Linux on zSeries

How to use Execute-in-Place Technology with Linux on z/VM March 23, 2005

Linux Kernel 2.6 (April 2004 stream)

SC33-8283-00

Linux on zSeries

How to use Execute-in-Place Technology with Linux on z/VM March 23, 2005

Linux Kernel 2.6 (April 2004 stream)

SC33-8283-00

Note
Before using this information and the product it supports, read the information in "Notices" on page 43.

First Edition (March 2005)

This edition applies to Linux kernel 2.6 (April 2004 stream) and to all subsequent releases and modifications until otherwise indicated in new editions.

This edition replaces LNUX-H6DS-00. SC33-8283-00 is the Linux on zSeries kernel 2.6 (April 2004 stream) equivalent to LNUX-HTDS which applies to the Linux on zSeries kernel 2.4, June 2003 stream.

Contents

Summary of Changes v

About this publication vii
Who should read this document vii
How this document is organized vii
Where to get more information viii

Chapter 1. Introduction to DCSS 1
Which data you can share 1
Where a DCSS can reside 1
DCSS above guest storage 2
DCSS in a storage gap 2
How programs run from a DCSS 4
Granularity of shared data 5
Sharing directories 5
Sharing individual files 5
Limitations and trade-offs you must be aware of . . 6
Using multiple DCSSs 7
Considerations for sharing files or directories that are not on the root file system 7
Requirements 7

Chapter 2. Setting up a DCSS 9
Task 1: Planning the DCSS content 9
Identifying individual files to be shared 9
Identifying directories to be shared 12
Task 2: Planning the size and location of the DCSS 13
Task 3: Setting up your guest for accessing the DCSS 14
Defining a storage gap 14
Extending the addressable address range beyond the guest storage 15
Task 4: Providing a script to over-mount shared data on startup 15
Over-mounting individual files 16
Over-mounting entire directories 16
Task 5: Creating a DCSS with the shared data . . . 17
Using the DCSS block device driver to copy data to the DCSS 17
Performing an IPL to load data to the DCSS . . 21
Task 6: Testing the DCSS 25
Task 7: Activating execute-in-place. 26

Chapter 3. Making your Linux guests use the DCSS. 29

Chapter 4. Updating the software on a DCSS 31
Updating software on an existing DCSS 31
Replacing a DCSS 32

Appendix. Scripts used for setting up a DCSS 33
copylibs.sh 34
mkxipimage.sh 36
update.sh 37
xipinit.sh 40
xipinit-fw.sh 41

Notices 43
Trademarks 44

Summary of Changes

This edition is related to the March 23th 2005 code drop. It constitutes a major rewrite of and replaces LNUX-H6DS-00.

- DCSSs can now be positioned in unused ranges within guest storage (storage gaps).
- DCSS content can now be at the granularity of individual files rather than entire directories.
- This edition includes a description of how you can set up a DCSS using the DCSS block device driver.

About this publication

This document describes how you can keep down the memory requirements and boost the performance of a virtual Linux™ server farm by using a z/VM® discontiguous saved segment (DCSS). DCSSs can be used by Linux instances that run as z/VM guests on an IBM® @server zSeries® mainframe.

You can find the latest version of this document on developerWorks at:
`ibm.com/developerworks/linux/linux390/april2004_documentation.shtml`

If you are routed to the top-level page: On the left navigation bar, under Kernel 2.6, click **April 2004 stream**. On the April 2004 stream page, click **Documentation**.

Who should read this document

This document is intended for Linux administrators and system programmers in charge of a virtual Linux server farm that runs under z/VM.

Note
This document is intended for expert users. Be sure you understand the implications of using a DCSS before you attempt to perform the tasks described in this document.

How this document is organized

Chapter 1, "Introduction to DCSS," on page 1 introduces the concepts of DCSSs and discusses the options you have when setting up a DCSS for your Linux guests.

Chapter 2, "Setting up a DCSS," on page 9 provides step-by-step procedures for planning and setting up a DCSS. For some of the tasks there are alternate procedures you can use, depending on the options you choose for:

- The location of your DCSS in storage
- The granularity of the data in your DCSS
- The method you prefer for writing data to your DCSS

The introduction to each of these alternate procedures clearly indicates to which option it applies.

Chapter 3, "Making your Linux guests use the DCSS," on page 29 summarizes what you need to do for each Linux guest to make it use the DCSS, after you have put the DCSS in place.

Chapter 4, “Updating the software on a DCSS,” on page 31 describes how you can update a DCSS that is already in use.

“Scripts used for setting up a DCSS,” on page 33 contains the scripts that are referred to in the main part of this document. You can also find a tarball with these scripts at:

`ibm.com/developerworks/linux/linux390/april2004_documentation.shtml`

If you are routed to the top-level page: On the left navigation bar, under Kernel 2.6, click **April 2004 stream**. On the April 2004 stream page, click **Documentation**.

Where to get more information

For more information on the CP commands and guest storage, refer to *CP Command and Utility Reference*, SC24-6081.

For information on DCSSs refer to *z/VM Saved Segments Planning and Administration*, SC24-6056

For more information on the DCSS block device driver refer to *Device Drivers, Features, and Commands*. You can obtain the latest version of this book on developerWorks at:

`ibm.com/developerworks/linux/linux390/april2004_documentation.shtml`

If you are routed to the top-level page: On the left navigation bar, under Kernel 2.6, click **April 2004 stream**. On the April 2004 stream page, click **Documentation**.

For information on the xip2 file system, visit:

`http://linuxvm.org/Patches/`

Chapter 1. Introduction to DCSS

This section explains what a z/VM DCSS is and how DCSSs can help to boost the performance of virtual Linux server farms.

All operating system instances that run concurrently on a zSeries mainframe vie for some of the available physical memory. When multiple operating system instances need the same data, that data might get loaded into memory several times.

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 allow all Linux instances that need the data to share this memory segment.

A major part of the memory required by a Linux server is used for binary application files and for shared library files. The potential for saving memory by sharing application binaries and libraries depends on the respective applications. In some test scenarios, the same resources could support four times the number of Linux guests for a given guest performance after a DCSS was put in place.

Which data you can share

The shared data must be identical across all sharing Linux instances.

- Application files can only be shared by Linux instances that use the same version of an application.
- Shared data must be read-only. To prevent Linux instances from interfering with one another they are not given write permission for the DCSS.
- Applications and libraries that are frequently used by numerous Linux instances are good candidates for sharing.

The following items are not suitable for sharing:

- Files or directories that are being written to.

  Note: If you choose to share entire directories, the subdirectories of a shared directory are also shared. Therefore, you cannot write to a subdirectory of a shared directory.

- Scripts.

  Sharing scripts is ineffective because they are typically interpreted and not executed directly. Scripts include, for example, files with extension .pl, .py, .sh.

- The /etc directory.

  You cannot share Linux instance specific data. The /etc directory holds, for example, instance specific network configuration data, like a Linux instance's IP address.

Where a DCSS can reside

All guests access the DCSS with the same real addresses. There are two alternative placements for a DCSS to avoid conflicts with the guest's addressing:

- The DCSS addresses are set to be above the highest storage of all individual guests
- The DCSS is placed in the address range between non-contiguous storage extents (referred to as a *storage gap* in this document)

The *guest storage* is the amount of memory that z/VM presents to a guest operating system as the guest machines's real memory.

DCSS above guest storage

Figure 1 shows two Linux guests with their storage. The "mem=" kernel parameter enables Linux to handle real addresses beyond what it would normally consider its real memory. To make the entire DCSS addressable, the value for "mem" must be at the upper limit of the DCSS, like Linux A in Figure 1, or above the upper limit of the DCSS, like Linux B in Figure 1.

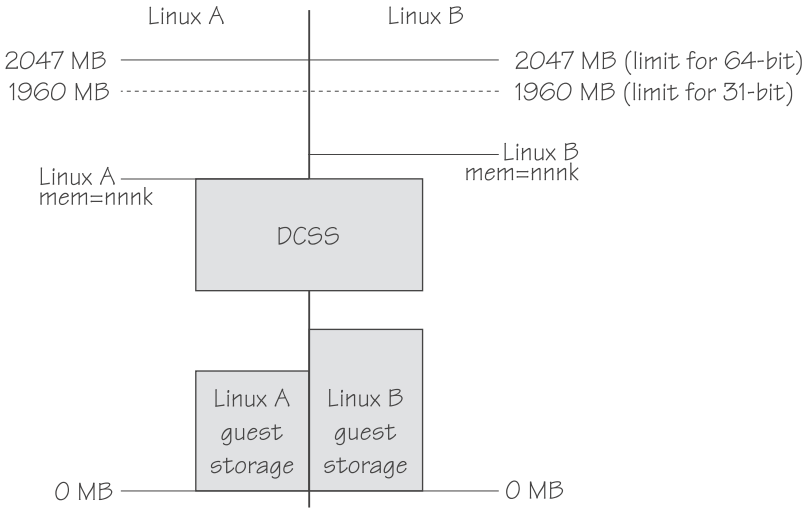

Figure 1. DCSS above Linux guest storage

z/VM supports DCSSs up to 2047 megabyte only. While DCSSs for 64-bit Linux guests can go right up to 2047 megabyte, 31-bit Linux guests require the address range from 1960 megabyte to 2048 megabyte (2 gigabyte) for virtual memory allocations. The maximum DCSS size is:

64-bit: 2047 MB – <*largest guest storage size*>

31-bit: 1960 MB – <*largest guest storage size*>

The guests' storage must be kept small enough to allow for the DCSS between the storage size of the guest with the largest storage and the 2047 megabyte or 1960 megabyte ceiling.

