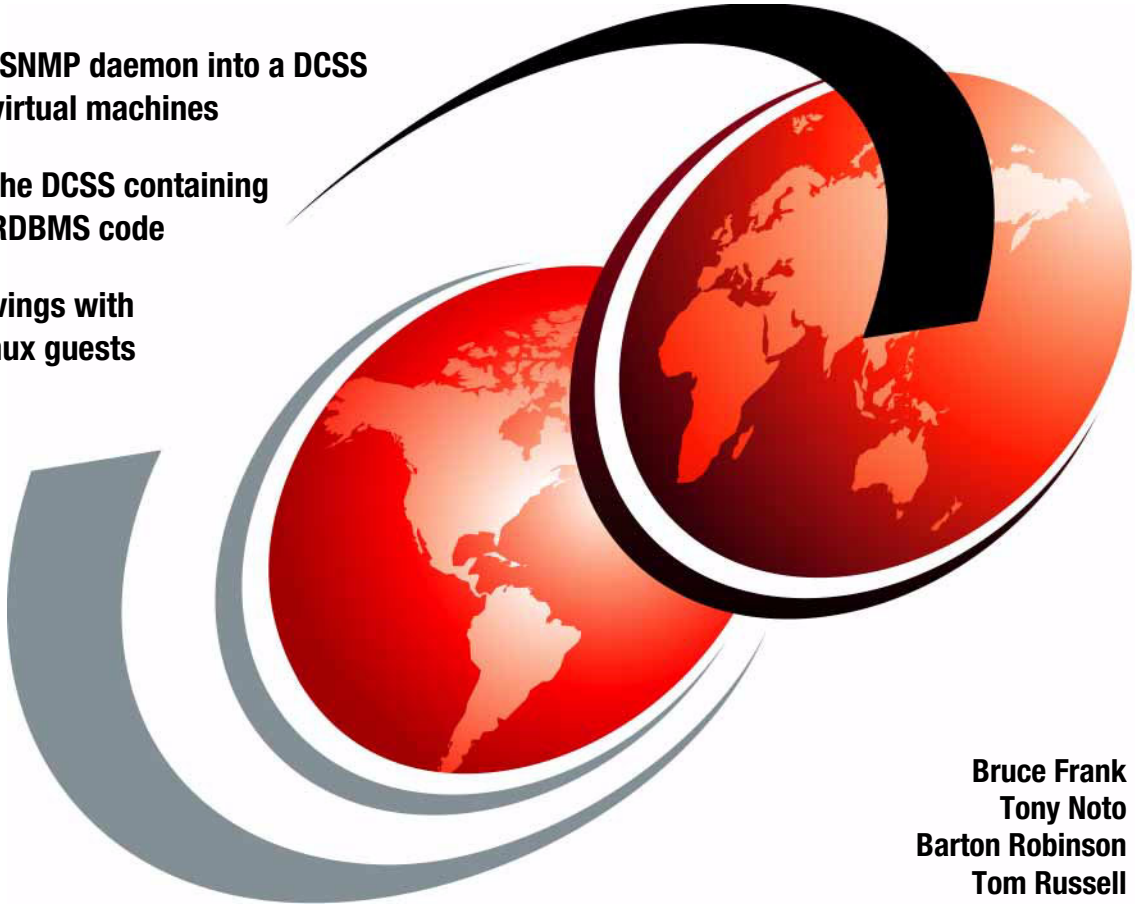
Using Discontiguous Shared Segments and XIP2 Filesystems

With Oracle Database 10*g* on Linux for IBM System z

- Putting the SNMP daemon into a DCSS to support virtual machines
- Setting up the DCSS containing the Oracle RDBMS code
- Memory savings with multiple Linux guests



Bruce Frank
Tony Noto
Barton Robinson
Tom Russell

Redbooks

ibm.com/redbooks

International Technical Support Organization

Using Discontiguous Shared Segments and XIP2 Filesystems: With Oracle Database 10*g* on Linux for IBM System z

June 2006

SG24-7285-00

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

First Edition (June 2006)

This edition applies to Oracle Version 10g, Release 1.

Contents

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.

Trademarks

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

developerWorks®	Redbooks (logo)  ™	z/VM®
IBM®	System z™	
Redbooks™	VM/ESA®	

The following terms are trademarks of other companies:

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

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

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

Preface

Linux® on IBM® System z™ offers many advantages to customers who rely on IBM mainframe systems to run their businesses. Linux on IBM System z takes advantage of the qualities of service in the IBM System z hardware and in z/VM® — making it a robust industrial strength Linux. This provides an excellent platform for consolidating Oracle databases that exist in your enterprise.

This IBM Redbook describes our experiences in boosting performance and reducing memory requirements by using discontiguous saved segments (DCSS) and the execute in place (XIP2) filesystem technology.

This book provides an overview of DCSS and XIP2 in Chapter 1, and describes our experiences gained while installing and testing Oracle Database 10g for Linux on IBM System z, namely:

- Using DCSS and XIP2 with Oracle:
  - Our tests with the Simple Network Management Protocol (SNMP) daemon are described in Chapter 2, “Putting the SNMP daemon into a DCSS” on page 5.
  - Our tests with Oracle are described in Chapter 3, “Using DCSS for Oracle” on page 23.
  - The measurements we made are described in Chapter 4, “DCSS measurements” on page 37.

Interested readers include database consultants, installers, administrators, and system programmers.

The team that wrote this redbook

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

Bruce Frank is a System z Oracle Specialist in the IBM/Oracle International Competency Center at IBM San Mateo.

Tony Noto is a z/VM Specialist with Velocity Software, Mountain View, California. He has been involved with software development and system programming on IBM mainframes since 1974 and has been working in the VM environment since 1983, specializing in VM security and performance.

Barton Robinson is President of Velocity Software, Inc. He started working with VM in 1975, specializing in performance starting in 1983. His previous publication experience includes the *VM/HPO Tuning Guide* published by IBM, and the *VM/ESA Tuning Guide* published by Velocity Software. He is the author and developer of ESAMAP and ESATCP.

Tom Russell is a System z Specialist with IBM Canada. He spent several years on a special assignment with Oracle in Redwood Shores, California.

Thanks to the following people for their contributions to this project:

Lydia Parziale
International Technical Support Organization, Poughkeepsie Center

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 email to:

  redbook@us.ibm.com

- Mail your comments to:

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

1

DCSS and XIP2 overview

This chapter provides an overview of DCSS and XIP2:

In further chapters, we provide information about using DCSS in an Oracle environment with these topics:

- Our tests with the SNMP daemon are described in Chapter 2, “Putting the SNMP daemon into a DCSS” on page 5.
- Our tests with Oracle described in Chapter 3, “Using DCSS for Oracle” on page 23.
- The measurements we made described in Chapter 4, “DCSS measurements” on page 37.

1.1 Using DCSS in Linux SLES9

The high level objective of this project is to document techniques that use features of z/VM which a customer can implement that can reduce the storage requirement for the second and subsequent Linux servers.

Each Linux instance under z/VM loads its own copies of programs, and manages its own storage. When multiple copies of Linux are running on the same copy of z/VM there is an opportunity to use functions unique to z/VM and System z to share memory pages between the multiple Linux machines.

A Discontiguous Shared Segment (DCSS) is a function of z/VM that allows a segment of memory to be loaded into a virtual machine by command. This segment of memory can be discontiguous to the virtual machine's address space, and a shared copy is loaded at the same address in all virtual machines that load the same DCSS.

A segment is a 1 MB portion of real storage as defined by the ESA/390 architecture. A z/VM saved segment is a range of pages of virtual storage consisting of a number of memory segments where you can hold data or reentrant code (programs). A discontiguous saved segment (DCSS) is a saved segment that can be embedded above the virtual machine's defined storage size.

In a virtual server farm with similar Linux instances there is often a considerable amount of data that is required by all instances. A DCSS enables z/VM to load such data into a designated segment of physical memory and allows all Linux instances that need the data to share this memory segment.

1.2 Using the XIP2 filesystem in Linux SLES9

The execute in place (XIP2) filesystem is provided in SuSE SLES9. This filesystem allows programs residing in a DCSS to be executed directly, sharing the resident pages with other Linux guests.

You can find the latest version of the book *How to use Execute-in-Place Technology with Linux on z/VM,* SC33-8287 at:

http://www-128.ibm.com/developerworks/linux/linux390/october2005_documentation.html

If you are routed to the top-level page, on the left navigation bar, under *Kernel 2.6*, click **October 2005 stream**. On the October 2005 stream page, click **Documentation**.

The book discusses sharing complete directories in a DCSS as well as using a DCSS to share individual files. We will show both here in this redbook. You can also download a tarball containing shell scripts that you can tailor to set up the DCSS environment. We modified these scripts slightly for our environment. The modified versions are listed in Appendix B, “Scripts used to set up DCSS” on page 65.

1.3 DCSS for SNMP

Every Linux guest uses the SNMP daemon. This daemon listens on TCP/IP port 161 and will reply with data on the activity in the Linux guest. Our monitor, ESAMON, sends SNMP requests to each active guest every minute. By putting the SNMP daemon into a DCSS, each Linux guest will use the same copy of the pages where the program is loaded.

Plan

As discussed in Chapter 2, we analyzed the amount of disk space that the SNMP daemon takes, and decided that 8 MB was enough to hold the shared segment containing the SNMP daemon. We also decided to use the technique for sharing individual files as opposed to sharing directories.

The DCSS is executable code. In order to address these pages, Linux must have built page and segment tables to address the memory where the segment will be loaded. These memory management data structures are normally only built for the amount of memory in the virtual machine. Our Linux guests were 256 MB each.

Define

Under z/VM we defined the segment NETSNMP. We reduced the size of the virtual machine by 8 MB and defined the DCSS to be loaded with an end address of 256 MB. The Linux boot parms were changed to ensure that the Linux guest could address storage from zero to 256 MB. See Chapter 2, “Putting the SNMP daemon into a DCSS” on page 5 for the details.

1.4 DCSS for Oracle modules

Plan

After many experiments, we decided that the most effective way to use the DCSS for Oracle is to share an entire directory, and only put into the DCSS the code that we knew would be executed. In our tests we got the best results by putting only two modules (**`oracle`** and **`sqlplus`**) in the DCSS.

The standard scripts for implementing DCSS as supplied from developerWorks® assume that the code loaded in the DCSS is in the root file system. Because $ORACLE_HOME is not in the root file system, we modified the techniques slightly. Chapter 3, “Using DCSS for Oracle” on page 23 describes the process we used for defining and implementing this DCSS.

1.5 Oracle and XIP2 considerations

The execute in place (XIP2) filesystem allows Linux servers to share programs, executing the code directly out of the DCSS. The objective is that with many Linux servers sharing storage, programs only need to be resident once and should allow for significant storage savings.

There are trade-offs that must be understood before implementing XIP2. In order to access the DCSS, the Linux kernel must be configured with enough virtual and real page management structures to address the DCSS pages. These additional control blocks can sometimes cost more than the savings generated by sharing the DCSS. This is discussed in detail in Chapter 4, “DCSS measurements” on page 37.

ESALPS Version 4.1, the Linux Performance Suite, was provided by Velocity Software for this analysis. More information about ESALPS can be found in Appendix A, “ESALPS” on page 61.

2

Putting the SNMP daemon into a DCSS

This chapter documents the process that we used to put selected files that support the SNMP daemon in Linux. Because every Linux machine uses SNMP, we put it into a DCSS so that a single copy could support all of the virtual machines.

In this chapter, we discuss:

- Creating the DCSS segment
- Attaching the DCSS segment to the Linux guest, creating a file system on the DCSS and putting the programs into it that we want to share
- Making changes to the Linux guest so that the DCSS is allocated at boot time, and correcting files that are overmounted by the DCSS version.
- Rebooting the machine to begin using the DCSS copy of the shared routines.

2.1 DCSS creation for the SNMP daemon

The following is the procedure used for creating the “direct file” DCSS.

The DCSS creation requires a virtual machine with Class E privileges. There are other methods of doing this without giving the Linux machine authority class E. For this project, we found it simpler to give the Linux machine class E, and do all of the CP segment definition from the Linux machine.

Information about how to use the execute in place (XIP2) filesystem was received from the developerWorks site for documentation and sample code:

http://www-128.ibm.com/developerworks/linux/linux390/october2005_documentation.html

A tarball of the sample scripts that we used can be downloaded from here.

2.2 Sizing the DCSS

To make effective use of the DCSS, you should ensure that it is sized correctly. The following is the procedure that we used to build a DCSS that was the minimum size to hold the number of files that we knew would be used. The shell scripts that we used, as well as the methodology, was downloaded from the developerWorks Web site referred to in Figure . The scripts were put in the home directory of the user doing the work.

To determine the size of the processes, run the **copylibs.sh** script that was downloaded from the developerWorks Web site. This execution will copy all snmp programs from the current $PATH to a directory /dcss. For example, the following is used to copy the SNMP daemon programs.

Example 2-1 Sample execution of copylibs.sh

```
linux9:~ # xip-howto-scripts/copylibs.sh -f snmpd -d /dcss
option f is set with snmpd
snmpd
option d is set with /dcss
/dcss
/usr/local/lib/libnetsnmpmibs.so.5
/usr/local/lib/libnetsnmpmibs.so.5 is a link
/usr/local/lib/libnetsnmpmibs.so.5.2.1
/usr/local/lib/libnetsnmpagent.so.5
/usr/local/lib/libnetsnmpagent.so.5 is a link
/usr/local/lib/libnetsnmpagent.so.5.2.1
/usr/local/lib/libnetsnmphelpers.so.5
/usr/local/lib/libnetsnmphelpers.so.5 is a link
```

```
/usr/local/lib/libnetsnmphelpers.so.5.2.1
/usr/local/lib/libnetsnmp.so.5
/usr/local/lib/libnetsnmp.so.5 is a link
/usr/local/lib/libnetsnmp.so.5.2.1
/lib/libdl.so.2
/usr/lib/librpm-4.1.so
/usr/lib/librpmdb-4.1.so
/usr/lib/librpmio-4.1.so
/lib/tls/librt.so.1
/lib/tls/libpthread.so.0
/lib/libz.so.1
/lib/libz.so.1 is a link
/lib/libz.so.1.2.1
/usr/lib/libpopt.so.0
/usr/lib/libpopt.so.0 is a link
/usr/lib/libpopt.so.0.0.0
/usr/lib/libbz2.so.1
/usr/lib/libbz2.so.1 is a link
/usr/lib/libbz2.so.1.0.0
/lib/tls/libm.so.6
/lib/tls/libc.so.6
/lib/ld.so.1
/lib/ld.so.1 is a link
/lib/ld-2.3.3.so
```

Note: Libraries which were loaded with libdl will not be copied, be sure that you copy these manually.

The execution of the script in Example 2-1 on page 6 copies all the SNMP daemon files to a new directory /dcss. Now, to determine how much space the files take up, issue the **du -sk** command, as shown in Example 2-2. This will show how much space (in kilobytes) is used in the /dcss directory.

Example 2-2 Calculating size of /dcss directory

```
linux9:~ # du -sk /dcss
7024    /dcss
```

Each file has some overhead. Adding 4 KB per file provides for this additional overhead. To determine how many files there are in the DCSS, issue the command as shown in Example 2-3:

Example 2-3 Counting the files to go in the DCSS

```
linux9:~ # find /dcss type f -print | wc -l
17
```

The DCSS is allocated in one MB increments so the size will be rounded up to the next whole megabyte. Now with the size of the programs loaded, and the number of files, the DCSS storage requirement can be calculated as demonstrated in Example 2-4.

Example 2-4 Calculating the size of the DCSS

```
Number of files * 4 KB = 17 * 4 = 68 KB
Size of files + file overhead = 7024 KB + 68 KB = 7092 KB
```

Rounding 7092 KB up to the next MB results in 8 MB. Thus the DCSS needs to be 8 MB to support the SNMP daemon process used in this example.

