworks.⁶⁵ The error arises in I/O tests done while Linux is booting. Once through the booting process there is no need for special handling. The 50 millisecond delay is arbitrary; we used it and it worked for us. We did not experiment with smaller delays.

If the *iodelay* is not sufficient or is not done, the symptoms (during booting) are:

```
...
dasd: waiting for responses
dasd: giving up, enable late devices manually
...
Kernel panic: VFS: Unable to mount root fs....
```

The operational requirement, until a corrected *dasd* driver is used, is to set an I/O delay before SuSE Linux for S/390 is booted and (optionally) remove the delay after the root file system is mounted during the booting process. (If you do not watch the booting process, you can simply wait until it finishes and displays a login prompt on the hardware console.)

9.3 Shell scripts for FLEX-ES

We needed two FLEX-ES startup scripts. The first script, which we placed in */usr/flexes/rundir/SLinux.sh*, was as follows:

```
flexes SLinux.syscf
echo "rdmem /mnt/cdrom/boot/vmldr.ikr 0" | flexesccli localhost SLinux
echo "rdmem /mnt/cdrom/boot/initrd 800000" | flexesccli localhost SLinux
echo "remem /mnt/cdrom/boot/parmfile 10480" | flexesccli localhost SLinux
flexesccli localhost SLinux
```

This shell script starts a S/390 instance named *SLinux*. It then copies three files from the first SuSE CD (assumed to be mounted in the CD-ROM drive) directly into S/390 memory. This is used only to start the SuSE installation process. A somewhat similar technique is used by zSeries machines to boot Linux from the HMC. The memory addresses (0, 800000, and 10480) are from */mnt/cdrom/boot/suse.ins* on the first SuSE CD. They are not unique to FLEX-ES usage.

⁶⁵ This fix had not been integrated into the SuSE release we used. There is a set of updates available with the SuSE 8.0 SLES release that we assume contains this fix. Once the *dasd* fix is installed the **iodelay** command is not needed.

The second shell script, which we placed in /usr/flexes/rundir/SLinux.sh, was as follows:

```
flexes SLinux.syscf
xhost +
flexesccli localhost SLinux
```

This shell script starts the same S/390 instance, but does not load files into S/390 memory. (The **xhost +** command is not required unless you want to use a GUI interface with SuSE Linux. You could specify more secure operands than “+” if needed.)

9.4 Start installation

We placed the first SuSE CD in the CD-ROM drive and started the installation, as follows:

```
$ cd rundir                    (We keep FLEX-ES working files here)
$ su                          (Must be root)
# sh shLVLNX                  (If not already run)
# resadm -s SLinux.rescf
# xhost +                     (We need this later)
# exit                        (Return to user flexes)
$ sh SLinux.sh                (Start shell script; takes several seconds)
flexes> restart               (Perform S/390 restart function)
```

This is an unusual startup, where no IPL is needed. The necessary programs are already in S/390 memory, placed there by the shell script. The **restart** function gives control to the program at a fixed address in S/390 memory. The SuSE installation program obtains control and issues many messages to the S/390 hardware console. These messages appear in the window with the *flexes>* prompt.

Whenever such messages appear, we should press Enter (with the desktop focus in this window) to regain the *flexes>* prompt.⁶⁶ This is done many times during the installation dialog and we do not show these Enter steps in the following description. To reply to a message, we regain the *flexes>* prompt and use the CLI command **hwc** to send the message.

The initial installation dialog was as follows:⁶⁷

```
Linux version 2.4.19-3suse-SMP (root@s390z06) (gcc version 3.2) #1 SMP Wed Nov 6 ...
We are running native (31 bit mode)
This machine has an IEEE fpu
...
(Several pages of messages, including a long pause while decompressing a RAMdisk)
...
Please select the type of your network device:
0) no network
1) OSA Token Ring
2) OSA Ethernet
3) OSA-Gigabit Ethernet or OSA-Express Fast Ethernet
4) Channel To Channel
5) Escon
6) IUCV
8) Hipersockets
9) Show subchannels and detected devices
Enter your choice (0-9):
(Press Enter to regain the flexes> prompt)
flexes> hwc 2                  (Select OSA Ethernet)
```

⁶⁶ It is not actually necessary to regain the *flexes>* prompt before entering CLI commands. If you simply enter the command (without regaining the *flexes>* prompt) the console listing can be confusing when read later.

⁶⁷ The dialog shown here was not captured from screen images. It was entered “freehand” and your displayed dialog might have slightly different spacing, punctuation, and indentation.

```

First 16 possible OSA / OSA-2 channel devices detected
... (more text)
Enter the read channel device number, e.g., 'FC20': (Enter auto)
flexes> hwc auto
.... (a page of messages, including what appear to be error messages)
.... (long pauses--over one minute--while interfaces are checked)
eth0 is available, continuing with network setup
Please enter your full host name, e.g., 'linux.example.com':
flexes> hwc t23.itsoefs.com (Not connected to an external network)
Please enter your IP address, e.g., 192.168.0.1:
flexes> hwc 192.168.0.112 (Address we want for SuSE)
Please enter the net mask, e.g., 255.255.255.0:
flexes> hwc 255.255.255.0
Please enter the broadcast address if different from (192.168.0.255):
flexes> hwc (Enter a null line)
Please enter the gateway's IP address, e.g., 192.168.0.254:
flexes> hwc 192.168.0.110 (Address of Red Hat Linux)
Please enter the IP address of the DNS server or 'none' for no DNS: (Enter none)
flexes> hwc none
Please enter the MTU (Maximum Transfer Unit).
Leave blank for 1492:
flexes> hwc (Enter a null line)
Configuration for eth0 will be:
Full host name : t23.itsoefs.com
IP address : 192.168.0.112
Net mask : 255.255.255.0
Broadcast address : 192.168.0.255
Gateway address : 192.168.0.110
MTU size : 1492
Is this correct (Yes/No):
flexes> hwc Yes
....
Please enter the temporary installation password:
flexes> hwc root1 (Select a temporary password for root)
...
(Long pause.
An ifconfig command is issued automatically and pings are sent to our own
address (192.168.0.112) and to the gateway we specified (192.168.0.110).)
...
Network setup finished, running inetd...
You should be able to login via telnet now, for ssh wait a few seconds.
Temporary host keys (only for installation) are being generated now.

```

Do not **telnet** to the new system at this time. Wait for all the host keys, DSA keys, and RSA keys to be generated. This takes several minutes.

```

...
Generation of temporary installation host keys finished.
...
Please specify the installation source
...
1) NFS
2) SAMBA
3) FTP
0) Abort
Choice:
flexes> hwc 3 (Select FTP)
Please enter the IP-Number of the host providing the installation media
flexes> hwc 192.168.0.110 (Address of Red Hat Linux)
...
(The system pings the host you named)

```

```

...
Please enter the directory of the installation media:
flexes> hwc /mnt/cdrom (Assume SuSE CD#1 is mounted)
Is the following correct?
Installation source: ftp
IP-Address: 192.168.0.110
Directory: /mnt/cdrom
Yes/No:
flexes> hwc yes
Please enter the username for the FTP-access (for anonymous just press enter):
flexes> hwc helper (The userid with root as home)
Please enter the password for the FTP-access (for anonymous just press enter):
flexes> hwc xxxxx (Whatever password you assigned)
Is the following correct?
FTP User: helper
FTP Password: xxxxx
Yes/No:
flexes> hwc yes
Which terminal do you want to use?
1) X-Windows
2) VNC (VNC-Client or Java enabled Browser)
3) ssh
Choice:
flexes> hwc 1 (Select X Windows)
Please enter the IP-Number of the host running the X-Server:
flexes> 192.168.0.110 (Address of Red Hat Linux)
...
(There is an automatic ping sent to the indicated address.)
...
(Many dots on the screen, to indicate progress, and a variety of messages.)
...
(After a long pause a new window in the Red Hat gnome desktop should open and
display an initial YaST screen.)

```

9.5 Main installation steps

A SuSE license was first displayed in the newly opened X window:

```

click Accept
select English(US) and click Accept

```

The next steps involved DASD setup. Please read the text before attempting to follow our examples.

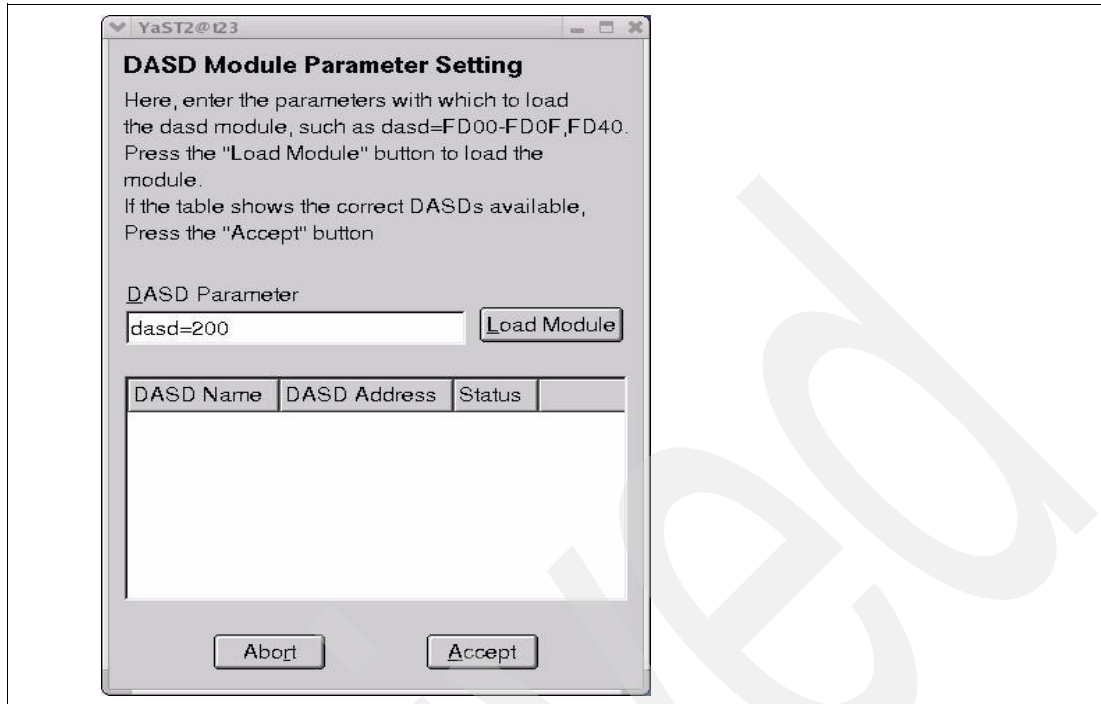


Figure 9-1 DASD Module Parameter Setting

DASD initialization

STOP and read this section about DASD initialization before you do anything. If this is done incorrectly, you may need to start the complete installation again.

YaST presented the DASD Module Parameter Setting panel, as shown in Figure 9-1. We entered *dasd=200* in the DASD parameter box and clicked **Load Module**. *Do not click Accept yet.*

We opened a new terminal window on the Red Hat gnome desktop and created a **telnet** connection to SuSE (at IP address 192.168.0.112):

```
$ telnet 192.168.0.112
t23 login: root                                (Log in as root)
Password: root1                                (Use the temporary root password we selected)
# cat /proc/dasd/devices                       (We should see device 200)
0200(ECKD) at (94: 0) is dasda .....
# dasdfmt -b 4096 -f /dev/dasda                (Format the drive)
Type "yes" to continue, no will leave the disk untouched: yes
```

Formatting may take a while; ours took about 10 minutes.⁶⁸ We then used **fdasd** to partition the drive into a large partition (for root) and a small swap partition. You might notice that **fdasd** is a close relative of **fdisk**:

```
# fdasd /dev/dasda
n                                                (Add a partition)
 2 - 45000                                       (Tracks for the large partition)
n                                                (Add another partition)
```

⁶⁸ We had problems when we tried to repeat the **dasdfmt** step during a later installation. The **dasdfmt** program was unable to write the IPL record and the target disk was unusable. We found it necessary to stop FLEX-ES, reformat the volume (with the FLEX-ES **ckdfmt** program), and go through the whole SuSE initialization again before we could run **dasdfmt** again. We suggest that you do not try to **dasdfmt** a volume a second time. You can repartition it with **fdasd** without using **dasdfmt** again.

```

45001 - 50084                (Tracks for the small partition)
t                            (Change partition type)
2                            (Type 2 is Linux swap)
p                            (Display partitions)
w                            (Write partition table and exit)
# exit                      (Exit from telnet session; close window)

```

After completing the **fdisk** step, we exited from the telnet window. We clicked **Accept** on the YaST DASD Module Parameter Setting panel. The partition size we used for swap, about 5000 tracks, was quite arbitrary. It is approximately 240 MB and seemed a reasonable size.

On the next YaST panel, we clicked **New Installation** and **OK**.

On the main YaST panel, we clicked on the word **Partitioning** that is just above a red message in the center of the panel. This started the Expert Partitioner with the panel shown in Figure 9-2. In this panel:

- Select **/dev/dasda1** and click **edit**.
- In the edit panel select **Format**, File System **Ext3**, Mount Point **/**, and click **OK**.
- Select **/dev/dasda2** and click **edit**.
- In the edit panel select **Format**, **swap**, and click **OK**.
- Click **Next** on the Expert Partitioner panel.

This exited from the Expert Partitioner.

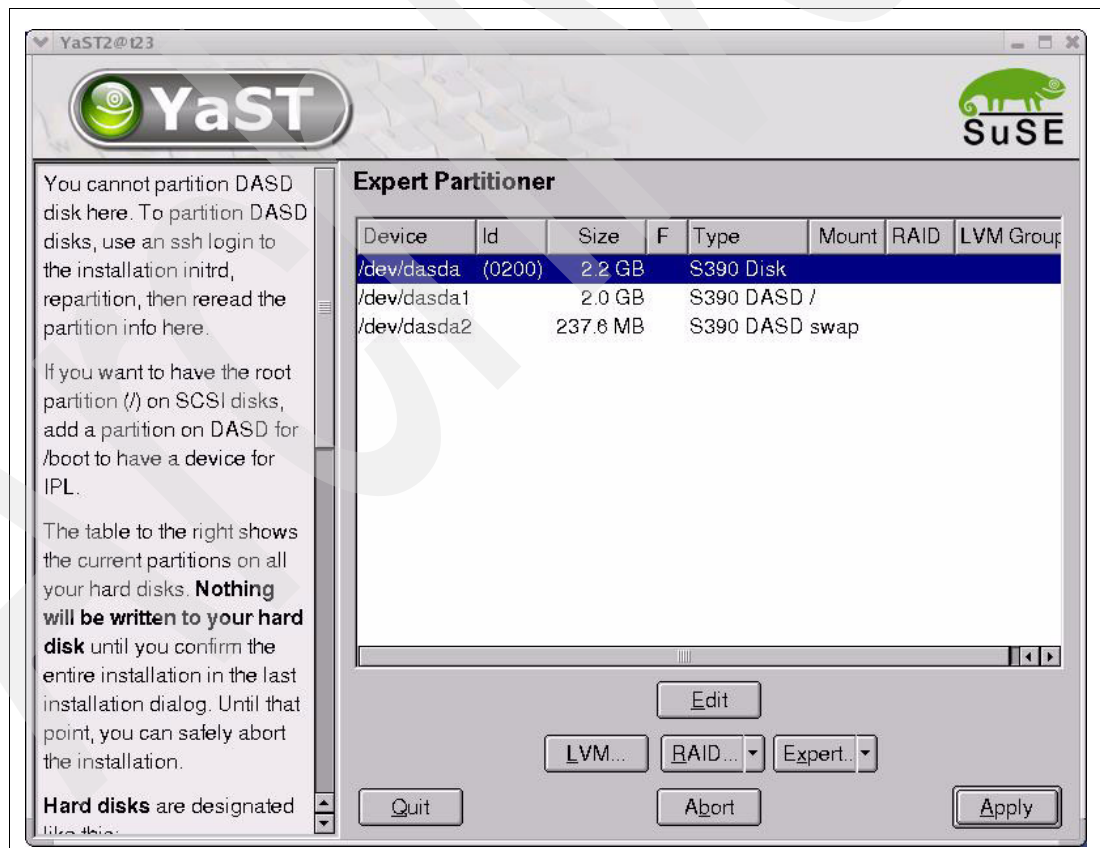


Figure 9-2 Expert Partitioner

Package selection

We clicked on the word **Software** in the main YaST panel. This opened a window with the options:

- ▶ Minimum graphical system (without KDE)
- ▶ Default system
- ▶ Detailed selection

We selected *Detailed selection* and this opened another window with high-level choices on the left and detailed package lists on the right, as shown in Figure 9-3. We discovered that the selection “C/C++ Compiler and Tools” was too large for our single 3390-3 drive. We selected Gnome, Help, Graphical Base System, YaST2, Analyzing Tools, and SLES Administration Tools. The panel indicated these would use about 73 percent of our disk space. Note that the operation of these panels is slow in this environment. Wait after clicking on an option—do not click again until the first selection completes.

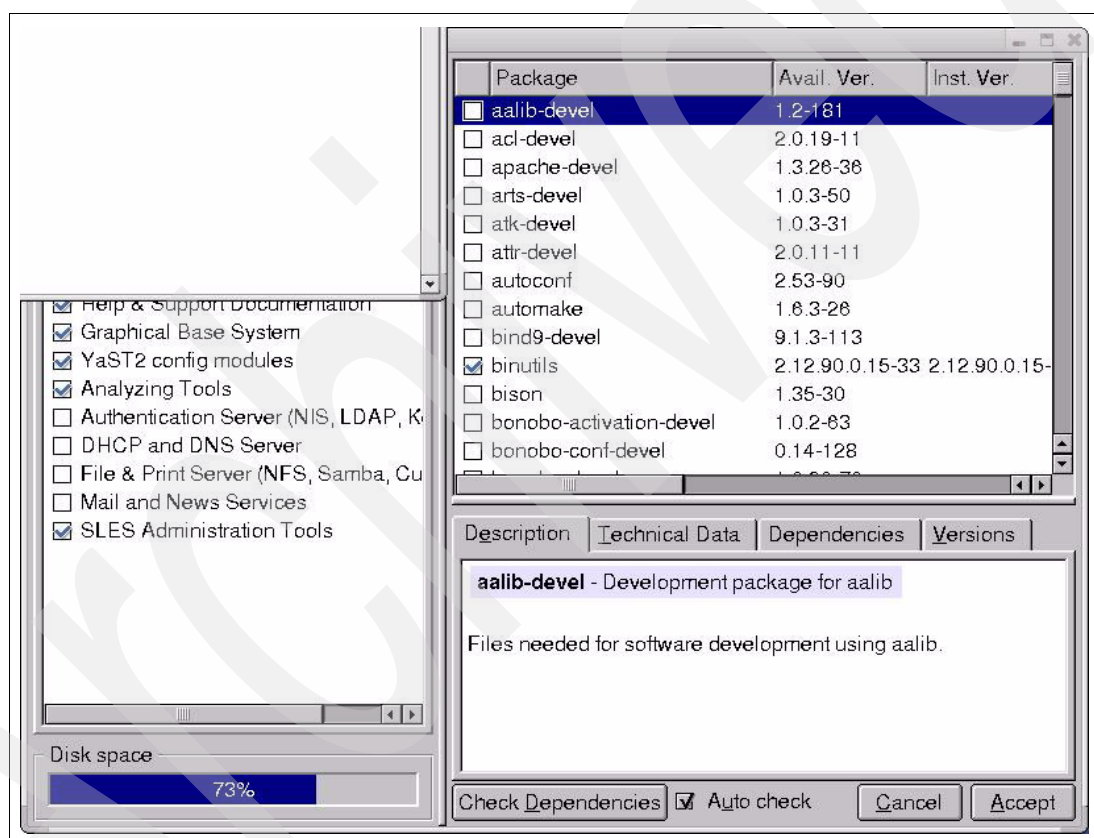


Figure 9-3 Part of Software Selection panel

We eventually clicked on Accept for the Software selection and this returned us to the main YaST panel. We then clicked on Accept in this panel to begin installation. YaST produced the following messages:

```
YaST2 has obtained all the information
required to install SuSE Linux.
The installation will be carried out according
to settings made in the previous dialogs.
To commit the installation and all choices made
so far, choose "Yes". Choose "no" to return
to the previous dialog.
Start installation?
```

Click on “Yes, install”.