DCSS in a storage gap

You can define your guests' storage as one or more non-contiguous storage extents. Defining non-contiguous storage extents implicitly defines non-addressable storage ranges between the extents. For convenience, this document refers to such non-addressable storage ranges as *storage gaps*.

A DCSS can be placed into an address range that is within a storage gap for all Linux guests that are to share the DCSS. A storage gap is the preferred DCSS placement for 64-bit Linux guests.

Figure 2 illustrates a DCSS at an address range that is a storage gap for two 64-bit Linux guests.

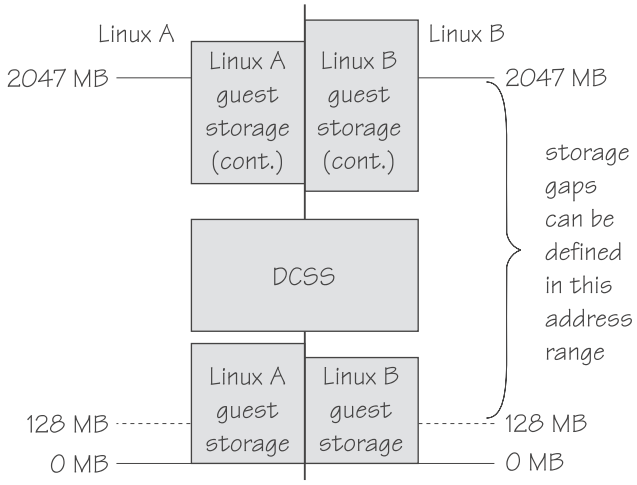

Figure 2. DCSS in a storage gap — 64-bit

To avoid conflicts with data structures that Linux guests require at low memory locations, you are advised to define your storage gaps above 128 megabyte. Always use full megabyte boundaries as starting and ending addresses for the storage gaps.

For 64-bit Linux guests you can assign guest storage above 2 gigabyte (2048 megabyte). Locating the DCSS in a storage gap allows a large DCSS to reside below 2 gigabyte without constraining the size of the 64-bit Linux guest storage.

As shown in Figure 3, for 31-bit Linux guests both guest storage and the storage gap surrounding the DCSSs must be below 1960 megabyte. A large DCSS, therefore, constrains your guest storage sizes.

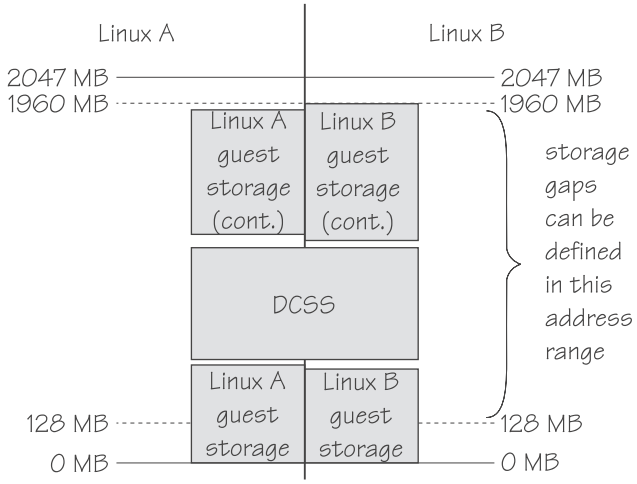

Figure 3. DCSS in a storage gap — 31-bit

While storage gaps for 64-bit Linux guests can go up to 2047 megabyte, 31-bit Linux guests require the address range from 1960 megabyte to 2 gigabyte for virtual memory allocations. The maximum DCSS size is:

64-bit: 2047 MB – 128 MB = 1919 MB

31-bit 1960 MB – *<largest guest storage size>*

How programs run from a DCSS

z/VM allows you to define DCSSs that can be loaded with data and saved on z/VM's spool space. Guest operating systems can load a DCSS into their own address space. If multiple guest operating system instances load the same DCSS, z/VM loads it into memory once only.

While the kernel and all other data that are not shared run from the storage of each guest, shared user space programs are mapped to the DCSS and run directly from the DCSS.

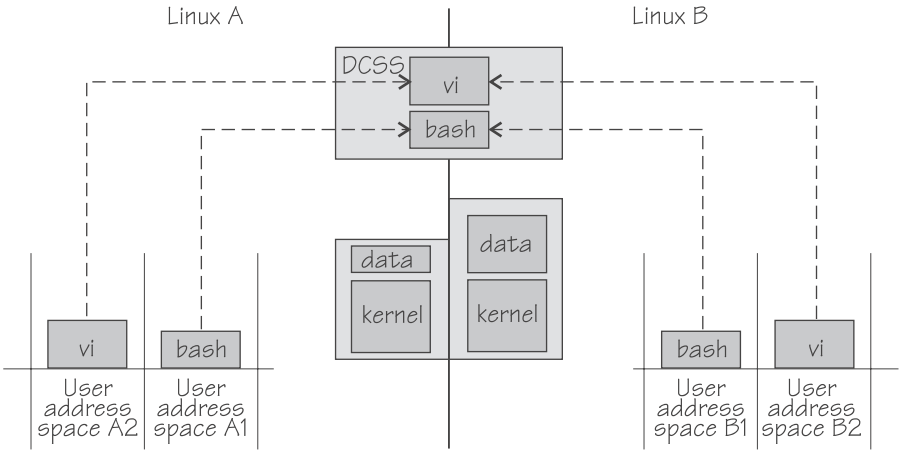

Figure 4. Shared programs run directly from the DCSS

Figure 4 shows a simple example where bash and vi are shared between two Linux guests, Linux A and Linux B.

Linux can access a DCSS through the execute-in-place file system (xip2). xip2 is based on the second extended file system (ext2). To replace directories in the Linux file system with shared content from a DCSS, the shared directories must be over-mounted.

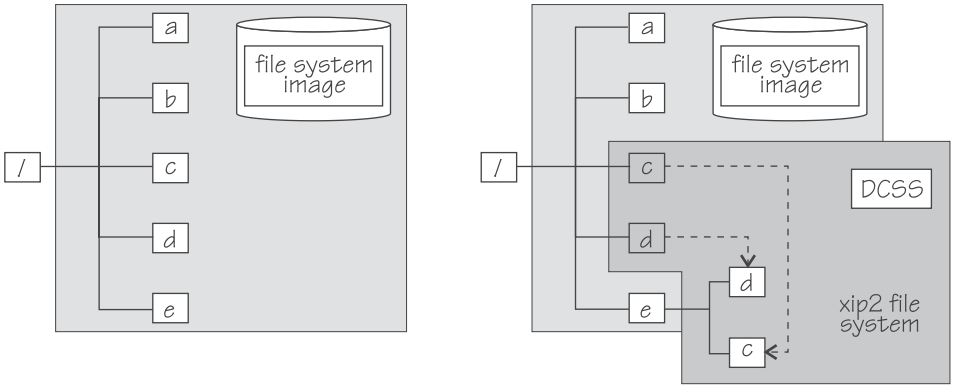

Figure 5. xip2 over-mounting directories

On the left side, Figure 5 on page 4 shows a Linux file system with the corresponding DASD volume where it resides as a file system image file. On the right, two of the directories, c and d, have been over-mounted with a xip2 file system. The xip2 file system is mounted on a spare directory, e. Any calls to c are being redirected to e/c and calls to d are redirected to e/d.

Linux loads shared libraries and application files through the mmap() operation. mmap() maps the contents of a file into the application's address space. The xip2 file system performs this operation by mapping the contents of the DCSS into the application's address space while other file systems use a page cache. This feature is called execute-in-place because the file contents are not physically copied to a different memory location.

Execute-in-place allows application files and libraries to be accessed without I/O operations. Avoiding I/O operations increases performance. Running applications directly in a shared memory segment saves memory.

Granularity of shared data

A DCSS can contain shared data at the granularity of:
- Entire directories
- Individual files

With the scripts and methods described in this document you cannot create DCSSs that combine both granularities. This section helps you to decide which granularity is more suitable for your needs.

Sharing directories

You can share data at the granularity of entire directories as illustrated in Figure 5 on page 4.

Advantages: Sharing entire directories allows you to share a large number of files with only a few bind-mount operations, and therefore with only a little overhead in memory consumption.

Disadvantages: Every shared directory and its subdirectories are read-only. You cannot share read-only files in directories that also contain files that are written to.

Sharing files at the granularity of entire directories is advantageous if:
- You want to share a large number of small files.
- You do not want to share files in directories that are written to.

Sharing individual files

The left side of Figure 6 on page 6 shows a Linux file system with the corresponding DASD volume where it resides as a file system image. Directories /c and /d are shown with a small number of the files they contain. Some of these files are to be shared.