2.3 Determining DCSS addresses

As the DCSS in this example is 8 MB, the next step is to determine the starting address of the DCSS. For this example, the virtual machine size is 256 MB. To get the most benefit in this case from the DCSS, the virtual machine will be reduced in size by 8 MB and the DCSS will be placed just above the virtual machine. The DCSS starting address will be:

256 MB - 8 MB = 248 MB

For creating the DCSS, the page number must be determined to supply to the **CP command** that creates the DCSS. The starting address of 248 MB must be converted to hex, and this is then converted to the page number. The DCSS start address of 248 MB is address hex “F800000”. The end address is hex “FFFFFFF”, or 256 MB minus 1. The page address is then the hexadecimal value truncated, so in this case, the starting page is F800, and ending page is FFFF.

For different applications, a different sized DCSS might be more appropriate.

2.4 Tailoring Linux storage size to support the DCSS

The virtual machine should now be reduced by 8 MB, and then Linux needs to be configured to address the DCSS. By default, Linux will build page management control blocks for the amount of real memory that it detects when it boots in the virtual machine. To increase this value, so that the pages to be used by the DCSS will be addressable, the configuration file “ /etc/zipl.conf” requires modification. To do this, we use the standard **vi**, as shown in Example 2-5 on page 9.

Example 2-5 Changing the zipl.conf file

```
cd /etc
vi zipl.conf
```

add “**mem=256M**” to the parameter line (shown in Example 2-6):

Example 2-6 zipl.conf file

```
parameters = "dasd=0.0.0205,0.0.0306 root=/dev/dasda1 selinux=0
TERM=dumb elevator=cfq mem=256M"
```

Run **zipl** to update everything. The output from the **zipl command** is shown in Example 2-7.

Example 2-7 Output from zipl command

```
linux9s:/etc # zipl
Using config file '/etc/zipl.conf'
Building bootmap in '/boot/zipl'
Adding IPL section 'ipl' (default)
Preparing boot device: dasda (0205).
Done.
```

You may want to use the -V parameter to get details on zipl's actions. The output from the **zipl command** with the -V parameter is shown in Example 2-8.

Example 2-8 Output from zipl -V command

```
linux9s:/etc # zipl -V
Using config file '/etc/zipl.conf'
Target device information
 Device.......................: 5e:00
 Partition....................: 5e:01
 Device name..................: dasda
 DASD device number...........: 0205
 Type.........................: disk partition
 Disk layout..................: ECKD/compatible disk layout
 Geometry - heads.............: 15
 Geometry - sectors...........: 12
 Geometry - cylinders.........: 1113
 Geometry - start.............: 24
 File system block size.......: 4096
 Physical block size..........: 4096
 Device size in physical blocks..: 200316
Building bootmap in '/boot/zipl'
Adding IPL section 'ipl' (default)
 kernel image......: /boot/image at 0x10000
 kernel parmline...: 'dasd=0.0.0205,0.0.0306 root=/dev/dasda1 selinux=0
TERM=dumb elevator=cfq mem=256M' at 0x1000
 initial ramdisk...: /boot/initrd at 0x800000
Preparing boot device: dasda (0205).
Syncing disks...
Done.
```

2.5 Creating the DCSS

We chose to create the DCSS from the Linux Virtual Machine. In order to have the authority to create and populate a DCSS, the z/VM directory entry for this Linux machine must define the Linux guest as having Class E privileges. There must also be a **NAMESAVE** statement in the directory entry. These privileges allow the Linux machine to define the DCSS. In Example 2-9 on page 11, we called the DCSS “NETSNMP”. The parameters for the **defseg** command were calculated in 2.3, “Determining DCSS addresses” on page 8.

Example 2-9 Defining the DCSS

```
#CP DEFSEG NETSNMP F800-FFFF SR
HCPNSD440I Saved segment NETSNMP was successfully defined in fileid 0149.
```

Validate the NSS is done with the QUERY command as shown in Example 2-10.

Example 2-10 Saving the DCSS to create the storage image

```
#CP Q NSS MAP
FILE FILENAME FILETYPE MINSIZE  BEGPAG ENDPAG TYPE CL #USERS PARMREGS VMGROUP
0096 DEMOMON  DCSS       N/A    04000  04FFF   SN  R  00003   N/A       N/A
0092 ESAWEB   DCSS       N/A    03800  038FF   SN  R  00007   N/A       N/A
0097 ZMON     DCSS       N/A    02800  037FF   SN  A  00000   N/A       N/A
...
0149 NETSNMP  DCSS       N/A    0F800  0FFFF   SR  S  00000   N/A       N/A
```

Note that after the **saveseg command** (see Example 2-11), the class of the DCSS has changed from "S" for Skeleton to "A" for Active.

Example 2-11 Results of the "saveseg netsnmp" command

```
#CP SAVESEG NETSNMP
HCPNSS440I Saved segment NETSNMP was successfully saved in fileid 0149.
CP Q NSS MAP
FILE FILENAME FILETYPE MINSIZE  BEGPAG ENDPAG TYPE CL #USERS PARMREGS VMGROUP
0096 DEMOMON  DCSS       N/A    04000  04FFF   SN  R  00003   N/A      N/A
0092 ESAWEB   DCSS       N/A    03800  038FF   SN  R  00007   N/A      N/A
0097 ZMON     DCSS       N/A    02800  037FF   SN  A  00000   N/A      N/A
...
0149 NETSNMP  DCSS       N/A    0F800  0FFFF   SR  A  00000   N/A      N/A
```

2.6 Resize your virtual machine and then reboot

Now that you have saved the DCSS, you have to change the size of your virtual machine so that its pages will not overlap with the DCSS pages when the DCSS is loaded. To do this, shutdown your system, resize the virtual machine, and then reboot. Example 2-12 demonstrates the command required to do this.

Example 2-12 Shutdown, change storage size and reboot

```
linux9s:~ # shutdown -h now
Broadcast message from root (pts/0) (Thu Dec 22 10:57:27 2005):
```

```
The system is going down for system halt NOW!
 ...
The system will be halted immediately.
 ...
Power  down.
HCPGSP2630I The virtual machine is placed in CP mode due to a SIGP stop
and store status from CPU 00.
DEF STOR 248M
STORAGE = 0248M
Storage cleared - system reset.
IPL 205
```

Changing the virtual machine size with the define storage command works immediately, but the change will be lost if you logoff the virtual machine. To make this a permanent change, update the CP directory entry for this virtual machine. The size of the machine is on the USER statement. This will ensure that the virtual machine size will be correct through system restarts.

After your system is back up, ensure the "mem=" parameter is present and correct by using the command, **cmdline**, as shown in Example 2-13.

Example 2-13 Verify zipl changes were done correctly

```
linux9s:~ # cat /proc/cmdline
dasd=0.0.0205,0.0.0306 root=/dev/dasda1 selinux=0 TERM=dumb elevator=cfq
mem=256M BOOT_IMAGE=0
```

2.7 Setup for the XIP2 DCSS

One of the shell scripts downloaded from the developerWorks Web site shown in Figure on page 6 is "**xipinit-fw**". Copy this script to /sbin and modify it to point to the eventual mount point of the XIP2 file system and to tell it the name of the DCSS (in this case “NETSNMP”). This is demonstrated in Example 2-14.

Example 2-14 Customizing xipinit.fw

```
cp /root/xip-howto-scripts/xipinit-fw.sh /sbin/xipinit-fw
vi /sbin/xipinit-fw
...
#mount point of xipimage
MPXIPIMAGE="/xip"
#name of xipimage
XIPIMAGE="NETSNMP"
```

Create the mount point with the **mkdir** command (see Example 2-15).

Example 2-15 Create the directory for the mount point

```
mkdir /xip
```

Now load the DCSS driver with the **modprobe** command. This driver is supplied in the Linux distribution, and allows you to treat the DCSS as a disk so that you can read and write data to it. We will use this driver to create a filesystem on the DCSS and copy the files we want to share into this file system. The command in Example 2-16 loads the block driver.

Example 2-16 Load the DCSS block mode driver

```
linux9s:~ # modprobe dcssblk
```

In order to define devices to use this driver we need to define some nodes. To do this we need the driver's major number. To determine this number we look at the /proc/devices file. Example 2-17 shows us that the dcssblk driver is there and that the major number is 252.

Example 2-17 Find the driver's major number

```
linux9s:~ # cat /proc/devices
Character devices:
  1 mem
  4 ttyS
  4 ttysclp
  5 /dev/tty
  5 /dev/console
  5 /dev/ptmx
 10 misc
 43 ctctty
128 ptm
136 pts
227 ttyTUB
228 fs3270
Block devices:
  1 ramdisk
  7 loop
  8 sd
  9 md
 65 sd
 66 sd
 67 sd
 68 sd
 69 sd
 70 sd
```

```
71 sd
94 dasd
128 sd
129 sd
130 sd
131 sd
132 sd
133 sd
134 sd
135 sd
252 dcssblk
253 device-mapper
254 mdp
```

The next step shown in Example 2-18 is to create the device nodes, if required.

Example 2-18 Verify device nodes and create if necessary

```
linux9s:~ # ls -l /dev/dcss*
/bin/ls: /dev/dcss*: No such file or directory

linux9s:~ # mknod /dev/dcssblk0 b 252 0
linux9s:~ # mknod /dev/dcssblk1 b 252 1
linux9s:~ # mknod /dev/dcssblk2 b 252 2
linux9s:~ # mknod /dev/dcssblk3 b 252 3

linux9s:~ # ls -l /dev/dcss*
brw-r--r--  1 root root 252, 0 Oct 12 17:15 /dev/dcssblk0
brw-r--r--  1 root root 252, 1 Oct 12 17:16 /dev/dcssblk1
brw-r--r--  1 root root 252, 2 Oct 12 17:16 /dev/dcssblk2
brw-r--r--  1 root root 252, 3 Oct 12 17:16 /dev/dcssblk3
```

2.8 Load the DCSS storage with the files to be shared

At this point, we have created a DCSS segment that will be loaded from address 248 MB to 256 MB. The machine is defined to be 248 MB in size and rebooted at this size. The Linux command line has been modified to support addresses up to 256 MB.

The dcssblk driver is used to manipulate the DCSS (Example 2-19).

Example 2-19 Use the dcssblk driver to load the DCSS into storage

```
linux9s:~ # echo "NETSNMP" > /sys/devices/dcssblk/add
extmem info:segment_load: loaded segment NETSNMP range 0f800000 ..0fffffff type
SR in shared mode
```

```
dcssblk info: Loaded segment NETSNMP, size = 8388608 Byte, capacity = 16384 512
byte) sectors
Dec 12 17:06:14 linux9s kernel: extmem info:segment_load: loaded segment
NETSNMP range 0f800000 .. 0fffffff type SR in shared mode
Dec 12 17:06:14 linux9s kernel: dcssblk info: Loaded segment NETSNMP,
size = 838 8608 Byte, capacity = 16384 (512 Byte) sectors
```

We now must get exclusive access to the DCSS while we are writing into it. We again use the dcssblk driver to do this (Example 2-20).

Example 2-20 Get exclusive access to the DCSS

```
linux9s:~ # echo 0 > /sys/devices/dcssblk/NETSNMP/shared
```

Now that we have exclusive access, we can define a filesystem on the DCSS (Example 2-21).

Example 2-21 Make a file system on the DCSS

```
linux9s:~ # mke2fs -b 4096 /dev/dcssblk0
mke2fs 1.36 (05-Feb-2005)
Filesystem label= OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
2048 inodes, 2048 blocks
102 blocks (4.98%) reserved for the super user
First data block=0
1 block group
32768 blocks per group, 32768 fragments per group
2048 inodes per group
Writing inode tables: done
Writing superblocks and filesystem accounting information: done
This filesystem will be automatically checked every 37 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
linux9s:~ #
```

Example 2-22 shows mounting the new file system on /xip and copying the files into it.

Example 2-22 Copy of new file

```
linux9:~ # mount /dev/dcssblk0 /xip
linux9s:/dcss # cp -va /dcss/* /xip
'/dcss/lib' -> '/xip/lib"
'/dcss/lib/libdl.so.2' -> '/xip/lib/libdl.so.2'
'/dcss/lib/tls' -> '/xip/lib/tls'
'/dcss/lib/tls/librt.so.1' -> '/xip/lib/tls/librt.so.1'
'/dcss/lib/tls/libpthread.so.0' -> '/xip/lib/tls/libpthread.so.0'
'/dcss/lib/tls/libm.so.6' -> '/xip/lib/tls/libm.so.6'
'/dcss/lib/tls/libc.so.6' -> '/xip/lib/tls/libc.so.6'
'/dcss/lib/libz.so.1.2.1' -> '/xip/lib/libz.so.1.2.1'
'/dcss/lib/ld-2.3.3.so' -> '/xip/lib/ld-2.3.3.so'
'/dcss/usr' -> '/xip/usr'
'/dcss/usr/local' -> '/xip/usr/local'
'/dcss/usr/local/sbin' -> '/xip/usr/local/sbin'
'/dcss/usr/local/sbin/snmpd' -> '/xip/usr/local/sbin/snmpd'
'/dcss/usr/local/lib' -> '/xip/usr/local/lib'
'/dcss/usr/local/lib/libnetsnmpmibs.so.5.2.1' ->
'/xip/usr/local/lib/libnetsnmpmibs.so.5.2.1'
'/dcss/usr/local/lib/libnetsnmpagent.so.5.2.1' ->
'/xip/usr/local/lib/libnetsnmpagent.so.5.2.1'
'/dcss/usr/local/lib/libnetsnmphelpers.so.5.2.1' ->
'/xip/usr/local/lib/libnetsnmphelpers.so.5.2.1'
'/dcss/usr/local/lib/libnetsnmp.so.5.2.1' ->
'/xip/usr/local/lib/libnetsnmp.so.5.2.1'
'/dcss/usr/lib' -> '/xip/usr/lib'
'/dcss/usr/lib/librpm-4.1.so' -> '/xip/usr/lib/librpm-4.1.so'
'/dcss/usr/lib/librpmdb-4.1.so' -> '/xip/usr/lib/librpmdb-4.1.so'
'/dcss/usr/lib/librpmio-4.1.so' -> '/xip/usr/lib/librpmio-4.1.so'
'/dcss/usr/lib/libpopt.so.0.0.0' -> '/xip/usr/lib/libpopt.so.0.0.0'
'/dcss/usr/lib/libbz2.so.1.0.0' -> '/xip/usr/lib/libbz2.so.1.0.0'
```

Now save the DCSS (Example 2-23). This will take the data in the current storage copy of the DCSS and schedule it for writing out to the z/VM SPOOL. Again we use the dcssblk driver to do this.

Example 2-23 Save the segment

```
linux9:~ # echo 1 > /sys/devices/dcssblk/NETSNMP/save
Dec 12 17:06:14 linux9s kernel: dcssblk info: Loaded segment NETSNMP,
size = 838 8608 Byte, capacity = 16384 (512 Byte) sectors
dcssblk info: Segment NETSNMP is currently busy, it will be saved when
it becomes idle...
Dec 12 17:17:41 linux9s kernel: dcssblk info: Segment NETSNMP is
currently busy, it will be saved when it becomes idle...
```

Unmount the file system. This means that no-one is connected to the DCSS any more, so it will be written out (Example 2-24).

Example 2-24 Free the segment so that it will be written

```
umount /xip
HCPNSD440I Saved segment NETSNMP was successfully defined in fileid 0150.
HCPNSS440I Saved segment NETSNMP was successfully saved in fileid 150.
dcssblk info: Segment NETSNMP became idle and is being saved now
Dec 12 17:20:26 linux9s kernel: dcssblk info: Segment NETSNMP became
idle and is being saved now
Finish unloading it:
```

We have now finished using the segment as a disk. We can release it, and then mount it using the new XIP2 file system. Example 2-25 demonstrates how this is done.