The installation of the packages took 2 hours 20 minutes on our T23 ThinkPad. Only the first SuSE CD was used. This is relatively slow, possibly due to the complex YaST shell script that manages the process.⁶⁹ The YaST messages continue with:

```
Your system will now be shutdown.
After shutdown, reload the system with the
load address of your root DASD.
( Click “OK” at this point. The YaST2 window on the Red Hat desktop closes. Log files
are written. This takes a short time.)
```

A few additional messages appeared on the hardware console (the *flexes*> window) and then we IPLed the new system:

```
Power down                                (Last message from SuSE. Wait for it.)
(Press Enter to regain the flexes> prompt)
flexes> iodelay 200 50                    (Important if using older dasd module)
flexes> ipl 200                            (IPL the SuSE disk)
...
(The normal Linux startup messages appear. This is a long startup that executes a
number of configuration scripts and so forth.)
...
*
* Please return to your X-server screen to finish installation
*
...
(More long automatic startup functions follow this message)
(Eventually an X Window is opened on the Red Hat desktop. The installation
process remembers the IP address from the initial dialog. Our xhost +
command should still be in effect.)
...
(Wait until the “wrist watch” icon disappears from the YaST window.)
```

The YaST window asked us to enter a new password for *root*. It then asked us to define another user. This additional user may be needed later, so it should be defined. We defined user *ogden*.

YaST then performed more automatic configuration steps and spent at least 10 minutes saving various configuration files. After displaying Network Interface and Printer options (which we skipped) it waited for us to click **Next**. This caused the YaST window to close and SuSE automatically rebooted itself.

9.6 First use

SuSE Linux needed about five minutes to complete the automatic reboot. It then displayed a login prompt on the hardware console:

```
....
(...many booting messages....)
t23 login:
flexes> iodelay 200 0                    (We can turn off the delay now)
flexes>
```

At this point we attempted to **telnet** to SuSE from a Red Hat gnome window; the connection was rejected. We used **ssh** from a terminal window on the Red Hat desktop, using the

⁶⁹ The first package installed, “howtoenh-2002.9.6”, is especially slow and may appear to be hung. Be patient.

command **ssh root@192.168.0.112**. (We could also log into the system from the *flexes*> window, using *hwc* prefixes. The **ssh** login was much more convenient.)

We decided to start YaST2 again and use it to configure the system to allow **telnet** connections. From the **ssh** login we issued:

```
# export DISPLAY=192.168.0.110:0          (Point to the Red Hat system)
# yast2 &
```

After a pause, the YaST2 Control Center window opened on the Red Hat desktop. We performed the following two steps:

```
Select Security and Users (in the left panel in the window)
Click Security Settings (in the right panel in the window)
    In a new window, select Security Configuration and select Level 1
Click Finish

Select Network/Basic (inetd) (in the right panel in the main window)
Select On with custom configuration and click Next
    Select the telnet line and click Activate or Deactivate
Click Finish
```

This caused **inetd** to restart. After this we could **telnet** into the system, but not as **root**. We logged in using user ID *ogden* and then used **su** to acquire **root** authority. This gave us full access to our SuSE system with either **telnet** or **ssh**. Obviously, **ssh** is the better choice on an open network.

We used **shutdown now** to terminate SuSE Linux for S/390. (If this is being done from the hardware console, be certain to remember the *hwc* prefix, otherwise you will **shutdown FLEX-ES**.)

Using ssh

The first time we used **ssh**, we received messages saying the authenticity of the host could not be verified and asking for permission to continue. We replied “yes” and **ssh** continued. Later use of **ssh** to the same SuSE installation worked without displaying these warnings.

We later reinstalled SuSE and found we could not log into the system with **ssh**, which issued a *host key verification failed* message. This message was issued because the new SuSE installation created a new connection key used by **ssh**. If this happens to you, you can find an **.ssh** directory in your home directory on the system running **ssh**. (This was Red Hat Linux, in our case, and our home directory was /usr/flexes.) In the **.ssh** directory, look for file **known_hosts**. This file should have a line (containing an encryption key) for your SuSE system. Delete this line and you should then be able to **ssh** to SuSE again. If you do not use **ssh** for any other purpose, you could delete the whole **.ssh** directory to let **ssh** reinitialize your connection to SuSE.

9.7 Routine use

Our routine startup for FLEX-ES and SuSE Linux for S/390, using the environment just described, goes like this:

```
$ su                                (Normal FLEX-ES startup)
# cd rundir
# sh shLVLNX                        (Need to start LVM and create raw link)
# resadm -s SLinux.rescf
# xhost +                           (In case we want to use SuSE GUI)
# exit                               (Leave root)
$ cd rundir
```

```

$ sh SLinux.sh                                (Our startup script)
flexes> iodelay 200 50
flexes> ipl 200
( wait about 5 minutes until the logon prompt appears )
t23 login:
flexes> iodelay 200 0                          (Turn off delay)
flexes>

```

At this point we open another Red Hat gnome terminal window and **telnet 192.168.0.112**. We log in as *ogden* (or whatever user IDs we have created) and **su** to *root* as needed. We can set the DISPLAY variable and start a YaST2 window on the Red Hat desktop, as we did during our first use.

If you have applied updates and patches that include the current dasd driver, you will not need the **iodelay** commands.

9.8 Left undone

This chapter described the installation of a very basic SuSE Linux for S/390 under FLEX-ES. It did not describe the application of the SP2 set of fixes or further SuSE customization.

Installation of all the available software packages overflows a single 3390-3. A recommended solution is to use LVM to combine several 3390-3 volumes into a single Logical Volume.

There are many sources of information about additional SuSE setup and customization. During SuSE installation (in the Software selection function) one of the options is for *redbooks*. This causes many IBM redbooks to be loaded as part of the SuSE system. We recommend this option.

Archived

Tape usage

FLEX-ES provides three general ways to use S/390 tape devices. These are through:

- ▶ Channel-attached tapes, using parallel or ESCON® channels to connect to “real” S/390 tape drives.
- ▶ SCSI-attached drives that are used to emulate 3420, 3480, or 3490E drives, as appropriate for the physical characteristics of the hardware. (Other IBM drive tapes are also emulated, but these are the three major families of drives.)
- ▶ Linux files that are used to emulate tape drives. Three formats are supported: FakeTape, AWSTAPE, and OMA.

This chapter discusses various aspects of tape usage under FLEX-ES.

10.1 SCSI tape setup

Using SCSI tape drives requires that you know the SCSI *name* of the devices. This is sometimes not obvious. FLEX-ES uses Linux SCSI generic device interfaces and names for tape I/O; an example of a name is `/dev/sg1`.

The following command may be useful:

```
# cat /proc/scsi/scsi
Attached devices:
Host: scsi0 Channel: 00 Id: 00 Lun: 00
  Vendor: FUJITSU Model: MAG3182MP   Rev: 5506
  Type:   Direct-Access              ANSI SCSI revision: 03
Host: scsi0 Channel: 00 Id: 01 Lun: 00
  Vendor: FUJITSU Model: M1016B M2483B Rev: 0014
  Type:   Sequential-Access          ANSI SCSI revision: 02
Host: scsi1 Channel: 00 Id: 00 Lun: 00
  Vendor: LITE-ON  Model: LTR-12102C  Rev: 0000
  Type:   CD-ROM                     ANSI SCSI revision: 02
```

SCSI devices are listed in order here, starting with device zero. `/dev/sg0` is a disk drive; `/dev/sg1` is a tape drive; and so forth. The only tape drive on this system is `/dev/sg1`, not

/dev/sg0. Note that FLEX-ES uses the /dev/sgx node interface, not the /dev/stx node interface.

If you are not certain that you have the correct name, you can try:

```
# /sbin/scsi_info /dev/sg1
SCSI_ID="0,1,0"
MODEL="FUJITSU M1016B M2483B"
FW_REV="0014"
```

The output from this command should verify that you have the correct *sg* number.

Linux tape utilities use names such as /dev/st0 and /dev/st1. Unfortunately, the numeric part of the *sg* and *st* name is not necessarily the same for a given drive; you may need to experiment a little to find the correct *st* number. Also, remember to turn on external drive(s) before turning on PC power. This ensures that the PC BIOS recognizes the drive.

10.1.1 Linux tape commands

You can use Linux tape utility commands with SCSI-attached tape drives, but do this only if the tape drive is not mounted on a FLEX-ES device (or if FLEX-ES is not running). If you are having problems determining if your tape drive is properly connected, these commands can be useful. Typical commands are:

```
# mt -f /dev/st1 fsf          (Forward space a file)
# mt -f /dev/st1 rewind
# mt -f /dev/st1 eject        (There is no RUN command)
```

Again, note the use of *st1* device names for Linux use. FLEX-ES *always* uses *sg0* (or *sg1*, *sg2*,...) names.

10.1.2 Device characteristics

Tape device names can be a little confusing for 3480/90 drives. Table 10-1 may help explain the names and characteristics.

Table 10-1 3480/90 tape devices

Common name	Characteristic	IODF definition	FLEX-ES cu	FLEX-ES device
3480	18 track	3480	3480	3480
3480 compressed	18 track IDRC	3480+COMPACT	3490	3490
3490	18 track IDRC	3480+COMPACT	3490	3490
3490E	36 track	3490	3490	3490-E

Some SCSI tape drives can handle all the combinations in the table. How should such devices be defined? For example, z/OS may think it has a 3480 drive, but the user loads a 36-track cartridge in a drive that can handle both 18 and 36-track tapes. This might cause the drive to return sense bytes that conflict with the IODF definition. The result is a device error.

Another factor is the firmware in the tape drive. Most SCSI 3480/90 drives allow the user to specify various characteristics. These settings can also affect the id and sense bytes generated during operation and possibly create a mismatch that is sensed by z/OS I/O routines.

10.2 FLEX-ES FakeTape on z/OS

FakeTape⁷⁰ emulates tape devices using Linux disk files instead of tape drives. Provided the appropriate tape devices are defined in the z/OS IODF configuration data set, FakeTape will emulate any type of tape drive from 3420 to 3490-E. Because FakeTape always writes and reads the same format to/from Linux, it operates at the same speed for all different emulated tape device types. We ran several tape jobs using IEBGENER, IEBCOPY and DFDSS and they all performed well. Our z/OS and FLEX-ES definitions included a 3480 tape drive at address 560.

This is one of the jobs we executed:

```
//P390T    JOB 1,OGDEN,MSGCLASS=X
//BACKUP   EXEC PGM=IEBGENER
//SYSPRINT DD SYSOUT=*
//SYSIN    DD DUMMY
//SYSUT1   DD DISP=SHR,DSN=SYS1.PROCLIB(JES2)
//SYSUT2   DD DSN=PROCLIB,DISP=(NEW,KEEP),
//          UNIT=560,LABEL=(1,SL),VOL=SER=222222
```

We submitted the job and an IEF233A mount request was issued by z/OS. The console interaction went like this:

```
$HASP373 P390T STARTED - INIT 1
IEF244I P390T - UNABLE TO ALLOCATE 1 UNIT(S)
IEF877E P390T NEEDS 1 UNIT(S)
FOR SYSUT2
FOR VOLUME 222222
OFFLINE
0560
02 IEF238D P390T - REPLY DEVICE NAME OR 'CANCEL'
2,560

IEF503I UNIT 0560 I/O ERROR
IEF234E D 0560
IEF233A M 0560,222222,P390T,,PROCLIB
```

(at this point, go to the flexescli window and issue the command
flexes> **mount 560 /tmp/222222**)

```
IEC512I LBL ERR 0560, ,NL,222222,SL,P390T.....
IEC704A L 0560,222222,SL,NOCOMP,P390T.....
03 IEC704A REPLY 'VOLSER,OWNER INFORMATION', 'M' OR 'U'
3,222222
```

As seen in this interaction, we submitted our job. The tape drive was offline and z/OS requested that we provide a device. We answered with the device address, 560. We next received a mount request for volume 222222 (taken from our JCL) on unit 560. At this point, we switched to the CLI window (with the *flexes* prompt) and entered **mount 560 /tmp/222222**. This file name was quite arbitrary; if you plan to use many emulated tape volumes, you will probably want to create a directory for them (and perhaps a whole file system for them).

z/OS attempted to read the tape label (using an empty /tmp/222222 file) and received an error. This causes normal recovery processing for tape labels. We entered a volser (222222) in response to message IEC704A, and the job ran correctly. z/OS rewound and unloaded the “tape” at the end of the job. This logically unmounted the file and made the tape drive logically not ready.

⁷⁰ FakeTape is a trademark of Fundamental Software, Incorporated.

We later read the tape and printed (to SYSPRINT) the tape contents. The process was about the same, except our tape file (/tmp/222222) now had a correct label and was immediately accepted after we entered another *flexes*> **mount 560 /tmp/222222** command.

10.3 Tape resource options

Server files used by FakeTape are normal files in the server's file systems. With a little planning—for space and naming—a whole tape library can be emulated using inexpensive server disks. A file (in the server file system) is equivalent to a tape volume. You can compress these files, ftp them, write them on CD-ROMs, and so forth, provided they are restored to their normal form when used again by FakeTape.

FakeTape can automatically recognize and handle data in AWSTAPE and AWSOMA formats, as produced by P/390-related systems. You should be able to exchange such files via ftp, although we did not try this. You can use the FLEX-ES utility **initawstape** to initialize a file in AWSTAPE format and then write output data to it. By default, output tapes are initialized in FakeTape format.

Several special options are available for tape resource definitions:

```
c3480: cu 3480
options 'maxwritesize=200,allowdisconnects,allowmountccws,autoloader'
device(00) 3480 OFFLINE
end c3480
```

The maxwritesize option can be important. It specifies (in megabytes) the maximum size of an output tape. If your output approaches this size, FLEX-ES will signal an end-of-tape reflective marker. The S/390 program would usually write EOVS trailer labels at this point. In any case, FakeTape will continue to write additional records but signal end-of-tape for every additional write request. If you do not specify a maxwritesize value, two exposures occur:

- ▶ You might consume all the free space in the server file system if you are writing to a FakeTape (or AWSTAPE) device.
- ▶ You will never have end-of-volume processing if your output device does not provide an end-of-reel reflective marker or the cartridge equivalent. FakeTape output does not provide this unless you use the maxwritesize option. (FakeTape will signal end-of-tape when the output file system indicates it is full, even when the maxwritesize option is not specified. The S/390 operating system, however, may not be able to write any trailer records in this situation.)

In our opinion, you should always have a maxwritesize specification when using FakeTape. This can be included in the FLEX-ES **mount** command.⁷¹ For example:

```
flexes> mount 560 /holding/tape/222222 'maxwritesize=50'
```

(The maxwritesize parameter also applies to DAT tapes. DAT drives do not signal that they are near the end of the tape (to permit trailer records to be written). They signal end-of-tape and nothing more can be written. A well-selected maxwritesize can overcome this limitation.)

The other options are more specialized and you may never need them with FakeTape.

⁷¹ You can change all device options except DASD trackcachesize.

10.4 Using a 4mm tape drive

We ran the same job we used for FakeTape (“FLEX-ES FakeTape on z/OS” on page 147) using a 4mm drive in our xSeries EFS system. Before submitting the job, we used this command in the *flexes*> window:

```
flexes> mount 560 /dev/sg0
```

This assigns the 4mm drive to the FLEX-ES resource at S/390 address 560. (This address is appropriate in the AD system, where it is generated as a 3480 tape drive. /dev/sg0 was the correct name for our 4mm drive, but your generic tape names may be different.) When the job was submitted, we received a mount request on the z/OS console, inserted a scratch 4mm tape, went through the normal standard label creation messages, and watched the job run. The 4mm tape was unloaded at the end of the job. (We later ran another job to read the tape, just to verify that it was correct.)

After creating a standard label 4mm tape on our EFS system running z/OS, we read the tape (without problems) on a 4mm drive attached to a P/390 system running z/OS.

10.5 Using an Overland T490E

We attached an Overland T490E SCSI tape drive to a SCSI PCI adapter in our xSeries EFS system. This is an older drive that was often used with P/390 systems. It can read 3480 and 3480 IDRC tapes (18 tracks) and 3490 tapes (36 tracks); it can also write all of these modes. Our xSeries EFS system also had a 4mm SCSI drive. Linux sensed the 4mm drive first and addressed it as /dev/sg0. The Overland drive was /dev/sg1.

We could use FLEX-ES mount commands to associate a tape drive with a FLEX-ES instance, or we could hard code the associations in the FLEX-ES definition files. We decided to hard code the definitions. The z/OS system we used had an appropriate 3490E drive defined at address 500 and a 3480 drive at address 560.⁷² Our associated FLEX-ES definitions included:

```
system S14A:
.....
cu devad(0x560,1) path(2) resource(CU3480)
cu devad(0x500,1) path(2) resource(CU3490)
.....
end S12A

resources R12A:
.....
CU3480: cu 3480
interface local(1)
device(00) 3480 /dev/sg0                (the 4mm drive)
end CU3480

CU3490: cu 3490
interface local(1)
device(00) 3490-E /dev/sg1              (the Overland drive)
end CU3490
.....
end R14A
```

Both drives were online after we IPLed z/OS, and we had no problems using them.

⁷² We could use HCD to define or change these addresses, but we found it easier to adapt the device addresses to the existing IODF definitions.

The Overland drive is similar to the IBM 3490-F01 drive. Both have an extensive list of firmware options that can be set from the panel of the drive. We used the following options with the Overland drive:

Option	Description	Our setting
42	Power-up restart	01 = Unload cartridge
43	Format Display	01 = Enable
44	Write Synchronization	02 = Sync on 2nd consecutive file mark
46	Number of Read retries	05 = 16 retries
47	Number of Write retries	05 = 16 retries
48	Lockout switches	00 = Host & panel unlocked
49	Block size	00 = Variable Block Mode
50	SCSI address	We used address 04; nothing else on the adapter
51	Interface parity	01 = Enable
52	Device type	00 = 80h
53	Rewind ready	02 = Disable
54	Synchronous Transfer	04 = Disable
55	EOT mode reporting	00 = Standard
57	2's complement residue	02 = No
58	FSC report size	02 = 1 byte display
60	Vendor ID	11 = OVERLAND
61	Product ID	32 = T490E
62	Density	03 = 3490E
63	Interface Options	00 = SCSI2 (not 08 = IBM 3490E)
65	18 track write	00 = Enabled
68	Default transfer rate	00 = Sync 5MB/sec
69	IDRC default	01 = Compression OFF

10.6 Using the `scsitfake` program

The `scsitfake` program copies from a SCSI-attached tape drive to a Linux file and the file is properly formatted for FakeTape usage. In a sense, this converts a “real” tape (a 3480 cartridge, for example, if you have a SCSI-attached 3480 tape drive) to a Linux file in a format that can be used just like a “real” tape. The Linux file can be written on CD, sent by ftp, or managed in all the ways that apply to simple Linux files.

This is a very useful utility service. Most EFS systems do not have SCSI-attached tape drives, especially 3480/3490-compatible drives. A single system with such a tape drive can “convert” tapes to Linux files, burn these files on CDs, and send the CD to another EFS user. The FakeTape file on the CD can then be used just like the original tape (for read-only operations, of course).

Consider the following example. We have a stand-alone utility tape on a 3480 tape cartridge. This tape contains the stand-alone versions of the ICKDSF initialization program and the DFDSS restore program. We want to convert this tape cartridge to a CD file so we can distribute the stand-alone utilities to EFS users who do not have “real” tape drives.⁷³

The `scsitfake` program must be used by *root*. We earlier determined that our 3480/3490 tape drive was `/dev/sg1`. (See “SCSI tape setup” on page 145 for details about this step.) We verified that there was ample space in our `/tmp`. We then inserted the tape cartridge in the drive, waited for it to load, and issued the following commands:

```
$ su                                (Change to root)
# scsitfake /dev/sg1 /tmp/sautap    (The /tmp file name is arbitrary)
20596 blocks and tape mark copied
```

⁷³ These users could, in turn, use the utilities to restore backup tapes that had been converted to FakeTape format CD files.

```

0 blocks and tape mark copied
1 blocks and tape mark copied
0 blocks and tape mark copied
0 blocks and tape mark copied
0 blocks and tape mark copied
SCSI sense data 00 - 11 (f0000800 0100000c 00000000)
SCSI sense data 12 - 23 (0000d200 00000000 00000000)
Aborting scsitfake -- error reading from SCSI device (errno = 5)
# mt -f /dev/st1 eject (Rewind and unload the tape cartridge)

```

FLEX-ES need not be running to use **scsitfake**. (If it is running, it should not have the tape drive allocated to its use.) Like all the FLEX-ES functions, this program uses the SCSI *sg* interface instead of the *st* interface; that is, our tape drive was `/dev/sg1` and not `/dev/st1`. We placed the resulting FakeTape file in `/tmp/sautap`, and this file name was completely arbitrary.