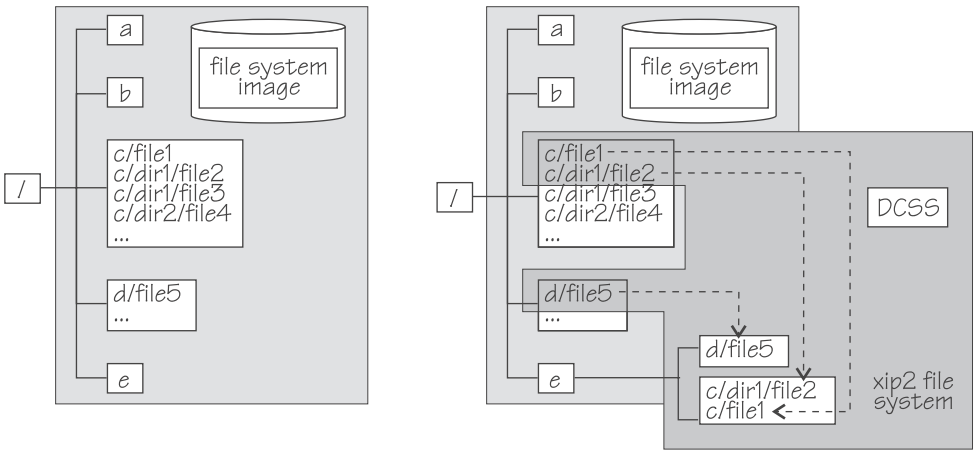

Figure 6. xip2 over-mounting individual files

On the right side, a DCSS contains individual files, `/c/file1`, `/c/dir1/file2`, and `/d/file5`. Mounting the xip2 file system from this DCSS over-mounts only these individual files. With the DCSS mounted on a mount point e, calls to these files are redirected to `/e/c/file1`, `/e/c/dir1/file2`, and `e/d/file5`, respectively. Calls to other files in directories c and d are not affected.

Advantages: Sharing individual files allows you to be very specific about what you want to share and what you do not want to share. Directories with shared read-only files are not limited to reading, new files can be created in them at any time, and files that have not been selected to be shared can be written to.

Disadvantages: Sharing files at the granularity of individual files requires a bind-mount operation for each shared file. Every bind-mount operation consumes up to 750 byte of storage for each guest that shares the DCSS. For a large number of shared files, there can be considerable storage consumption on account of bind-mount operations.

Sharing files at the granularity of individual files is advantageous if:

- You want to share a small number of files.
- You want to share files that reside in directories that are written to.
- You want to share files that are spread over a large number of directories, while numerous files in these directories do not need to be shared.

Limitations and trade-offs you must be aware of

- DCSSs occupy spool space. When you update a DCSS, multiple versions of the DCSS might coexist. Ensure that your available spool space is at least a multiple of the DCSS size.
- The address range where the DCSS is located is the same for all guest operating system instances and needs to be set when the DCSS is created.
- The address range must not overlap with the guest storage (perceived as physical memory by Linux) of any operating system instance that accesses it.
- Linux needs memory management data structures to access data on the DCSS. These data structures occupy 13 MB (31-bit Linux kernel) or 26 MB (64-bit Linux kernel) per GB DCSS. These data structures are subject to z/VM paging and are accessed infrequently, but be aware that DCSS resource consumption

increases with size. Sharing files or directories that contain rarely used data can degrade the performance and scalability of a server farm.

- Sharing individual files rather than entire directories can lead to an increase in storage consumption of up to 750 byte for each file. These data structures are subject to z/VM paging and are accessed infrequently, but be aware that resource consumption increases with the number of shared files.
- Some of the tasks you need to perform to set up or update a DCSS include commands that might take considerable time to complete. Be sure that the z/VM watchdog is not active on any Linux guest that you are using to perform these tasks. The watchdog might time out and restart your Linux guest.

Using multiple DCSSs

You can set up multiple DCSSs to be used by the same Linux guest. A Linux guest can concurrently use multiple DCSSs above the storage limit or multiple DCSSs in storage gaps, but not a combination of the two. To use multiple DCSSs concurrently, you need to ensure that the address ranges of any two used DCSSs do not overlap.

This document describes using a single DCSS only.

Considerations for sharing files or directories that are not on the root file system

You might want to share a file or directory that is not on the root file system (the file system mounted on /) but on a separate partition. Sharing files or directories that are on file systems other than the root file system is beyond the scope of this document.

If you want to share such files or directories, you need to be able to adapt the steps in this document. In particular, you need to be able to modify the startup scripts of your Linux distribution and adapt your xipinit or xipinit-fw script (see "Task 4: Providing a script to over-mount shared data on startup" on page 15) to ensure that:

- Any file systems with directories or files to be over-mounted are mounted at the beginning of your script, before the bind mount operations.
- The file systems with directories or files to be over-mounted are not mounted again after the DCSS has been mounted.

Requirements

To be able to use a DCSS:

- All Linux instances that share a DCSS must be guest operating systems of the same z/VM.
- Your Linux distribution needs to provide the execute-in-place file system xip2 kernel module.

To be able to set up or update a DCSS with the DCSS block device driver, your Linux kernel also needs:

- The DCSS block device driver, either built-in or as a separate module.

Chapter 2. Setting up a DCSS

This section describes what you need to do to put a DCSS in place.

Perform the following tasks to provide a DCSS for your virtual Linux server farm:

1. Plan the DCSS content.
2. Plan the size and location of the DCSS.
3. Set up the guest for accessing the DCSS.
4. Provide a script to over-mount shared data on startup.
5. Create the DCSS with the shared data.
6. Test the DCSS.
7. Activate execute-in-place.

Task 1: Planning the DCSS content

This task helps you to decide what to include in a DCSS.

You might want to use the DCSS to share entire read-only file system directories or to share the read-only files of applications and other individual read-only files.

- If you want to share applications and individual files, proceed with "Identifying individual files to be shared."
- If you want to share entire directories, proceed with "Identifying directories to be shared" on page 12.

You need to plan your DCSS such that it fits into the address space of all guest operating system instances that are going to use it (see "Task 2: Planning the size and location of the DCSS" on page 13).

Identifying individual files to be shared

This section helps you to decide what to include in a DCSS if you want to share the read-only files of applications and other individual files. If you want to share entire directories, skip this section and perform the steps in "Identifying directories to be shared" on page 12 instead.

1. List the applications with the highest memory consumption on your Linux guests. These applications are your candidates for sharing. Issue:

   ```
   # ps -e -ouser,pid,size,args | sort +2n -3
   ```

   Processes with the highest memory consumption are listed last in the command output.

   You get the maximum benefit from sharing applications that are used frequently by a large number of Linux guests.
2. Create an empty directory to serve as a repository for the files to be shared.
3. For each application you consider sharing, find out the space needed to accommodate the read-only application files.
   a. Get a copy of the copylibs.sh script (see "copylibs.sh" on page 34). See "Scripts used for setting up a DCSS," on page 33 for information on where you can obtain a copy of this script.

b. Run copylibs.sh to identify the read-only files of a particular application and to copy them to the directory you created in step 2 on page 9. Issue a command of this form:

```
# copylibs.sh -f <application> -d <destination_directory>
```

The script assumes that the application is in the path. Add the application to the path if necessary.

Example: This command copies the read-only files of the httpd Web server to a directory `/dcss`

```
# copylibs.sh -f httpd -d /dcss
```

Result: copylibs.sh determines the directory where the application resides and copies the application's read-only files to the destination directory. The script replicates the application's subdirectory structure. Symbolic links are resolved and files are copied to a subdirectory that corresponds to their actual location. If the destination directory does not exist, copylibs.sh creates it.

Example: Figure 7 shows a simplified example where an application consists of five files:

- `/usr/bin/abc/symlink` is a symbolic link that points to a read-only file `/lib/read3.so`
- `/usr/bin/abc/write1` is a file that can be written to
- `/usr/bin/abc/read1` is a read-only file
- `/usr/bin/abc/xyz/read2` is a read-only file
- `/usr/bin/abc/xyz/write2` is a file that can be written to

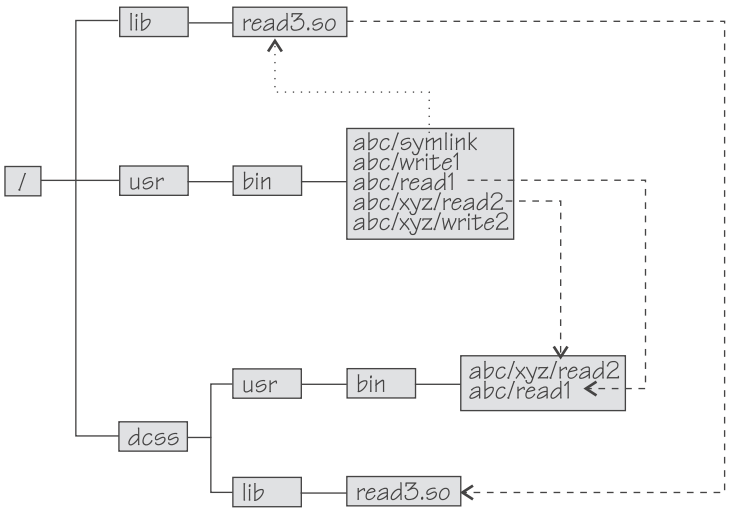

Figure 7. Example: files copied by copylibs.sh

copylibs.sh resolves the symbolic link and copies the read-only files to the destination directory:

- `/dcss/lib/read3.so`
- `/dcss/usr/bin/abc/read1`
- `/dcss/usr/bin/abc/xyz/read2`

c. Determine the space occupied by the copied files:

```
# du -sk <destination_directory>
```

Example:

```
# du -sk /dcss
61761
```

The output shows the space in kilobyte.

Note: The script might not copy all shared objects used by the application. For example, the Apache HTTP Server uses a shared object, `/usr/lib64/apache2` that copylibs.sh does not copy. You can copy such objects as individual files to include them in the DCSS (see step 4).

d. Repeat steps 3b on page 10 and 3c for each application. Find the size of each application from the increment in the output of the `du -sk` command.