Example 2-25 Release the DCSS

```
echo NETSNMP > /sys/devices/dcssblk/remove
linux9s:/dcss # ls -l /sys/devices//dcss*
total 0
drwxr-xr-x  2 root root    0 Dec 12 17:25 .
drwxr-xr-x  7 root root    0 Dec 12 16:56 ..
--w-------  1 root root    0 Dec 12 17:06 add
-rw-r--r--  1 root root 4096 Dec 12 17:06 detach_state
--w-------  1 root root    0 Dec 12 17:25 remove
```

Note that the DCSS has been saved into a new file. You can now remove the original one (Example 2-26) to free the space in SPOOL used by the original DCSS.

Example 2-26 Delete the old segment

```
#cp purge nss 149
```

Now that the DCSS has been written out, and it contains the files we wanted, we can mount it. Example 2-27 demonstrates mounting the DCSS with the XIP2 file system.

Example 2-27 Mount the DCSS with the XIP2 file system

```
linux9s:~ # echo "NETSNMP" > /sys/devices/dcssblk/add
linux9s:~ # mount -t ext2 -o ro,xip /dev/dcssblk0 /xip
```

On the Virtual Machine console you get the messages that the segment was loaded (Example 2-28).

Example 2-28 Messages from loading segment

```
extmem info:segment_load: loaded segment NETSNMP range 0f800000 ..
0fffffff type SR in shared mode
Dec 12 17:28:08 linux9s kernel: extmem info:segment_load: loaded segment
NETSNMP
range 0f800000 .. 0fffffff type SR in shared mode
```

Check /proc/mounts to verify that the filesystem is mounted at the /xip mount point (Example 2-29).

Example 2-29 Checking the mount points

```
linux9s:/dcss # cat /proc/mounts
rootfs / rootfs rw 0 0
/dev/root / ext3 rw 0 0
proc /proc proc rw 0 0
sysfs /sys sysfs rw 0 0
devpts /dev/pts devpts rw 0 0
tmpfs /dev/shm tmpfs rw 0 0
none /xip xip2 ro 0 0
```

These files should now be available for any use. You can verify that they are readable by using the **tar** command to copy them (to /dev/null). Example 2-30 demonstrates how to get to the **tar** command and use it to verify that the files are, indeed, accessible.

Example 2-30 Verify files are accessible

```
linux9s:/dcss # cd /xip
linux9s:/xip # tar cvf /dev/null *
lib/
lib/libdl.so.2
lib/tls/
lib/tls/librt.so.1
lib/tls/libpthread.so.0
lib/tls/libm.so.6
lib/tls/libc.so.6
lib/libz.so.1.2.1
lib/ld-2.3.3.so
lost+found/
usr/
usr/local/
usr/local/sbin/
usr/local/sbin/snmpd
usr/local/lib/
usr/local/lib/libnetsnmpmibs.so.5.2.1
usr/local/lib/libnetsnmpagent.so.5.2.1
usr/local/lib/libnetsnmphelpers.so.5.2.1
usr/local/lib/libnetsnmp.so.5.2.1
usr/lib/
usr/lib/librpm-4.1.so
usr/lib/librpmdb-4.1.so
usr/lib/librpmio-4.1.so
usr/lib/libpopt.so.0.0.0
usr/lib/libbz2.so.1.0.0
```

Now that we know that the DCSS is connected and files in it can be used, we test the "overmounting". This is done so that when any of the programs in this list are fetched, the DCSS copy will be used. The overmounting is done by the "xipinit-fw" shell script that we downloaded from the developerWorks Web site. Example 2-31 demonstrates using the xipinit-fw shell script to test the overmounting.

Example 2-31 Overmount the modules in the DCSS

```
umount /xip
linux9s:~ # /sbin/xipinit-fw
Usage: init 0123456SsQqAaBbCcUu
linux9s:~ # cat /proc/mounts
rootfs / rootfs rw 0 0
/dev/root / ext3 rw 0 0
proc /proc proc rw 0 0
sysfs /sys sysfs rw 0 0
devpts /dev/pts devpts rw 0 0
tmpfs /dev/shm tmpfs rw 0 0
none /xip xip2 ro 0 0
none /lib/libdl.so.2 xip2 ro 0 0
none /lib/tls/librt.so.1 xip2 ro 0 0
none /lib/tls/libpthread.so.0 xip2 ro 0 0
none /lib/tls/libm.so.6 xip2 ro 0 0
none /lib/tls/libc.so.6 xip2 ro 0 0
none /lib/libz.so.1.2.1 xip2 ro 0 0
none /lib/ld-2.3.3.so xip2 ro 0 0
none /usr/local/sbin/snmpd xip2 ro 0 0
none /usr/local/lib/libnetsnmpmibs.so.5.2.1 xip2 ro 0 0
none /usr/local/lib/libnetsnmpagent.so.5.2.1 xip2 ro 0 0
none /usr/local/lib/libnetsnmphelpers.so.5.2.1 xip2 ro 0 0
none /usr/local/lib/libnetsnmp.so.5.2.1 xip2 ro 0 0
none /usr/lib/librpm-4.1.so xip2 ro 0 0
none /usr/lib/librpmdb-4.1.so xip2 ro 0 0
none /usr/lib/librpmio-4.1.so xip2 ro 0 0
none /usr/lib/libpopt.so.0.0.0 xip2 ro 0 0
none /usr/lib/libbz2.so.1.0.0 xip2 ro 0 0
```

Once we know that the overmount script works properly, we have to modify Linux so that this overmounting is done at boot time. We change the call to **init** in "zipl.conf" to use the overmount script "xipinit-fw" (see Example 2-32). This script will run at boot time to do the overmounting and then execute the standard Linux **init** routine.

Example 2-32 Update zipl.conf with the call to xipinit-fw and run zipl

```
linux9s:/etc #cd /etc
linux9s:/etc #vi zipl.conf
parameters = "dasd=0.0.0205,0.0.0306 root=/dev/dasda1 selinux=0
TERM=dumb elevator=cfq mem=256M init=/sbin/xipinit-fw"
linux9s:/etc # zipl -V
Using config file '/etc/zipl.conf'
Target device information
 Device.......................: 5e:00
 Partition....................: 5e:01
 Device name..................: dasda
 DASD device number...........: 0205
 Type.........................: disk partition
 Disk layout..................: ECKD/compatible disk layout
 Geometry - heads.............: 15
 Geometry - sectors...........: 12
 Geometry - cylinders.........: 1113
 Geometry - start.............: 24
 File system block size.......: 4096
 Physical block size..........: 4096
 Device size in physical blocks..: 200316
Building bootmap in '/boot/zipl'
Adding IPL section 'ipl' (default)
 kernel image......: /boot/image at 0x10000
 kernel parmline...: 'dasd=0.0.0205,0.0.0306 root=/dev/dasda1 selinux=0
TERM=dumb elevator=cfq mem=256M init=/sbin/xipinit-fw' at 0x1000
 initial ramdisk...: /boot/initrd at 0x800000
Preparing boot device: dasda (0205).
Syncing disks...
Done.
```

This can be tested by rebooting the system with the **reboot** command.

You should see something like Example 2-33 on the virtual machine console during the boot of the system. This indicates that the DCSS has been loaded and the files are being mounted.

Example 2-33 Messages during boot

```
extmem info:segment_load: loaded segment NETSNMP range 0f800000 ..
0fffffff typ SR in shared mode
```

When the system is back up, check to see if the modules are mounted. Example 2-34 demonstrates that the **mount** command does not show them:

Example 2-34 Mount command does not show overmounts

```
linux9s:~ # mount
/dev/dasda1 on / type ext3 (rw,acl,user_xattr)
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
tmpfs on /dev/shm type tmpfs (rw)
devpts on /dev/pts type devpts (rw,mode=0620,gid=5)
```

But using cat /proc/mounts will, as shown in Example 2-35:

Example 2-35 Use cat command to see overmounts

```
linux9s:~ # cat /proc/mounts
rootfs / rootfs rw 0 0
/dev/root / ext3 rw 0 0
none /xip xip2 ro 0 0
none /lib/libdl.so.2 xip2 ro 0 0
none /lib/tls/librt.so.1 xip2 ro 0 0
none /lib/tls/libpthread.so.0 xip2 ro 0 0
none /lib/tls/libm.so.6 xip2 ro 0 0
none /lib/tls/libc.so.6 xip2 ro 0 0
none /lib/libz.so.1.2.1 xip2 ro 0 0
none /lib/ld-2.3.3.so xip2 ro 0 0
none /usr/local/sbin/snmpd xip2 ro 0 0
none /usr/local/lib/libnetsnmpmibs.so.5.2.1 xip2 ro 0 0
none /usr/local/lib/libnetsnmpagent.so.5.2.1 xip2 ro 0 0
none /usr/local/lib/libnetsnmphelpers.so.5.2.1 xip2 ro 0 0
none /usr/local/lib/libnetsnmp.so.5.2.1 xip2 ro 0 0
none /usr/lib/librpm-4.1.so xip2 ro 0 0
none /usr/lib/librpmdb-4.1.so xip2 ro 0 0
none /usr/lib/librpmio-4.1.so xip2 ro 0 0
none /usr/lib/libpopt.so.0.0.0 xip2 ro 0 0
none /usr/lib/libbz2.so.1.0.0 xip2 ro 0 0
proc /proc proc rw 0 0
sysfs /sys sysfs rw 0 0
devpts /dev/pts devpts rw 0 0
tmpfs /dev/shm tmpfs rw 0 0
linux9s:~ #
```

Your system is now up and running with the SNMP daemon in an XIP2 DCSS. If you make the same changes to all your Linux images, then the snmp code will be shared among all these images.

3

Using DCSS for Oracle

This chapter discusses the method we used to set up a DCSS containing the Oracle RDBMS code, so that multiple Linux guests could share this code.

The method of implementing is very similar to the implementation of a DCSS as discussed in Chapter 2, however there are a few differences.

The documentation for the XIP2 file system is in *How to use Execute-in-Place Technology with Linux on z/VM,* SC33-8287, which you can find at:

http://www-128.ibm.com/developerworks/linux/linux390/october2005_documentation.html

This documentation, and the supporting sample shell scripts assume that the code to be loaded in the DCSS is located in the root file system. Because Oracle is an application, and is not in the root file system, we modified the sample shell scripts, and used a different method of initializing the DCSS. This chapter documents this approach.

3.1 Sizing the DCSS

After many experiments, we decided that the most effective way to use the DCSS is to share an entire directory, and only put the code that we knew would be executed into the DCSS. In our tests we obtained the best results by putting only two modules in the DCSS.

To make effective use of the DCSS, you should ensure that the DCSS is sized correctly. We decided to create a directory and put only the **oracle**, and **sqlplus** binaries in it. We used the **ls** command to determine the size of these modules, as demonstrated in Example 3-1.

Example 3-1 Get size of modules

```
pazxxt19:/local/10g/db/dcss # ls -al
total 87748
drwxr-xr-x   2 root   root     4096 Feb  9 17:39 .
drwxr-xr-x  62 oracle dba      4096 Dec  9 12:46 ..
-rwxr-x--x   1 oracle dba  89736931 Oct 28 09:26 oracle
-rwxr-x--x   1 oracle dba     12131 Oct 28 09:27 sqlplus
```

We decided that 100 MB was big enough to hold the two modules that we wanted to put into the DCSS.

3.2 Determining DCSS addresses

As the DCSS in Example 3-1 is 100 MB, the next step is to determine the starting address of the DCSS. For this example, the virtual machine size is 512 MB. To get the most benefit in this case from the DCSS, the virtual machine will be reduced in size by 100 MB and the DCSS will be placed just above the virtual machine. The DCSS starting address will be:

512 MB - 100 MB = 412 MB

For creating the DCSS, the page number must be determined to supply to the CP command that creates the DCSS. The starting address of 412 MB must be converted to hex, and this then converted to the page number. The DCSS start address of 248 MB is address hex “F800000”. The end address is hex “1FFFFFFF”, or 512 MB minus 1. The page address is then the hexadecimal value truncated, so in this case, starting page is 19C00, and ending page is 1FFFF.

For different applications, a different sized DCSS could be more appropriate.

3.3 Tailoring Linux storage size to support the DCSS

The virtual machine should now be reduced by 100 MB, and then Linux needs to be configured to address the DCSS. The configuration file “/etc/zipl.conf”: requires modification. To do this, we use the standard **vi**, as shown in Example 3-2.

Example 3-2 Changing the zipl.conf file

```
cd /etc
vi zipl.conf
```

Add “**mem=512M”** to the parameter line as shown in Example 3-3.

Example 3-3 zipl.conf file

```
parameters = "root=/dev/dasdb1 selinux=0 TERM=dumb elevator=deadline
dasd=200-209,300-30f,400-40f,fixedbuffers mem=512M"
```

Run **zipl** to update everything. You may want to use the -V parameter to get details on zipl's actions, as shown in Example 3-4.

Example 3-4 Output from zipl -V command

```
zipl -V
Using config file '/etc/zipl.conf'
Target device information
  Device......................: 5e:00
  Partition...................: 5e:01
  Device name.................: dasda
  DASD device number..........: 0200
  Type........................: disk partition
  Disk layout.................: ECKD/compatible disk layout
  Geometry - heads............: 15
  Geometry - sectors..........: 12
  Geometry - cylinders........: 200
  Geometry - start............: 24
  File system block size......: 4096
  Physical block size.........: 4096
  Device size in physical blocks..: 35976
Building bootmap in '/boot/zipl'
Adding IPL section 'ipl' (default)
  kernel image......: /boot/image at 0x10000
  kernel parmline...: 'root=/dev/dasdb1 selinux=0 TERM=dumb elevator=deadline
dasd=200-209,300-30f,400-40f,fixedbuffers mem=512M' at 0x1000
  initial ramdisk...: /boot/initrd at 0x1000000
Preparing boot device: dasda (0200).
Syncing disks...
```

```
Done.
```

3.4 Creating the DCSS

We chose to create the DCSS from CMS. In order to have the authority to create and populate a DCSS, the z/VM directory entry for the CMS user must be defined as Class E privileges. There must also be a **NAMESAVE** statement in the directory entry. These privileges allow the CMS user to define the DCSS. In Example 3-5, we called the DCSS “ORADCSS”.

Example 3-5 Defining the DCSS

```
cp defseg oradcss 19c00-1ffff sr
Saved segment ORADCSS was successfully defined in fileid 0501.
```

Validating the NSS is done with the **QUERY** command, or simply **Q**, as shown in Example 3-6.

Example 3-6 Saving the DCSS to create the storage image

```
#CP Q NSS MAP
FILE FILENAME FILETYPE MINSIZE  BEGPAG ENDPAG TYPE CL #USERS PARMREGS VMGROUP
0401 LNXSEG1  DCSS       N/A    54000  7FEFF   SR  A  00000   N/A       N/A
0473 NETSNMP  DCSS       N/A    5C000  5FFFF   SR  A  00000   N/A       N/A
...
0475 ORADCSS  DCSS       N/A    19C00  1FFFF   SR  S 00000   N/A       N/A
```

Note that after the **saveseg command** in Example 3-7, the class of the DCSS has changed from “S” for Skeleton to “A” for Active.