Any program that copies unknown tapes can have a problem detecting the end of the valid data on the tape. In our example, all the valid data was in the first file (ending with the first tape mark). The single block following two tape marks was probably the “PID trailer” found on some IBM distribution tapes. After a few additional tape marks, the program encountered tape errors and quit. The resulting FakeTape file ends at the point where the **scsitfake** program encountered errors.

Note that the Linux tape utility function that we used to remove the tape cartridge uses *st* addresses instead of *sg* addresses. (There is no need to use this utility, of course. In this case, we could have simply pressed the Unload button on our tape drive.)

We had a FLEX-ES definition file⁷⁴ that included a 3270 terminal (at address 700, although this is not relevant) and a 3490 tape drive at address 420. The tape drive resource definition was:

```

CU3490: cu 3490
interface local(1)
device(00) /dev/sg1
end CU3490

```

We started the FLEX-ES resource manager and S/390 instance in the normal manner and then issued the following commands at the CLI prompt:

```

flexes> mount 420 /tmp/sautap
flexes> ipl 420

```

We then pressed Enter on the 3270 session and received the first output from the ICKDSF stand-alone program. We entered a second **ipl 420** command and started the restore stand-alone program (which was also in the first file of the tape), just as if we were working from a “real” tape drive.

We then burnt a CD with the FakeTape file:⁷⁵

```
# mkisofs -R /tmp/sautap | cdrecord -v fs=6m dev=0,0 -
```

Comments about burning CDs in our environment are in “CD recording under Linux” on page 3. To verify that the file was still useful, we then mounted the CD and issued the following CLI commands:

```

flexes> mount 420 /mnt/cdrom/sautap (sautap was the file name on the CD)
flexes> ipl 420

```

⁷⁴ This particular definition was for use with the TPF operating system and not the z/OS we use for most of our examples.

⁷⁵ We sent the file, via ftp, to a ThinkPad that had a CD-RW unit and burnt the CD there.

The programs IPLed and operated as before.

Writing awstape format

You can use the following method to have **scsitfake** write in *awstape* format. This is useful if you need to use the resulting file with an MP3000 or P/390 system:

```
# cd /usr/flexes/examples          (Where initawstape is located)
# initawstape /tmp/awstape123      (Initialize emulated tape file)
# scsitfake /dev/sg1 /tmp/awstape123 (Copy from SCSI tape drive)
```

The first command creates a new emulated volume. (The directory and file name are arbitrary.) The new volume contains a control record that indicates it is in *awstape* format. The **scsitfake** program senses that the output file already exists and is in a specific format; it then continues to write output in this format.

SNA over Ethernet

This chapter was contributed by Michael Ryan of Fundamental Software, Inc., Fremont, California.

The following discussion illustrates how to make VTAM connections among multiple z/OS systems on an Ethernet LAN using FLEX-ES emulated device types *osasna* and *xcasna*. A Token Ring configuration would be nearly identical, but would define FLEX-ES device types *osasnaTR* and *xcasnaTR* respectively.

The purpose of this document is to show z/OS FLEX-ES users how to correctly configure SNA-over-Ethernet connections; we illustrate how to define the FLEX-ES resources, the MVS Input/Output Definition File (IODF), and how to make the basic VTAM SSCP-SSCP connections. The lab exercise also configures cross-domain logon for TSO and JES2 Network Job Entry (NJE).

Both *osasna* and *xcasna* LAN nodes are known to VTAM as XCA (external communication adapter) major nodes, regardless of whether the FLEX-ES emulated device is an *xcasna* (3172-type) or *osasna* (OSA-1 type). In fact, the VTAM XCA major node definitions do not change if the emulated device type is changed from *xcasna* to *osasna* or vice-versa. The FLEX-ES and z/OS device definitions do change, however, as we shall see.

11.1 Network lab

We set up three servers, two running OpenUnix 8⁷⁶ and one running Red Hat Linux 8.0. On server x370, we ran one z/OS 1.4 system and named it z370. This system connects to our LAN with an Ethernet NIC defined as an *xcasna* device. On server x232, we ran two z/OS 1.4 systems in separate FLEX-ES instances, one named z232 and the other named zсна. The systems share DASD but not JES2 spool. These two instances connect to the LAN using two dedicated Ethernet NICS defined as *xcasna* devices. Neither of the cards is used for any other Linux or MVS networking beyond the *xcasna* connections. On server xeon, we ran a z/OS 1.4 base sysplex with three members, zos1, zos2, and zos3; we connected zos1 to the

⁷⁶ ITSO note: The use of FLEX-ES under OpenUnix is not described in more recent IBM Redbooks. It is included here for completeness of the operational environment being described.

Ethernet LAN with an osasna device. For reasons we will explore later, neither zos1 nor zos2 required an Ethernet connection to our LAN.

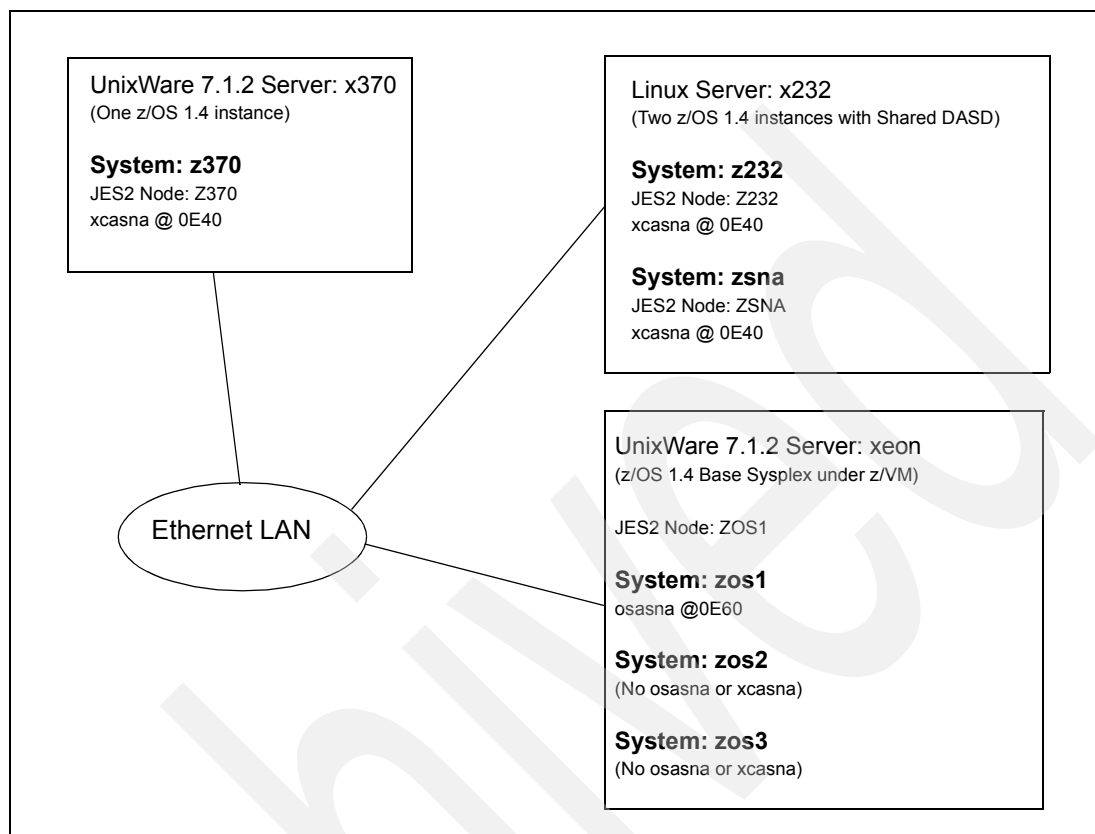


Figure 11-1 Server configuration in the lab

11.2 Selecting and initializing the adapters

SNA over LAN can use its own, dedicated Ethernet adapter or can ‘piggyback’ onto an adapter that has either a server or an MVS TCP/IP stack (or both) already installed. Each system that will connect via SNA protocols over the LAN is identified by the combination of the Medium Access Control address of the Network Interface Card (MACADDR) and a Service Access Point (SAPADDR). SAP’s are arbitrarily assigned but start at four and must be evenly divisible by four.

On server xeon, we chose /dev/net5 as the SNA-over-Ethernet adapter. This adapter was installed via the UnixWare 7.1.2 **netcfg** utility and was not assigned a UnixWare 7.1.2 home address. Installing the adapter this way adds an entry to the /etc/confnet.d/inet/interface file, which must be commented out by inserting the pound sign (#) at the beginning of the line:

```
#net:5:0.0.0.0:/dev/net5:netmask 255.0.0.0 broadcast 0.255.255.255 perf 24576
245761:add_interface:
```

To obtain the MACADDR of this adapter, we used the UnixWare 7.1.2 **ndstat** command:

```
bash-2.03# ndstat /dev/net5
Device      SNPA/MAC address  Factory Address
/dev/net5    00:02:b3:c8:84:e0 00:02:b3:c8:84:e0
              FRAMES
Unicast Multicast Broadcast Error Octets Queue Length
```

```
-----
In:      0      0      0      0      0      0
Out:     0      0      0      0      0      0
```

The MACADDR of /dev/net5 is shown under Factory Address and is 0002B3C884E0.

We chose to dedicate an adapter to SNA-over-Ethernet on server xeon, but we did not need to do so. On this system, /dev/net0 is in use by the UnixWare 7.1.2 TCP/IP stack, and /dev/net1 is defined as a FLEX-ES 3172 emulated control unit that is used for z/OS TCP/IP. We could have defined an osasna or xcasna device using either of these adapters; one NIC can carry both a TCP/IP stack and SNA-over-Ethernet connections. One restriction is that one NIC cannot carry both an MVS and a UnixWare 7.1.2 TCP/IP stack. (This restriction does not exist with Linux).

On server x370, we chose the NIC on /dev/net3, and we used **ndstat** to get its MAC address.

On server x232, we dedicated eth1 and eth2 to SNA-over-Ethernet. We used the **ifconfig** command (an example is shown here) to get the MAC addresses of both adapters:

```
[root@x232 root]# ifconfig eth1
eth1  Link encap:Ethernet HWaddr 00:02:B3:A6:E2:FA
      BROADCAST MULTICAST MTU:1500 Metric:1
      RX packets:0 errors:0 dropped:0 overruns:0 frame:0
      TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
      collisions:0 txqueuelen:100
      RX bytes:0 (0.0 b) TX bytes:0 (0.0 b)
      Interrupt:18 Base address:0xd000
```

The MACADDR of eth1 is shown in the HWaddr field and is 0002B3A6E2FA. To make the adapters available to Linux applications, we initialized them with the **ifconfig** command as follows:

```
# ifconfig eth1 10.0.0.1 mtu 1500
# ifconfig eth2 10.0.0.2 mtu 1500
```

The adapters cannot be opened by MVS unless this initialization is done each time that Linux is booted. This can be done by defining the adapters to Linux and assigning addresses such as 10.0.0.1. This address will never be used (we assume), but it provides an IP address to allow Linux to initialize the adapter.

11.3 A word about MACADDRES

We have seen several installations attempt to use byte-reversal to transform Ethernet MAC addresses into canonical form. In the Linux and UnixWare 7.1.2 environments, however, universal MAC addresses are usually already in canonical form. Therefore, when using xcasna or osasna devices in these environments, it is well to remember this rule:

*The Ethernet universal MAC address specified to VTAM is the manufacturer's MAC address as printed on the adapter card or displayed by either the **ifconfig** or **ndstat** commands. This address is coded in the XCA major node definition in VTAMLST as-is, without any modification.*

For example, the **ndstat**-displayed MAC address of 00:02:b3:c8:84:e0 is coded on a VTAM PU statement as MACADDR=0002B3C884E0.

11.4 Selecting VTAM configuration information

Member ACTSTR00 in SYS1.VTAMLST⁷⁷ on each system contains information that must be tailored to identify each SSCP node (each VTAM) and the LAN itself. Each system must have a unique SSCP name, PU name, and subarea number. The systems must share a common network id. The following list shows the values we selected for each system:

Server	MVS System	SSCPID	HOSTSA	SSCPNAME	HOSTPU
xeon	zos1	1	1	ZOS1SSCP	ZOS1\$PU
xeon	zos2	2	2	ZOS2SSCP	ZOS2\$PU
xeon	zos3	3	3	ZOS3SSCP	ZOS3\$PU
x232	zsna	6	6	ZSNASSCP	ZSNA\$PU
x232	z232	8	8	Z232SSCP	Z232\$PU
x370	z370	7	7	Z370SSCP	Z370\$PU

The SSCPID uniquely identifies a VTAM node; HOSTSA assigns the subarea number.

For NJE and cross-domain logon to function, each system must have a unique NJE application ID (APPLID), unique local non-SNA terminal names, and unique application names for TSO, CICS, IMS, etc. We named the terminals like the systems, and named the applications after their subareas. For the sake of simplicity, we will confine the examples to TSO.

MVS System	JES2 Node#	JES2 Name	JES2 ACB Name	TSO APPLID	Sub-area	Local Terminals
zos1	1	ZOS1	ZOS1NJE1	ZOS1TSO	1	LCL701-LCL71F
zos2	1	ZOS1	ZOS2NJE1	ZOS2TSO	2	ZS2701-ZS271F
zos3	1	ZOS1	ZOS3NJE1	ZOS3TSO	3	ZS3701-ZS371F
zsna	4	ZSNA	ZSNANJE1	ZSNATSO	6	ZSN701-ZSN71F
z232	3	Z232	Z232NJE1	Z232TSO	8	X23701-X2371F
z370	2	Z370	Z370NJE1	Z370TSO	7	ZS3701-ZS371F

Note that the three sysplex members, because they are in a multi-access-spool (MAS) configuration, share the same JES2 node number and node name. Each TSO APPLID is set in the application major node, on the APPL statement that defines the TSO ACB name. This APPL statement will have the operand ACBNAME=TSO.

11.5 Developing the adapter and device matrix

We developed the following list to describe the resources to be used. zos2 and zos3 are not in this table because they have no SNA-over-Ethernet connections. Their connectivity is achieved via VTAM RTP (Rapid Transit Protocol):

MVS System	FLEX-ES Resource	MVS CUA	MVS Device Type	Ethernet Adapter	MACADDR	SAP
zos1	XEON.OSASNA1	0E60	OSA	/dev/net5	0002B3C884E0	4
z232	X232.XCA1	0E40	CTC	eth1	0002B3A6E2FA	4
zsna	X232.XCA2	0E40	CTC	eth2	0002B3A6E2FB	8
z370	X370.XCA1	0E40	CTC	/dev/net3	0002B3AC6E27	4

Each SNA-over-Ethernet port must be assigned a unique SAPADDR (unique on a given NIC). SAP numbers begin at four and are evenly divisible by four. We arbitrarily used a SAP 8 for zsna's network interface card.

⁷⁷ Editor's note: Some z/OS implementations place these members in data sets other than SYS1.VTAMLST. References to SYS1.VTAMLST imply the use of whatever data set is used for this logical purpose.

11.6 Defining the FLEX-ES resources

VTAM configurations are very similar for any of the OSA or XCA FLEX-ES devices, but the MVS hardware definitions in the Input/Output Definition File (IODF) are often different. FLEX-ES XCA devices are defined as device type CTC in the IODF; FLEX-ES OSA devices are defined as device type OSA in the IODF.

An XCA device may be connected to any local channel in a FLEX-ES resource definition; OSA devices, however, require a new type of channel path that was introduced in FLEX-ES release 6.2.9, the *localosa* channel. The new channel path is necessary because a “real” OSA-1 device defines its own channel and control unit path; when z/OS reads configuration from a *localosa* channel, FLEX-ES correctly identifies the path as an OSA path.

On server xeon, we defined an osasna resource using our naming list and matrix (only the relevant parts of the configuration file are shown in Figure 11-2). We discuss the *localosa* channel type in more detail later.

```
channel(3) localosa

cu devad(0x0e60,01) path(3) resource(XEON.OSASNA1)

XEON.OSASNA1: cu osasna
interface local(1) options 'adapternumber=1'
device(0x00) osasna /dev/net5
end XEON.OSASNA1
```

Figure 11-2 FLEX-ES definition extract for instance zos1 on server xeon

On server x370, we also defined the resources using the same base information, as shown in Figure 11-3.

```
channel(2) local

cu devad(0x0e40,01) path(2) resource(Z370.XCA1)

Z370.XCA1: cu xcasna
interface local(1) options 'adapternumber=1'
device(00) xcasna /dev/net3
end Z370.XCA1
```

Figure 11-3 FLEX-ES definition extract for instance z370 on server x370

For information on how to configure VTAM for an XCA major node, we consulted examples in the IBM Redbook, *Connectivity on a PC Server System/390*, SG24-4624.

On server x232, we defined address E40 twice—once in instance z232 and once in instance zсна. “FLEX-ES resource definitions” on page 181 shows the complete FLEX-ES definitions used for all the systems in this discussion. The relevant definition lines are:

```
#FLEX-ES resource definition extract for instance z232 on server x232.
cu devad(0x0e40,01) path(2) resource(X232.XCA1)

X232.XCA1: cu xcasna
interface local(1) options 'adapternumber=1'
device(00) xcasna eth1
end X232.XCA1
```

```
#FLEX-ES resource definition extract for instance zсна on server x232.  
cu devad(0x0e40,01) path(2) resource(X232.XCA2)
```

```
X232.XCA2:cu xcasna  
interface local(1) options 'adapternumber=1'  
device(00) xcasna eth2  
end X232.XCA2
```

11.7 MVS device definitions

We used the HCD dialogs available under TSO ISPF to define the LAN devices to MVS. We did not need to create any channel paths or control units; we only needed to define the devices.

We first needed to know what IODF was used to IPL the system, since it is the proper starting point when adding new devices. To determine which data set to use, we first used the MVS operator command **D IPLINFO**:

```
00-Z0S1 d iplinfo  
IEE254I 13.42.05 IPLINFO DISPLAY FRAME LAST F E SYS=Z0S1  
SYSTEM IPLED AT 12.15.39 ON 06/28/2003  
RELEASE z/OS 01.04.00 LICENSE =z/OS  
USED LOAD01 IN SYS1.IPLPARM ON 0302  
ARCHLVL =2 MTLSHARE =N  
IEASYM LIST =01  
IEASYS LIST =00 (0P)  
IODF DEVICE 0302  
IPL DEVICE 0300 VOLUME ZP4RS1
```

From this display, we learned that member LOAD01 of data set SYS1.IPLPARM on device 302 was used to IPL the system. We obtained the volume serial number of device 302 with the MVS **d u** command:

```
00-Z0S1 d u,,,302,1  
IEE457I 13.47.29 UNIT STATUS FRAME LAST F E SYS=Z0S1  
UNIT TYPE STATUS VOLSER VOLSTATE  
0302 3390 A ZP4SYS PRIV/RSDNT
```

We then used the ISPF **browse** command to view the LOAD01 member of SYS1.IPLPARM on volume ZP4SYS, as shown in Figure 11-4 on page 159.

[illegible]

Figure 11-4 Browse entry panel

The SYS1.IPLPARM(LOAD01) member displayed as follows:

```

000001 *-+---1---2---+---3---+---4---+---5---+---6---+---7-
000002 IODF      07 SYS1      ZOS      10
000003 SYSCAT    ZP4SYS113CCATALOG.ZP4.MASTER
000004 SYSPARM    00
000005 IEASYM     01
000006 PARMLIB   ZPLEX.PARMLIB                      ZP4RS1
000007 PARMLIB   ADCD.ZOSV1R4.PARMLIB                ZP4RS1
000008 PARMLIB   SYS1.PARMLIB                        ZP4RS1
000009 NUCLEUS   1
000010 SYSPLEX   ZPLEX
*****Bottom of Data*****

```

The first three parameters of line 2 (line 1 is a comment) tell z/OS initialization which IODF to use. The file must be cataloged in the master catalog. The third column is the first-level qualifier of the data set. The first and second columns are combined to form the second-level qualifier. The system was IPLed using SYS1.IODF07.