4. Copy any individual read-only files or libraries that you know are used frequently. Issue a command like this for each individual file you want to include in the DCSS:

```
# copylibs.sh -f <file> -d <destination_directory>
```

Example: To copy `/usr/lib64/apache2` to `/dcss/usr/lib64/apache2` issue:

```
# copylibs.sh -f /usr/lib64/apache2 -d /dcss
```

5. Find the total size of your DCSS.

```
# du -sk <destination_directory>
```

Example:

```
# du -sk /dcss
882402
```

6. Allow some space for file system metadata. As a conservative estimate, use 4 KB per shared file.

a. Count the files in the DCSS.

Example:

```
# find /dcss | wc -l
21043
```

b. Calculate the additional space requirement.

Example: 21043 × 4 KB = 84172 KB

Tip: Add extra space to allow for future software updates, like security fixes.

Example: The example of the previous steps yields 882402 KB + 84172 KB = 966574 KB as an estimate for the space requirement. Adding extra space for future updates results in a space requirement of: 1 GB = 1024 MB = 1048576 KB.

Hint: DCSSs at the granularity of individual files are often much smaller than 1 GB.

Note: Be aware that each file in your DCSS will be mounted with the **mount --bind** command, and that each mounted file consumes up to 750 bytes of guest storage.

Tip: You can use the destination directory with the files copied in this task as a source for creating the DCSS. However, you cannot remove applications that you have already added. If you decide not to include an application that you have already added, repeat all of step 3 on page 9, omitting the unwanted application.

Proceed with "Task 2: Planning the size and location of the DCSS" on page 13.

Identifying directories to be shared

This section helps you to decide what to include in a DCSS if you want to share entire directories. If you want to share applications and individual files, omit this section and perform the steps in "Identifying individual files to be shared" on page 9 instead.

1. The following list helps you to find directories that are candidates for sharing:
   - Issue the following command to find all regular files with execute permission:

     ```
     # find / -perm -100 -type f
     ```

     Directories that contain any of these files are candidates for sharing.
   - Check the PATH environment variable of the superuser and a normal user. The standard binary directories defined therein are candidates for sharing. Check the directories' contents and include directories that are referred by symbolic links.
   - The major library directories as defined in `/etc/ld.so.conf` are candidates for sharing. For 64-bit Linux, distributions also include the lib64 directories. Check the directories' contents and include directories that are referred by symbolic links.

   The following list helps you to eliminate candidates that are not suitable:
   - Be sure not to share directories that are written to. Within the context of this description, sharing includes subdirectories. You cannot write to a subdirectory of a shared directory.
   - Sharing scripts is ineffective because they are typically interpreted instead of being executed directly. Scripts include, for example, files with extensions .pl, .py, .sh.
   - Do not include the /etc directory.

   You get the most benefit from sharing directories with programs that are frequently used by a large number of the sharing operating system instances.
2. Calculate the space requirements for sharing the directories you have selected.
   a. Issue the following command to find the space occupied by each directory:

      ```
      # du -sk <directories>
      ```

   b. Build the sum of the individual spaces to find the total space occupied by the directories.
   c. Allow some space for file system metadata. As a conservative estimate, use 4 KB per shared file.

      Tip: Add extra space to allow for future software updates, like security fixes.

Example: Assume that the directories to be shared are: `/lib`, `/usr/lib`, `/usr/X11R6/lib`, `/bin`, `/sbin`, `/usr/X11R6/bin`, and `/usr/bin` and that you obtain this space information:

```
# du -sk /lib /usr/lib /usr/X11R6/lib /bin /sbin /usr/X11R6/bin /usr/bin
46596 /lib
562972 /usr/lib
97372 /usr/X11R6/lib
5752 /bin 7768 /sbin
16956 /usr/X11R6/bin
152424 /usr/bin
```

The space used by the directories adds up to 882072 KB. Adding about 10% for metadata and contingencies results in a space requirement of 1 GB = 1024 MB = 1048576 KB.

Task 2: Planning the size and location of the DCSS

This task describes how to determine suitable page frame numbers for the start and end of the DCSS.

1. Determine the maximum size of the DCSS. The maximum size depends on:
   - Whether you are running 31-bit or 64-bit Linux kernels
   - Whether you want to place your DCSS above the storage of your Linux guests or in a common storage gap
   - The storage size of the largest guest that is to share the DCSS

   Use Table 1 to determine the maximum DCSS size according to your environment. For details on how these formulas are obtained see "DCSS above guest storage" on page 2 and "DCSS in a storage gap" on page 2.

   Table 1. Maximum DCSS size

Kernel	DCSS location	Maximum DCSS size
64-bit	DCSS in a storage gap	1919 MB
	DCSS above guest storage	2047 MB – *<largest guest storage size>*
31-bit	either location	1960 MB – *<largest guest storage size>*

   Example: Consider a number of Web servers with 64-bit Linux kernels where some have been assigned 256 MB guest storage and others have been assigned 512 MB guest storage. This example allows the following DCSS size:
   - For a DCSS in a storage gap: 1919 MB (independent of the guest storage sizes)
   - For a DCSS above guest storage: 2047 MB – 512 MB = 1535 MB

   The same size Web servers with 31-bit Linux kernels would allow:
   - 1960 MB – 512 MB = 1448 MB
2. Ensure that the required size (see "Task 1: Planning the DCSS content" on page 9) does not exceed the maximum DCSS size for your environment. If necessary, remove files or directories from the list of items you want to share. Remove items where you expect the least benefit, that is, where the data is used least frequently.
3. Decide on a start and end address for the DCSS. The start and end addresses must be on a page boundary (4 KB, on a mainframe) and in hexadecimal notation.
   - For a DCSS in a storage gap choose an address at or above 128 MB.

- For a DCSS above the guest virtual the start address must be at or above the virtual storage size of the largest guest.

Examples: The following start and end address provide for a DCSS of 1024 MB.

Start address: 512 MB => 0x20000000
End address: 512 MB + 1024 MB - 1 B => 0x5FFFFFFF

Tip: Round to easily remembered hexadecimal numbers because you will need to use them later in several commands.

If the DCSS is to be located above the guests storage, this start address limits the guest storage to a maximum of 512 MB. If the DCSS is to be located in a storage gap, guest storage can be allocated below 512 MB and starting from 1024 MB above the DCSS.

4. Calculate the page frame number for the start and end address. You do this by first rounding the address down to the nearest multiple of 4096 and then dividing this number by 4096.

   Hint: In hexadecimal notation this is accomplished by dropping the last three digits.

   Example: Start and end address 0x20000000 and 0x5FFFFFFF result in start and end frame numbers 0x20000 and 0x5FFFF.

Task 3: Setting up your guest for accessing the DCSS

This task describes the steps to make the DCSS address range available to the Linux kernel.

Proceed according to your intended DCSS location:

- If you want to place your DCSS in a storage gap, follow the steps in "Defining a storage gap."
- If you want to place your DCSS above the guest storage, follow the steps in "Extending the addressable address range beyond the guest storage" on page 15.

Defining a storage gap

This section describes how to set up your guest if you want to locate your DCSS in a storage gap. If you want to locate your DCSS above the virtual storage, skip this section and perform the steps in "Extending the addressable address range beyond the guest storage" on page 15 instead.

To define a storage gap for your guest, establish a CMS session with the VM guest that you want to set up for accessing the DCSS. Then add a line of this form to your PROFILE.EXEC:

```
DEF STOR CONFIG 0.<gap_start>M <gap_end>M.<guest_storage>M
```

where:

<gap_start>
defines the lower boundary of the storage gap. The lower boundary must be lower than the start address of the DCSS but not below 128 megabytes.

<gap_end>
defines the upper boundary of the storage gap. The upper boundary must be higher than the end address of the DCSS. For 64-bit Linux guests the upper boundary can be up to 2047 MB for 31-bit Linux guests it can be up to 1960 MB.

<guest_storage>
specifies the size of the storage extent above the storage gap.

For more information on the DEF STOR CONFIG command, refer to *CP Command and Utility Reference*, SC24-6081.

Example: This example defines a storage gap between 500 and 1600 megabytes and sets your guest storage to 750 megabytes (500 megabytes below the storage gap and another 250 above).

```
DEF STOR CONFIG 0.500M 1600M.250M
```

Extending the addressable address range beyond the guest storage

This section describes how to set up your guest if you want to locate your DCSS above the guest storage. If you want to locate your DCSS in a storage gap, omit this section and follow the instructions in "Defining a storage gap" on page 14 instead.

Before your Linux kernel can use the DCSS above the guest storage, it must be aware of the extended address space that covers the DCSS.

Perform the following steps to set the mem kernel parameter accordingly:

1. Start up your Linux system.
2. Add a `mem=<value>` parameter to your kernel parameter line (parmfile). The value must cover the entire DCSS, that is, it must be equal to or above the DCSS end address.

   The value can be in either of these forms: in byte, in the form `<x>k`, in the form `<y>M`, or in the form `<z>G`, where *<x>*, *<y>*, and *<z>* represent the value in kilobyte, megabyte, and gigabyte, respectively.

   Example: To accommodate a DCSS with an end address 0x5FFFFFFF add `mem=1536M`
3. Run **zipl** with the new parameter file.
4. Reboot Linux.
5. Issue the following command to verify that Linux is using the new parameter:

   ```
   # cat /proc/cmdline
   ```

Task 4: Providing a script to over-mount shared data on startup

This task provides information on sample scripts that you can use to over-mount files or directories with the content of a DCSS before Linux accesses them.