Example 3-7 Results of the “saveseg oradcss” command.

```
#CP saveseg oradcss
Saved segment ORADCSS was successfully saved in fileid 0475.
CP Q NSS MAP
FILE FILENAME FILETYPE MINSIZE  BEGPAG ENDPAG TYPE CL #USERS PARMREGS VMGROUP
0401 LNXSEG1  DCSS       N/A    54000  7FEFF   SR  A  00000   N/A       N/A
0473 NETSNMP  DCSS       N/A    5C000  5FFFF   SR  A  00000   N/A       N/A
...
0475 ORADCSS  DCSS       N/A    60000  7FEFF   SR  A  00000   N/A       N/A
```

3.5 Set the Linux size and boot Linux

Now that you have saved the DCSS, you have to change the size of your Linux virtual machine so that its pages don't overlap with the DCSS pages when the DCSS is loaded. To do this, either change the directory for the Linux virtual machine, or use the **define storage command in the Linux machine**. Changing the virtual machine size with the **define storage** command works immediately, but the change will be lost if you logoff the virtual machine.

After your system is booted, check to ensure the "mem=" parameter is present and correct by using **cmdline**, as shown in Example 3-8.

Example 3-8 Verify zipl changes were done correctly

```
pazxxt19:~ # cat /proc/cmdline
dasd=0.0.0205,0.0.0306 root=/dev/dasda1 selinux=0 TERM=dumb elevator=cfq
mem=512M BOOT_IMAGE=0
```

3.6 Resize your virtual machine and reboot

Now that you have saved the DCSS, you have to change the size of your virtual machine so that its pages will not overlap with the DCSS pages when the DCSS is loaded. To do this, shutdown your system, resize the virtual machine, and reboot. Issue the commands shown in Example 3-9 to do this Linux.

Example 3-9 Shutdown, change storage size and reboot

```
      pazxxt19:~ # shutdown -h now
      Broadcast message from root (pts/0) (Thu Dec 22 10:57:27 2005):
      The system is going down for system halt NOW!
       ...
      The system will be halted immediately.

...

      Power  down.
      HCPGSP2630I The virtual machine is placed in CP mode due to a SIGP stop
      and store status from CPU 00.
      DEF STOR 412M
      STORAGE = 0412M
      Storage cleared - system reset.
      IPL 205
```

Changing the virtual machine size with the **define storage** command works immediately, but the change will be lost if you logoff the virtual machine. To make this a permanent change, update the CP directory entry for this virtual machine.

The size of the machine is on the **USER** statement. This will ensure that the virtual machine size will be correct through system restarts.

After your system is back up, ensure the mem= parm is present and correct by using the **cmdline** command, as shown in Example 3-10.

Example 3-10 Verify zipl changes were done correctly

```
pazxxt19:~ # cat /proc/cmdline
dasd=0.0.0205,0.0.0306 root=/dev/dasda1 selinux=0 TERM=dumb elevator=cfq
mem=256M BOOT_IMAGE=0
```

3.7 Setup for the XIP2 DCSS

This DCSS differs from the SNMP DCSS discussed in 2.7, “Setup for the XIP2 DCSS” on page 12, in that it is being defined in an LVM outside of the root file system. We can use a similar technique to fill the DCSS with the programs to be shared, but we cannot use the same technique to initialize the DCSS after booting. In Example 3-11, we will create a directory to hold the modules we want to populate the DCSS.

We then create a directory under “oracle\home” called “dcss” in which we move the modules that we will be sharing with “dcss”. Before the move, we copy the modules for backup purposes. Once we move the modules we create a symbolic link to the new location of these modules.

Example 3-11 Create the directory for the mount point

```
mkdir $ORACLE_HOME/dcss
cp $ORACLE_HOME/bin/oracle $ORACLE_HOME/bin/oracle.org
cp $ORACLE_HOME/bin/sqlplus $ORACLE_HOME/bin/sqlplus.org
mv oracle $ORACLE_HOME/dcss/oracle
mv sqlplus $ORACLE_HOME/dcss/sqlplus
ln -s $ORACLE_HOME/dcss/oracle $ORACLE_HOME/bin/oracle
ln -s $ORACLE_HOME/dcss/sqlplus $ORACLE_HOME/bin/sqlplus
```

Now load the DCSS driver with the **modprobe** command. This driver is supplied in the Linux distribution, and allows you to treat the DCSS as a disk so that you can read and write data to it. We will use this driver to create a filesystem on the DCSS and copy the files we want to share into this filesystem. Example 3-12 shows the **modprobe** command that will load the block driver.

Example 3-12 Load the DCSS block mode driver

```
pazxxt19:~ # modprobe dcssblk
```

In order to define devices to use this driver, we need to define some nodes. To do this we need the driver's major number. To determine this number we look at the /proc/devices file, as shown in Example 3-13. We see that the dcssblk driver is there and the major number is 252.

Example 3-13 Find the driver's major number

```
pazxxt19:~ # cat /proc/devices
Character devices:
  1 mem
  4 ttyS
  4 ttysclp
  5 /dev/tty
  5 /dev/console
  5 /dev/ptmx
  9 st
 10 misc
 21 sg
128 ptm
136 pts
227 ttyTUB
228 fs3270

Block devices:
  1 ramdisk
  7 loop
  8 sd
  9 md
 11 sr
 65 sd
 66 sd
 67 sd
 68 sd
 69 sd
 70 sd
 71 sd
 94 dasd
128 sd
129 sd
130 sd
131 sd
132 sd
133 sd
134 sd
135 sd
252 dcssblk
253 device-mapper
254 mdp
```

The next step is to create the device nodes, if this is required. Example 3-14 demonstrates the command used to verify whether or not the device nodes need to be created. In this example, we see that the output from the **ls -l /dev/dcss*** command returns “No such file or directory”, so the next step would be to create the device nodes, as indicated in the example by the use of the **mknod** commands for each node that we want to create.

Example 3-14 Verify device nodes and create if necessary

```
pazxxt19:~ # ls -l /dev/dcss*
/bin/ls: /dev/dcss*: No such file or directory

pazxxt19:~ # mknod /dev/dcssblk0 b 252 0
pazxxt19:~ # mknod /dev/dcssblk1 b 252 1
pazxxt19:~ # mknod /dev/dcssblk2 b 252 2
pazxxt19:~ # mknod /dev/dcssblk3 b 252 3

pazxxt19:~ # ls -l /dev/dcss*
brw-r--r--  1 root root 252, 0 Oct 12 17:15 /dev/dcssblk0
brw-r--r--  1 root root 252, 1 Oct 12 17:16 /dev/dcssblk1
brw-r--r--  1 root root 252, 2 Oct 12 17:16 /dev/dcssblk2
brw-r--r--  1 root root 252, 3 Oct 12 17:16 /dev/dcssblk3
```

3.8 Load the DCSS storage with the files to be shared

At this point, we have created a DCSS segment that will be loaded from address 412 MB to 512 MB. The machine is defined to be 412 MB in size and rebooted at this size. The Linux command line has been modified to support addresses up to 512 MB.

The dcssblk driver is used to manipulate the DCSS (Example 3-15).

Example 3-15 Use the dcssblk driver to load the DCSS into storage

```
extmem info:segment_load: loaded segment ORADCSS range 0000000019c00000 ..
000000001fffffff type SR in shared mode
dcssblk info: Loaded segment ORADCSS, size = 104857600 Byte, capacity = 204800
(512 Byte) sectors
Feb  9 17:37:02 pazxxt19 kernel: extmem info:segment_load: loaded segment
ORADCSS range 0000000019c00000 .. 000000001fffffff type SR in shared mode
Feb  9 17:37:02 pazxxt19 kernel: dcssblk info: Loaded segment ORADCSS, size =
104857600 Byte, capacity = 204800 (512 Byte)sectors
```

We now must get exclusive access to the DCSS while we are writing into it. We again use the dcssblk driver to do this (Example 3-16).

Example 3-16 Get exclusive access to the DCSS

```
pazxxt19:~ # echo 0 > /sys/devices/dcssblk/ORADCSS/shared
```

Now that we have exclusive access, we can define a filesystem on the DCSS (Example 3-17).

Example 3-17 Make a filesystem on the DCSS

```
pazxxt19:~ # mke2fs -b 4096 /dev/dcssblk0
mke2fs 1.36 (09-Feb-2005)
Filesystem label= OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
2048 inodes, 2048 blocks
102 blocks (4.98%) reserved for the super user
First data block=0
1 block group
32768 blocks per group, 32768 fragments per group
2048 inodes per group
Writing inode tables: done
Writing superblocks and filesystem accounting information: done
This filesystem will be automatically checked every 37 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
pazxxt19:~ #
```

Example 3-18 shows how we mount the new filesystem under /mnt to create the directory structure and copy the modules. We need to make sure we have the same directory structure we have in the Oracle home directory, this is why we will be creating local/10g/db under /mnt so when we mount the DCSS filesystem it will overmount on /local/10g/db/dcss, where /local/10g/db is the Oracle home directory.

Example 3-18 Mount new filesystem

```
pazxxt19:~ # mount /dev/dcssblk0 /local/10g/db/dcss
pazxxt19:~ # cp -va /dcss/* /local/10g/db/dcss

pazxxt19:~# export ORACLE_HOME=/local/10g/db
pazxxt19:~ # mount /dev/dcssblk0 /mnt
pazxxt19:~ # cd /mnt
pazxxt19:/mnt # mkdir -p local/10g/db/dcss
pazxxt19:/mnt # cp -av $ORACLE_HOME/dcss/* local/10g/db/dcss
`/local/10g/db/dcss/oracle' -> `local/10g/db/dcss/oracle'
```

```
`/local/10g/db/dcss/sqlplus' ->
`local/10g/db/dcss/sqlplus'
```

Example 3-19 shows the next step, saving the DCSS. This will take the data in the current storage copy of the DCSS and schedule it for writing out to the z/VM SPOOL. Again we use the dcssblk driver to do this.

Example 3-19 Save the segment

```
pazxxt19:/ # echo 1 > /sys/devices/dcssblk/ORADCSS/save
dcssblk info: Segment ORADCSS is currently busy, it will be saved when it
become
s idle...
Feb 20 14:24:39 pazxxt19 kernel: dcssblk info: Segment ORADCSS is currently
busy , it will be saved when it becomes idle...
```

Unmount the file system as shown in Example 3-20. This means that no one is connected to the DCSS any more, so it will be written out.

Example 3-20 Free the segment so that it will be written

```
pazxxt19:/ # umount /mnt
HCPNSD440I Saved segment ORADCSS was successfully defined in fileid 0506.
HCPNSS440I Saved segment ORADCSS was successfully saved in fileid 0506.
dcssblk info: Segment ORADCSS became idle and is being saved now
Feb 20 14:26:33 pazxxt19 kernel: dcssblk info: Segment ORADCSS became idle and
is being saved now
```

We have now finished using the segment as a disk. We can release it, and then mount it using the new XIP2 filesystem (Example 3-21).

Example 3-21 Free the DCSS after populating it

```
echo ORADCSS> /sys/devices/dcssblk/remove
pazxxt19:/dcss # ls -l /sys/devices/dcss*
total 0
drwxr-xr-x  2 root root    0 Dec 12 17:25 .
drwxr-xr-x  7 root root    0 Dec 12 16:56 ..
--w-------  1 root root    0 Dec 12 17:06 add
-rw-r--r--  1 root root 4096 Dec 12 17:06 detach_state
--w-------  1 root root    0 Dec 12 17:25 remove
```

Unlike the example shown in Chapter 2, “Putting the SNMP daemon into a DCSS” on page 5, we are sharing a complete directory in this DCSS. The example script “xipinit.sh” is the supplied code which initializes a DCSS, doing the “overmounting” that makes the files available. We copy this file to /etc/init.d

with a new name, and modify it to point to our environment. We then modify rc2.d, rc3.d, and rc5.d to cause these scripts to be executed after the boot sequence (see Example 3-22).

Example 3-22 Update xipinit.sh and put it into /etc/init.d

```
pazxxt19:~ # cd /etc/init.d
pazxxt19:/etc/init.d # cp /local/scripts/xip-howto-scripts/xipinit.sh
init.oradcss
vi init.dcss
```

using vi set:

```
#mount point of xipimage
ROMOUNT="/oraxip"
#name of xipimage
XIPIMAGE="ORADCSS"
#name of device node
RODEV="/dev/dcssblk0"

RODIRS=/local/10g/db/dcss
# make sure it ends with ,
RODIRS="$RODIRS",

Also inset a # in the last line to make the call to init a comment:
# run real init
#$FAKE exec /sbin/init "$@"
```

In directory /etc/init.d, there are directories rc2.d, r3.d and r5.d which contain a sequence of scripts that get executed when Linux enters runlevel 2, 3 or 5. When we installed Oracle on this Linux machine, a file named “S17init.cssd” was created in these three directories. This shell script initializes the cssd component of Oracle. Our shell, “script init.oradcss”, must get invoked before the cssd.

initialization. These initialization scripts get executed in sequence. So we rename the Oracle script from “S17init.cssd” to “S18init.cssd” and insert our script as “S17init.oradcss” (see Example 3-23).

Example 3-23 Rename the init call for Oracle cssd

```
pazxxt19:/etc/init.d # find . -name S17init.cssd
./rc2.d/S17init.cssd
./rc3.d/S17init.cssd
./rc5.d/S17init.cssd

pazxxt19:/etc/init.d # mv ./rc2.d/S17init.cssd ./rc2.d/S18init.cssd
pazxxt19:/etc/init.d # mv ./rc3.d/S17init.cssd ./rc3.d/S18init.cssd
pazxxt19:/etc/init.d # mv ./rc5.d/S17init.cssd ./rc5.d/S18init.cssd
```

Example 3-24 shows how we insert a call to our initialization routine as “S17init.oradcss”. This will get executed before the initialization of Oracle cssd because it is now named “S18init.cssd”.

Example 3-24 Insert call to our initialization routine in sequence

```
pazxxt19:/etc/init.d # ln -s /etc/init.d/init.oradcss ./rc2.d/S17init.oradcss
pazxxt19:/etc/init.d # ln -s /etc/init.d/init.oradcss ./rc3.d/S17init.oradcss
pazxxt19:/etc/init.d # ln -s /etc/init.d/init.oradcss ./rc5.d/S17init.oradcss
```

A reboot of the machine should load and initialize the Oracle DCSS. Example 3-25 demonstrates this.

Example 3-25 Boot up message

```
<notice>start services (local init.oradcss)
Feb 20 14:44:13 pazxxt19 kernel: extmem info:segment_load: loaded segment
ORADCSS range 0000000019c00000  .. 000000001fffffff type SR in shared mode.
```

3.9 Use same segment in a “hole” in memory

Now that this segment is built to load from 412 MB to 512 MB, it could be used by other, larger Linux guests. We had to add mem=512M to the “zipl.conf” file so that Linux could address up to 512 MB even though the machine was only 412 MB in size. If we remove the mem= parameter from “zipl.conf”, Linux will be able to address the size of the machine where it is IPLed.

z/VM allows you to define storage with a “hole” in the physical memory using the config form of the define storage command as shown in Example 3-26.

Example 3-26 Defining storage with a hole

```
def stor config 0.412M 512M.1536m
STORAGE = 1948M
Storage Configuration:
0.412M 512M.1536M
Extent Specification               Address Range
---------------------------------- ----------------------------------
                0.412M             0000000000000000 - 0000000019BFFFFF
             512M.1536M            0000000020000000 - 000000007FFFFFFF
```

A machine defined this way can load the same ORADCSS as created earlier in this chapter because the DCSS will be loaded in the hole in the address space. The mem= parameter on the Linux guest should not be specified. With the above definition of storage Linux will be able to address up to 2 GB.

4

DCSS measurements

This chapter describes the tests and measurements we made to show that DCSS was actually saving memory with multiple guests when using SNMP and Oracle.

4.1 Oracle and XIP2 objectives

The execute in place (XIP2) filesystem allows Linux servers to share programs, executing the code directly out of the DCSS. The objective is that with many Linux servers sharing storage, programs only need to be resident once and should allow for significant storage savings.