Armed with this information, we started the HCD dialog and selected option number 1, specifying SYS1.IODF07 as the starting point, as shown in Figure 11-5 on page 160.

```
OS/390 Release 9 HCD
Command ===>_____
                Hardware Configuration
Select one of the following.

_1 1. Define,modify,or view configuration data
   2. Activate or process configuration data
   3. Print or compare configuration data
   4. Create or view graphical configuration report
   5. Migrate configuration data
   6. Maintain I/O definition files
   7. Query supported hardware and installed UIMs
   8. Getting started with this dialog
   9. What's new in this release

For options 1 to 5,specify the name of the IODF to be used.

I/O definition file ...'SYS1.IODF07'+
```

Figure 11-5 The initial HCD screen

Since we wanted to add a new device, we selected option 5 from the Define, Modify, or View Configuration Data menu, as shown in Figure 11-6.

[illegible]

Figure 11-6 Define, Modify or View Configuration Data dialog

We were then presented with the list of devices defined for this system. We pressed F11 to add a new device. New information cannot be added to an existing IODF, however. HCD must copy the existing IODF to a 'work' IODF before any changes can be made. As soon as we pressed F11 for the first time, we were prompted to allow HCD to make the copy, as shown in Figure 11-7 on page 161.

```

Esxxxxxxxxxxxxxxxx Create Work I/O Definition File sxxxxxxxxxxxxxxxxN
e
e
e The current IODF is a production IODF and therefore cannot be
e updated.To create a new work IODF based on the current
e production IODF,specify the following values.
e
e IODF name ..... 'SYS1.IODF07.WORK'
e
e Volume serial number ._____+
e
e Space allocation ...1024 (Number of 4K blocks)
e
e Activity logging ...Yes (Yes or No)
e
e F1=Help F2=Split F3=Exit F4=Prompt F9=Swap F12=Cancel
DxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxM

```

Figure 11-7 Initial Create Work IODF dialog

We elected to change the name to 'SYS1.IODF08.WORK' and we changed the space allocation and activity logging fields as shown in Figure 11-8.

```

Esxxxxxxxxxxxxxxxx Create Work I/O Definition File sxxxxxxxxxxxxxxxxN
e
e
e The current IODF is a production IODF and therefore cannot be
e updated.To create a new work IODF based on the current
e production IODF,specify the following values.
e
e IODF name ..... 'SYS1.IODF08.WORK'
e
e Volume serial number .ZP4SYS +
e
e Space allocation ...256 (Number of 4K blocks)
e
e Activity logging ...No (Yes or No)
e
e F1=Help F2=Split F3=Exit F4=Prompt F9=Swap F12=Cancel
DxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxM

```

Figure 11-8 Modified Create Work IODF dialog

After the work IODF was created, HCD then displayed the Add Device dialog, as shown in Figure 11-9 on page 162.

```

Esxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx Add Device xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxN
e
e
e Specify or revise the following values.
e
e Device number . . . . . (0000 -FFFF)
e Number of devices . . . . .
e Device type . . . . . +
e
e Serial number . . . . .
e Description . . . . .
e
e Volume serial number . . . . . (for DASD)
e
e Connected to CUs . . . . . +
e
e
e F1=Help F2=Split F3=Exit F4=Prompt F5=Reset F9=Swap
e F12=Cancel
DsxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxM

```

Figure 11-9 Initial Add Device dialog

To create a z/OS device address for the osasna device at device number 0E60, we filled out the dialog as shown in Figure 11-10.

```

Esxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx Add Device xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxN
e
e
e Specify or revise the following values.
e
e Device number . . . . . E60 (0000 -FFFF)
e Number of devices . . . . . 1
e Device type . . . . . OSA +
e
e Serial number . . . . .
e Description . . . . .
e
e Volume serial number . . . . . (for DASD)
e
e Connected to CUs . . . . . +
e
e
e F1=Help F2=Split F3=Exit F4=Prompt F5=Reset F9=Swap
e F12=Cancel
DsxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxM

```

Figure 11-10 Modified Add Device dialog

When we pressed ENTER, we received a dialog prompt asking whether the new device should be added to the z/OS configuration. The dialog lists all of the OS configurations we have defined (in this case there is only one). This is shown in Figure 11-11 on page 163

```

Esxxxxxxxxx Define Device to Operating System Configuration sxxxxxxxxxN
e Row 1 of 1 e
e Command ==> _____ Scroll ==>PAGE e
e e
e Select OSs to connect or disconnect devices,then press Enter. e
e e
e Device number . : 0E60 Number of devices : 1 e
e Device type . . : OSA e
e e
e / Config.ID Type Description Defined e
e / ZOS MVS zos lpar-based sysplex e
e *****Bottom of data ***** e

```

Figure 11-11 Define Device to Operating System Configuration selection

We selected “ZOS” by typing a forward slash (‘/’) in the column marked ‘/’ and pressing ENTER. When the dialog shown in Figure 11-12 was displayed, we selected ‘1’ for connect.

```

Esxxxxxxxxx Actions on selected operating systems sxxxxxxxxxN
e e
e Select by number or action code and press Enter. e
e e
e 1_ 1.Select (connect,change). . . . . (s) e
e 2.Disconnect from OS . . . . . (n) e
e e
e e
e F1=Help F2=Split F3=Exit F9=Swap F12=Cancel e
DsxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxM

```

Figure 11-12 Actions on selected operating systems dialog

When the dialog shown in Figure 11-13 was presented, we simply pressed Enter.

```

Esxxxxxxxxxxxxxxxxxxxxxxxx Define Device Parameters / Features sxxxxxxxxxxxxxxxxN
e Row 1 of 3 e
e Command ==> _____ Scroll ==>PAGE e
e e
e Specify or revise the values below. e
e e
e Configuration ID . ZOS zos lpar-based sysplex e
e Device number . . : 0E40 Number of devices : 1 e
e Device type . . . : OSA e
e e
e Parameter / e
e Feature Value P Req. Description e
e OFFLINE No Device considered online or offline at IPL e
e DYNAMIC Yes Device has been defined to be dynamic e
e LOCANY No UCB can reside in 31 bit storage e
e *****Bottom of data ***** e

```

Figure 11-13 Initial Define Device Parameters / Features dialog

We then had the chance to add the device to the system’s esoterics, as shown in Figure 11-14 on page 164. Again, we just pressed Enter.

[illegible]

Figure 11-14 Modified Define Device Parameters / Features dialog

We were then returned to the *Define Device to Operating System Configuration* dialog; had we defined more operating system configurations, we could have selected them and connected the device in the same way. This time, instead of selecting an operating system configuration, we simply pressed Enter, and the process was complete.

11.8 Adding an XCA device

An xcasna device is defined to z/OS as a channel-to-channel adapter with one device address.

Defining an XCA follows each step we just described and the steps are identical except for the Add Device dialog that defines the device, shown in Figure 11-15 on page 165.

```

Essssssssssssssssssssssssssssssssss Add Device sssssssssssssssssssssssssssssssN
e
e
e Specify or revise the following values.
e
e Device number . . . . . E40_ (0000 -FFFF)
e Number of devices . . . . . 1_
e Device type . . . . . CTC _____ +
e
e Serial number . . . . . _____
e Description . . . . . _____
e
e Volume serial number . . . . . _____ (for DASD)
e
e Connected to CUs . . _____ +
e
e
e F1=Help F2=Split F3=Exit F4=Prompt F5=Reset F9=Swap
e F12=Cancel
DssssssssssssssssssssssssssssssssssM

```

Figure 11-15 Add Device dialog

When we had finished adding the device, we pressed F3 until we returned to the Hardware Configuration menu, shown in Figure 11-16, where we started.

```

Hardware Configuration
Select one of the following.
2  1. Define,modify,or view configuration data
   2. Activate or process configuration data
   3. Print or compare configuration data
   4. Create or view graphical configuration report
   5. Migrate configuration data
   6. Maintain I/O definition files
   7. Query supported hardware and installed UIMs
   8. Getting started with this dialog
   9. What's new in this release

For options 1 to 5,specify the name of the IODF to be used.

I/O definition file . . . 'SYS1.IODF08.WORK' +

```

Figure 11-16 HCD primary menu

The remaining task is to turn the work IODF into a production IODF. To begin that process, we selected item 2, Activate or Process Configuration Data, as shown in Figure 11-17 on page 166.

```

E sssss Activate or Process Configuration Data sssss N
e
e
e Select one of the following tasks.
e
e _1 1. Build production I/O definition file
e    2. Build IOCDs
e    3. Build IOCP input data set
e    4. Create JES3 initialization stream data
e    5. View active configuration
e    6. Activate or verify configuration
e       dynamically
e    7. Activate configuration sysplex-wide
e    8. Activate switch configuration
e    9. Save switch configuration
e   10. Build OS configuration data set
e   11. Build and manage S/390 microprocessor
e       IOCDs and IPL attributes
e
e F1=Help F2=Split F3=Exit F9=Swap
e F12=Cancel
D sssssssssssssssssssssssssssssssssssssssssssssssss M

```

Figure 11-17 Activate or Process Configuration Data dialog

We selected option 1 and received the prompt shown in Figure 11-18.

```

E sssssssssssss Build Production I/O Definition File sssssssssssss N
e
e
e Specify the following values, and choose how to continue.
e
e Work IODF name . . . : 'SYS1.IODF08.WORK'
e
e Production IODF name . 'SYS1.IODF08'
e Volume serial number. ZP4SYS +
e
e Continue using as current IODF:
e 2 1.The work IODF in use at present
e   2.The new production IODF specified above
e
e
e
e F1=Help F2=Split F3=Exit F4=Prompt F9=Swap F12=Cancel
D sssssssssssssssssssssssssssssssssssssssssssssssss M

```

Figure 11-18 Build Production I/O Definition File dialog

We supplied the file name 'SYS1.IODF08' for the new production IODF and pressed ENTER. We received the prompt shown in Figure 11-19 on page 167 and pressed Enter again.

```

Esxxxxxxxxxxxxxxxxxxxx Define Descriptor Fields xxxxxxxxxxxxxxxxxxxxxxxN
e
e
e Specify or revise the following values.
e
e Production IODF name . : 'SYS1.IODF08'
e
e Descriptor field 1 . . . SYS1
e Descriptor field 2 . . . IODF08
e F1=Help F2=Split F3=Exit F5=Reset F9=Swap
e F12=Cancel
DxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxM

```

Figure 11-19 Define Descriptor Fields dialog

We then received a confirmation that the new production IODF had been created. We pressed F3 the number of times needed to return to the ISPF main menu.

We next used the ISPF editor to modify line 2 of member LOAD01 of SYS1.IPLPARM on ZP4SYS, as shown in Figure 11-20. We changed the '07' to '08' so that at the next IPL the system would use SYS1.IODF08, the IODF we just created.

```

EDIT SYS1.IPLPARM(LOAD01) - 01.07 Columns 00001 00080
Command ==> Scroll==>PAGE
***** Top of Data *****
000001 *---+---1---+---2---+---3---+---4---+---5---+---6---+---7-
000002 IODF 08 SYS1 ZOS 10
000003 SYSCAT ZP4SYS113CCATALOG.ZP4.MASTER
000004 SYSPARM 00
000005 IEASYM 01
000006 PARMLIB ZPLEX.PARMLIB ZP4RS1
000007 PARMLIB ADCD.ZOSV1R4.PARMLIB ZP4RS1
000008 PARMLIB SYS1.PARMLIB ZP4RS1
000009 NUCLEUS 1
000010 SYSPLEX ZPLEX

```

Figure 11-20 IPLPARM modification

We then IPLed the system to ensure that the osasna device was available to VTAM.

11.9 Defining the SSCPs, the subareas, and the network

Before the SSCPs can go into session, they must be named and assigned unique IDs. Member ATCSTR00 of VTAMLIB contains the VTAM start options. We used the values from Tables 1-3 to enter the values unique to each system. Every system must specify the same NETID.

The VTAM starting member for system zos1 is shown in Figure 11-21 on page 168.

```

EDIT      ZPLEX.ZOS1.VTAMLST(ATCSTR00) - 10.22      Columns 00001 00072
Command ==>                                         Scroll ==>PAGE
*****Top of Data *****
000100 CONFIG=00,SUPP=NOSUP,                          X
000200 SSCPID=01,NOPROMPT,                             X
000300 HOSTSA=1,MAXSUBA=31,                             X
000400 SSCPNAME=ZOS1SSCP,HOSTPU=ZOS1$PU,               X
000500 NETID=FLEX,                                     X
000600 NODETYPE=NN,                                    X
000700 DYNLU=YES,                                       X
000800 CRPLBUF=(208,,15,,1,16),                        X
000900 IOBUF=(400,508,19,,1,20),                       X
001000 LFBUF=(104,,0,,1,1),                             X
001100 LPBUF=(64,,0,,1,1),                             X
001200 SFBUF=(163,,0,,1,1)

```

Figure 11-21 System zos1 VTAM start options (ATCSTR00)

The values for SSCPID, HOSTSA, SSCPNAME, and HOSTPU came from the tables we presented in “Selecting VTAM configuration information” on page 156 and “Developing the adapter and device matrix” on page 156. We set NETID to an arbitrary string. The remaining values are CBIPO defaults. The start members for the other systems are shown in Figure 11-22 through Figure 11-26 on page 169.

```

CONFIG=00,SUPP=NOSUP,                          X
SSCPID=02,NOPROMPT,                             X
HOSTSA=2,MAXSUBA=31,                             X
SSCPNAME=ZOS2SSCP,HOSTPU=ZOS2$PU,               X
NETID=FLEX,                                     X
NODETYPE=NN,                                    X
DYNLU=YES,                                       X
CRPLBUF=(208,,15,,1,16),                        X
IOBUF=(400,508,19,,1,20),                       X
LFBUF=(104,,0,,1,1),                             X
LPBUF=(64,,0,,1,1),                             X
SFBUF=(163,,0,,1,1)

```

Figure 11-22 System zos2 VTAM start options (ATCSTR00)

```

CONFIG=00,SUPP=NOSUP,                          X
SSCPID=03,NOPROMPT,                             X
HOSTSA=3,MAXSUBA=31,                             X
SSCPNAME=ZOS3SSCP,HOSTPU=ZOS3$PU,               X
NETID=FLEX,                                     X
NODETYPE=NN,                                    X
DYNLU=YES,                                       X
CRPLBUF=(208,,15,,1,16),                        X
IOBUF=(400,508,19,,1,20),                       X
LFBUF=(104,,0,,1,1),                             X
LPBUF=(64,,0,,1,1),                             X
SFBUF=(163,,0,,1,1)

```

Figure 11-23 System zos3 VTAM start options (ATCSTR00)

CONFIG=00,SUPP=NOSUP,	X
SSCPID=08,NOPROMPT,	X
HOSTSA=8,MAXSUBA=31,	X
SSCPNAME=Z232SSCP,HOSTPU=Z232\$PU,	X
NETID=FLEX,	X
NODETYPE=NN,	X
DYNLU=YES,	X
CRPLBUF=(208,,15,,1,16),	X
IOBUF=(400,508,19,,1,20),	X
LFBUF=(104,,0,,1,1),	X
LPBUF=(64,,0,,1,1),	X
SFBUF=(163,,0,,1,1)	

Figure 11-24 System z232 VTAM start options (ATCSTR00)

CONFIG=00,SUPP=NOSUP,	X
SSCPID=06,NOPROMPT,	X
HOSTSA=6,MAXSUBA=31,	X
SSCPNAME=ZSNASSCP,HOSTPU=ZSNA\$PU,	X
NETID=FLEX,	X
NODETYPE=NN,	X
DYNLU=YES,	X
CRPLBUF=(208,,15,,1,16),	X
IOBUF=(400,508,19,,1,20),	X
LFBUF=(104,,0,,1,1),	X
LPBUF=(64,,0,,1,1),	X
SFBUF=(163,,0,,1,1)	

Figure 11-25 System zsna VTAM start options (ATCSTR00)

CONFIG=00,SUPP=NOSUP,	X
SSCPID=07,NOPROMPT,	X
HOSTSA=7,MAXSUBA=31,	X
SSCPNAME=Z370SSCP,HOSTPU=Z370\$PU,	X
NETID=FLEX,	X
NODETYPE=NN,	X
DYNLU=YES,	X
CRPLBUF=(208,,15,,1,16),	X
IOBUF=(400,508,19,,1,20),	X
LFBUF=(104,,0,,1,1),	X
LPBUF=(64,,0,,1,1),	X
SFBUF=(163,,0,,1,1)	

Figure 11-26 System z370 VTAM start options (ATCSTR00)

11.10 CDRM major nodes

Each system defining an XCA major node also activates a CDRM (Cross Domain Resource Manager) node that defines other VTAM systems in the network. Because zos1 has responsibility to engage RTP sessions with zos2 and zos3 as needed, zos2 and zos3 do not need to be in the CDRM node list. They also neither define nor activate a CDRM node of their own.

Each of the other four systems has a CDRM major node; the definitions are identical with the exception of the first statement in the definition, which names the CDRM major node on the activating system. The member names in the various VTAMLST partitioned data sets are ZOS1CDRM, ZSNACDRM, Z232CDRM, and Z370CDRM. The major node is activated by being listed in ATCCON00.

For simplicity's sake we list only one of the CDRM members, in Figure 11-27. Each of the four SSCPs listed in the node definition activates its own version of the major node.

ZOS1CDRM	VBUILD	TYPE=CDRM	
ZOS1SSCP	CDRM	SUBAREA=01,	X
		ISTATUS=ACTIVE,	X
		CDRSC=OPT,	X
		CDRDYN=YES,	X
		RECOVERY=YES	
Z370SSCP	CDRM	SUBAREA=07,	X
		ISTATUS=ACTIVE,	X
		CDRSC=OPT,	X
		CDRDYN=YES,	X
		RECOVERY=YES	
Z232SSCP	CDRM	SUBAREA=08,	X
		ISTATUS=ACTIVE,	X
		CDRSC=OPT,	X
		CDRDYN=YES,	X
		RECOVERY=YES	
ZSNASSCP	CDRM	SUBAREA=06,	X
		ISTATUS=ACTIVE,	X
		CDRSC=OPT,	X
		CDRDYN=YES,	X
		RECOVERY=YES	

Figure 11-27 CDRM node used by zos1, zsna, z232, and z370

We specified CDRDYN=YES and CDRSC=OPT to permit CDRM to dynamically define cross-domain or cross-network resources as needed. RECOVERY=YES and ISTATUS=ACTIVE are the defaults.

11.11 Defining the VTAM XCA major nodes

To make the SNA connection between the two systems, each system must declare an XCA major node consisting of VBUILD, PORT, GROUP, LINE, and PU statements. The PORT statement defines the hardware on the local system: the MVS device address, the type of adapter (Token Ring or Ethernet), and the SAPADDR that can be used by other systems to contact this PORT. The PU statement(s) each define a system on the LAN that can be contacted using the defined PORT.

The major node definition is contained in a member of that system's VTAMLST and is activated by being listed in member ATCCON00.

11.11.1 Defining the SAPADDRS

The combination of a MACADDR and a SAPADDR uniquely identifies a VTAM host node on the LAN. One point of confusion is that SAPADDR is specified on both the PORT and the PU statements in the VTAMLST member that defines an XCA major node.

The SAPADDR specified on the PORT statement is the access point where VTAM will listen. Thus, if we specified a SAPADDR of 8 on the PORT statement on system xeon, then xeon would begin to listen for packets on access point 8.

A PORT statement also specifies an MVS device number on the CUADDR operand; that device number, in turn, is connected to a specific NIC by a FLEX-ES emulated control unit definition. The PORT CUADDR specification thus determines the MACADDR on which VTAM listens.

While the PORT statement defines the addresses of the activating system, the PU statements define the addresses of other hosts that may be contacted on the LAN. Each PU statement specifies both MACADDR and SAPADDR and thus uniquely addresses another VTAM SSCP on the network.