In this context, over-mounting a directory or file means replacing it with the contents of the DCSS at system startup. Shared items must be over-mounted before being accessed. If an individual file or a file in a shared directory is accessed before being over-mounted, the accessed file might remain in memory, and so waste memory, after the corresponding file or directory is over-mounted.

Proceed according to how you have planned the DCSS in "Task 1: Planning the DCSS content" on page 9:

- If your DCSS is to contain individual files, follow the steps in "Over-mounting individual files" on page 16.

- If your DCSS is to contain entire directories, follow the steps in "Over-mounting entire directories."

Over-mounting individual files

This section applies if you want to over-mount individual files. If you want to over-mount entire directories, skip this section and perform the steps in "Over-mounting entire directories" instead.

You can use the xipinit-fw.sh script (see "xipinit-fw.sh" on page 41) to over-mount individual files. See "Scripts used for setting up a DCSS," on page 33 for information on where you can obtain a copy of this script.

This script will have to run before `/sbin/init`. On system startup, xipinit-fw.sh runs instead of `/sbin/init`. First xipinit-fw.sh over-mounts the shared files and then it runs `/sbin/init`.

Perform these steps to tailor the script to your environment:

1. Find the line `MPXIPIMAGE=""` and specify the mount-point for the DCSS.

   Example: `MPXIPIMAGE="/mnt"`
2. Find the line `XIPIMAGE=""` and specify the name of your DCSS.

   Example: `XIPIMAGE="LNXSHARE"`
3. Place the script in the `/sbin` directory, name it xipinit-fw.sh, and ensure that the system administrator root is permitted to run it.

The tailored xipinit-fw.sh will be needed in "Task 7: Activating execute-in-place" on page 26.

Proceed with "Task 5: Creating a DCSS with the shared data" on page 17.

Over-mounting entire directories

This section applies if you want to over-mount entire directories. If you want to over-mount individual files, omit this section and perform the steps in "Over-mounting individual files" instead.

Directories that are on the root file system need to be over-mounted before running the first program on system startup: `/sbin/init`.

Directories on other file systems (for example, `/usr/bin` if `/usr` is on a separate DASD) need to be over-mounted right after the startup scripts for your kernel have mounted the other file systems. If you want to share directories that are not on the root file system, you need to modify the startup scripts. Because startup scripts are distribution-specific, this document does not describe how to do this.

You can use the xipinit script (see "xipinit.sh" on page 40) to over-mount directories on the root file system (the file system mounted on `/`). See "Scripts used for setting up a DCSS," on page 33 for information on where you can obtain a copy of this script.

On system startup, xipinit runs instead of `/sbin/init`. First xipinit over-mounts the shared directories and then it runs `/sbin/init`.

Perform these steps to tailor the script to your environment:

1. Specify your DCSS. In line 12, `ROOPTIONS="ro,memarea=XIP2502S"`, replace `XIP2502S` with the name of your DCSS.
2. Specify the directories you want to be shared. In line 13,
   `RODIRS=/lib,/usr/lib,/usr/X11R6/lib,/bin,/sbin,/usr/X11R6/bin,/usr/bin,`

   replace the value for `RODIRS` with a comma separated list of the directories you want to be shared.

   Note: Be sure to end the line with a comma character.
3. Ensure that line 18 specifies the mount point for your DCSS file system. As a default, `ROMOUNT=/mnt` specifies /mnt as the mount point. If you want to mount your DCSS file system at a different mount point, replace /mnt with your preferred mount point.
4. Place the script in the `/sbin` directory, name it xipinit, and ensure that the system administrator root is permitted to run it.

The script you have tailored in this task will be needed:

- In "Creating a file system image with the content of the DCSS" on page 21 if you create the file system image with the provided script.
- In "Task 7: Activating execute-in-place" on page 26.

Task 5: Creating a DCSS with the shared data

This task describes how you can create a DCSS and copy the shared data to it. Proceed according to the method you want to use for copying your data to the DCSS:

- If you want to use the DCSS block device driver, follow the steps in "Using the DCSS block device driver to copy data to the DCSS."
- If you want to use a file system image on an IPL device as the source for the data in the DCSS, follow the steps in "Performing an IPL to load data to the DCSS" on page 21.

Using the block device driver is technically simpler but requires class E privileges for the Linux guest.

If you do not have class E privileges, the technically more complex IPL method might be more convenient from an organizational point of view. You still need the cooperation of an operator with class E privileges but the actions that require class E privileges can be performed from a suitable guest that does not need to have Linux installed.

Using the DCSS block device driver to copy data to the DCSS

This section describes how to create a DCSS if you want to use the DCSS block device driver to copy your data to the DCSS. This task consists of two subtasks:

- "Creating the DCSS" on page 18
- "Copying data to the DCSS using the DCSS block device driver" on page 19

If you want to load your DCSS with data by performing an IPL, skip this task and perform the steps in "Performing an IPL to load data to the DCSS" on page 21 instead.

Before you start:

- You need to know size of the DCSS you want to share.

- You need to know the start and end address of your DCSS.
- You need privilege class E for your z/VM guest. This guest must be set up to allow a temporary increase of storage using a define store command.
- You need the DCSS block device driver, either as a module or built into the kernel.

Creating the DCSS

Perform these steps to create the DCSS (see *CP Command and Utility Reference*, SC24-6081 for more information on the commands used in these steps):

1. Take down Linux and IPL CMS in your guest machine.
2. From CMS, use the **defseg** command to define the DCSS. Issue a command of this form:

   ```
   # defseg <name of the DCSS> <first page number>-<last page number> sr
   ```

   The DCSS name is restricted to 8 characters. The page frame numbers must be in hexadecimal notation. For more details on the **defseg** command, refer to the z/VM CP documentation.

   Example:

   ```
   # defseg lnxshare 20000-5ffff sr
   ```

3. Verify that the DCSS has been defined. Issue:

   ```
   # query nss map
   ```

   The DCSS has been defined correctly if the output shows your segment with the address limits that you have specified.
4. Define your virtual guest storage sufficiently large to cover the entire DCSS. Issue a command of the form:

   ```
   # define stor <value>
   ```

   Specify a value greater than the last page number you used in the **defseg** statement in step 2.

   Example: To allow for a DCSS with an upper limit of 1536 MB issue:

   ```
   # define stor 1536M
   ```

   Tip: Increasing the size to 2G covers any possible DCSS.
5. Save the DCSS by issuing a command of this form:

   ```
   # saveseg <name of the DCSS>
   ```

   This command can take several minutes to complete.

   Example: To save a DCSS named LNXSHARE issue:

   ```
   # saveseg LNXSHARE
   ```

6. Verify that the DCSS was saved correctly. Issue:

   ```
   # query nss map
   ```

Result: The save operation has completed successfully if the class (column title: CL) has changed from skeleton (S) to active (A).

7. Log off from your z/VM guest. Next time you IPL an operating system in the guest, the guest storage will be reset to its previous size.

Now you are ready to boot your Linux guest and use the DCSS block device driver to copy the directories or files to the DCSS.

Copying data to the DCSS using the DCSS block device driver

You cannot use the xip2 file system to write to a DCSS because it is designed to be read-only. Because xip2 and ext2 have compatible file system structures, you can use the extended file system (ext2) in combination with the DCSS block device driver instead.

1. Ensure that the DCSS block device driver is available. Unless the DCSS block device driver has been built into your kernel or has already been loaded, load the dcssblk module. Issue:

```
# modprobe dcssblk
```

   Result: The device driver has been loaded successfully if `cat /proc/devices` shows an entry "dcssblk". The number preceding the device driver name is the major number of your DCSS block device.

2. Ensure that there are device nodes for your DCSS block devices. If they are not created for you, for example by udev, create them with the **mknod** command. The following lines create 4 nodes (to update a DCSS one node is sufficient).

```
# mknod /dev/dcssblk0 b <major number> 0
# mknod /dev/dcssblk1 b <major number> 1
# mknod /dev/dcssblk2 b <major number> 2
# mknod /dev/dcssblk3 b <major number> 3
```

   In the commands, replace <*major number*> with the major number from `/proc/devices`.

   The examples for the following steps use the nodes with standard device names as created by the commands in this step. If your distribution provides different device nodes, you can use these nodes instead.

3. Load the DCSS by adding it to the block device driver. Issue a command of this form:

```
# echo "<name of DCSS>" > /sys/devices/dcssblk/add
```

   With the device nodes of step 2, the first DCSS you add becomes `/dev/dcssblk0`, the second `/dev/dcssblk1`, and so on.

   Result: Loading a DCSS creates a subdirectory with the name of the DCSS within `/sys/devices/dcssblk`. This subdirectory contains three attributes and a symbolic link, block. The symbolic link points to the block device that corresponds to the DCSS. The save and shared attributes are used in the steps that follow.