There are trade-offs that must be understood before implementing XIP2. The trade-off is between the costs of addressing additional storage, the cost of the DCSS, and then the savings in Linux storage. At a minimum, the savings in Linux storage *must* be larger than the costs in addressing the DCSS. Otherwise, storage will be lost for each server utilizing the DCSS, missing our objective. The improvement in Linux storage for one server will not exceed the cost of the DCSS itself, but will be hopefully covered on the second or third server utilizing the DCSS. If the savings are so low and the cost of the DCSS is very high, then it may take more servers utilizing the DCSS than exist, again missing the objective.

Recognizing that the objectives are:

- Minimizing the cost of storage addressability
- Minimizing the cost of the DCSS
- Maximizing the Linux storage savings

Each of these should be measured, managed and are not completely intuitive. Much of the analysis is to understand Linux storage, since the first measurements using “common knowledge” proved to not reduce storage as much as expected. It was found that performance improvements using XIP2 could not be taken for granted.

ESALPS Version 4.1, the Linux Performance Suite was provided by Velocity Software for this analysis. More information about ESALPS can be found in the Appendix A, “ESALPS” on page 61.

4.2 Experiment objectives

Installations running a few large (2 GB - 6 GB) servers will not see a large gain from implementing an XIP2 file system in a DCSS. The objective of our experimentation, therefore, is for installations that want to run 20 or 30 Oracle servers in the 256M to 512M range will see the real benefit from adding servers without adding more real storage.

4.3 XIP2 analysis findings

For Oracle servers, the XIP2 filesystem in a DCSS was shown to reduce storage when created correctly. It was also found that the implementation could create storage overheads that exceeded the benefit. Most of the study was in determining how to optimize the XIP2 implementation. This is the first time this kind of analysis has been published, possibly the first time ever done. There were several surprises that, without measurements, would never have been realized. It became obvious that measuring both the Linux storage down to the process level and the z/VM storage in a constrained environment gave a different set of answers than had been expected.

There are two ways that the XIP2 feature may be used. One method is to load an entire directory into the DCSS. This was found to be very ineffective due both to the high cost of addressability and the inflexibility of what files can be shared.

The other is to load individual files into the DCSS. This was found to be most effective in using the XIP2 DCSS feature. The DCSS should be loaded with only frequently used programs and it should have a minimal amount of free space. This is slightly more complex, but provides much more measurable benefits.

Another finding was in where to place the DCSS. Three methods of placing a DCSS may be used:

- Adding a DCSS at the address just above the Linux servers
- Adding a DCSS at the highest possible storage
- Putting the DCSS into a “hole”. Current z/VM allows defining a virtual machine with a gap in the middle of virtual machines and thus reserving a place for the DCSS. For installations that want to implement a DCSS when using machines that are 2 GB or larger, this is the only option because, at least up to z/VM V5.2, DCSS must be below the 2 GB line.

Each method has reasons for doing it that way. However, adding a DCSS to an existing Linux server adds storage to the Linux server and no savings will be obtained, in fact the storage requirements will increase. Linux will always use all available storage, the only way to force a reduction in storage requirements is to reduce the virtual machine storage size. In implementing a DCSS with XIP2, the Linux server storage MUST be reduced, with a recommendation of reducing the storage by the size of the DCSS.

The implementation most likely to have minimal storage savings is when adding the DCSS in high storage.

The most significant benefit found in these experiments using XIP2 and DCSS was to carefully pack the DCSS with only frequently used programs, place the

DCSS in as low a storage position as possible, and then reduce the size of the virtual machine.

The value in including libraries in the DCSS is likely positive, but further experiments are needed to validate this benefit.

4.4 Linux storage

This section examines storage overheads and how to measure them.

4.4.1 XIP2 storage overheads

There are three measurable costs associated with using a DCSS for XIP2 that can be greater than the benefits without management. Two of the three measurable costs are related to addressability. To address additional storage, Linux needs to increase its page management structure tables and page tables. Thus, having a DCSS in high storage may cost more storage in overhead than save in shared storage.

4.4.2 Storage addressability overhead

In the XIP2 in DCSS measurements, the “mem=” parameter was used to increase the amount of storage that Linux can address when adding a DCSS outside the virtual machine. This parameter builds the page management structures required to address additional storage that Linux perceives when it boots. This would not be necessary if using the “hole” approach. Several experiments were done with just changing the “mem=” parameter to show the storage costs in a static environment and to understand the storage costs.

- Addressability overhead shows up in the size of the kernel. This increases due to page management structure tables. Measurements showed the following costs for 31-bit Linux:
- Page management structures (struct page entries) for 31-bit Linux overhead is 8 MB per gigabyte of addressed storage (measured in kernel storage). The entries are 32 bytes per page.
- An extra megabyte per gigabyte of overhead was identified, but the cause of this was not determined.

The costs using 64-bit Linux are double that of when using 31-bit Linux and therefore needs to be managed even more carefully. Page management structure overhead is 16 MB per gigabyte (measured in kernel storage). The entries are 64 bytes per page.

4.4.3 Linux storage analysis

ESALPS provides several measures of storage. In the following example of an ESAUCD2 report, the measures of storage are:

- Kernel - this includes the kernel and page management structures. This is the difference in size between what Linux reports as “total Real Storage” and the virtual machine size.
- Available - storage that is not in use by Linux. This is always low. The only value that this number has is when it is large; this shows that Linux has not been able to find any data to put into storage.
- Buffer - storage in the write buffer, typically small
- Cache - dynamic storage in use for caching data and other Linux storage items. Note that this includes the Oracle SGA, as well as all executables.
- Overhead - storage in use by control blocks and page tables.

An experiment was performed in analyzing just the impact of “mem=”. The following report shows two 31-bit SUSELNX servers that are identical except for the defined memory size. Note the difference in the total real storage size, by increasing addressability to 1G from 256M; the total storage size has decreased 8 MB, the kernel size has increased 8 MB.

Example 4-1 ESAUCD2 report

```
Report: ESAUCD2      LINUX EXAMPLE           Linux Test
-----------------------------------------------------
Node/    <------------------Mega Bytes--------------->
Time/    <--Real Storage--> . <----Storage in Use----->
Date     Total  Avail Used  . Shared Buffer Cache Ovrhd
-------- ------ ----- ----- . ------ ------ ----- -----
10:26:00                    .
SUSELNX1  239.3 175.8  63.5 .      0   18.6  26.1  18.8
SUSELNX2  247.3 182.8  64.5 .      0   18.5  26.2  19.8
```

The kernel size is the difference between the virtual machine size, in this case, 256 MB for both servers and the reported Total Real Storage size.

What this analysis shows is that if a DCSS was defined at the 512M mark for 512M, the mem= would have to be 1.G, adding a cost of storage of 8 MB to an existing 256 MB 31-bit server.

4.5 Linux virtual storage analysis

Before analyzing storage impacts of the DCSS, it was noticed that the mem= parameter in Linux had impacted Linux storage requirements in different areas. The kernel size increased for one, as did the overhead storage. The following table shows the results of several simple experiments measuring the impact of “mem=”.

These measurements were taken to help understand how to measure the impact of increasing Linux addressability. ESALPS was used to show the kernel size and the overhead storage size for several different configurations first in 31-bit mode, and then in 64-bit mode.

All values in the following tables are in megabytes.

4.5.1 31-bit Linux analysis

Table 4-1 Memory analysis

mem=	Kernel Size	Cache Size	Buffer Size	Overhead Storage
256	8.3	25.1	3.8	16.7
256	8.3	25.2	3.7	16.8
512	10.6	18.9	3.9	18.0
528	10.8	25.2	3.7	19.6
1024	15.2	25.2	3.8	19.6
1040	15.3	25.2	4.0	22.2
1552	19.8	25.1	4.1	22.4

In this study, the virtual machine was 256 MB. The objective was to determine addressability costs. By analyzing this data, and the slope of the line, the kernel size increases at a rate of 9 MB per GB of defined storage. The overhead storage also increased at a rate of 4 MB per GB. Thus, adding a 256 MB DCSS to a 256 MB virtual machine would cost about 3.5 MB of Linux virtual storage. It was later determined that this storage is referenced and not paged out by VM.

4.5.2 64-bit Linux analysis

From the following two samples extracted from an ESAUCD2 report, the first set is three machines, all with machine size of 1024, and mem=2047. The second set is 512 MB servers, but with mem=1024. The static storage then for the kernel

is 43 MB for the first, and 27 MB for the second. Arithmetic says that the cost of the 1 GB extra of memory addressability costs 16 MB of Linux virtual storage, and further arithmetic would indicate that the fixed cost of the 64-bit kernel is about 11 MB (27 MB - 16 MB).

Example 4-2 ESAUCD2 report

```
Report: ESAUCD2      LINUX U
Monitor initialized:
----------------------------
Node/    <------------------
Time/    <--Real Storage-->
Date     Total  Avail Used
-------- ------ ----- -----
----------------------------
PAZXXT19  981.1 826.3 154.8
PAZXXT21  981.1 751.1 230.0
PAZXXT20  981.1 776.4 204.7
----------------------------
PAZXXT19  485.1  10.3 474.8
PAZXXT20  485.1  11.6 473.5
PAZXXT21  485.1  10.2 475.0
```

4.6 VM real storage analysis

One potential error in the mem= analysis is that if Linux does not reference pages, then the extra page management structure tables are paged out by z/VM and would not take storage other than during initialization.

4.6.1 Measuring mem= impact on VM working set

The following analysis shows first with 64 MB addressable, and the second with 1024 MB. Both measurements were done with a 64 MB Linux server, with the server being tested running a minimal configuration, including netsnmp for instrumentation. A second server was started with a very large virtual machine that caused memory contention to force our test server to be paged out, leaving only the referenced pages resident.

The test then came when a new process was started to see if there was an impact on storage that was different between the two tests. The ”top” process was used as its characteristics should be very familiar to Linux users.

This analysis shows that the VM working set increases about 8 MB when the mem=1024M is used, as expected for 31-bit Linux. 64-bit Linux would be double.

Therefore using a DCSS at an address just below the 2 GB line for a 512 MB server for a 64-bit server will add 24 MB to its virtual storage requirements *and* its working set. Thus the savings must be much more than 24 MB per server for this to be worthwhile. If the same DCSS was compressed to 100 MB, and put at the 512 line, then the added overhead would be less than 2 MB, and much more significant savings are then possible.

4.6.2 Test 1, mem=64M

The following report shows the active process analysis during the experiment. The ESALNXP report shows each active process, its process ID, parent ID, CPU and storage information as provided in the Linux process table. Following is the z/VM storage analysis. Note that the SNMP daemon and some background Linux tasks (pdflush, slpd) were running throughout the measurement. Then top was started at 14:57. This made bash and sshd active (to run the terminal session), touching a lot of pages.

The interval at 14:55 has the least amount of processes running and should have the lowest working set.

Example 4-3 ESALNXP report for test 1

```
Report: ESALNXP      LINUX HOST Process Statistics Report
------------------------------------------------------------------------
node/     <-Process Ident-> <-----CPU Percents----------> <Stg (k)>
 Name     ID    PPID   GRP   Tot  sys user syst usrt nice  Size RSS
--------- ----- ----- ----- ---- ---- ---- ---- ---- ---- ---- ----
14:53:00
SUSELNX1      0     0     0  9.5  8.6  1.0    0    0    0 135K  53K
 pdflush   1509     4     0  0.1  0.1    0    0    0    0    0    0
 snmpd     1817     1  1816  2.6  2.3  0.3    0    0    0 7060 4016
 slpd      1832     1  1832  0.1  0.1  0.0    0    0    0 3588 1244
------------------------------------------------------------------------
14:54:00
SUSELNX1      0     0     0 10.0  8.1  1.8    0    0    0 135K  53K
 init         1     0     0  0.5  0.4  0.0    0    0    0  628  100
 kblockd/     6     4     0  0.1  0.1    0    0    0    0    0    0
 pdflush   1509     4     0  0.4  0.4    0    0    0    0    0    0
 snmpd     1817     1  1816  4.0  3.0  1.0    0    0    0 7060 4016
 slpd      1832     1  1832  0.2  0.2  0.0    0    0    0 3588 1244
------------------------------------------------------------------------
14:55:00
SUSELNX1      0     0     0  6.3  4.0  2.3    0    0    0 135K  53K
 snmpd     1817     1  1816  2.6  1.6  1.0    0    0    0 7060 4016
------------------------------------------------------------------------
14:56:00
SUSELNX1      0     0     0  9.7  6.2  3.5    0    0    0 135K  53K
```

```
snmpd     1817     1  1816  3.8  2.4  1.4    0    0    0 7060 4016
 sshd      2073  1868  2073  0.3  0.3  0.0    0    0    0 8392 2576
------------------------------------------------------------------
14:57:00
   SUSELNX1      0     0     0 13.3  8.8  4.5    0    0    0 137K  54K
    kjournal   277     1     1  0.2  0.2    0    0    0    0    0    0
    snmpd     1817     1  1816  2.4  1.7  0.8    0    0    0 7060 4016
    sshd      2073  1868  2073  0.6  0.4  0.2    0    0    0 8392 2580
    bash      2076  2073  2076  0.6  0.5  0.1    0    0    0 3204 1952
    top       2095  2076  2095  2.0  1.1  0.9    0    0    0 2132 1100
   ------------------------------------------------------------------
   14:58:00
   SUSELNX1      0     0     0 19.9 12.5  7.4    0    0    0 137K  54K
    kjournal   277     1     1  0.1  0.1    0    0    0    0    0    0
    snmpd     1817     1  1816  5.2  3.8  1.3    0    0    0 7060 4016
    sshd      2073  1868  2073  0.7  0.4  0.3    0    0    0 8392 2580
    top       2095  2076  2095  2.7  1.1  1.6    0    0    0 2132 1100
```

The next part of the test was to show what happens on the VM side. Note that the numbers used for Linux are in “k”, and the numbers for z/VM are in pages (4k).

The following ESAUSR2 report shows resource requirements by virtual machine. In analyzing the SUSELNX1 server during the experiment, its total resident storage started at 16K pages, the full 64 MB of the virtual machine. When there was contention this dropped to 1160 pages which would be just the referenced pages for when just Linux and the SNMP daemon were running. 1160 pages is assumed to be the minimum working set for this configuration.

When pdflush started, the VM resident pages went up to 1460 pages, and when top started, the resident pages went up to 4242 pages. When top starts, top runs under bash, and sshd runs to validate the logon.

Example 4-4 ESAUSR2 report for test 1

```
   Report: ESAUSR2      User Resource     Linux Test
   ------------------------------------------------------------------
            <---CPU time--> <-Pages-->  <------Paging (pages)------->
   UserID   <(seconds)> T:V <Resident>  <---Allocated---> <---I/O--->
   /Class   Total  Virt Rat Totl Activ  Total ExStg  Disk  Read Write
   -------- ----- ----- --- ---- -----  ----- -----  ----- ----- -----
   14:47:00
   SUSELNX1  2.48  2.03 1.2  16K 16222      0     0     0     0     0
   (All storage resident)
   14:48:00
   SUSELNX1  2.47  2.03 1.2  16K 16222      0     0     0     0     0
   14:49:00
   SUSELNX1  3.05  2.28 1.3 5584  5584  11037   907 10130  1021 10145
```

```
(Contention starts, pushes 10,130 pages to disk)
14:50:00
SUSELNX1  4.98  4.15 1.2 1879  1879  14937    21 14916  2916  5033
14:51:00
SUSELNX1  2.68  1.92 1.4 1189  1189  15658   197 15461  2974  2028
14:52:00
SUSELNX1  5.63  5.01 1.1 1196  1196  15754   366 15388  3082  2271
14:53:00
SUSELNX1  3.89  3.13 1.2 1160  1160  15819   410 15409  2431  1907
(Minimum storage requirement, working set starts to stabalize)
14:54:00
SUSELNX1  3.33  2.63 1.3 1461  1461  15498   195 15303   143    52
(kernel processes start: init,kblockd,pdflush)
14:55:00
SUSELNX1  3.33  2.67 1.2 1331  1331  15630   362 15268    37     0
(storage starts to drop
14:56:00
SUSELNX1  3.70  2.94 1.3 3910  3910  15405   144 15261  2361     0
(Start “top” – cost 10MB)
14:57:00
SUSELNX1  5.04  4.40 1.1 4135  4135  15056   136 14920   217     0
```

The conclusion from this is that our minimal configuration running just Linux and netsnmp without a DCSS was about 1160 pages. When top was started, from the z/VM perspective, the working set adds about 3000 pages. Linux reports the combined size of top, bash and sshd as about 5.6 MB or 1400 pages. The remainder of the working set increase is not completely explained but likely in large part due to kernel data areas being referenced by top.