For system z370 to contact system zos1, the MACADDR specified on its PU definition for system zos1 must be the MACADDR of the NIC identified by system zos1's PORT CUADDR operand. The SAPADDR on z370's PU definition for system zos1 must match the SAPADDR specified on system zos1's PORT SAPADDR specification.

11.11.2 XCA major node definition for zos1

We defined the XCA major node on system zos1 as shown in Figure 11-28.

OSAE60E	VBUILD	TYPE=XCA	
OPE60E	PORT	CUADDR=E60,ADAPNO=1,SAPADDR=4,MEDIUM=CSMACD, DELAY=0,TIMER=30	X
OGE60E	GROUP	DIAL=NO	
OLZ232	LINE	USER=SNA,ISTATUS=ACTIVE	
OPZ232	PU	ISTATUS=ACTIVE, MACADDR=0002B3A6E2FA, PUTYPE=5, SAPADDR=4, SUBAREA=08, TGN=1	X X X X X
OLZ370	LINE	USER=SNA,ISTATUS=ACTIVE	
OPZ370	PU	ISTATUS=ACTIVE, MACADDR=0002B3AC6E27, PUTYPE=5, SAPADDR=4, SUBAREA=07, TGN=1	X X X X X
OLZSNA	LINE	USER=SNA,ISTATUS=ACTIVE	
OPZSNA	PU	ISTATUS=ACTIVE, MACADDR=0002B3A6E2FB, PUTYPE=5, SAPADDR=8, SUBAREA=06, TGN=1	X X X X X

Figure 11-28 Member OSAE60E in ZPLEX.ZOS1.VTAMLST on zos1

The PORT macro describes the OSA adapter on server xeon that system zos1 uses to connect to the LAN. Note that the 'ADAPNO=1' field matches the "options 'adapternumber=1'" statement on the FLEX-ES emulated control unit definition XEON.OSASNA1. The CUADDR matches the device number specified in the FLEX-ES system definition and when we ran the HCD dialogs. MEDIUM=CSMACD specifies an Ethernet adapter.

DELAY=0 tells VTAM not to wait before transmitting low-priority data to the PU. TIMER=30 (the default) tells VTAM how long to wait for a response after the node is activated.

The PU and LINE macros describe the other systems on the LAN that can be reached via the PORT. Each is uniquely identified by MACADDR and SAPADDR; note that these values are from Table 1. For each PU statement, the MACADDR is the MAC address of the NIC in the other system; the SAPADDR is the SAPADDR specified on the PORT statement on the other system.

PUTYPE=5 tells VTAM that the PU being contacted is a host; TGN=1 (the default) names the transmission group that represents the connection between this node and the partner. Had we specified something other than the default, we would have needed to code the CPNAME and NETID parameters as well.

11.11.3 XCA major node definition for z370

We defined the XCA major node on system z370 for its xcasna connection as shown in Figure 11-29.

XCAE40E	VBUILD	TYPE=XCA	
XPE40E	PORT	CUADDR=E40,ADAPNO=1,SAPADDR=4,MEDIUM=CSMACD, DELAY=0,TIMER=30	X
XGE40E	GROUP	DIAL=NO	
XLZ232	LINE	USER=SNA,ISTATUS=ACTIVE	
XPZ232	PU	ISTATUS=ACTIVE, MACADDR=0002B3A6E2FA, PUTYPE=5, SAPADDR=4, SUBAREA=08, TGN=1	X X X X X
XLZ0S1	LINE	USER=SNA,ISTATUS=ACTIVE	
XPZ0S1	PU	ISTATUS=ACTIVE, MACADDR=0002B3C884E0, PUTYPE=5, SAPADDR=4, SUBAREA=01, TGN=1	X X X X X
XLZSNA	LINE	USER=SNA,ISTATUS=ACTIVE	
XPZSNA	PU	ISTATUS=ACTIVE, MACADDR=0002B3A6E2FB, PUTYPE=5, SAPADDR=8, SUBAREA=06, TGN=1	X X X X X

Figure 11-29 Member XCAE40E in ZPLEX.Z370.VTAMLST on z370

11.11.4 XCA major node definition for z232

We defined the XCA major node on system z232 as shown in Figure 11-30 on page 173.

XCAE40E	VBUILD	TYPE=XCA	
XPE40E	PORT	CUADDR=E40,ADAPN0=1,SAPADDR=4,MEDIUM=CSMACD, DELAY=0,TIMER=30	X
XGE40E	GROUP	DIAL=NO	
XLZ0S1	LINE	USER=SNA,ISTATUS=ACTIVE	
XPZ0S1	PU	ISTATUS=ACTIVE, MACADDR=0002B3C884E0, PUTYPE=5, SAPADDR=4, SUBAREA=01, TGN=1	X X X X X
XLZ370	LINE	USER=SNA,ISTATUS=ACTIVE	
XPZ370	PU	ISTATUS=ACTIVE, MACADDR=0002B3AC6E27, PUTYPE=5, SAPADDR=4, SUBAREA=07, TGN=1	X X X X X
XLZSNA	LINE	USER=SNA,ISTATUS=ACTIVE	
XPZSNA	PU	ISTATUS=ACTIVE, MACADDR=0002B3A6E2FB, PUTYPE=5, SAPADDR=8, SUBAREA=06, TGN=1	X X X X X

Figure 11-30 Member XCAE40E in ZPLEX.Z232.VTAMLST on z232

11.11.5 XCA major node definition for zsna

Finally, Figure 11-31 on page 174 illustrates how we defined the XCA major node on system zsna.

XCAE40	VBUILD	TYPE=XCA	
XPE40E	PORT	CUADDR=E40,ADAPNO=1,SAPADDR=8,MEDIUM=CSMACD, DELAY=0,TIMER=30	X
XGE40E	GROUP	DIAL=NO	
XLZOS1	LINE	USER=SNA,ISTATUS=ACTIVE	
XPZOS1	PU	ISTATUS=ACTIVE, MACADDR=0002B3C884E0, PUTYPE=5, SAPADDR=4, SUBAREA=01, TGN=1	X X X X X
XLZ370	LINE	USER=SNA,ISTATUS=ACTIVE	
XPZ370	PU	ISTATUS=ACTIVE, MACADDR=0002B3AC6E27, PUTYPE=5, SAPADDR=4, SUBAREA=07, TGN=1	X X X X X
XGZ232	LINE	USER=SNA,ISTATUS=ACTIVE	
XPZ232	PU	ISTATUS=ACTIVE, MACADDR=0002B3A6E2FA, PUTYPE=5, SAPADDR=4, SUBAREA=08, TGN=1	X X X X X

Figure 11-31 Member XCAE40E in ZPLEX.ZSNA.VTAMLST on zsna

11.12 Bringing up a link

On system z370, we varied the XCA node active and received the response shown in Figure 11-32.

```
Z370 v net,act,id=xcae40e
Z370 IST097I VARY ACCEPTED
Z370 IST093I XCAE40E ACTIVE
```

Figure 11-32 Vary Activate of the XCA node on z370

Not too much can be made of this response. We then went over to the zos1 system, and activated its OSA link. We received the response shown in Figure 11-33.

```
ZOS1 v net,act,id=osae60e
ZOS1 IST097I VARY ACCEPTED
ZOS1 IST093I OSAE60E ACTIVE
ZOS1 IST464I LINK STATION XGEP60 HAS CONTACTED Z370SSCP SA 7
ZOS1 IST093I XGEP60 ACTIVE
ZOS1 IST324I INACT IN PROGRESS WITH ID =Z370SSCP DUE TO DACTCDRM REQUEST
ZOS1 IST105I Z370SSCP NODE NOW INACTIVE
ZOS1 IST324I ACTIVATE IN PROGRESS WITH ID =Z370SSCP DUE TO ACTCDRM REQUEST
ZOS1 IST093I Z370SSCP ACTIVE
```

Figure 11-33 Vary Activate of the OSA node on zos1

This is what is expected. Once both sides of the link have been activated, the message we want to see is the IST464I message for the SSCP. As soon as zos1 showed Z370SSCP active, the system console on z370 displayed the messages shown in Figure 11-34.

```
Z370 IST464I LINK STATION XGEP60 HAS CONTACTED ZOS1SSCP SA 1
Z370 IST093I XGEP60 ACTIVE
Z370 IST093I ZOS1SSCP ACTIVE
```

Figure 11-34 Successful contact

At this point the two VTAMs are in session using SNA-over-Ethernet as the connecting medium. We configured and activated XCA major nodes on systems z232 and zsn1 in the same manner.

11.13 TSO cross-domain logon

A cross-domain logon uses a terminal attached to one system to access an application running on another system. A CDRM major node defines the systems that can participate. In order for the cross-domain logon to be successful, each system must specify unique application id and LU names. To set the TSO application id, we renamed the APPL statement that defines the TSO ACB, as shown in Figure 11-35.

```
ZOS1TSO APPL AUTH=(NOACQ,NOBLOCK,PASS,NOTCAM,NVPACE,TSO,NOP0), X
EAS=1,ACBNAME=TSO
*
```

Figure 11-35 Extract from zos1's major application node

We named the TSO applications after the system: ZOS1TSO, ZOS2TSO, ZOS3TSO, Z232TSO, ZSN1TSO, and Z370TSO. We assigned unique names to every local terminal on each system. VTAM requires the unique names in order to prevent duplicate sessions. Once the unique terminal names and TSO application names were in place, we could use a locally attached terminal on any of the six z/OS systems and log onto TSO on any other system.

11.14 RTP activation

To verify that RTP activation was effective, we used a local terminal attached to system zsn1 and requested a TSO session on zos3. The connection was made, and we were prompted for user ID and password. The console messages are shown in Figure 11-36.

```
ZOS3 IST1488I ACTIVATION OF RTP CNR00006 AS PASSIVE TO FLEX.ZOS1SSCP
ZOS1 IST1488I ACTIVATION OF RTP CNR00008 AS ACTIVE TO FLEX.ZOS3SSCP
ZOS3 $HASP100 FLEX ON TSQINRDR
ZOS3 $HASP373 FLEX STARTED
```

Figure 11-36 Activation messages

VTAM created the necessary resources and made the connection between ZOS1SSCP and ZOS3SSCP because they are both members of a base sysplex. Thus, any system on the LAN has access to all three members of the sysplex, and all members of the sysplex can access the systems on the LAN, provided system zos1 is operational.

11.15 JES2 network job entry

We also set up JES2 NJE using the LAN connections. When we finished, we could transmit and receive to or from any of the six systems and we could submit jobs, route their execution, and route their output among any of the six systems.

We defined an application major node on each system for JES2. Each APPL statement was named for its system. The ACBNAME (one of the definitions is shown in Figure 11-37) is referred to in the JES2 initialization statements and the JES2 \$SN command used to start a node session.

ZOS1NJE VBUILD TYPE=APPL	
ZOS1NJE1 APPL ACBNAME=ZOS1NJE1,AUTH=(ACQ,BLOCK,VPACE),EAS=1, VPACING=0	X

Figure 11-37 JES2 NJE application definition – member ZOS1NJE of ZPLEX.ZOS1.VTAMLST

The NJE network has four nodes, numbered 1-4, and named ZOS1, Z370, Z232, and Z370.

Since the sysplex is a MAS setup, the three members comprise only one node. Jobs are routed to the node with /*ROUTE XEQ statements; jobs are routed to a specific system in the sysplex (MAS) with /*JOBPARM SYSAFF statements. The NJE statements from JES2PARM on zos1 are as follows:

```
/* NJE statement extract from JES2PARM on zos1. */
LOGON(1)  APPLID=ZOS1NJE1,          /*ACCESS CONTROL BLOCK (ACB)NAME ohwnc*/
          LOG=N,                    /*Monitor VTAM interface (Y)      ohwnc*/
          TRACEIO=NO                /*Trace I/O Activity (YES)        ohwnc*/
LOGON(2)  APPLID=ZOS2NJE1,          /*ACCESS CONTROL BLOCK (ACB)NAME ohwnc*/
          LOG=N,                    /*Monitor VTAM interface (Y)      ohwnc*/
          TRACEIO=NO                /*Trace I/O Activity (YES)        ohwnc*/
LOGON(3)  APPLID=ZOS3NJE1,          /*ACCESS CONTROL BLOCK (ACB)NAME ohwnc*/
          LOG=N,                    /*Monitor VTAM interface (Y)      ohwnc*/
          TRACEIO=NO                /*Trace I/O Activity (YES)        ohwnc*/

NODE(1)NAME=ZOS1
NODE(2)NAME=Z370
NODE(3)NAME=Z232
NODE(4)NAME=ZSNA
APPL(ZOS1NJE1)NODE=1
APPL(ZOS2NJE1)NODE=1
APPL(ZOS3NJE1)NODE=1
APPL(Z370NJE1)NODE=2
APPL(Z232NJE1)NODE=3
APPL(ZSNANJE1)NODE=4
LINE(1)UNIT=SNA
LINE(2)UNIT=SNA
LINE(3)UNIT=SNA
LINE(4)UNIT=SNA
LINE(5)UNIT=SNA
LINE(6)UNIT=SNA
LINE(7)UNIT=SNA
NJEDF     DELAY=120,                /*Max.Msg Delay Time              ohwnc*/
          HDRBUF=(LIMIT=100,        /*Number of NJE header +trailer   ohwnc*/
          WARN=80),                /*buffers                          */
          JRNUM=2,                  /*Warning Threshold                ohwnc*/
          JTNUM=2,                  /*Number of job receivers          hwnc*/
          LINENUM=5,                /*Number of job xmitters           hwnc*/
          MAILMSG=NO,               /*Number of lines for NJE         hwnc*/
          /*Don't automatically issue ohwnc*/
```

```

/*notification message */
MAXHOP=0, /*Num.of iterations to limit hwnc*/
/*hoping in network */
/*0 means no hop counting */
NODENUM=50, /*Max.number of NJE nodes nc*/
OWNNODE=1, /*this node's number c*/
PATH=1, /*number of paths hwnc*/
RESTMAX=8000000, /*Max.resistance tolerance ohwnc*/
RESTNODE=150, /*this node's resistance ohwnc*/
RESTTOL=300, /*Alt.resistance tolerance ohwnc*/
SRNUM=2, /*number of sysout receivers hwnc*/
STNUM=2, /*number of sysout xmitters hwnc*/
TIMETOL=30 /*Time variation between clocks ohwnc*/

```

The statements from JES2PARM on z370 are as follows:

```

/* NJE statement extract from JES2PARM on z370. */
LOGON(1) APPLID=Z370NJE1, /*ACCESS CONTROL BLOCK (ACB)NAME ohwnc*/
LOG=N, /*Monitor VTAM interface (Y) ohwnc*/
/*or discontinue monitoring (N) */
TRACEIO=NO /*Trace I/O Activity (YES) ohwnc*/

NODE(1)NAME=ZOS1
NODE(2)NAME=Z370
NODE(3)NAME=Z232
NODE(4)NAME=ZSNA
APPL(ZOS1NJE1)NODE=1
APPL(ZOS2NJE1)NODE=1
APPL(ZOS3NJE1)NODE=1
APPL(Z232NJE1)NODE=3
APPL(ZSNANJE1)NODE=4
LINE(1)UNIT=SNA
LINE(2)UNIT=SNA
LINE(3)UNIT=SNA
LINE(4)UNIT=SNA
NJEDEF DELAY=120, /*Max.Msg Delay Time ohwnc*/
HDRBUF=(LIMIT=100, /*Number of NJE header +trailer ohwnc*/
/*buffers */
WARN=80), /*Warning Threshold ohwnc*/
JRNUM=2, /*Number of job receivers hwnc*/
JTNUM=2, /*Number of job xmitters hwnc*/
LINENUM=5, /*Number of lines for NJE hwnc*/
MAILMSG=NO, /*Don't automatically issue ohwnc*/
/*notification message */
MAXHOP=0, /*Num.of iterations to limit hwnc*/
/*hoping in network */
/*0 means no hop counting */
NODENUM=50, /*Max.number of NJE nodes nc*/
OWNNODE=2, /*this node's number c*/
PATH=1, /*number of paths hwnc*/
RESTMAX=8000000, /*Max.resistance tolerance ohwnc*/
RESTNODE=150, /*this node's resistance ohwnc*/
RESTTOL=300, /*Alt.resistance tolerance ohwnc*/
SRNUM=2, /*number of sysout receivers hwnc*/
STNUM=2, /*number of sysout xmitters hwnc*/
TIMETOL=30 /*Time variation between clocks ohwnc*/

```

The NJE JES2PARM statements from zсна are as follows:

```

/* NJE statement extract from JES2PARM on zсна. */
LOGON(1) APPLID=ZSNANJE1, /*ACCESS CONTROL BLOCK (ACB)NAME ohwnc*/
LOG=N, /*Monitor VTAM interface (Y) ohwnc*/

```

```

                                /*or discontinue monitoring (N)      */
                                /*Trace I/O Activity (YES)           ohwnc*/
                                /*                                  */
                                /*                                  */
TRACEIO=NO

NODE(1)NAME=ZOS1
NODE(2)NAME=Z370
NODE(3)NAME=Z232
NODE(4)NAME=ZSNA
APPL(ZOS1NJE1)NODE=1
APPL(ZOS2NJE1)NODE=1
APPL(ZOS3NJE1)NODE=1
APPL(Z370NJE1)NODE=2
APPL(Z232NJE1)NODE=3
LINE(1)UNIT=SNA
LINE(2)UNIT=SNA
LINE(3)UNIT=SNA
LINE(4)UNIT=SNA

NJedef    DELAY=120,          /*Max.Msg Delay Time           ohwnc*/
           HDRBUF=(LIMIT=100, /*Number of NJE header +trailer ohwnc*/
           WARN=80),          /*buffers                       */
           JRNUM=2,           /*Warning Threshold            ohwnc*/
           JTNUM=2,           /*Number of job receivers hwnc */
           LINENUM=5,         /*Number of job xmitters       hwnc*/
           MAILMSG=NO,        /*Number of lines for NJE      hwnc*/
           MAXHOP=0,          /*Don't automatically issue    ohwnc*/
           NODENUM=50,        /*notification message         */
           OWNNODE=4,         /*Num.of iterations to limit   hwnc*/
           PATH=1,            /*hoping in network            */
           RESTMAX=8000000,    /*0 means no hop counting      */
           RESTNODE=150,       /*Max.number of NJE nodes      nc*/
           RESTTOL=300,        /*this node's number           c*/
           SRNUM=2,           /*number of paths              hwnc*/
           STNUM=2,           /*Max.resistance tolerance     ohwnc*/
           TIMETOL=30,         /*this node's resistance       ohwnc*/
                                /*Alt.resistance tolerance     ohwnc*/
                                /*number of sysout receivers   hwnc*/
                                /*number of sysout xmitters    hwnc*/
                                /*Time variation between clocks ohwnc*/
                                /*Times are in 1/100 sec.unless spec'd*/

```

The NJE JES2PARM statements from z232 are as follows:

```

/* NJE statement extract from JES2PARM on z232. */
LOGON(1)  APPLID=Z232NJE1,    /*ACCESS CONTROL BLOCK (ACB)NAME ohwnc*/
          LOG=N,              /*Monitor VTAM interface (Y)      ohwnc*/
                                /*or discontinue monitoring (N)    */
                                /*Trace I/O Activity (YES)        ohwnc*/
                                /*                                  */
                                /*                                  */
TRACEIO=NO

NODE(1) NAME=ZOS1
NODE(2) NAME=Z370
NODE(3) NAME=Z232
NODE(4) NAME=ZSNA
APPL(ZOS1NJE1) NODE=1
APPL(ZOS2NJE1) NODE=1
APPL(ZOS3NJE1) NODE=1
APPL(Z370NJE1) NODE=2
APPL(ZSNANJE1) NODE=4
LINE(1) UNIT=SNA
LINE(2) UNIT=SNA
LINE(3) UNIT=SNA
LINE(4) UNIT=SNA

```

NJEDEF	DELAY=120,	/*Max.Msg Delay Time	ohwnc*/
	HDRBUF=(LIMIT=100,	/*Number of NJE header +trailer	ohwnc*/
		/*buffers	*/
	WARN=80),	/*Warning Threshold	ohwnc*/
	JRNUM=2,	/*Number of job receivers	hwnc*/
	JTNUM=2,	/*Number of job xmitters	hwnc*/
	LINENUM=5,	/*Number of lines for NJE	hwnc*/
	MAILMSG=NO,	/*Don't automatically issue	ohwnc*/
		/*notification message	*/
	MAXHOP=0,	/*Num.of iterations to limit	hwnc*/
		/*hoping in network	*/
		/*0 means no hop counting	*/
	NODENUM=50,	/*Max.number of NJE nodes	nc*/
	OWNNODE=3,	/*this node's number	c*/
	PATH=1,	/*number of paths	hwnc*/
	RESTMAX=8000000,	/*Max.resistance tolerance	ohwnc*/
	RESTNODE=150,	/*this node's resistance	ohwnc*/
	RESTTOL=300,	/*Alt.resistance tolerance	ohwnc*/
	SRNUM=2,	/*number of sysout receivers	hwnc*/
	STNUM=2,	/*number of sysout xmitters	hwnc*/
	TIMETOL=30	/*Time variation between clocks	ohwnc*/

We changed the OWNNODE= statement on each system to match the node number we had assigned; we then let the SMF system ID match the node name list in order to select a node number. We had to cold start the systems in order for the change of OWNNODE to become effective.