   Example:

```
# echo "LNXSHARE" > /sys/devices/dcssblk/add
# ls -l /sys/devices/dcssblk/LNXSHARE
total 0
lrwxrwxrwx  1 root root    0 Oct 26 19:29 block -> ../../../block/dcssblk0
-rw-r--r--  1 root root 4096 Oct 26 19:29 detach_state
-rw-r--r--  1 root root 4096 Oct 26 19:29 save
-rw-r--r--  1 root root 4096 Oct 26 19:29 shared
```

4. Change the access mode of the DCSS from “share” (default) to “exclusive-writable”. Issue a command of this form:

```
# echo 0 > /sys/devices/dcssblk/<name of DCSS>/shared
```

Example:

```
# echo 0 > /sys/devices/dcssblk/LNXSHARE/shared
```

5. Create an ext2 file system on the block device that is associated with the DCSS. Use a block size of 4 KB.

Example:

```
# mke2fs -b 4096 /dev/dcssblk0
```

6. Mount the file system in the DCSS on a spare mount point.

Example:

```
# mount /dev/dcssblk0 /mnt
```

7. Copy the directories or files you want to be shared to the file system on the DCSS. Proceed according to the granularity at which you are sharing data:
   - If you want to share individual read-only files, you will have used copylibs.sh (see “Identifying individual files to be shared” on page 9) to copy the files to a separate destination directory. You can copy the entire destination directory to the file system image:

     Example:

     ```
     # cp -a /dcss/* /mnt
     ```

     Alternatively, you can use copylibs.sh to directly copy the files to the file system on the DCSS. Specify the file system’s mount point as the target directory.

     Example:

     ```
     # copylibs.sh -f httpd -d /mnt
     # copylibs.sh -f ...
     ...
     ```

   - If you want to share entire directories, copy these directories to the file system on the DCSS.

     Example:

     ```
     # cp -a /bin /mnt
     # cp -a /lib /mnt
     # mkdir /mnt/usr
     # cp -a /usr/bin /mnt/usr
     # cp -a /usr/lib /mnt/usr
     ```

8. Save the DCSS.

You create a save request by issuing:

```
# echo 1 > /sys/devices/dcssblk/<name of DCSS>/save
```

The save request is performed after the device is unmounted. You can cancel an existing save request by issuing:

```
# echo 0 > /sys/devices/dcssblk/<name of DCSS>/save
```

9. Unmount the file system.

   Example:

   ```
   # unmount /mnt
   ```

10. Remove the device. For example, remove the entry in sysfs by issuing:

    ```
    # echo <name of DCSS> > /sys/devices/dcssblk/remove
    ```

Proceed with "Task 6: Testing the DCSS" on page 25.

Performing an IPL to load data to the DCSS

This section describes how to create a DCSS if you want to perform an IPL to load data to the DCSS. This task consists of two subtasks:

- "Creating a file system image with the content of the DCSS"
- "Creating a DCSS from the image file" on page 24

If you want to use the DCSS block device driver to copy your data to the DCSS, omit this task and perform the steps in "Using the DCSS block device driver to copy data to the DCSS" on page 17 instead.

Before you start:

- You need to know the start and end address of your DCSS.
- You need to know the contents and size of the file system you want to share.
- You need an empty disk with sufficient space to hold the file system image.

Creating a file system image with the content of the DCSS

This task describes how to create a file system image on disk that contains the directories to be shared. This disk is needed temporarily only, for initializing the DCSS.

Perform these steps to create a file system image:

1. Prepare the disk as an ordinary Linux volume using the **dasdfmt** command.
2. Create one single large partition on the disk using the **fdasd** command.

   Tip: If you are sharing data at the granularity of entire directories, you can use a script to perform the remaining steps of this task for you (see "Using a script to create a file system image" on page 23).
3. Create an empty file with the size of the DCSS you want to create. Create the file using the **dd** Linux command line utility.

   Example: To create a file filesystem of 1 GB (=> 262144 pages) for a mount point `/mnt` issue:

   ```
   # dd if=/dev/zero of=/mnt/filesystem bs=4096 count=262144
   ```

4. Create an empty ext2 file system in the empty file using the **mke2fs** command.

   Example: To create an empty ext2 file system in the file created in the example for the previous step issue:

   ```
   # mke2fs -b 4096 /mnt/filesystem
   ```

   The **mke2fs** tool informs you that "filesystem" is not a block device and asks if you really want to create a file system in it. Respond with "yes".

5. Create a mount point for the file system.

   Example: Issue the following command to create a mount point `/xipimage`:

   ```
   # mkdir /xipimage
   ```

6. Mount the file system on the mount point you have created.

   Example: To mount a xip2 file system image `/mnt/filesystem` on a mount point `/xipimage` issue:

   ```
   # mount -t ext2 /mnt/filesystem /xipimage -o loop
   ```

7. Copy the directories or files you want to be shared to the file system image. Proceed according to the granularity at which you are sharing data:

   - If you want to share individual read-only files, you will have used copylibs.sh (see "Identifying individual files to be shared" on page 9) to copy the files to a separate destination directory. You can copy this destination directory to the file system image:

     Example:

     ```
     # cp -a /dcss /xipimage
     ```

     Alternatively, you can use copylibs.sh to directly copy the files to the file system image. To copy files directly to the file system image, specify the file system image's mount point as the target directory.

     Example:

     ```
     # copylibs.sh -f httpd -d /xipimage
     # copylibs.sh -f ...
     ...
     ```

   - If you want to share entire directories, copy these directories to the files system image.

     Example:

     ```
     # cp -a /bin /xipimage/bin
     # cp -a /lib /xipimage/lib
     # mkdir /xipimage/usr
     # cp -a /usr/bin /xipimage/usr/bin
     # cp -a /usr/lib /xipimage/usr/lib
     ```

8. Unmount the file system.

   Example:

   ```
   # unmount /xipimage
   ```

9. Prepare the DASD with the image file for IPL. Use the **zipl** command where option -t identifies the directory that contains the image file and option -s identifies the image file and the start address of the DCSS.

   Example: To specify an image file `/mnt/filesystem` and a DCSS that starts at 0x20000000 issue:

   ```
   # zipl -t /mnt -s /mnt/filesystem,0x20000000
   ```

10. Take down Linux.

You now have a disk that can be IPLed to create the DCSS.

Using a script to create a file system image: This section describes how to use a script for creating a file system image. Using the script is an alternative to performing the steps 3 on page 21 through 8 on page 22 of "Creating a file system image with the content of the DCSS" on page 21 if your DCSS contains data at the granularity of entire directories.

Before you start:

- All directories you want to share are located on the root file system.
- The script `/sbin/xipinit` needs to be present with the RODIRS variable set to include the directories to be shared (see the example in "Task 4: Providing a script to over-mount shared data on startup" on page 15)
- You need to know the name of the file system image and its size in megabyte.

Perform the following steps to use the mkxipimage.sh script (see "mkxipimage.sh" on page 36) for creating a file system image on a disk:

1. Get a copy of mkxipimage.sh. See "Scripts used for setting up a DCSS," on page 33 for information on where you can obtain this script.
2. Permit user root to run mkxipimage.sh.
3. Run mkxipimage.sh as user root.

   Note: The image file is created in the directory from where you run the script. Be sure that there is sufficient free space to hold the image file.
4. When prompted, enter the file name for the image file to be created.
5. When prompted, enter the size of the image file to be created.
6. Confirm that the script has run successfully. Verify that:
   - The script output shows a reasonable amount of free disk space.
   - The image file exists and has the correct size.
   - The log-file `mkxipimage.log` does not contain any error messages.
7. Mount the file system image using the loop device.

   Example: To mount an xip2 file system `/mnt/filesystem` on a mount point `/xipimage` issue:

   ```
   # mount -t ext2 /mnt/filesystem /xipimage -o loop
   ```

8. Check that everything is properly copied to the image file.
9. Unmount the file system image.

Creating a DCSS from the image file

This task describes how to copy a file system from a disk to a DCSS.

Before you start:

- You need your file system image on a dedicated DASD (see "Creating a file system image with the content of the DCSS" on page 21).
- You need privilege class E on z/VM.

Perform these steps from a z/VM guest with privilege class E. The guest does not need to have Linux installed.

1. IPL CMS in your guest machine.
2. Attach the DASD that you have prepared as an IPL device in "Creating a file system image with the content of the DCSS" on page 21.
3. From CMS, use the **defseg** command to define the DCSS. Issue a command of this form:

   ```
   # defseg <name of the DCSS> <first page number>-<last page number> sr
   ```

   The DCSS name is restricted to 8 characters. The page frame numbers must be in hexadecimal notation. For more details on the **defseg** command, refer to the z/VM CP documentation.

   Example:

   ```
   # defseg lnxshare 20000-5ffff sr
   ```

4. Verify that the DCSS has been defined. Issue:

   ```
   # query nss map
   ```

   The DCSS has been defined correctly if the output shows your segment with the address limits that you have specified.
5. Define your virtual guest storage sufficiently large to cover the entire DCSS. Issue a command of the form:

   ```
   # define stor <value>
   ```

   Specify a value greater than the last page number you used in the **defseg** statement in step 3.

   Example: To allow for a DCSS with an upper limit of 1536 MB issue:

   ```
   # define stor 1536M
   ```

   Tip: Increasing the size to 2G covers any possible DCSS.
6. IPL the DASD. Issue a command of the form:

   ```
   # ipl <device number> clear
   ```

   Result: The boot loader installed on the disk loads the contents of the DCSS into the z/VM virtual guest memory and then enters a disabled wait state.
7. Save the DCSS by issuing a command of this form:

   ```
   # saveseg <name of the DCSS>
   ```

This command can take several minutes to complete.

Example: To save a DCSS named LNXSHARE issue:

```
# saveseg LNXSHARE
```

8. Verify that the DCSS was saved correctly. Issue:

```
# query nss map
```

Result: The save operation has completed successfully if the class (column title: CL) has changed from skeleton (S) to active (A).