When all processes were active and part of the working set, the working set was 4135 pages; this is what we hope to reduce using XIP2.

4.6.3 Test 2, mem=1024M

In the second test with mem=1024M to validate the numbers, the increase when top was started was consistent. The storage requirements in this data is consistently 8 MB greater. There is no difference in extra costs when top was started. The base cost is about 3000 pages instead of the 1160 pages prior, or about 7.6 MB. This is consistent with the increase in kernel size due to the page management structure tables.

This validates the conclusion that the cost in addressing additional storage impacts both real storage and virtual storage.

Example 4-5 ESALNXP report for Test2

```
Report: ESALNXP      LINUX HOST Process Statistics Report
```

```
------------------------------------------------------------------
node/     <-Process Ident-> <-----CPU Percents----------> <Stg (k)>
 Name     ID    PPID   GRP   Tot  sys user syst usrt nice  Size RSS
--------- ----- ----- ----- ---- ---- ---- ---- ---- ---- ---- ----
16:30:00
SUSELNX1      0     0     0  5.3  3.4  1.9    0    0    0 198K  78K
 snmpd     1823     1  1822  1.7  1.1  0.6    0    0    0 7060 4016
------------------------------------------------------------------
16:31:00
SUSELNX1      0     0     0  6.8  4.8  1.9    0    0    0 198K  78K
 snmpd     1823     1  1822  2.2  1.6  0.6    0    0    0 7060 4016
------------------------------------------------------------------
16:32:00
SUSELNX1      0     0     0  6.3  3.9  2.4    0    0    0 198K  78K
 snmpd     1823     1  1822  2.1  1.3  0.8    0    0    0 7060 4016
------------------------------------------------------------------
16:33:00
SUSELNX1      0     0     0  5.1  3.3  1.8    0    0    0 198K  78K
 snmpd     1823     1  1822  1.6  1.0  0.6    0    0    0 7060 4016
------------------------------------------------------------------
16:36:00
SUSELNX1      0     0     0 14.4  8.6  5.8    0    0    0 200K  79K
snmpd     1823     1  1822  1.7  1.1  0.6    0    0    0 7060 4016
 sshd      2082  1875  2082  0.7  0.5  0.2    0    0    0 8392 2580
 bash      2085  2082  2085  0.3  0.3  0.0    0    0    0 3204 1952
 top       2104  2085  2104  2.1  1.0  1.1    0    0    0 2132 1100
------------------------------------------------------------------
16:37:00
SUSELNX1      0     0     0 13.1  7.6  5.5    0    0    0 200K  79K
 snmpd     1823     1  1822  1.5  0.9  0.6    0    0    0 7060 4016
 sshd      2082  1875  2082  0.8  0.6  0.1    0    0    0 8392 2580
 top       2104  2085  2104  2.0  0.9  1.2    0    0    0 2132 1100
```

The following user report shows the resident storage that decreased to referenced pages only, and then increased when top was started.

Example 4-6 ESAUSR2 report for test2

```
Report: ESAUSR2      User Resource U    Linux Test
Monitor initialized:                   First record analyzed: 01/
------------------------------------------------------------------
         <---CPU time--> <--------Ma <------Paging (pages)------->
UserID   <(seconds)> T:V <Resident>  <---Allocated---> <---I/O--->
/Class   Total  Virt Rat Totl Activ  Total ExStg  Disk  Read Write
-------- ----- ----- --- ---- -----  ----- -----  ----- ----- -----
16:25:00
SUSELNX1  2.56  2.09 1.2  16K 15876      0     0     0     0     0
16:26:00
```

```
SUSELNX1  2.51  2.08 1.2  10K 10430   5446  5371    75     0    75
16:27:00
SUSELNX1  3.33  2.86 1.2 3725  3725  12481  9137  3344   330  3269
16:28:00
SUSELNX1  2.29  1.85 1.2 2984  2984  13345  9135  4210   123   866
(working set at lowest point with just netsnmp)
16:29:00
SUSELNX1  2.27  1.84 1.2 2997  2997  13339  9129  4210    65     0
16:30:00
SUSELNX1  2.26  1.83 1.2 3109  3109  13339  9129  4210    54     0
16:31:00
SUSELNX1  2.40  1.94 1.2 3169  3169  13339  9129  4210    60     0
16:32:00
SUSELNX1  2.28  1.85 1.2 3169  3169  13339  9129  4210     0     0
16:33:00
SUSELNX1  2.26  1.83 1.2 3169  3169  13339  9129  4210     0     0
(working set is very stable with just netsnmp)
16:34:00
SUSELNX1  2.29  1.85 1.2 3745  3745  13327  9117  4210   564     0
16:35:00
SUSELNX1  2.35  1.89 1.2 5354  5354  13391  9181  4210  1673     0=
(top is started, working set increaes about 3200 pages)
16:36:00
SUSELNX1  4.02  3.44 1.2 6011  6011  13840  9630  4210  1106     0
16:37:00
SUSELNX1  3.96  3.41 1.2 6022  6022  13829  9619  4210     0     0
16:38:00
SUSELNX1  3.93  3.39 1.2 6027  6027  13824  9614  4210     0     0
```

4.6.4 Implementing XIP2 in DCSS to reduce Linux Resident Storage

The next step was to look for the “best case” opportunity to minimize the overall real storage requirements by implementing XIP2 in a DCSS.

The methodology used for packing the DCSS was to evaluate the ESALNXC report for this server to determine which processes should be loaded into a DCSS. This report shows all the resident processes, not just the active ones, and their parent/child relationship. Child processes are indented from the parent. The RSS, Resident Storage, is the relevant number. This is the amount of virtual storage currently addressable by the process. Of the resident storage, some will be private, and some “shared”, which can then be in the DCSS. Only experimentation and measurements will show the savings. Of these processes, the idle processes do not impact the referenced pages that VM will keep resident. Therefore, putting inactive processes into a DCSS has little benefit unless used by multiple servers. Note that processes with a zero RSS are kernel processes and would not be included in a DCSS.

Example 4-7 ESALNXC report

```
Report: ESALNXC      LINUX Process Conf Report
-------------------------------------------------
Node/        <-Process Ident-> <Process> <Storage(k)
 Name        ID    PPID   GRP  Path      Size  RSS
------------ ----- ----- ----- -------- -----  ----
SUSELNX1
 init            1     0     0 init [3]   628    99
 *               4     1     0              0     0
  khelper        5     4     0 khelper      0     0
  kblockd/       6     4     0 kblockd/     0     0
  pdflush     1516     4     0              0     0
 kjournal      277     1     1 pdflush      0     0
 rc            557     1   557 /sbin/sy  2616   144
  S14hwsca    1921   557   557 top       2616   256
   hwbootsc   1929  1921   557 sshd: ro  2612   208
    hwscan    1953  1929   557 -bash     2880  1064
 blogd         568     1   568 /sbin/kl  9824    76
 syslogd      1798     1  1798 /sbin/kl  1640   728
 snmpd        1817     1  1816 /sbin/re  7060  4016
 resmgrd      1830     1  1830 /sbin/po  1496   516
 portmap      1831     1  1831 /usr/sbi  1516   580
 syslogd      1841     1  1841 /sbin/re  1640   196
 klogd        1844     1  1844 /sbin/po  1596   224
 sshd         1868     1  1868 /sbin/mi  5304  1972
  sshd        2073  1868  2073 -bash     8392  2580
   bash       2076  2073  2076 top       3204  1952
    top       2095  2076  2095 sshd: ro  2132  1104
   sshd       6524  1868  6524 -bash     8392  2576
    bash      6527  6524  6527 pdflush   3208  1996
```

From this report, four processes were chosen for inclusion into a DCSS, snmpd, top, bash and sshd. These four were used during the previous experiments, setting a baseline and should provide a measurable impact. The target storage was the approximately 4200 pages when both snmpd and top (and sshd) were running.

The four processes required a 13 MB DCSS to be created, which was located at 64 MB, just above the Linux virtual machine. The following sample when all four processes were running shows that each process was greatly reduced in resident virtual storage.

Example 4-8 ESALNXP report for test 2

```
Report: ESALNXP      LINUX HOST Process Statistics Report
-------------------------------------------------------------------
node/     <-Process Ident-> <-----CPU Percents----------->  <Stg (k)>
 Name     ID    PPID   GRP   Tot  sys user syst usrt nice   Size RSS
```

```
--------- ----- ----- ----- ---- ---- ---- ---- ---- ----  ---- ----
SUSELNX1      0     0     0 10.6  6.2  4.4    0    0    0  131K  15K
 kjournal   279     1     1  0.1  0.1    0    0    0    0     0    0
 snmpd     1865     1  1864  1.3  0.7  0.6    0    0    0  7248 1948
 sshd      2125  1923  2125  0.6  0.5  0.2    0    0    0  8392  740
 bash      2128  2125  2128  0.2  0.2  0.0    0    0    0  3204  592
 top       3171  2128  3171  3.3  1.7  1.5    0    0    0  2132  288
```

In measuring the results, from the Linux perspective, there was a savings in process resident virtual storage of about 4.7 MB.

Table 4-2 Results

Process	Without DCSS	With DCSS	Savings (MB)
snmpd	4016K	1948K	2.1
sshd	2580	740K	.5
bash	1952	592K	1.3
top	1104	288K	.8
Total:			4.7

The snmpd process went from 4016K storage to 1948K storage, a savings of 2.1 MB. Thus with a 13 MB dcss, the virtual storage savings was 4.7 MB.

4.6.5 Impact of DCSS on z/VM real storage

The first analysis was with just the SNMP daemon running. The pages resident for z/VM varied much more in this experiment than prior experiments. We believe that the use of XIP2 increased the use of the pdflush kernel process, increasing the storage requirements, with storage varying between 500 and 1400 pages. The overall storage requirement in the equivalent experiment without XIP2 varied from 1160 to 1460 pages. Thus the savings was as much as 600 pages, 2.4 MB The conclusion from this experiment is that the savings in the SNMP daemon virtual storage closely matched the savings in the working set.

Example 4-9 ESUSR2 report

```
Report: ESAUSR2      User Resource Linux Test
---------------------------------------------------------------------
         <---CPU time--> <-Pages---> <------Paging (pages)------->
UserID   <(seconds)> T:V <Resident>  <---Allocated---> <---I/O--->
/Class   Total  Virt Rat Totl Activ  Total ExStg  Disk  Read Write
SUSELNX1  4.21  3.63 1.2  16K 16186      0     0     0     0     0
16:29:00
SUSELNX1  5.04  4.46 1.1  16K 16186      0     0     0     0     0
```

```
16:30:00
SUSELNX1  2.90  2.31 1.3 1290  1290  15046 12465  2581   641  2759
16:31:00
SUSELNX1  5.59  5.04 1.1 1198  1198  15657  4912 10745  3548 10838
16:32:00
SUSELNX1  3.55  2.89 1.2  785   785  16417  1051 15366  2675  6475
16:33:00
SUSELNX1  5.90  5.35 1.1 1111  1111  16548  1206 15342  2547  1813
16:34:00
SUSELNX1  3.26  2.56 1.3  981   981  16667  1342 15325    35    15
16:35:00
SUSELNX1  4.64  3.96 1.2 1402  1402  16505  1232 15273   311     0
16:36:00
SUSELNX1  3.37  2.69 1.3  925   925  17015  1709 15306    89    89
16:37:00
SUSELNX1  4.68  3.99 1.2  738   738  17373  1993 15380   795   678
16:38:00
SUSELNX1  4.63  3.95 1.2  498   498  17639  2342 15297   155    48
```

Note the different processes running at different intervals. As these are more or less normal processes, their overhead is to be expected.

Example 4-10 ESALNXP report

```
Report: ESALNXP      LINUX HOST Process Statistics Report
------------------------------------------------------------------------
node/     <-Process Ident-> <-----CPU Percents----------> <Stg (k)>
 Name     ID    PPID   GRP   Tot  sys user syst usrt nice  Size RSS
--------- ----- ----- ----- ---- ---- ---- ---- ---- ---- ---- ----
SUSELNX1      0     0     0  5.4  4.4  0.9    0    0    0 129K  15K
 kjournal   279     1     1  0.1  0.1    0    0    0    0    0    0
 snmpd     1865     1  1864  4.0  3.2  0.8    0    0    0 7248 1948
 slpd      1887     1  1887  0.1  0.1    0    0    0    0 3588  436
------------------------------------------------------------------------
16:33:00
SUSELNX1      0     0     0  9.0  4.5  0.8  1.9  0.7    0 129K  15K
 init         1     0     0  0.2  0.2    0    0    0    0  628  116
 khelper      5     4     0  3.0  0.4    0  1.9  0.7    0    0    0
 pdflush     99     4     0  0.2  0.2    0    0    0    0    0    0
 kjournal   279     1     1  0.3  0.3    0    0    0    0    0    0
 snmpd     1865     1  1864  4.0  3.1  0.8    0    0    0 7248 1948
 slpd      1887     1  1887  0.3  0.3  0.0    0    0    0 3588  436
```

The relevant information from this test was just saving the SNMP daemon in a DCSS could be shown to reduce real storage requirements by as much as 2 MB.

The second part of the analysis was to show the value of the other three processes. The top process was started at 16:44. Note the jump in storage to

2550 pages, but after initialization this storage drops down and then varies between 1321 and 2559 pages. The equivalent prior measurement had a working set of between 3900-4200 pages. The saving represented from these four processes then is about 2500-2600 pages, or about 10 MB, from a 13 MB dcss. With the variability in how Linux kernel processes run, this number is likely extremely optimistic as the virtual storage savings are much smaller than the real storage savings. But it is very obvious that the storage savings from the DCSS was significant in terms of percent of working set.

Example 4-11 ESAUSR2 report

```
Report: ESAUSR2      User Resource U    Linux Test
Monitor initialized:                   First record analyzed: 01/
------------------------------------------------------------------
         <---CPU time--> <--------Ma <------Paging (pages)------->
UserID   <(seconds)> T:V <Resident>  <---Allocated---> <---I/O--->
/Class   Total  Virt Rat Totl Activ  Total ExStg  Disk  Read Write
-------- ----- ----- --- ---- -----  ----- -----  ----- ----- -----
16:37:00
SUSELNX1  4.68  3.99 1.2  738   738  17373  1993 15380   795   678
16:38:00
SUSELNX1  4.63  3.95 1.2  498   498  17639  2342 15297   155    48
16:39:00
SUSELNX1  4.42  3.76 1.2 1119  1119  17527  2283 15244     1     0
16:44:00
SUSELNX1  5.82  5.03 1.2 2550  2550  18346  3163 15183  2311     0
16:45:00
SUSELNX1  5.04  4.27 1.2 1559  1559  19321  4283 15038   129     0
16:46:00
SUSELNX1  6.33  5.56 1.1 1321  1321  19461  4492 14969    39     0
16:47:00
SUSELNX1  6.31  5.52 1.1 1321  1321  19482  4528 14954    69    33
16:48:00
SUSELNX1  4.79  4.09 1.2 1420  1420  19407  4462 14945    33     0
```

4.6.6 XIP2 DCSS guidelines from non-Oracle test

The following guidelines were derived from this test:

- Evaluate the processes to include in the DCSS from the ESALNXP and ESALNXC reports; picking the proper candidates will result in reducing storage requirements.
- Pick the highly used processes. The SNMP daemon is always a good candidate when ESALPS is used.
- Carefully pack the DCSS with these processes to minimize the size of DCSS.