11.15.1 NJE verification

On z370, zsna, and z232 we placed the following commands in the automatic startup script:

```
$SLOGON1
$SLINE1
```

We then used the console at zos1 to start the NJE network, producing the following console activity:

```
ZOS1 $slogon1
ZOS1 $HASP881 LOGON1 APPLID=ZOS1NJE1,STATUS=INACTIVE
ZOS1 $HASP091 LOGON1 IS ACTIVE
ZOS1 $sline1-3
ZOS1 $HASP880 LINE1 UNIT=SNA,STATUS=INACTIVE
ZOS1 $HASP880 LINE2 UNIT=SNA,STATUS=INACTIVE
ZOS1 $HASP880 LINE3 UNIT=SNA,STATUS=INACTIVE
ZOS1 $sn,a=z370nje1
ZOS1 $HASP000 OK
ZOS1 $HASP200 Z370 STARTED ON LNE3 SESSION Z370NJE1 BFSZ=03840
ZOS1 $HASP524 L3.JT1 INACTIVE
ZOS1 $HASP524 L3.JT2 INACTIVE
ZOS1 $HASP534 L3.ST1 INACTIVE
ZOS1 $HASP534 L3.ST2 INACTIVE
ZOS1 $sn,a=z232nje1
ZOS1 $HASP000 OK
ZOS1 $HASP200 Z232 STARTED ON LNE2 SESSION Z232NJE1 BFSZ=03840
ZOS1 $HASP524 L2.JT1 INACTIVE
ZOS1 $HASP524 L2.JT2 INACTIVE
ZOS1 $HASP534 L2.ST1 INACTIVE
ZOS1 $HASP534 L2.ST2 INACTIVE
ZOS1 $sn,a=zsnaenje1
```

```

ZOS1 $HASP000 OK
ZOS1 $HASP200 ZSNA STARTED ON LNE1 SESSION ZSNANJE1 BFSZ=03840
ZOS1 $HASP524 L1.JT1 INACTIVE
ZOS1 $HASP524 L1.JT2 INACTIVE
ZOS1 $HASP534 L1.ST1 INACTIVE
ZOS1 $HASP534 L1.ST2 INACTIVE.

```

To verify that NJE was operating properly we transmitted a message from zos3 to a user on z370, with the following results at the TSO terminal:

```

INMX000I 0 message and 1 data records sent as 4 records to Z370.IBMUSER
INMX001I Transmission occurred on 06/28/2003 at 19:54:09.
READY

```

On zos1, we saw the following message traffic indicating the file had been sent:

```

ZOS1 $HASP530 FLEX ON L3.ST1 4 RECORDS
ZOS1 $HASP534 L3.ST1 INACTIVE

```

On z370, we saw the following message indicating the file had been received:

```

19.54.39 $HASP540 FLEX ON L1.SR1 FROM FLEX AT ZOS1 4 RECORDS

```

Finally, we logged onto z370 and submitted a job to be run on sysplex member zos3 at node ZOS1, producing the following output:

```

SDSF OUTPUT DISPLAY TESTNJE JOB00851 DSID 2 LINE 0 COLUMNS 02-133
COMMAND INPUT ==>SCROLL ==>PAGE
J E S 2 J O B L O G --S Y S T E M Z O S 3 --N O D E Z O S 1
19.59.20 JOB00851 ----SATURDAY,28 JUN 2003 ----
19.59.20 JOB00851 $HASP373 TESTNJE STARTED -INIT SYS -CLASS A -SYS ZOS3
19.59.20 JOB00851 $HASP395 TESTNJE ENDED
- - - - - JES2 JOB STATISTICS - - - - -
  28 JUN 2003 JOB EXECUTION DATE
    4 CARDS READ
    21 SYSOUT PRINT RECORDS
    0 SYSOUT PUNCH RECORDS
    1 SYSOUT SPOOL KBYTES
    0.00 MINUTES EXECUTION TIME
    1 //TESTNJE JOB 0,FLEX,CLASS=A,NOTIFY=FLEX
JOB00851
/*ROUTE XEQ ZOS1
00011015
/*JOBPARM SYSAFF=ZOS3
00012015
  2 //S1 EXEC PGM=IEFBR14
00020009
IEF142I TESTNJE S1 -STEP WAS EXECUTED -COND CODE 0000
IEF373I STEP/S1 /START 2003179.1959
IEF374I STEP/S1 /STOP 2003179.1959 CPU OMIN 00.02SEC SRB OMIN 00.00SEC VIRT
4K SYS 232K EXT 4K SYS 10960K
IEF375I JOB/TESTNJE /START 2003179.1959
IEF376I JOB/TESTNJE /STOP 2003179.1959 CPU OMIN 00.02SEC SRB OMIN 00.00SEC

```

As should be expected, the job was routed to the ZOS1 node, where an initiator on system zos3 picked it up for execution. The job output was then returned to the spool on z370, where we displayed it with SDSF.

11.16 FLEX-ES resource definitions

This section provides complete listings of the FLEX-ES definitions used on all the systems discussed in this chapter.

11.16.1 The base sysplex

We present the complete FLEX-ES resource files for all three servers. On server xeon, we defined resources in one file and then used a separate file to define the system instance. Server xeon ran z/VM and we initialized all three members of the sysplex as z/VM guest machines.

Server xeon: Resources

```
#####
# Unit-Record
#####

XEON.3211A: cu 3211 interface local(3) cloned
           device( 0x00 ) 3211 OFFLINE
end XEON.3211A

XEON.2821A:cu 2821 interface local(3) cloned
           device( 0x00 ) 2540R OFFLINE
           device( 0x01 ) 2540P OFFLINE
           device( 0x02 ) 1403-N1 OFFLINE
           device( 0x03 ) 1403-N1 OFFLINE
end XEON.2821A

XEON.3203A: cu 3203 interface local(3) cloned
           device( 0x00 ) 3203-5 OFFLINE
end XEON.3203A

XEON.4245A: cu 4245 interface local(3) cloned
           device( 0x00 ) 4245 OFFLINE
end XEON.4245A

#####
# Tape
#####

XEON.3490A: cu 3490 interface local(1)
           device( 0x00 )          3490-E /dev/rmt/ctape1
           device( 0x01 - 0x0f ) 3490-E OFFLINE
end XEON.3490A

XEON.3490B: cu 3490 interface local(1)
           device( 0x00 - 0x0f ) 3490-E OFFLINE
end XEON.3490B

XEON.3480A: cu 3480 interface local(1)
           device( 0x00 - 0x0f ) 3480 OFFLINE
end XEON.3480A

XEON.3480B: cu 3480 interface local(1)
           device( 0x00 - 0x0f ) 3480 OFFLINE
end XEON.3480B

XEON.3480C: cu 3480 interface local(1)
```

```
device( 0x00 - 0x0f ) 3480 OFFLINE
end XEON.3480C
```

```
XEON.AFLIB1: cu 3480 interface local(1)
device( 0x00 - 0x0f ) 3480 OFFLINE
end XEON.AFLIB1
```

```
XEON.AFLIB2: cu 3480 interface local(1)
device( 0x00 - 0x0f ) 3480 OFFLINE
end XEON.AFLIB2
```

```
XEON.3422A: cu 3422 interface local(1)
device( 0x00 - 0x0f ) 3422 OFFLINE
end XEON.3422A
```

```
#####
# 3278
#####
```

```
XEON.3274A: cu 3274 interface local(1)
device ( 0x00 ) 3278 x20
device ( 0x01 ) 3278 x21
device ( 0x02 ) 3278 x22
device ( 0x03 ) 3278 x23
device ( 0x04 ) 3278 x24
device ( 0x05 ) 3278 x25
device ( 0x06 ) 3278 x26
device ( 0x07 ) 3278 x27
end XEON.3274A
```

```
XEON.3274B: cu 3274 interface local(1)
device ( 0x00 ) 3278 x700-1
device ( 0x01 ) 3278 tso1-1
device ( 0x02 ) 3278 tso1-2
device ( 0x03 ) 3278 tso1-3
device ( 0x04 ) 3278 tso1-4
device ( 0x05 ) 3278 tso1-5
device ( 0x06 ) 3278 tso1-6
device ( 0x07 ) 3278 tso1-7
device ( 0x08 - 0x1f ) 3278 tso1
end XEON.3274B
```

```
XEON.3274C: cu 3274 interface local(1)
device ( 0x00 ) 3278 x700-2
device ( 0x01 ) 3278 tso2-1
device ( 0x02 ) 3278 tso2-2
device ( 0x03 ) 3278 tso2-3
device ( 0x04 ) 3278 tso2-4
device ( 0x05 ) 3278 tso2-5
device ( 0x06 ) 3278 tso2-6
device ( 0x07 ) 3278 tso2-7
end XEON.3274C
```

```
XEON.3274D: cu 3274 interface local(1)
device ( 0x00 ) 3278 x700-3
device ( 0x01 ) 3278 tso3-1
device ( 0x02 ) 3278 tso3-2
device ( 0x03 ) 3278 tso3-3
device ( 0x04 ) 3278 tso3-4
device ( 0x05 ) 3278 tso3-5
```

```

    device ( 0x06 ) 3278 tso3-6
    device ( 0x07 ) 3278 tso3-7
end XEON.3274D

XEON.FC00: cu 3274 interface local(1)
    device ( 0x00 ) 3278 fc00
end XEON.FC00

XEON.FC01: cu 3274 interface local(1) # zos1 mstcons
    device ( 0x00 ) 3278 fc01
end XEON.FC01

XEON.FC02: cu 3274 interface local(1) # zos2 mstcons
    device ( 0x00 ) 3278 fc02
end XEON.FC02

XEON.FC03: cu 3274 interface local(1) # zos3 mstcons
    device ( 0x00 ) 3278 fc03
end XEON.FC03

#####
# Network
#####

XEON.OSA1: cu osa interface local(1) options 'adapternumber=0' #zos1 tcpip
    device( 0x00 ) osa /dev/net1
    device( 0x01 ) osa OFFLINE
end XEON.OSA1

XEON.OSA2: cu osa interface local(1) options 'adapternumber=0' #zos2 tcpip
    device( 0x00 ) osa /dev/net6
    device( 0x01 ) osa OFFLINE
end XEON.OSA2

XEON.OSA3: cu osa interface local(1) options 'adapternumber=0' #zos3 tcpip
    device( 0x00 ) osa /dev/net5
    device( 0x01 ) osa OFFLINE
end XEON.OSA3

XEON.OSASNA1: cu osasna interface local(1) options 'adapternumber=1'
    device( 0x00 ) osasna /dev/net5
end XEON.OSASNA1

#####
# DASD
#####

XEON.3990A: cu 3990 options 'trackcachesize=1200 '
    interface local(3)
    interface network(3)

    device( 0x00 ) 3390-3 /aos/systems/xeon/z4res1
    device( 0x01 ) 3390-3 /aos/systems/xeon/z4res2
    device( 0x02 ) 3390-3 /aos/systems/xeon/zplex1
    device( 0x03 ) 3390-3 /aos/systems/xeon/zwork3
    device( 0x04 ) 3390-3 /aos/systems/xeon/zs4uss
    device( 0x05 ) 3390-3 /aos/systems/xeon/zs4pg0
    device( 0x06 ) 3390-3 /aos/systems/xeon/zs4pg1
    device( 0x07 ) 3390-3 /aos/systems/xeon/zs4pg2
    device( 0x08 ) 3390-3 /aos/systems/xeon/zs4sms

```

```

device( 0x09 ) 3390-3 /aos/systems/xeon/z4ckpt
device( 0x0a ) 3390-3 /aos/systems/xeon/zs4log
device( 0x0b ) 3390-3 /aos/systems/xeon/zs4cds
device( 0x0c ) 3390-3 /aos/systems/xeon/sp1001
device( 0x0d ) 3390-3 /aos/systems/xeon/aflib1
device( 0x0e ) 3390-3 /aos/systems/xeon/aflib2
device( 0x0f ) 3390-3 /aos/systems/xeon/aflib3
device( 0x10 ) 3390-3 /aos/systems/xeon/sp1002
device( 0x11 ) 3390-3 /aos/systems/xeon/zzres1
device( 0x12 ) 3390-3 /aos/systems/xeon/zzres2
device( 0x13 ) 3390-3 /aos/systems/xeon/zz39m1
device( 0x14 ) 3390-3 /aos/systems/xeon/zwork1
device( 0x15 ) 3390-3 /aos/systems/xeon/z4uss1
device( 0x16 ) 3390-3 /aos/systems/xeon/z4db21
device( 0x17 ) 3390-2 /aos/systems/xeon/z4cic1
device( 0x18 ) 3390-2 /aos/systems/xeon/z4ims1
device( 0x19 ) 3390-3 /aos/systems/xeon/sares1
device( 0x1a ) 3390-3 /aos/systems/xeon/z4was1
device( 0x1b ) 3390-3 /aos/systems/xeon/z4was2
device( 0x1c ) 3390-3 /aos/systems/xeon/z4dis1
device( 0x1d ) 3390-3 /aos/systems/xeon/z4dis2
device( 0x1e ) 3390-3 /aos/systems/xeon/z4dis3
device( 0x1f ) 3390-3 /aos/systems/xeon/z4dis4
end XEON.3990A

```

```

XEON.3990B: cu 3990
interface local(3)
interface network(3)

```

```

device( 0x00 ) 3390-3 /aos/systems/xeon/aotso1
device( 0x01 ) 3390-3 /aos/systems/xeon/zwork2
device( 0x02 ) 3390-3 /aos/systems/xeon/ts0001
device( 0x03 ) 3390-3 /aos/systems/xeon/ts0002
device( 0x04 ) 3390-3 /aos/systems/xeon/ts0003
device( 0x05 ) 3390-3 /aos/systems/xeon/ts0004
device( 0x06 ) 3390-3 /aos/systems/xeon/ts0005
device( 0x07 ) 3390-3 /aos/systems/xeon/ts0006
device( 0x08 - 0x1f ) 3390-3 OFFLINE
end XEON.3990B

```

```

XEON.3990E: cu 3990 options 'trackcachesize=1200'
interface local(3)
interface network(1)

```

```

device( 0x00 ) 3390-3 /aos/systems/xeon/zp4rs1
device( 0x01 ) 3390-3 /aos/systems/xeon/zp4rs2
device( 0x02 ) 3390-3 /aos/systems/xeon/zp4sys
device( 0x03 ) 3390-3 /aos/systems/xeon/zp4ck1
device( 0x04 ) 3390-3 /aos/systems/xeon/zp4ck2
device( 0x05 ) 3390-3 /aos/systems/xeon/zp4sms
device( 0x06 ) 3390-3 /aos/systems/xeon/zp4cds
device( 0x07 ) 3390-3 /aos/systems/xeon/zp4log
device( 0x08 ) 3390-3 /aos/systems/xeon/zp4pg0
device( 0x09 ) 3390-3 /aos/systems/xeon/zp4pg1
device( 0x0a ) 3390-3 /aos/systems/xeon/zp4pg2
device( 0x0b ) 3390-3 /aos/systems/xeon/zp4wk1
device( 0x0c ) 3390-3 /aos/systems/xeon/zp4wk2
device( 0x0d ) 3390-3 /aos/systems/xeon/zp4wk3
device( 0x0e ) 3390-3 /aos/systems/xeon/zp4us1
device( 0x0f ) 3390-3 /aos/systems/xeon/zp4ds1

```

```

device( 0x10 ) 3390-3 /aos/systems/xeon/zp4ds2
device( 0x11 ) 3390-3 /aos/systems/xeon/zp4ds3
device( 0x12 ) 3390-3 /aos/systems/xeon/zp4ds4
device( 0x13 ) 3390-3 /aos/systems/xeon/zp4sp1
device( 0x14 ) 3390-3 /aos/systems/xeon/zp4sp2
device( 0x15 ) 3390-3 /aos/systems/xeon/zp4db2
device( 0x16 ) 3390-3 /aos/systems/xeon/zp4cic
device( 0x17 ) 3390-3 /aos/systems/xeon/zp4ims
device( 0x18 ) 3390-3 /aos/systems/xeon/af0lib
device( 0x19 ) 3390-3 /aos/systems/xeon/af0mnt
device( 0x1a - 0x1f ) 3390-3 OFFLINE
end XEON.3990E

```

```

XEON.3990F: cu 3990 options 'trackcachesize=1000'
interface local(3)
interface network(1)

```

```

device( 0x00 ) 3390-3 /aos/systems/adcd/z4res1
device( 0x01 ) 3390-3 /aos/systems/adcd/z4res2
device( 0x02 ) 3390-3 /aos/systems/adcd/os39m1
device( 0x03 ) 3390-3 /aos/systems/adcd/z4uss1
device( 0x04 ) 3390-3 /aos/systems/adcd/z4db21
device( 0x05 ) 3390-2 /aos/systems/adcd/z4ims1
device( 0x06 ) 3390-2 /aos/systems/adcd/z4cic1
device( 0x07 ) 3390-3 /aos/systems/adcd/z4dis1
device( 0x08 ) 3390-3 /aos/systems/adcd/z4dis2
device( 0x09 ) 3390-3 /aos/systems/adcd/z4dis3
device( 0x0a ) 3390-3 /aos/systems/adcd/z4dis4
device( 0x0b ) 3390-3 /aos/systems/adcd/z4was1
device( 0x0c ) 3390-3 /aos/systems/adcd/z4was2
device( 0x0d ) 3390-3 /aos/systems/adcd/sares1
device( 0x0e - 0x1f ) 3390-3 OFFLINE
end XEON.3990F

```

```

XEON.3990D: cu 3990
interface local(3)
interface network(3)

```

```

device( 0x00 ) 3390-3 /aos/systems/xeon/lrx390
device( 0x01 ) 3390-3 /aos/systems/xeon/lrx001
device( 0x02 ) 3390-3 /aos/systems/xeon/lrxfs1
device( 0x03 ) 3390-3 /aos/systems/xeon/lrxfs2
device( 0x04 ) 3390-3 /aos/systems/xeon/lrxfs3
device( 0x05 ) 3390-3 /aos/systems/xeon/lrxfs4
device( 0x06 ) 3390-3 /aos/systems/xeon/lrxfs5
device( 0x07 ) 3390-3 /aos/systems/xeon/lrxfs6
device( 0x08 ) 3390-3 /aos/systems/xeon/lrxfs7
device( 0x09 ) 3390-3 /aos/systems/xeon/lrxfs8
device( 0x0a - 0x1f ) 3390-3 OFFLINE
end XEON.3990D

```

```

#####
#CTCs for base sysplex
#####

```

```

XEON.CTC01: cu ctc interface local(2)
device(00) ctc
end XEON.CTC01
XEON.CTC02: cu ctc interface local(2)
device(00) ctc