9. Log off from your z/VM guest. Next time you IPL an operating system in the guest, the guest storage will be reset to its previous size.

The file system image on disk is no longer required. z/VM saves the DCSS to the spool space to make it persistent across z/VM IPLs.

Task 6: Testing the DCSS

This task describes how you can assure that your DCSS has been set up correctly.

Perform the following steps to test the DCSS:

1. Mount the xip2 file system. Use a command of this form:

```
# mount -t xip2 -o ro,memarea=<name of the DCSS> none <mount point>
```

This command can take several minutes to complete.

Example: To mount a DCSS named LNXSHARE to /mnt issue:

```
# mount -t xip2 -o ro,memarea=LNXSHARE none /mnt
```

2. Issue the following command to verify that the file system has been mounted correctly:

```
# cat /proc/mounts
```

The file system has been mounted correctly if a line indicates that the mount point is active with the xip2 file system.

3. Verify that all files that you have copied to the mount point are now accessible. Issue:

```
# cd <mount point>
# tar -cvf /dev/null *
```

where *<mount point>* is your mount point.

If you encounter any problems, check the following:

- Issue the following command from CP to show the DCSS memory range:

```
# query nss
```

- If you are using a storage gap, verify that the entire DCSS fits within the gap.
- If you have located the DCSS above the guest storage:

  - Verify that the mem parameter is active and covers the entire DCSS.
  - Verify that the virtual machine memory is smaller than the start address of the DCSS.
- Verify that the file system size does not exceed the size of the DCSS.

If you have a problem that is not caused by these possible reasons, your DCSS is most probably not set up correctly. Use the CMS **purge** command to remove the DCSS and repeat the tasks "Task 1: Planning the DCSS content" on page 9 through "Task 6: Testing the DCSS" on page 25.

Issue a command like this:

```
# purge nss name <name of the DCSS>
```

where *<name of the DCSS>* is the name of your DCSS.

Example:

```
# purge nss name LNXSHARE
```

Task 7: Activating execute-in-place

This task describes how to assure that your shared directories or individual files are over-mounted with the DCSS content at startup.

If all directories or files to be shared are located on the root file system, you just need the xipinit.sh or xipinit-fw.sh script in place as described in "Task 4: Providing a script to over-mount shared data on startup" on page 15.

If you also want to share directories or individual files that are not on your root file system, you have to ensure that all directories to be shared, or the directories containing the individual files to be shared, are mounted to their correct positions before their content is accessed. To do this you need to modify the startup scripts of your Linux distribution.

Perform these steps to activate execute-in-place:

1. Depending on whether you are sharing individual files or entire directories, test your xipinit-fw.sh or xipinit.sh script. Run your script as user root.
   - If you are sharing individual files, issue `/sbin/xipinit-fw`.

     Example:

     ```
     # /sbin/xipinit
     Usage: init 0123456SsQqAaBbCcUu
     ```

   - If you are sharing entire directories, issue `/sbin/xipinit`.

     Example:

```
# /sbin/xipinit
mounting read-only segment
binding directory /lib
binding directory /usr/lib
binding directory /usr/X11R6/lib
binding directory /bin
binding directory /sbin
binding directory /usr/X11R6/bin
binding directory /usr/bin
Usage: init 0123456SsQqAaBbCcUu
```

2. Enter `cat /proc/mounts` to confirm that the directories or files to be shared have been over-mounted with the xip2 file system. The output dependings on the directories or files you are sharing.

   Examples:

   - The following is part of a sample output when sharing individual files. Depending on the number of shared files, the output might be lengthy:

     ```
     # cat /proc/mounts
     none /mnt xip2 ro 0 0
     none /usr/bin/less xip2 ro 0 0
     none /sbin/fsck xip2 ro 0 0
     none /bin/bash xip2 ro 0 0
     ...
     ```

   - The following is a sample output when sharing entire directories:

     ```
     # cat /proc/mounts
     none /mnt xip2 ro 0 0
     none /lib xip2 ro 0 0
     none /usr/lib xip2 ro 0 0
     none /usr/X11R6/lib xip2 ro 0 0
     none /bin xip2 ro 0 0
     none /sbin xip2 ro 0 0
     none /usr/X11R6/bin xip2 ro 0 0
     none /usr/bin xip2 ro 0 0
     ```

   Note: Because `/sbin/xipinit` or `/sbin/xipinit-fw` does not add the mounted directories to `/etc/fstab`, the **mount** command does not reflect these mounts properly!

   If your `xipinit` or `xipinit-fw` script does not work as intended, revisit "Task 4: Providing a script to over-mount shared data on startup" on page 15.

3. Depending on the granularity at which you are sharing data, add `init=/sbin/xipinit-fw` (individual files) or `init=/sbin/xipinit` (entire directories) to your kernel parameter line (parmfile).
4. Run **zipl** with the new parameter file.
5. Reboot Linux.

Execute-in-place is now active and your Linux instance makes use of the DCSS.

Chapter 3. Making your Linux guests use the DCSS

This task describes these steps you need to perform on each Linux instance that you want to use the DCSS.

Perform these steps to enable a Linux kernel to use a DCSS:

1. If your DCSS is located above the guest storage, change the kernel parameter line (see "Task 3: Setting up your guest for accessing the DCSS" on page 14). Skip this step if your DCSS is located in a storage gap.
2. Depending on the granularity at which you are sharing data, get a copy of `/sbin/xipinit-fw` (individual files) or `/sbin/xipinit` (entire directories) and activate it on your Linux ("Task 7: Activating execute-in-place" on page 26).

Chapter 4. Updating the software on a DCSS

This task describes how you can update the data on a DCSS.

Proceed according to the granularity at which you are sharing data and the extend of the intended changes:

- If you want to make minor changes to a DCSS with shared data at the granularity of entire directories, follow the steps in "Updating software on an existing DCSS."
- If you are sharing data at the granularity of individual files or if you are making major updates to a DCSS of any granularity, follow the steps in "Replacing a DCSS" on page 32.

Updating software on an existing DCSS

This task describes how you can perform minor software updates by writing to files on an existing DCSS file system. If you are intending to make major updates or if your DCSS contains data at the granularity of individual files, omit this section and perform the steps in "Replacing a DCSS" on page 32 instead.

Before you start:

- You need the DCSS block device driver, either as a module or built into the kernel.

  You cannot use the execute in-place file system (xip2) to write to a DCSS because it is designed to be read-only. Because xip2 and the extended file system (ext2) have compatible file system structures, you can use the ext2 instead. ext2 uses the DCSS block device driver to access the file system in memory.
- You need a z/VM guest with Class E user privileges to perform step 3.
- The updated file system image must fit into the existing DCSS.

To update data on an existing DCSS, perform the following steps:

1. Restart your Linux guests without activating execute-in-place. For each guest that shares the DCSS:
   a. Remove the "init=" parmeter from the kernel parameter line (parmfile).
   b. Run **zipl** with the new parameter file.
   c. Boot Linux using the new boot configuration.
2. Perform the required software update on each guest that shares the DCSS.

   Perform the updates according to your distribution documentation, as you would on any platform. Be sure to make the *same updates* on each guest. For example, install or update the same packages with the same version. Do not continue until all updates are completed on all guests that share the DCSS.
3. Run the script update.sh to write the updates to your DCSS. Only run the update.sh once, on only one of your Linux guests.
   a. Get a copy of the update.sh script (see "update.sh" on page 37). See "Scripts used for setting up a DCSS," on page 33 for information on where you can obtain this script.
   b. Run the script as superuser root. The script parses your xipinit file, accesses the DCSS, and updates the DCSS according to your xipinit file.

4. Restart your Linux guests with execute-in-place activated. For each guest that shares the DCSS:
   a. Add `init=/sbin/xipinit` to your kernel parameter line (parmfile).
   b. Run **zipl** with the new parameter file.
   c. Reboot Linux.

Your Linux guests now use the updated DCSS.

Replacing a DCSS

This section describes how to replace your DCSS with a new DCSS that provides the updated software. You can use this update method for any DCSS.

Tip: If you are making minor updates to a DCSS that contains shared data at the granularity of entire directories, you can also proceed as described in "Updating software on an existing DCSS" on page 31 instead of replacing the DCSS.

To replace a DCSS, perform the following steps:

1. Restart your Linux guests without activating execute-in-place. For each guest that shares the DCSS:
   a. Remove the "init=" parmeter from the kernel parameter line (parmfile).
   b. Run **zipl** with the new parameter file.
   c. Boot Linux using the new boot configuration.
2. Perform the required software update on each guest that shares the DCSS.

   Perform the updates according to your distribution documentation, as you would on any platform. Be sure to make the *same updates* on each guest. For example, install or update the same packages with the same version. Do not continue until all updates are completed on all guests that share the DCSS.
3. Create a new DCSS as described in Chapter 2, "Setting up a DCSS," on page 9.
4. Restart your Linux guests with execute-in-place activated. For each guest that shares the DCSS:
   a. Add `init=/sbin/xipinit` to your kernel parameter line (parmfile).
   b. Run **zipl** with the new parameter file.
   c. Reboot Linux.

Your Linux guests now use the new DCSS.

Appendix. Scripts used for setting up a DCSS

This appendix contains the scripts that are referred to in the main part of this document:

- copylibs.sh
- mkxipimage.sh
- update.sh
- xipinit.sh
- xipinit-fw.sh

You can download a tarball with copies of these scripts from:
`ibm.com/developerworks/linux/linux390/april2004_documentation.shtml`

If you are routed to the top-level page: On the left navigation bar, under Kernel 2.6, click **April 2004 stream**. On the April 2004 stream page, click **Documentation**.