- Locate the DCSS in as low a storage position as possible to reduce overhead.
- Measure it. *Note*, in several early experiments, no differences in storage were perceived. As the DCSS had been implemented “by the book”, the first conclusion was that XIP2 had no value. Resulting analysis showed that there were errors in the implementation scripts and the DCSS and XIP2 were in fact not being used. Only the measurements caught this problem.

4.7 Packing the XIP2 DCSS for Oracle

The first experiments were with small servers, and now the following is Oracle in larger servers.

4.7.1 Linux storage management

Linux storage management is designed to be efficient. When a program is loaded, only the pages referenced are made resident in virtual storage. If two programs load the same storage, then the pages used by both processes are shared and do not consume extra storage as more processes utilize the already loaded pages.

Thus for Oracle, in the following (abbreviated) process report, there are many Oracle processes, but with current instrumentation levels of Linux, it is unclear to how much storage is actually shared, and how much is resident in total to support Oracle. From this data that comes from the standard Linux process table, all we know is how much storage a given process can address currently. In looking at the Oracle processes, the largest in this case has 87M resident (87K * 1024). The other processes are sharing some amount of storage, of which potentially most of is part of the 87M.

Example 4-12 ESLNXP report

```
Report: ESALNXP      LINUX HOST Process Statistics Report
Monitor initialized: 01/12/06 at 15:27:11 on 2084
--------------------------------------------------------------------
node/     <-Process Ident-> <-----CPU Percents----------->  <Stg (k)>
 Name     ID    PPID   GRP   Tot  sys user syst usrt nice   Size RSS
--------- ----- ----- ----- ---- ---- ---- ---- ---- ----  ---- ----
PAZXXT21      0     0     0  0.4  0.3  0.0    0    0    0   12M   1M
 snmpd     2571     1  2570  0.3  0.3  0.0    0    0    0  9784 5644
 oracle    7848     1  7848    0    0    0    0    0    0  282K  15K
 oracle    7850     1  7850    0    0    0    0    0    0  281K  13K
 oracle    7852     1  7852    0    0    0    0    0    0  284K  33K
 oracle    7854     1  7854  0.0  0.0    0    0    0    0  296K  15K
 oracle    7856     1  7856    0    0    0    0    0    0  282K  17K
```

```
oracle    7858     1  7858     0     0     0     0     0     0  283K   52K
oracle    7860     1  7860     0     0     0     0     0     0  281K   20K
oracle    7862     1  7862     0     0     0     0     0     0  282K   31K
oracle    7864     1  7864     0     0     0     0     0     0  282K   13K
oracle    7866     1  7866     0     0     0     0     0     0  282K   12K
oracle    7872     1  7872     0     0     0     0     0     0  281K   14K
oracle    7876     1  7876     0     0     0     0     0     0  285K   58K
oracle    7878     1  7878     0     0     0     0     0     0  281K   24K
oracle    7880     1  7880     0     0     0     0     0     0  282K   87K
```

There are two methods of determining what files should be moved to the XIP2 DCSS. One method which was used for the Oracle measurements was to move a complete file directory containing all of the object code for Oracle. The DCSS was created at the 1.5 GB address for 512 MB. It appeared that storage requirements actually increased in using XIP2 in a DCSS. Now we know that the extra addressability cost at least 24 MB, and that the overall savings in shared storage was not much more than that.

The second and better method is to select the programs used most by the Linux servers and include those programs into a DCSS.

Note that any installation using the scripts provided on the IBM Web site may have encountered a bug, in which case users could have believed they were using the programs in shared mode from the DCSS, but were in fact using the programs off disk instead. This has been corrected on the IBM download page as a result of this research. Version 2 of the XIPINIT-fw shell script should be used.

4.7.2 Impact of DCSS on z/VM real storage

The pages in the DCSS that are referenced will stay resident. Pages in the DCSS that are in use by more than one Linux server result in savings. Pages that are rarely referenced may increase the storage requirements with little savings. The number of pages of the DCSS resident as well as the number of Linux servers with the DCSS attached (and in use) is provided on the ESADCSS display and report. In this case, the ORADCSS is a 512 MB DCSS (131,000 pages) with 26,000 pages resident.

Note that if this was a constrained environment, some amount of the DCSS would have been paged out.

Example 4-13 ESADCSS report

```
Report: ESADCSS      NSS/DCSS Analysis Linux
------------------------------------------------------------------------
                                        <-Users-> <--------Number of Pag
```

```
         Spool <Creation-> <First Use>      Non-       Non---resident-
Name        ID Date  Time  Date  Time  Shrd Shrd Saved Data <2GB >2GB HO
-------- ----- ----- ----- ----- ----- ---- ---- ----- ---- ---- ----
13:34:00
CMS        469 08/24 09:09 10/30 09:06   33    0  1298    0  687    0
CMSFILES   454 05/23 11:12 10/30 09:06    4    0   768    0  301    0
CMSPIPES   451 05/23 11:12 10/30 09:06   36    0   256    0   94    0
CMSVMLIB   458 08/22 15:43 10/30 09:06   36    0   256    0  113    0
ESAMON     348 11/10 13:21 10/30 09:06    4    0     0  16K 9772    0
GCS        440 05/20 13:33 10/30 09:06    3    0   120  917  939    0
HELPSEG    466 08/22 16:12 10/31 13:42    1    0   256    0  230    0
INSTSEG    455 05/23 11:11 10/30 09:06   36    0   768    0  298    0
MONDCSS    350 11/10 13:32 10/30 09:06    2    0     0  16K  106    0
NLSAMENG   464 08/22 15:44 10/30 09:06   25    0   256    0    2    0
ORADCSS    492 11/07 13:04 11/07 13:11    3    0  131K    0  26K    0
SCEE       446 05/23 11:12 10/30 09:06    6    0   256    0   27    0
SCEEX      439 05/23 11:12 10/30 09:06    6    0  1792    0  279    0
SVM        453 05/23 11:12 10/30 09:06    2    0   256    0   39    0
VTAM       149 04/17 11:06 10/30 09:06    3    0   256    0   70    0
                                         ---- ---- ----- ---- ---- ----
 Totals:                                 200    0  138K  34K  39K    0
```

4.8 Measuring XIP2 - file directories

The following experiments were done using 512 MB virtual machines, a 512 MB DCSS, and a 220 MB SGA. The DCSS was created using a file directory approach rather than selecting specific programs, and at the 512 MB address.

The kernel size change is very measurable, and very consistent. The overhead storage measurement is dependant on the point in time of the script where the measurement was taken. There is a cost of 9 MB for the added 512 MB addressability.

The two ESALNXP report extracts show just the largest Oracle process in terms of resident storage to show before and after the DCSS was implemented. The first report extract shows that the resident storage grows over time and peaks at 184 MB. In the next experiment, it peaked at 146 MB, a 40 MB improvement.

Example 4-14 ESALNXP report

```
Report: ESALNXP      LINUX HOST Process Statistics Report
---------------------------------------------------------------
node/     <-Process Ident-> <-----CPU Percents----->  <Stg (k)>
 Name     ID    PPID   GRP   Tot  sys user syst usrt  Size RSS
--------- ----- ----- ----- ---- ---- ---- ---- ----  ---- ----
 oracle    3358     1  3358  0.2  0.2  0.0    0     0 370K  46K
```

```
 oracle    3358     1  3358  0.4  0.3  0.1    0     0 370K  63K
 oracle    3358     1  3358  0.8  0.4  0.4    0     0 370K 101K
 oracle    3358     1  3358  0.8  0.4  0.4    0     0 370K 141K
 oracle    3358     1  3358  0.9  0.4  0.5    0     0 370K 176K
 oracle    3358     1  3358  0.9  0.4  0.5    0     0 370K 183K
 oracle    3358     1  3358  1.0  0.4  0.5    0     0 370K 184K
 oracle    3358     1  3358  0.9  0.4  0.5    0     0 370K 184K
 oracle    3358     1  3358  0.8  0.3  0.5    0     0 370K 184K
 oracle    3358     1  3358  0.8  0.4  0.4    0     0 370K 184K
Report: ESALNXP      LINUX HOST Process Statistics Report
Monitor initialized: 01/12/06 at 15:27:11 on 2084
------------------------------------------------------------
node/     <-Process Ident-> <-----CPU Percents----->  <Stg (k)>
 Name     ID    PPID   GRP   Tot  sys user syst usrt  Size RSS
--------- ----- ----- ----- ---- ---- ---- ---- ----  ---- ----
 oracle    3158     1  3158  0.8  0.5  0.3    0     0 370K  59K
 oracle    3158     1  3158  1.2  0.7  0.5    0     0 370K 101K
 oracle    3158     1  3158  0.4  0.2  0.2    0     0 370K 117K
 oracle    3158     1  3158  0.2  0.1  0.1    0     0 370K 146K
```

The results of this experiment show that savings were not as large as had hoped and actually non-existent for this data. Addressing the DCSS cost 9 MB per server, that we know is real storage, the cost of the 512 MB DCSS was measured to have 30 MB resident. And as we did not reduce the servers virtual storage, Linux immediately consumed any storage that was now available and used it for caching data. The lessons learned from this measurement are:

- Manage the contents of the DCSS. Creating large DCSS without contents that are highly used or unreferenced costs in Kernel storage without benefit.
- Minimizing the size of the DCSS and packing it with highly shared files will provide the most benefit.
- Place the DCSS in as low a storage address as possible.
- Reduce the virtual machine size. For example, if a 100 MB DCSS was used, then drop the virtual machine size from 512 MB to 412 MB, with the DCSS loaded at address 412 MB.

The conclusion then is that best case for Oracle, with a 100 MB DCSS, the cost is about 1 MB for addressability, plus the resident pages of the DCSS. Realistically, if we can load the DCSS with Oracle, SQL as well as the SNMP daemon and other processes utilized, the virtual storage requirements might drop 60 MB in which case reducing the virtual machine size by 60 MB would be reasonable. For a 512 MB server, this is over a 10% savings, and in a constrained environment where z/VM is removing unreferenced pages is likely much more.

Note that none of the Oracle measurements were in a storage constrained environment so that there is no valid comparisons of the VM resident storage, this should be in a follow on project.

4.9 Measuring XIP2 - direct files

This experiment has two servers. the T20 server has an active DCSS, the T21 server has no DCSS. All servers are 400 MB. The DCSS is 100 MB and starts at 400 MB. The DCSS was packed with just two components, the Oracle and sqlplus processes.

For this experiment there was no contention, so from a VM perspective there is no difference in resident storage requirements. In all cases, the T20 and T21 machines had all 400M (100K pages) resident.

Example 4-15 ESAUSR2 report

```
Report: ESAUSR2      User Resource Utilization Linux Test
------------------------------------------------------------------------
         <---CPU time--> <--------Main St <------Paging (pages)------->
UserID   <(seconds)> T:V <Resident> Lock  <---Allocated---> <---I/O--->
/Class   Total  Virt Rat Totl Activ  -ed  Total ExStg  Disk  Read Write
-------- ----- ----- --- ---- ----- ----  ----- ----- ----- ----- -----
14:42:00 145.3 143.9 1.0 5.0M 5013K 1190    20M 2605K   17M   929  2705
PAZXXT20 15.02 14.96 1.0 101K  101K 14.0     34    34     0     0     0
PAZXXT21 13.26 13.15 1.0 101K  101K 13.0      5     5     0     0     0
------------------------------------------------------------------------
14:43:00 279.6 278.1 1.0 5.0M 5014K 1189    20M 2605K   17M  1144     0
PAZXXT20 58.73 58.62 1.0 101K  101K 13.0     19    19     0     0     0
PAZXXT21 58.69 58.57 1.0 101K  101K 13.0      2     2     0     0     0
------------------------------------------------------------------------
14:44:00 276.0 274.6 1.0 5.0M 5014K 1189    20M 2604K   17M   656   600
PAZXXT20 57.91 57.82 1.0 101K  101K 13.0     21    21     0     0     0
PAZXXT21 56.95 56.83 1.0 101K  101K 13.0      2     2     0     0     0
------------------------------------------------------------------------
14:45:00 202.5 201.2 1.0 5.0M 5011K 1189    20M 2604K   17M  1089  2255
PAZXXT21 37.61 37.49 1.0 101K  101K 13.0      3     3     0     0     0
PAZXXT20 33.59 33.53 1.0 101K  101K 13.0     20    20     0     0     0
```

One sample of the process data report looks like the following. With the Oracle processes sharing storage, it is difficult to come up with extremely accurate conclusions. However, these three servers ran exactly the same workload. If you take the largest Oracle process (in the RSS/resident storage) column, the T21 server is 60M and the T20 server is 30M, which would make a 30 MB reduction in virtual storage requirements seem likely.

Example 4-16 ESALNXP Report

```
Report: ESALNXP      LINUX HOST Process Statistics Report
-------------------------------------------------------------------
node/     <-Process Ident-> <-----CPU Percents----------->  <Stg (k)>
 Name     ID    PPID   GRP   Tot  sys user syst usrt nice   Size RSS
--------- ----- ----- ----- ---- ---- ---- ---- ---- ----  ---- ----
14:42:00
PAZXXT21     0     0     0  5.1  0.6  0.1  1.2  3.2    0    6M 682K
 init        1     0     0  4.3    0    0  1.1  3.2    0   696  288
 kswapd0   232     1     1  0.2  0.2    0    0    0    0     0    0
 snmpd    2571     1  2570  0.4  0.3  0.1    0    0    0  9784 5644
 nscd     2859     1  2859  0.0  0.0    0    0    0    0   43K  912
 bash     3258  3257  3258  0.0  0.0    0  0.0    0    0  4788 2360
 oracle   3288     1  3288  0.1  0.1  0.0    0    0    0  296K  15K
 oracle   3290     1  3290  0.0  0.0    0    0    0    0  281K  15K
 oracle   3313     1  3313  0.0    0  0.0    0    0    0  283K  60K
PAZXXT20     0     0     0 12.4  0.4  0.2  2.6  9.3    0   15M   1M
 init        1     0     0 11.9  0.0    0  2.6  9.3    0   696  288
 pdflush   231     4     0  0.0  0.0    0    0    0    0     0    0
kswapd0    232     1     1  0.0  0.0    0    0    0    0     0    0
 snmpd    2575     1  2574  0.2  0.2  0.0    0    0    0  9788 5648
 bash     3265  3264  3265  0.0    0  0.0    0  0.0    0  4788 2360
 oracle   3295     1  3295  0.2  0.1  0.1    0    0    0  296K 7088
 oracle   3320     1  3320  0.0    0  0.0    0    0    0  282K  30K
```

To look at several points over time, the following report shows only the Oracle and sqlplus processes. For the T21 server, the Oracle (process ID 3313) process starts at 60M and goes to 67M. For the T20 server, the Oracle (process ID 3320) process go from 30M to 38M. As this is the same workload, a savings again of 30M of virtual storage appears obvious. The sqlplus is a much smaller process, it added another 1.6M in savings.