```

```

end XEON.CTC02
XEON.CTC03: cu ctc interface local(2)
    device(00) ctc
end XEON.CTC03
XEON.CTC04: cu ctc interface local(2)
    device(00) ctc
end XEON.CTC04
XEON.CTC05: cu ctc interface local(2)
    device(00) ctc
end XEON.CTC05
XEON.CTC06: cu ctc interface local(2)
    device(00) ctc
end XEON.CTC06
XEON.CTC07: cu ctc interface local(2)
    device(00) ctc
end XEON.CTC07
XEON.CTC08: cu ctc interface local(2)
    device(00) ctc
end XEON.CTC08
XEON.CTC09: cu ctc interface local(2)
    device(00) ctc
end XEON.CTC09
XEON.CTC10: cu ctc interface local(2)
    device(00) ctc
end XEON.CTC10
XEON.CTC11: cu ctc interface local(2)
    device(00) ctc
end XEON.CTC11
XEON.CTC12: cu ctc interface local(2)
    device(00) ctc
end XEON.CTC12
XEON.CTC13: cu ctc interface local(2)
    device(00) ctc
end XEON.CTC13
XEON.CTC14: cu ctc interface local(2)
    device(00) ctc
end XEON.CTC14
XEON.CTC15: cu ctc interface local(2)
    device(00) ctc
end XEON.CTC15
XEON.CTC16: cu ctc interface local(2)
    device(00) ctc
end XEON.CTC16
XEON.CTC17: cu ctc interface local(2)
    device(00) ctc
end XEON.CTC17
XEON.CTC18: cu ctc interface local(2)
    device(00) ctc
end XEON.CTC18
XEON.CTC19: cu ctc interface local(2)
    device(00) ctc
end XEON.CTC19
XEON.CTC20: cu ctc interface local(2)
    device(00) ctc
end XEON.CTC20

end xeon

```

Server xeon: System xeon

```
system xeon:
```

```

feature lpar
lparnum(1)
memsize(0x1E0000)
cachesize(0x2000)
tracesize(0x100)
instset(z)

cpu(0)
cpu(1)

channel(0)localbyte
channel(1)local
channel(2)local
channel(3)localosa

#UR
cu devad(0x0005,01) path(0) resource(XEON.3211A)
cu devad(0x000C,04) path(0) resource(XEON.2821A)
cu devad(0x0010,01) path(0) resource(XEON.3203A)
cu devad(0x0018,01) path(0) resource(XEON.4245A)
#tape
cu devad(0x0180,16) path(1) resource(XEON.3422A)
cu devad(0x0560,16) path(2) resource(XEON.3490A)
cu devad(0x0570,16) path(2) resource(XEON.3490B)
cu devad(0x0600,16) path(2) resource(XEON.AFLIB1)
cu devad(0x0610,16) path(2) resource(XEON.AFLIB2)
cu devad(0xFF00,16) path(2) resource(XEON.3480A)
cu devad(0xFF10,16) path(2) resource(XEON.3480C)
#dasd
cu devad(0x0300,32) path(1) resource(XEON.3990E)
cu devad(0x0320,32) path(1) resource(XEON.3990A)
cu devad(0x0340,32) path(1) resource(XEON.3990B)
cu devad(0x1200,32) path(1) resource(XEON.3990D)
#3270
cu devad(0x0700,32) path(2) resource(XEON.3274B)
cu devad(0xFC01,01) path(2) resource(XEON.FC01)
#osa
cu devad(0x0e00,02) path(3) resource(XEON.OSA1)
cu devad(0x0e60,01) path(3) resource(XEON.OSASNA1)
#sysplex ctc
cu devad(0x1031,01) path(1) resource(XEON.CTC01)
cu devad(0x1032,01) path(1) resource(XEON.CTC02)
cu devad(0x1033,01) path(1) resource(XEON.CTC12)
cu devad(0x1034,01) path(1) resource(XEON.CTC11)
cu devad(0x1035,01) path(1) resource(XEON.CTC20)
cu devad(0x1036,01) path(1) resource(XEON.CTC19)
cu devad(0x1037,01) path(1) resource(XEON.CTC09)
cu devad(0x1038,01) path(1) resource(XEON.CTC10)

end xeon

```

Server x232: Resources and systems z232 and zсна

For server x232, we used one file to contain both system definitions as well as the resource definitions. Since the systems had one volume with the same VOLSER, we put each system's copy on a separate control unit.

```

#####
## System
#####

```

```

system z232:

memsize( 0x0a0000 )
cachesize( 0x2000 )
tracesize( 0x100 )
instset(esa)
cpu( 0 )

channel( 0 ) localbyte
channel( 1 ) local
channel( 2 ) local
channel( 3 ) localosa

cu devad(0x000c,04) path(0) resource(AOS2821.0A)
cu devad(0x0a80,32) path(1) resource(AOS3990A)
cu devad(0x0300,16) path(1) resource(AOS3990B)
cu devad(0x0310,01) path(1) resource(AOS3990C)
cu devad(0x0560,32) path(1) resource(AOS3480.0A)
cu devad(0x0500,16) path(1) resource(AOS3490.0A)
cu devad(0x0800,16) path(1) resource(AFMNT.A)
cu devad(0x0810,16) path(1) resource(AFMNT.B)
cu devad(0x0820,16) path(1) resource(AFMNT.C)
cu devad(0x0830,16) path(1) resource(AFMNT.D)
cu devad(0x0700,32) path(1) resource(AOS3274.0A)
cu devad(0x0e00,02) path(3) resource(AOSLCS.0A)
cu devad(0x0e40,01) path(2) resource(X232.XCA1)

end z232

system zсна:

memsize(0x0a0000)
cachesize(0x2000)
tracesize(0x100)
instset(esa)
cpu(0)

channel(0) localbyte
channel(1) local
channel(2) local
channel(3) localosa

cu devad(0x0a80,32) path(1) resource(AOS3990A)
cu devad(0x0300,16) path(1) resource(AOS3990B)
cu devad(0x0310,01) path(1) resource(AOS3990D)
cu devad(0x0700,32) path(1) resource(AOS3274.1A)
cu devad(0x0e40,01) path(2) resource(X232.XCA2)

end zсна

#####
#Resources
#####

resources x232:

AOS2821.0A: cu 2821
    interface local(1)
    device(00) 2540R OFFLINE
    device(01) 2540P OFFLINE

```

```

    device(02) 1403-N1 OFFLINE
    device(03) 1403-N1 OFFLINE
end AOS2821.0A

AOS3274.0A: cu 3274
    interface local(1)
    device( 00 ) 3278 master-0
    device( 0x01 - 0x1f ) 3278 tso-0
end AOS3274.0A

AOS3274.1A: cu 3274
    interface local(1)
    device( 00 ) 3278 master-1
    device(0x01 -0x1f ) 3278 tso-1
end AOS3274.1A

AOS3480.0A: cu 3480
    interface local(1)
    device ( 0x00 - 0x1f ) 3480 OFFLINE
end AOS3480.0A

AFMNT.A: cu 3480
    interface local(1)
    device ( 0x00 - 0x0f ) 3480 OFFLINE
end AFMNT.A

AFMNT.B: cu 3480
    interface local(1)
    device ( 0x00 - 0x0f ) 3480 OFFLINE
end AFMNT.B
AFMNT.C: cu 3480
    interface local(1)
    device ( 0x00 - 0x0f ) 3480 OFFLINE
end AFMNT.C

AFMNT.D: cu 3480
    interface local(1)
    device ( 0x00 - 0x0f ) 3480 OFFLINE
end AFMNT.D

AOS3490.0A: cu 3490
    interface local(1)
    device( 0x00 - 0x0f ) 3490-E OFFLINE
end AOS3490.0A

AOS3990A: cu 3990 options 'trackcachesize=3000'
    interface local(2)
    device(0x00) 3390-3 /dev/raw/vz4res1
    device(0x01) 3390-3 /dev/raw/vz4res2
    device(0x02) 3390-3 /dev/raw/vz4uss1
    device(0x03) 3390-2 /dev/raw/vz4cic1
    device(0x04) 3390-2 /dev/raw/vz4ims1
    device(0x05) 3390-3 /dev/raw/vz4db21
    device(0x06) 3390-3 /dev/raw/vz4dis1
    device(0x07) 3390-3 /dev/raw/vz4dis2
    device(0x08) 3390-3 /dev/raw/vz4dis3
    device(0x09) 3390-3 /dev/raw/vz4dis4
    device(0x0a) 3390-3 /dev/raw/vaotsol
    device(0x0b) 3390-3 /dev/raw/vzp4dmp
    device(0x0c) 3390-3 /dev/raw/vsares1

```

```

device(0x0d) 3390-3 /dev/raw/vz4was1
device(0x0e) 3390-3 /dev/raw/vz4was2
device(0x0f) 3390-3 /dev/raw/vz4pag0
device(0x10) 3390-3 /dev/raw/vz4pag1
device(0x11) 3390-3 /dev/raw/vz4pag2
device(0x12) 3390-3 /dev/raw/vz4pag3
device(0x13) 3390-3 /dev/raw/vz4ckpt
device(0x14) 3390-3 /dev/raw/vspool0
device(0x15) 3390-3 /dev/raw/vwork01
device(0x16) 3390-3 /dev/raw/vwork02
device(0x17) 3390-3 /dev/raw/vzzres1
device(0x18) 3390-3 /dev/raw/vzzres2
device(0x19) 3390-3 /dev/raw/vzz39m1
device( 0x01a - 0x01f ) 3390-3 OFFLINE
end AOS3990A

AOS3990B: cu 3990 options 'trackcachesize=3000'
interface local(2)
device(0x00) 3390-3 /dev/raw/vs4res1
device(0x01) 3390-3 /dev/raw/vs4res2
device(0x02) 3390-3 /dev/raw/vs4uss1
device(0x03) 3390-2 /dev/raw/vs4cic1
device(0x04) 3390-2 /dev/raw/vs4ims1
device(0x05) 3390-3 /dev/raw/vs4db21
device(0x06) 3390-3 /dev/raw/vs4dis1
device(0x07) 3390-3 /dev/raw/vs4dis2
device(0x08) 3390-3 /dev/raw/vs4dis3
device(0x09) 3390-3 /dev/raw/vs4dis4
device(0x0a) 3390-3 /dev/raw/vs4was1
device(0x0b) 3390-3 /dev/raw/vs4was2
device(0x0c) 3390-3 /dev/raw/vs4sp10
device( 0x0d - 0x0f ) 3390-3 OFFLINE
end AOS3990B

AOS3990C: cu 3990 interface local(1) options 'trackcachesize=500'
device( 0x00 ) 3390-3 /dev/raw/vos39m1
end AOS3990C

AOS3990D: cu 3990 interface local(1) options 'trackcachesize=500'
device( 0x00 ) 3390-3 /dev/raw/vsm39m1
end AOS3990D

AOSLCS.OA: cu osa
options 'adapternumber=0,ipaddress=192.168.100.30'
interface local(1)
device (00) osa eth0
device (01) osa OFFLINE
end AOSLCS.OA

X232.XCA1: cu xcasna interface local(1) options 'adapternumber=1'
device (00) xcasna eth1
end X232.XCA1

X232.XCA2: cu xcasna interface local(1) options 'adapternumber=1'
device (00) xcasna eth2
end X232.XCA2

end x232

```

Server x370: Resources

resources x370:

```
#####
# Unit-Record
#####

X370.3211A: cu 3211 interface local(3) cloned    #0005
            device( 0x00 ) 3211 OFFLINE
end X370.3211A

X370.2821A: cu 2821 interface local(3) cloned
            device( 0x00 ) 2540R OFFLINE          #000C
            device( 0x01 ) 2540P OFFLINE          #000D
end X370.2821A

X370.3203A: cu 3203 interface local(3) cloned
            device ( 0x00 ) 3203-5 OFFLINE        #000E
            device ( 0x01 ) 3203-5 OFFLINE        #000F
end X370.3203A

X370.4245A: cu 4245 interface local(3) cloned    #0018
            device( 0x00 ) 4245 OFFLINE
end X370.4245A

#####
# Tape
#####

X370.3490A: cu 3490 interface local(1)           #560
            device( 0x00 ) 3490-E /dev/rmt/ctape1
            device( 0x01 ) 3490-E /dev/rmt/ctape2
            device( 0x02 - 0x0f ) 3490-E OFFLINE
end X370.3490A

X370.3490B: cu 3490 interface local(3) cloned    #570
            device( 0x00 - 0x0f ) 3490-E OFFLINE
end X370.3490B

X370.3422A: cu 3422 interface local(3) cloned    #180
            device( 0x00 - 0x0f ) 3422 OFFLINE
end X370.3422A

X370.3480A: cu 3480 interface local(3) cloned    #FF00
            device( 0x00 - 0x0f ) 3480 OFFLINE
end X370.3480A

X370.AFLIB1: cu 3480 interface local(3) cloned   #600 -aflib
            device( 0x00 - 0x0f ) 3480 OFFLINE
end X370.AFLIB1

X370.AFLIB2: cu 3480 interface local(3) cloned   #610 -aflib
            device( 0x00 - 0x0f ) 3480 OFFLINE
end X370.AFLIB2

X370.AFMNTA: cu 3480 interface local(3) cloned   #800 -AFMNT
            device( 0x00 - 0x0f ) 3480 OFFLINE
end X370.AFMNTA

X370.AFMNTB: cu 3480 interface local(3) cloned   #810 -AFMNT
```

```

    device( 0x00 - 0x0f ) 3480 OFFLINE
end X370.AFMNTB

X370.AFMNTC: cu 3480 interface local(3) cloned    #820X370.3274C -AFMNT
    device( 0x00 - 0x0f ) 3480 OFFLINE
end X370.AFMNTC

X370.AFMNTD: cu 3480 interface local(3) cloned    #830X370.3274C -AFMNT
    device( 0x00 - 0x0f ) 3480 OFFLINE
end X370.AFMNTD

#####
# 3278
#####

X370.3274A: cu 3274 interface local(3) cloned    #020
    device( 0x00 ) 3278 vm_console
    device( 0x01 - 0x1f ) 3278 zvm
end X370.3274A

X370.3274B: cu 3274 interface local(3) cloned    #080
    device( 0x00 ) 3278 x080
    device( 0x01 ) 3278 x081
    device( 0x02 ) 3278 x082
    device( 0x03 ) 3278 x083
    device( 0x04 ) 3278 x084
    device( 0x05 ) 3278 x085
    device( 0x06 ) 3278 x086
    device( 0x07 ) 3278 x087
end X370.3274B

X370.3274C: cu 3274 interface local(3) cloned
    device( 0x00 ) 3278 x0700
    device( 0x01 - 0x1f ) 3278 tso
end X370.3274C

X370.FC00: cu 3274 interface local(1)
    device( 0x00 ) 3278 fc00
end X370.FC00

X370.FC01: cu 3274 interface local(1)
    device( 0x00 ) 3278 fc01
end X370.FC01

X370.FC02: cu 3274 interface local(1)
    device( 0x00 ) 3278 fc02
end X370.FC02

X370.FC03: cu 3274 interface local(1)
    device( 0x00 ) 3278 fc03
end X370.FC03

#####
# Communications
#####

X370.OSA1: cu osa interface local(1) options 'adapternumber=0'
    device(00) osa /dev/net2
    device(01) osa OFFLINE
end X370.OSA1

```

```

X370.OSA2: cu osasna interface local(1) options 'adapternumber=1'
    device(00) osasna /dev/net2
end X370.OSA2

X370.OSA3: cu osa interface local(1) options 'adapternumber=0'
    device(00) osa /dev/net4                                #3com
    device(01) osa OFFLINE
end X370.OSA3

X370.XCA1: cu xcasna interface local(1) options 'adapternumber=1'
    device(00) xcasna /dev/net3
end X370.XCA1

X370.3172A: cu 3172 interface local(1) options 'adapternumber=0'
    device ( 0x00 ) 3172 /dev/net5                            #3com
    device ( 0x01 ) 3172 OFFLINE
end X370.3172A

X370.CTCJES: cu ctc interface local(1) interface network(1)
    device( 0x00 ) ctc
end X370.CTCJES

#####
# DASD
#####

X370.3990A: cu 3990
    interface local(3)
    interface network(1)
    device( 0x00 ) 3390-3 /aos/systems/x370/430res
    device( 0x01 ) 3390-3 /aos/systems/x370/430w01
    device( 0x02 ) 3390-3 /aos/systems/x370/430w02
    device( 0x03 ) 3390-3 /aos/systems/x370/430w03
    device( 0x04 ) 3390-3 /aos/systems/x370/430w04
    device( 0x05 ) 3390-3 /aos/systems/x370/sles8d
    device( 0x06 ) 3390-3 /aos/systems/x370/dosres
    device( 0x07 ) 3390-3 /aos/systems/x370/syswk1
    device( 0x08 - 0x1f ) 3390-3 OFFLINE
end X370.3990A

X370.3990C: cu 3990
    interface local(3)
    interface network(3)
    device(0x00) 3390-3 /aos/systems/x370/as3res
    device(0x01) 3390-3 /aos/systems/x370/as3rs2
    device(0x02) 3390-3 /aos/systems/x370/as3sys
    device(0x03) 3390-3 /aos/systems/x370/as3pg2
    device(0x04) 3390-3 /aos/systems/x370/a3spl1
    device(0x05) 3390-3 /aos/systems/x370/as3us1
    device(0x06) 3390-3 /aos/systems/x370/as3pg0
    device(0x07) 3390-3 /aos/systems/x370/as3pg1
    device(0x08) 3390-3 /aos/systems/x370/static
    device(0x09) 3390-3 /aos/systems/x370/as3sms
    device(0x0a) 3390-3 /aos/systems/x370/as3log
    device(0x0b) 3390-3 /aos/systems/x370/a3ckpt
    device(0x0c) 3390-3 /aos/systems/x370/as3cds
    device(0x0d) 3390-3 /aos/systems/x370/ckdtst
    device(0x0e) 3390-3 /aos/systems/x370/z3dis1
    device(0x0f) 3390-3 /aos/systems/x370/z3dis2
    device(0x10) 3390-3 /aos/systems/x370/z3dis3

```

```

device(0x11) 3390-3 /aos/systems/x370/awork1
device(0x12) 3390-3 /aos/systems/x370/awork2
device(0x13) 3390-3 /aos/systems/x370/awork3
device(0x14) 3390-3 /aos/systems/x370/awork4
device(0x15) 3390-3 /aos/systems/x370/awork5
device(0x16) 3390-3 /aos/systems/x370/awork6
device(0x17) 3390-3 /aos/systems/x370/awork7
device(0x18) 3390-3 /aos/systems/x370/awork8
device( 0x019 - 0x01f ) 3390-3 OFFLINE
end X370.3990C

X370.3990D: cu 3990 interface local(1) options 'trackcachesize=1800'
device( 0x00 ) 3390-3 /aos/systems/x370/z4res1
device( 0x01 ) 3390-3 /aos/systems/x370/z4res2
device( 0x02 ) 3390-3 /aos/systems/x370/os39m1
device( 0x03 ) 3390-3 /aos/systems/x370/aotso1
device( 0x04 ) 3390-3 /aos/systems/x370/z4ckpt
device( 0x05 ) 3390-3 /aos/systems/x370/spool0
device( 0x06 ) 3390-3 /aos/systems/x370/z4pag0
device( 0x07 ) 3390-3 /aos/systems/x370/z4pag1
device( 0x08 ) 3390-3 /aos/systems/x370/z4pag2
device( 0x09 ) 3390-3 /aos/systems/x370/z4pag3
device( 0x0a ) 3390-3 /aos/systems/x370/z4dis1
device( 0x0b ) 3390-3 /aos/systems/x370/z4dis2
device( 0x0c ) 3390-3 /aos/systems/x370/z4dis3
device( 0x0d ) 3390-2 /aos/systems/x370/z4cic1
device( 0x0e ) 3390-2 /aos/systems/x370/z4ims1
device( 0x0f ) 3390-3 /aos/systems/x370/z4db21
device( 0x10 ) 3390-3 /aos/systems/x370/work01
device( 0x11 ) 3390-3 /aos/systems/x370/work02
device( 0x12 ) 3390-3 /aos/systems/x370/z4was1
device( 0x13 ) 3390-3 /aos/systems/x370/z4was2
device( 0x14 ) 3390-3 /aos/systems/x370/z4uss1
device( 0x15 ) 3390-3 /aos/systems/x370/sares1
device( 0x16 ) 3390-3 /aos/systems/x370/zp4dmp
device( 0x17 ) 3390-3 /aos/systems/x370/z4dis4
device( 0x18 ) 3390-3 /aos/systems/x370/z4plex
device( 0x19 - 0x1f ) 3390-3 OFFLINE
end X370.3990D

end x370

```

Server x370: Instance z370

```

system z370:

memsize(0x1a0000)
cachesize(0x2000)
tracesize(0x100)
instset(esa)

cpu(0)
cpu(1
)
channel(0)localbyte
channel(1)local
channel(2)local
channel(3)localosa

cu devad(0x000c,02) path(0) resource(X370.2821A)
cu devad(0x000f,02) path(0) resource(X370.3203A)

```

```
cu devad(0x0300,32) path(1) resource(X370.3990C)
cu devad(0x0a80,32) path(1) resource(X370.3990D)
cu devad(0x0560,16) path(1) resource(X370.3490A)
cu devad(0x0500,16) path(1) resource(X370.3490B)
cu devad(0x0800,16) path(1) resource(X370.AFMNTA)
cu devad(0x0810,16) path(1) resource(X370.AFMNTB)
cu devad(0x0820,16) path(1) resource(X370.AFMNTC)
cu devad(0x0830,16) path(1) resource(X370.AFMNTD)
cu devad(0x0700,32) path(1) resource(X370.3274C)
cu devad(0x0e00,02) path(3) resource(X370.0SA1)
cu devad(0x0e20,02) path(2) resource(X370.3172A)
cu devad(0x0e40,01) path(2) resource(X370.XCA1)
cu devad(0x0e60,01) path(3) resource(X370.0SA2)
cu devad(0x0e80,02) path(3) resource(X370.0SA3)
```

```
end z370
```

Archived

FAQ

Q: My logical volume (or slice) is for a 3390-3. I want to restore a 3390-2 to it. Can I do this?

A: Yes. If you “restore” with `ckdrestore` or `ckdconvaws` the utility will automatically reformat the emulated 3390 volume to the correct size. (It will not change the allocated size of the logical volume or slice, and this must be large enough to hold whatever model of 3390 you are attempting to restore.) Remember to specify 3390-2 in your resource definitions. The resource definition must specify the correct size.

Q: You mentioned the “FSI patches” for the Linux kernel. Are these published?

A: These patches are related to SMP usage by FLEX-ES. The source can be found in `/usr/src/linux-2.4.20.pset/ADDITIONS` (or the equivalent location for other kernel levels). This assumes you have installed the kernel source from FSI.

Q: RRS fails to start in the z/OS AD 1.4s distribution. Is this a FLEX-ES problem?

A: No, it is a z/OS problem. One solution is to rerun jobs `RRSDELLS` and `RRSLOGST` in `SYS1.P390.CNTL`.

Q: My 4mm tape drive sometimes immediately rejects a tape. Is this a Linux problem? A FLEX-ES problem?

A: How old is the tape? There may be a compatibility problem with very old tapes that cause newer drives (“DDS-4”) to immediately eject the cartridge. This is a hardware issue and is not related to Linux or FLEX-ES functions. Try using a newer tape cartridge; use one that is marked 120 meters (in length) or marked DDS-4 (assuming you have a DDS-4 drive).

Q: How can I receive z/OS PTFs? My EFS system does not have a tape drive.

A: A number of EFS users prefer <https://techsupport.services.ibm.com/server/fixes>. You need to create an IBM registration to use the site, but anyone can do this. You need to know the PTF number you want; there are no search facilities on this site. After requesting a PTF, you are sent instructions for downloading it.

Q: I receive the error message “Internal reconfiguration error” when I attempt to start x3270 in Red Hat 8.0. Is this a known problem?

A: We have not seen this problem; however we have seen reports that it is related to use of the INTEL 845 chip set. The report indicates that downloading the i845 video driver from an INTEL Web site resolved the problem. The Web site mentioned was <http://downloadfinder2.intel.com>.

Q: Can I use non-standard 3270 screen sizes when connecting through the Terminal Solicitor?

A: We have not tested this, but have been told that you need to change the DLOGMOD definition (in the LU definitions in VTAMLST) to use D4B32XX3. (This is for non-SNA terminals, which is how connections through the Terminal Solicitor appear to the S/390.)

Q: I am trying to make an Ethernet SNA connection to my existing P/390 network (including MP3000s) and the connections always fail. Is there anything unique about FLEX-ES in this case?

A: There are many things that might cause these errors. Common problems include:

- A MAC address on a P/390 (or MP3000) is entered in byte-reversed order. For a FLEX-ES system it is entered in normal order.
- P/390 Ethernet adapters can operate in DIX or IEEE802.3 mode, and these are not compatible. Normal usage is DIX for TCP/IP and IEEE802.3 for SNA. The mode is set by using the MPTS utility. An SNA network in which all members are using DIX will also work, although this is a nonstandard arrangement. FLEX-ES uses standard IEEE802.3 for SNA connections. You might verify that your P/390s and MP3000s are using IEEE802.3 for Ethernet connections used for SNA.

Q: I am developing a 64-bit exploitative z/OS program that needs to be loaded at a real address greater than 2G to verify testing. FLEX-ES, even in a z/Architecture instance, allows for real memory instances of not greater than 2G on IA32 platforms. How can I test that my program correctly operates in real addresses higher than 2G?

A: If you run z/OS under z/VM in a z/Architecture FLEX-ES instance, then you can define the virtual machine storage so that it has virtual storage at up to seven discontinuous extents including above the 2G line. Refer to the *z/VM CP Command and Utility Reference*, SC24-6008, under the "DEFINE STORAGE" command. Specifically, the "CONFIGuration" option. The command has a robust syntax. In the following example, the virtual machine's storage is defined to be from 0-256M, and then from 2G for a length of 16M, and then from 1M below the 4G line for a length of 17M. The program that the developer wants to test in a 64-bit address location can be loaded into either of the address ranges available above the 2G line. (Remember that, while these are virtual addresses to z/VM, z/OS will see them as real addresses.)

```
cp define storage config 0.256M 2G.16M 4G-1M.17M
STORAGE = 289M
Storage Configuration:
0.256M 2G.16M 4G-1M.17M
Extent Specification          Address Range
-----
0.256M                      0000000000000000 - 00000000FFFFFFFF
2G.16M                      0000000080000000 - 0000000080FFFFFFF
4G-1M.17M                   00000000FFF00000 - 0000000100FFFFFFF
```

Storage cleared - system reset.

It is important to note that only 289M of virtual machine storage are allocated. When z/OS is IPLed, the pages between 256M and 2G are marked offline (confirmable with a display matrix command on z/OS). The z/VM system does not have to manage page tables and page frames that would have resulted had the full address range from zero to greater than 2G been defined contiguously. While possible to define the contiguous storage, it would put a significant paging load on the system to have more than 2G of virtual z/OS machine size in a less than 2G of "real" memory configuration for the z/VM system.

Q: Should I update to each new FLEX-ES release?

A: In general, no. If you have a specific FLEX-ES problem, your supplier can tell you if a new release addresses this problem. Otherwise, if you have a stable system installed, there is no

reason to chase every FLEX-ES release. Most new releases contain relatively minor fixes (of course, these are major fixes if they address *your* problem). If a new release contains major new functionality or a substantial performance alternation your FLEX-ES supplier should inform you about this.

Archived

Archived

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information on ordering these publications, see “How to get IBM Redbooks” on page 201. Note that some of the documents referenced here may be available in softcopy only.

- ▶ *EFS Systems on a Linux Base: Getting Started*, SG24-7007
- ▶ *S/390 Partners in Development: OS/390 (and z/OS) New User's Cookbook*, SG24-6204
- ▶ *S/390 Partners in Development: Netfinity Enabled for S/390*, SG24-6501
- ▶ *NUMA-Q Enabled for S/390: Technical Introduction*, SG24-6215

Other publications

These publications from Fundamental Software, Inc., are also relevant as further information sources:

- ▶ *FLEX-ES Resource Language Reference*, FSIMM310
- ▶ *FLEX-ES System Programmer's Guide*, FSIMM300
- ▶ *FLEX-ES CLI Language Reference*, FSIMM210
- ▶ *FLEX-ES Operator's Guide*, FSIMM200
- ▶ *FLEX-ES Planning Guide*, FSIMM100

Online resources

These Web sites and URLs are also relevant as further information sources:

- ▶ Public Web site for Fundamental Software, Inc., of Fremont, California:
<http://www.funsoft.com>

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

Index

Symbols

\$SPRT1 command 82
/dev/raw 61
/etc/init.d/FLEXES 66
/etc/rc.d/rc2.d 66
/etc/xinitd.d/wu-ftpd 133

Numerics

1403-N1 printer 77
2540 card reader 77
2G line, above 198
3172 LAN connections 31
3174 connections 29
3215 console 111
3270 screen sizes 198
3270 terminal resources 90
3270 terminal, local, non-SNA 28
3480 146
3480 compressed 146
3490 146
3490E 146
4mm tape drive 149, 197
64-bit address location 198
64-bit operation 2

A

ACBNAME 176
active wait 86, 102
ACTSTR00 in SYS1.VTAMLST 156
AD system 63
Add Device dialog 162
address E40 42
ADDRSSU 75
ADDRSSU backups 76
ADDRSSU dump 69
AFP printer 79
allegiance, volume 109
Architectural Level Set 3 2
ATCSTR00 167
AWSCKD emulated disks 69
AWSOMA format 148
AWSTAPE 145
AWSTAPE format 148
awstape format 152

B

backup and restore 74
base sysplex 100, 153
BEGINRoute parameters 40
Booting from the Ultrabay 15
BPXPRM00 member of PARMLIB 118
burning a CD 3

C

Cable/DSL router 35
cachesize 17
cachesize parameter 22, 86
cachestats command 23
card reader 77, 83
CD recording 3
cdrecord 151
cdrecord command 4
CDRM major node 169
central storage 25
cfcomp command 88, 93
channel 87
CKD disk resources 90
CKD disk, verify 12
ckdbackup 71–72, 74–75
ckdcachestats command 21
ckdchk command 12
ckdconvaws 71, 197
ckdconvaws command 63
ckddump command 65
ckdfmt 73, 112
ckdfmt command 65
ckdrestore 71–73, 197
CLI commands 95
CLOCK00 member of PARMLIB 117
cloned parameter 93
COMMND00 member of PARMLIB 119
Compression 73
CONSOL00 member of PARMLIB 117
control unit types, emulated 88
Couple data set preparation 120
COUPLEnn members of PARMLIB 121
Coupling Facility (CF) 101
cpu parameters 86
cross-domain 175
CTCs, emulated 109
cu statement 87

D

DASD cache 18
dasd driver for Linux 134
David MacMillan 99
dd command 8, 71
dedicated keyword 87
devad 87
device types, emulated 88
device(00) parameter 90
devopt 90
devstate command 22
DHCP client 31, 34, 36
DHCP server 34, 36
disk cache 20, 58
disk caches 25

- disk layout 7
- disk partition 57
- DIX or IEEE802.3 mode 198
- Docking Station 4
- duplex printing 79

E

- E40, address in AD system 42
- emulated volumes, how many 22
- end statement 88
- essize 17
- essize, definition 86
- eth0, resource 92
- Ethernet adapter 154
- Ethernet hub 48
- Ethernet SNA 198
- ETR (External Timer Reference) emulation mode 100
- Expanded storage 25
- expanded storage 25
- ext2 or ext3 file system 57

F

- FakeTape 145
- fdasd command 138
- fdisk 6
- fdisk utility 58
- feature lpar 86
- file system 57
- firewall operation 48
- firmware options 150
- flexes command 8
- FLEX-ES definitions 85
- FLEX-ES definitions, printer 81
- FLEX-ES definitions, raw devices 63
- FLEX-ES instances, multiple 8
- FLEX-ES names, rules 94
- FLEX-ES resources, for SNA 157
- FLEX-ES startup, raw devices 64
- FLEX-ES system definition, z/VM 101
- flexescli program 95
- FSI patches 197
- ftp 75
- ftp for backups 75
- FTP server 133
- ftp to z/OS TCP/IP 31
- ftp.0.17-15.i386.rpm 133

G

- Gary Ehemann 66, 99
- Getting Started, IBM order number SG24-7007 1
- Global Resource Serialization (GRS) 101, 109
- gnome and kde desktops 9
- GRS 44
- gunzip 72–74
- gzip 72–73, 75
- gzip compression 69

H

- hard disk, cloning 8
- hardware console 52
- HCD dialogs 158
- HFS volume 124
- HiperSockets 32
- host key verification failed 142
- hotdr.sc shell script 83
- hotdr.sh 84
- http to z/OS HTTPD 31
- hub 48

I

- IBM 3490-F01 drive 150
- ICKDSF 112
- IEASYS00 member of PARMLIB 115
- IECIOS00 member of PARMLIB 119
- ifconfig command 155
- initawstape utility 148
- instruction cache 20
- instset parameter 86
- instset specification 2
- INTEL 845 chip set 197
- interface local(1) parameter 90
- interface network 47
- iodelay command 134
- IODF 29, 160, 165–167
- IP address 31
- IP addresses, permanent 34
- IP addressing plan 37
- ish (interactive shell) 126

J

- JES2 definitions 77, 82
- JES2 network job entry 176
- JES2PARM 77
- JES2PARM member of PARMLIB 119

K

- kernel updates 2

L

- LAN adapter, shared 92
- LAN adapter, sharing 27
- LAN adapters, multiple 27
- LAN resources 92
- laser printer 79
- Lexmark Optra L printer 80
- Linux desktop 30
- Linux file, standard 57
- Linux for S/390 131
- Linux kernel patches 197
- Linux paging 18
- Linux service 2
- Linux swapping 19
- Linux swapping rate 19
- Linux virtual storage 17

- load command 82
- local 87
- localosa channel 33
- logical volume 197
- Logical Volume Manager (LVM) 58
- logical volume sizes 61
- lpar feature 102
- lparnum 102
- lpr 81
- lpr command 80
- lvcreate 74
- LVM 74
- LVM and raw interface setup 132
- LVM modules 1
- LVM options 60
- LVM partition 59
- LVM partition, creating 59
- LVM, second hard disk 7
- lvremove 74
- lvremove command 65
- lvrename command 65
- lvscan command 65

M

- MAC address on a P/390 198
- MACADDR 154–155
- maxwritesize option 92, 148
- MDISK statements 108
- Medium Access Control address 154
- memory allocations 20
- memory definition 89
- Memory tuning 23
- memory usage, on server 17
- memsize 17
- memsize, definition 86
- Michael Ryan 99, 153
- minor number, raw nodes 61
- mkisofs 151
- mkisofs command 4
- mknod 61–62
- mknod command 132
- mount command 21, 91, 96, 148
- mount command, Linux 7
- MPTS number 92
- msgmgr module 2
- MSTJCL00 member of PARMLIB 119
- Multiple FLEX-ES instances 8
- MVS console 50
- MVS console, restarting 12

N

- Naming conventions, LVM and raw 60
- ndstat command 154
- netcfg utility 154
- Netgear Cable/DSL ProSafe Firewall/Print Server Model FR114P 35
- NETID 167
- Network Address Translation (NAT) 35
- network channel 47

- network channel definitions 43
- network id 156
- Networking limitations 32
- NFS files, for FLEX-ES use 42

O

- OMA 145
- OMVS 124
- OMVS couple data set 121
- onlcr option 81
- Open UNIX 8 99
- Operating FLEX-ES remotely 49
- options statement 92
- OSA channels 33
- OSA Ethernet, for Linux installation 135
- OSA-Express adapter 32
- osasna 153
- osasna control unit 102
- osasna device 109
- osasnaTR 153
- Overland T490E tape drive 149

P

- P/390 network 198
- P/390 system 149
- parallel port 80
- Parallel Sysplex 100–101
- path parameter 87
- PCI adapter slot, ThinkPad 5
- PCMCIA slots 4
- PCOM 30
- Performance implications, 64-bit operation 2
- ping command 48
- PORT statement 171
- print queue 80
- printer 77
- PROCLIB changes 119
- production FLEX-ES system 57
- PSW, display 12
- PTFs 197

R

- raw 74
- raw command 62, 64, 74
- raw device interfaces 90
- raw device nodes 61
- raw devices 24, 73
- raw disk devices 57
- raw disk interface 132
- Raw Print Queue 80
- real memory instances 198
- Real memory, server 26
- Red Hat 9.0 1
- Redbooks Web site 201
- Contact us x
- remote operation of FLEX-ES 49
- Remote resources 43
- remote z/OS operator console 49

- resadm command 93
- resadm command, options 94
- resadm -h commands 47
- rescf files 93
- resource definitions 88
- resource parameter 87
- restart function, S/390 135
- REXX code 111
- RNL changes 129
- Router administration 37
- Router service requests 41
- rpm command 3
- RRS fails to start 197
- RTP activation 175
- RTP using CTCS 123

S

- S/390 identification 5
- SAPADDR 156, 170
- SCSI generic device interfaces 145
- SCSI name 145
- SCSI-attached drives 145
- SCSI-attached tape drive 150
- scsitfake program 150, 152
- serial number, CPU 5
- ServeRAID modules 1
- Service Access Point (SAP) number 27
- set cpu command 23
- shared segment (of virtual memory) 25
- shared-DASD configuration 44
- shmmax, Linux variable 25
- shPRT shell script 80
- shutdown command 96
- SNA 3270 connections 29
- SNA over Ethernet 153
- SNA-over-Ethernet 102
- SSCP name 156
- ssh 49, 51
- ssh command 142
- ssid values 47
- standard Linux file 57
- STIDP instruction 5
- subarea number 156
- subareas 167
- SuSE 8.0 131
- SuSE customization 143
- SuSE license 137
- swap (paging) rates 19
- switch, LAN 48
- SYS1.IPLPARM 158, 167
- syscf files 93
- SYSCLONE 115
- sysplex 99, 153
- Sysplex couple data set 120
- sysplex naming 113
- sysplex operation 112, 127
- system definitions 85
- System Monitor 20

T

- T40 ThinkPads 1
- tape devices 145
- tape devices, emulated 147
- tape drive, 3480/90 149
- tape resources 91
- tape utility commands 146
- tar compression 69
- TCP/IP port 555 43
- TCP/IP profile (z/OS) 39
- TCPIP.PROFILE.TCPIP data set 39
- telnet to Linux (port 23) 31
- telnet to z/OS TCP/IP 31
- Terminal Solicitor 3, 27, 30, 49, 51, 91
- TN3270 clients 29
- TN3270 to Linux (port 24) 31
- TN3270 to z/OS TCP/IP 31
- Token Ring configuration 153
- top command 20
- tracesize parameter 86
- trackcachesize 20, 148
- TSO application id 175
- TSO networking 28
- tuning, sysplex 128

U

- Ultrabay slot 5
- Ultrabay, booting 15
- Ultrabay, second disk 6
- USB 2.0 ports 1

V

- vgcfbackup command 65
- vgchange command 61, 132
- vgcreate command 61
- vgdisplay command 65
- vmstat command 18
- VMUSERID filters 114
- VTAM configuration 156
- VTAM connections 153
- VTAM customization 123
- VTAM definitions 28, 30
- VTAM SSCP-SSCP connections 153

W

- WAN Ethernet connections 35
- Windows, TCP/IP configuration 41
- WLM couple data set 122
- writeback 20
- writeback cache 21
- writethrough cache 21
- writethroughcache parameter 20, 90
- WRKALLEG 109
- wu-ftpd package 133

X

- x235 server 99

x3270 51
x3270 errors 197
x3270 sessions 30
x3270 sessions on the desktop 3
XCA 33
XCA device 157, 164
XCA major node 170–173
xcasna 153
xcasnaTR 153
xcdroast 4
xhost + 134
xhost command 3

Y

YaST panel 140
YaST2 142
YaST2 Control Center 142

Z

z/Architecture 198
z/OS console 29
z/OS TCP/IP 31
z/VM 101, 103, 198
z/VM 4.3 99
z/VM guest 106
z/VM minidisks 106
zip/unzip compression 69

Archived



PartnerWorld for Developers, ITSO/EFS Project

EFS Systems on a Linux Base: Additional Topics



LVM and raw disks

TCP/IP approaches

Advanced topics

This IBM Redbook describes additional or advanced techniques for using FLEX-ES (a product of Fundamental Software, Inc., Fremont, California) with z/OS and various z/OS packages. Basic installation and use of FLEX-ES is described in *EFS on a Linux base; Getting Started*, SG24-7007. Both publications are intended primarily for members of the IBM S/390 PartnerWorld for Developers (PWD) organization and for internal IBM users of the ITSO/EFS package. This publication provides more detail for networking, local routers, tuning, use of raw devices, an experimental basic sysplex implementation, extensive SNA setup, and a basic Linux for S/390 installation.

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

SG24-7008-00

ISBN 0738499129