Example 4-17 ESALNXP report

```
Report: ESALNXP      LINUX HOST Process Statistics Report
-------------------------------------------------------------------
node/     <-Process Ident-> <-----CPU Percents----------->  <Stg (k)>
 Name     ID    PPID   GRP   Tot  sys user syst usrt nice   Size RSS
--------- ----- ----- ----- ---- ---- ---- ---- ---- ----  ---- ----
PAZXXT21     0     0     0  5.1  0.6  0.1  1.2  3.2    0    6M 682K
 oracle   3313     1  3313  0.0    0  0.0    0    0    0  283K  60K
PAZXXT20     0     0     0 12.4  0.4  0.2  2.6  9.3    0   15M   1M
oracle    3320     1  3320  0.0    0  0.0    0    0    0  282K  30K
14:43:00
PAZXXT21     0     0     0 99.2  1.7  0.9 27.2 69.4    0    6M 720K
 oracle   3286     1  3286  0.1  0.1  0.1    0    0    0  284K  38K
PAZXXT20     0     0     0 99.4  1.4  0.4 26.5 71.1    0   16M   1M
 oracle   3316     1  3316  0.0    0  0.0    0    0    0  285K  35K
```

```
14:44:00
PAZXXT21      0     0     0  100  1.2  0.6 18.9 79.7    0    7M 769K
 oracle    3313     1  3313  0.2  0.0  0.2    0    0    0  283K  65K
 sqlplus   6869  3328  3325  0.0    0  0.0    0    0    0   21K 6196
PAZXXT20      0     0     0 98.6  1.0  0.8 18.0 78.8    0   16M   1M
 oracle    3320     1  3320  0.3  0.0  0.3    0    0    0  282K  36K
 sqlplus   6976  3325  3324  0.0  0.0  0.0    0    0    0   22K 7840
14:45:00
PAZXXT21      0     0     0  1.9  1.0  0.9    0    0    0    7M 777K
 oracle    3313     1  3313  0.2  0.0  0.2    0    0    0  283K  67K
PAZXXT20      0     0     0  1.5  0.7  0.8    0    0    0   16M   1M
 oracle    3320     1  3320  0.2    0  0.2    0    0    0  282K  38K
```

The cost of the DCSS for this was 18K pages. As this was NOT a storage constrained environment, the cost is likely not this high.

Example 4-18 ESADCSS report

```
Report: ESADCSS      NSS/DCSS Analysis
-------------------------------------------------------------------
                                          <-Users-> <---------Number of
         Spool <Creation-> <First Use>      Non-       Non- <--reside
Name        ID Date  Time  Date  Time  Shrd Shrd Saved Data <2GB >2GB
-------- ----- ----- ----- ----- ----- ---- ---- ----- ---- ---- ----
14:42:00
CMS        469 08/24 09:09 01/12 14:38   33    0  1298    0  491    0
CMSFILES   454 05/23 11:12 01/12 14:38    4    0   768    0  144    0
CMSPIPES   451 05/23 11:12 01/12 14:38   36    0   256    0   79    0
CMSVMLIB   458 08/22 15:43 01/12 14:38   36    0   256    0   67    0
ESAMON     348 11/10 13:21 01/12 14:38    4    0     0  16K 2293    0
GCS        440 05/20 13:33 01/12 14:38    4    0   120  917  613    0
INSTSEG    455 05/23 11:11 01/12 14:38   36    0   768    0  122    0
MONDCSS    350 11/10 13:32 01/12 14:38    2    0     0  16K  145    0
NLSAMENG   464 08/22 15:44 01/12 14:38   25    0   256    0    2    0
ORADCSS    500 01/11 12:19 01/12 14:38    2    0 25600    0  18K    0
SCEE       446 05/23 11:12 01/12 14:38    6    0   256    0   14    0
SCEEX      439 05/23 11:12 01/12 14:38    6    0  1792    0   35    0
SVM        453 05/23 11:12 01/12 14:38    3    0   256    0   12    0
VTAM       149 04/17 11:06 01/12 14:38    4    0   256    0   66    0
                                         ---- ---- ----- ---- ---- ----
 Totals:                                  201    0 31882  34K  22K    0
```

The conclusion from this analysis duplicates prior conclusions that it is likely that at least a 10% in virtual storage can be attained, and though the real storage was not evaluated, it should be expected to reduce real storage by much more than that.

4.10 Conclusion

The savings are measurable and predictable. Further experiments should include analysis in a storage constrained environment. Also, including the operational libraries in the DCSS may also increase savings beyond what was measured.

A

ESALPS

For the measurements and analysis for this redbook, ESALPS (Linux Performance Suite) from Velocity Software was used for its unique capabilities and focuses on the z/VM and Linux environments.

ESALPS overview

ESALPS, the Linux Performance Suite, is a suite of products provided by Velocity Software for measurement and tuning of z/VM, Linux under z/VM, and applications in this environment. The four products that make up this suite are:

- ESAMAP, the VM Monitor Analysis Program, providing performance reports on all aspects of VM/ESA® and z/VM performance
- ESAMON, the VM Real Time Monitor, providing real time data collection and analysis of performance
- ESATCP, the network and Linux data collection program
- ESAWEB, a very fast VM based Web server

ESALPS addresses the infrastructure needs of the following:

- Performance: By providing full current and historical data for z/VM, Linux, and network performance characteristics, current performance and prior performance problems are easily analyzed and resolved. Reporting based on selecting specific periods of time allows in-depth performance analysis of performance problems.
- Capacity Planning: Long term data in the form of a performance database (PDB™) is needed as input to long term capacity planning and trend analysis. ESALPS meets the needs of all capacity planners by providing the PDB, as well as many interfaces to their favorite enterprise capacity management tools.
- Operations: ESALPS provides alerts on Linux, z/VM and network performance characteristics, either via 3270 interface, browser interface or via SNMP interface to the installation's management console.
- Accounting: By accurately accounting for CPU, storage and I/O, ESALPS can be used to account for virtual machines, or for Linux down to the process and application level.

ESALPS features

ESALPS provides the following features:

- Full function z/VM performance monitor allows for monitoring of all z/VM subsystems, including full reporting, real time analysis and management reporting.
- Network Monitor: Allows monitoring of your internet servers via SNMP.
- Linux Monitor: Allows monitoring of your Linux disks, storage and processes. In a virtual environment, ESALPS has the very unique capability for correcting

the process data that in a virtual environment are often incorrect, sometimes by an order of magnitude.

- Monitoring NT, SUN, HP: ESALPS will monitor any server that supports the standard MIBs. This includes network information, disk and storage information, and process data.
- Web serving: ESALPS includes a full function Web server to provide both browser access to the data but also to allow Web site support and CGI development on the VM platform.
- Alerts: Different types of alerts can be setup for network availability monitoring, VM subsystem errors and threshold exceptions, Linux errors and threshold exceptions.
- Full Performance Database: ESALPS provides both real time data and historical data for in-depth analysis. The performance data is collected daily with a one minute granularity based on the monitor interval. A longer term archive is collected usually with a granularity of 15 minutes. This PDB includes VM data, Linux data and network data.

Critical agent technology

One of the powerful features of ESALPS is the agent technology. Agents running inside Linux may be either active agents or passive agents. An active agent will wake up every few seconds on a regular schedule, collect data, possibly write data to a log, and then sleep. A passive agent will wait for an external data request and will not otherwise wake up and use resource.

The issue is that under z/VM, we expect to have maybe hundreds of Linux servers, of which many will be idle. With an active agent in each of them, the Linux servers are not idle and may consume excessive resources just by processing the active agent requests. Thus the only way to conserve resources is to eliminate the active agent from most or all of the servers.

With the passive technology enabled by using SNMP agents, ESALPS will know that a server is idle because of the z/VM data, and will not have to wake up an agent inside Linux, thus causing additional paging and CPU consumption, to determine and record the fact that the server is idle. The idle servers will not be measured until they become active.

The operational cost of agents may be quite high. On other non-virtualized servers, the cost is relatively insignificant as it only impacts the applications on that one server. In a virtualized environment where sharing resources is a primary objective, resources used by each agent reduces the resources available to all of the other servers. Minimizing the cost of running these agents

is critical to the success of monitoring many Linux servers under z/VM. The SNMP passive agent technology is believed to be the lowest cost agent technology available for Linux on z/VM.

NETSNMP extensions

The design of NETSNMP and SNMP host-specific MIBs was found lacking in a few areas. Velocity Software has extended the MIBs supported by NETSNMP. This agent can be downloaded from Velocity Software's Web site. This corrects measured values such as swap rate, and provides added process data thought necessary to meet the needs of this environment. This agent also is modified to use less resource than the NETSNMP that would come with the normal Linux distributions.

Standard interface

ESALPS uses standard interfaces for all data collection. The advantage to using the standard interfaces provided is that when there are a multitude of releases and distributions available, the standard interfaces will provide consistent data sources.

z/VM provides a "monitor interface" that has been available since 1988. Since then, this interface has provided a consistent view of performance of VM systems.

Network performance is collected using SNMP, the standard for network management.

NETSNMP, an open source software package provides host data for the Linux and other platforms.

VM Application data interface is used by applications to insert data into the monitor stream consistent with the standard monitor interface. ESATCP uses this interface to ensure consistent data collection that allows full integration of Linux and VM data.

B

Scripts used to set up DCSS

In this appendix we provide listings of the shell scripts we downloaded from developerWorks.

xipinit.sh

Example: B-1 xipinit.sh - Sets up to use a DCSS for a complete directory

```
#!/bin/sh
#
# xipinit.sh, Version 2
#
# (C) Copyright IBM Corp. 2002,2005
#
# /sbin/xipinit: use read only files from another file system
#
# default options

# internals
#set -v
#FAKE=echo

#mount point of xipimage
ROMOUNT=""
#name of xipimage
XIPIMAGE=""
#name of device node
RODEV=""

RODIRS="/lib,/usr/lib,/usr/X11R6/lib,/bin,/sbin,/usr/X11R6/bin,/usr/bin,"
# make sure it ends with ,
RODIRS="$RODIRS",

mount -t sysfs none /sys
if [ ! -e /sys/devices/dcssblk ]; then
  echo "xipinit: loading dcssblk module"
  /sbin/modprobe dcssblk
fi
echo $XIPIMAGE > /sys/devices/dcssblk/add

# mount ro file system to its mount point
echo "xipinit: mounting read-only segment"
$FAKE mount -t ext2 -o ro,xip "$RODEV" "$ROMOUNT"
# bind mount all ro dirs into rw filesystem
while [ -n "$RODIRS" ] ; do
  dir="${RODIRS%%,*}"
  RODIRS="${RODIRS#*,}"
  test -d "$dir" || continue
  echo "xipinit: binding directory" $dir
  $FAKE mount --bind "$ROMOUNT/$dir" "$dir"
done
umount /sys
```

```
# run real init
$FAKE exec /sbin/init "$@"
```

xipinit-fw.sh

Example: B-2 xipinit-fw - set up to use DCSS for selected files

```
#!/bin/bash
#
# xipinit-fw.sh, Version 3
#
# (C) Copyright IBM Corp. 2005


#mount point of xipimage
ROMOUNT=""
#name of xipimage
XIPIMAGE=""
#name of device node
RODEV=""

mount -t sysfs none /sys
if [ ! -e /sys/devices/dcssblk ]; then
  echo "xipinit-fw: loading dcssblk module"
  /sbin/modprobe dcssblk
fi
echo $XIPIMAGE > /sys/devices/dcssblk/add

echo "xipinit-fw: mounting read-only segment"
/bin/mount -t ext2 -o ro,xip $RODEV $ROMOUNT
echo "xipinit-fw: binding files"
for i in $(/usr/bin/find $ROMOUNT/*)
do
  if [[ -f $i ]]; then
       /bin/mount --bind $i ${i#$ROMOUNT}
  fi
done
umount /sys

exec /sbin/init $@
```

copylibs.sh

Example: B-3 copylibs.sh

```
#!/bin/bash
# copylibs.sh, Version 3
#
# (C) Copyright IBM Corp. 2005
#
#
# copies the binary and the libraries used to destination directory
# usage: ./copylibs.sh -f <executable|file> -d <destination directory>
#
#FAKE=echo


function getlibs {
   for i in `ldd $FILE | awk '{print $3}'`
   do
      echo $i
      if [[ -h $i ]]
      then
         echo $i is a link
         #LINKPATH=${i%/*}
         FILELINKEDTO=$(readlink -f $i)
              #FILELINKEDTOPATH=${FILELINKEDTO%/*}
         echo $FILELINKEDTO
         $FAKE cp -a --parent ${FILELINKEDTO} ${DESTINATION}
      elif [[ -e $i ]]
      then
         $FAKE cp -a --parent $i ${DESTINATION}
      fi
   done

   echo NOTE: Libraries which were loaded with libdl will not be copied.
   echo NOTE: be sure that you copy these extra.
}

###
###
# here the script starts
###
###

if [[ $# -ne 4 ]]
then
 echo "Usage: ./copylibs.sh -f <executable|file> -d <destination directory>"
 exit 1
fi
```

```
while getopts :f::d: OPT
do
   case $OPT in
       f )    if [[ ! $FILE && ${OPTARG:0:1} != '-' ]]
              then
                 echo "option f is set with $OPTARG"
                  FILE=$OPTARG
                 echo $FILE
         else
            echo error for option -f please check
            exit 1
         fi
         ;;
       d )    if [[ ! $DESTINATION && ${OPTARG:0:1} != '-' ]]
              then
            echo "option d is set with $OPTARG"
                 DESTINATION=$OPTARG
        echo $DESTINATION
         else
            echo error for option -d please check
        exit 1
         fi
        ;;
       :)     echo "no option set" ;exit 1 ;;
   esac
done

#here check if file (full qualified) exists
if [[ -e $FILE ]]
then
   if [[ $(file $FILE|grep ELF > /dev/null) -eq 0 ]]
   then
      getlibs
      # a dynamic shared object, so get all other libs needed
   fi
   #copy file itself first check if file is a symlink
   if [[ -h $FILE ]]
   then
      FILE=$(readlink -f $FILE)
   fi
   $FAKE cp -a --parent $FILE $DESTINATION
elif [[ ! -e $FILE ]]
then
   #check if file is an executable maybe found in PATH
   FILE=$(which $FILE)
   if [[ $? -ne 0 ]]
   then
```

```
      echo File not found, exiting...
      exit 1
   else
      $FAKE cp -a --parent $FILE $DESTINATION
      getlibs
   fi
fi
exit
```

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 71. Note that some of the documents referenced here may be available in softcopy only.

- *Experiences with Oracle Database on Linux on zSeries*, SG24-6552 (Oracle9*i*)
- *Linux for IBM zSeries and S/390: Distributions*, SG24-6264
- *Linux for S/390*, SG24-4987
- *Linux on IBM zSeries and S/390: ISP/ASP Solutions*, SG24-6299
- *Building Linux Systems under IBM VM*, REDP-0120
- *Linux on zSeries and S/390: Systems Management*, SG24-6820
- *z/VM and Linux on zSeries: From LPAR to Virtual Servers in Two Days*, SG24-6695
- *Linux Handbook A Guide to IBM Linux Solutions and Resources*, SG24-7000

Other publications

These publications are also relevant as further information sources:

- *How to use Execute-in-Place Technology with Linux on z/VM,* SC33-8287

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

T

V

X

Z

IBM Redbooks

Using Discontiguous Shared Segments and XIP2 Filesystems

(0.2”spine)
0.17”<->0.473”
90<->249 pages

Using Discontiguous Shared Segments and XIP2 Filesystems

With Oracle Database 10*g* on Linux for IBM System z

Redbooks

Putting the SNMP daemon into a DCSS to support virtual machines

Setting up the DCSS containing the Oracle RDBMS code

Memory savings with multiple Linux guests

Linux on IBM System z offers many advantages to customers who rely on IBM mainframe systems to run their businesses. Linux on IBM System z takes advantage of the qualities of service in the IBM System z hardware and in z/VM — making it a robust industrial strength Linux. This provides an excellent platform for consolidating Oracle databases that exist in your enterprise.

This IBM Redbook describes our experiences in boosting performance and reducing memory requirements by using discontiguous saved segments (DCSS) and the execute in place (XIP2) filesystem technology.

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

SG24-7285-00 ISBN 0738495514