If you copy any of the scripts from this appendix, confirm that copying and pasting does not change any characters.

copylibs.sh

You need copylibs.sh for "Task 1: Planning the DCSS content" on page 9 if you want to share the read-only parts of applications and individual files.

```
#!/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
```

Figure 8. copylibs.sh - 1/2

```
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
```

Figure 9. copylibs.sh (continued 2/2)

mkxipimage.sh

You can optionally use mkxipimage.sh for "Creating a file system image with the content of the DCSS" on page 21 if you want to share data at the granularity of entire directories.

```
#!/bin/sh
#
# mkxipimage.sh, Version 1
#
# (C) Copyright IBM Corp. 2002,2005
#
# This script generates an XIP image. It requires:
# - /sbin/xipinit in with up-to-date RODIRS defined
# - rights to mount the image to /mnt using loop device
# - enough diskspace ;)
echo "mkxipimage - generate a filesystem image for use with xip2"
echo "please enter file name for the image (max 8 chars)"
read FSIMAGE
echo "please enter the amount of megabytes to be used"
read FSIMAGE_SIZE
echo "reading /sbin/xipinit"
eval $(cat /sbin/xipinit | grep "RODIRS=" |grep -v 'RODIRS=\$' |grep -v "RODIRS=\"")
echo "generating image file" $FSIMAGE
echo generating $FSIMAGE with $FSIMAGE_SIZE megabytes >>mkxipimage.log 2>&1
dd if=/dev/zero of=$FSIMAGE bs=1048576 count=$FSIMAGE_SIZE >>mkxipimage.log 2>&1
echo creating ext2 filesystem in $FSIMAGE
mke2fs -b 4096 -F $FSIMAGE >>mkxipimage.log 2>&1
echo mounting $FSIMAGE to /mnt using loop
mount -t ext2 $FSIMAGE /mnt -o loop >>mkxipimage.log 2>&1
echo copying data to the filesystem image
while [ -n "$RODIRS" ] ; do
dir="${RODIRS%%,*}"
RODIRS="${RODIRS#*,}"
dir=$(echo $dir | sed "s/^\///g")
mkdir -p /mnt/$dir
cp -a /$dir/* /mnt/$dir >>mkxipimage.log 2>&1
done
echo free diskspace info:
df |grep "/mnt"
echo unmounting /mnt
umount /mnt
echo done
```

Figure 10. mkxipimage.sh

update.sh

You need update.sh for "Creating a file system image with the content of the DCSS" on page 21. In this task, you make minor updates by writing to an existing DCSS with data at the granularity of entire directories.

```
#!/bin/sh
#
# update.sh, Version 1
#
# (C) Copyright IBM Corp. 2005
#
# This script can be used to mount a DCSS for software updates. Requirements:
# - /sbin/xipinit in with up-to-date RODIRS defined
# - superuser privileges in Linux
# - class E privileges in z/VM

#temporary device node used
TEMPNODE=/tmp/dcssupdate.dev

#uncommend following 2 lines for debugging only
#set -v
#FAKE=echo


parse_xipinit() {
    #looks for xipinit script, and parses its settings
    # save parameters from environment
    _RODEV="$rodev"
    _ROFS="$rofs"
    _ROOPTIONS="$rooptions"
    _RODIRS="$rodirs"
    _MEMAREA="$memarea"
    _ROMOUNT="$romount"
    # now parse the script
    echo "reading" /sbin/xipinit
    eval $(cat /sbin/xipinit | grep "RODIRS=" |grep -v 'RODIRS=\$' |grep -v "RODIRS=\"")
    eval $(cat /sbin/xipinit | grep "ROFS=" |grep -v 'ROFS=\$' |grep -v "ROFS=\"")
    eval $(cat /sbin/xipinit | grep "RODEV=" |grep -v 'RODEV=\$' |grep -v "RODEV=\"")
    eval $(cat /sbin/xipinit | grep "ROOPTIONS=" |grep -v '_ROOPTIONS=' |grep -v "ROOPTIONS=\"\$")
    eval $(cat /sbin/xipinit | grep "ROMOUNT=")
    MEMAREA=$(echo $ROOPTIONS|awk 'match($0, /memarea=[^ ]+/){print substr($0, RSTART+8, RLENGTH-8)}')
    # override parameters with saved environment
    RODEV="${_RODEV:-$RODEV}"
    ROFS="${_ROFS:-$ROFS}"
    ROOPTIONS="${_ROOPTIONS:-$ROOPTIONS}"
    RODIRS=${_RODIRS:-$RODIRS}
    MEMAREA=${_MEMAREA:-$MEMAREA}
    ROMOUNT=${_ROMOUNT:-$ROMOUNT}
    # make sure it ends with ,
    RODIRS="$RODIRS",
}


check_blockdev() {
    #load dcss block device driver if needed
    test -d /sys/devices/dcssblk || ( echo "loading dcss block device driver" && modprobe dcssblk )
    if [ ! -d /sys/devices/dcssblk ]; then
 echo "error: could not initialize dcss block device driver"
 exit 1
    fi
    echo "dcss block device driver found"
}

load_segment() {
    #load the segment using the block device
    echo "loading segment" $MEMAREA
    $FAKE echo $MEMAREA >/sys/devices/dcssblk/add
    if [ ! -d /sys/devices/dcssblk/$MEMAREA ]; then
 echo "error: failed to load segment"
 exit 1
    fi
    echo "reloading segment" $MEMAREA "in nonshared mode"
    $FAKE echo 0 >/sys/devices/dcssblk/$MEMAREA/shared
    TEMPVAR0=$(cat /sys/devices/dcssblk/$MEMAREA/shared)
    if [ $TEMPVAR0 -ne 0 ]; then
 echo "error: cannot change segment to nonshared mode, maybe you don't have class E privileges?"
 exit 1
    fi
}
```

Figure 11. update.sh - 1/3

```
unload_segment() {
    #unload the segment again
    echo "unloading segment" $MEMAREA
    $FAKE echo $MEMAREA >/sys/devices/dcssblk/remove
    if [ -d /sys/devices/dcssblk/$MEMAREA ]; then
 echo "error: failed to unload segment. is it still busy?"
 exit 1
    fi
}

create_devnode() {
    #create a temporary device node for mounting
    if [ -e $TEMPNODE ]; then
 echo "error: temporary device node" $TEMPNODE "already exists."
 echo "if it still exists from a former run, please delete it and try again"
 exit 1
    fi
    if [ ! -f /sys/devices/dcssblk/$MEMAREA/block/dev ]; then
 echo "unexpected-error: cannot find block device directory in sysfs, please report to linux390@de.ibm.com"
 exit 1
    fi
    MAJOR=$(cat /sys/devices/dcssblk/$MEMAREA/block/dev |sed -e "s/\:[0-9]\{1,\}//g")
    MINOR=$(cat /sys/devices/dcssblk/$MEMAREA/block/dev |sed -e "s/[0-9]\{1,\}\://g")
    mknod $TEMPNODE b $MAJOR $MINOR
    if [ ! -b $TEMPNODE ]; then
 echo "error: cannot not create temporary device node" $TEMPNODE
        exit 1
    fi
}

delete_devnode() {
    #delete temporary device node again
    rm -f $TEMPNODE
    if [ -e $TEMPNODE ]; then
 echo "unexpected-error: cannot remove temporary device node" $TEMPNODE ", please report to linux390@de.ibm.com"
 exit 1
    fi
}

mount_segment() {
    #this mounts the segment on /mnt
    echo "mounting segment" $MEMAREA "on /mnt"
    $FAKE mount -t ext2 $TEMPNODE /mnt
    TEMPVAR0=$(cat /proc/mounts |grep $TEMPNODE| wc -l)
    if [ ! 1 -eq $TEMPVAR0 ]; then
 echo "error: cannot mount segment, is either already mounted or operation failed"
 umount $TEMPNODE
 delete_devnode
 unload_segment
 exit 1;
    fi
}

unmount_segment() {
    #unmounts the segment from /mnt
    echo "unmounting segment" $MEMAREA
    $FAKE umount $TEMPNODE
}
```

Figure 12. update.sh (continued 2/3)

```
update_segment() {
    #copies all data from disk to segment
    echo "updating software on segment"
    echo "phase 1: deleting old content"
    TEMPVAR0=$(echo $RODIRS)
    while [ -n "$TEMPVAR0" ]; do
 dir="${TEMPVAR0%%,*}"
 TEMPVAR0="${TEMPVAR0#*,}"
 if [ ! -n "$TEMPVAR0" ]; then
     continue
 fi
 echo "deleting directory" /mnt/$dir
 if [ ! -d "/mnt/$dir" ]; then
     echo "error:cannot find directory" /mnt/$dir
     echo "does not exist on dcss or is not a directory"
            unmount_segment
     delete_devnode
     unload_segment
     echo "cannot recover from earlier error - exit"
     exit 1
 fi
 $FAKE rm -rf /mnt/$dir
 if [ -d "/mnt/$dir" ]; then
     echo "error:cannot delete directory" /mnt/$dir
            unmount_segment
     delete_devnode
     unload_segment
     echo "cannot recover from earlier error - exit"
     exit 1
 fi
    done
    echo "phase 2: copy new content"
    TEMPVAR0=$(echo $RODIRS)
    while [ -n "$TEMPVAR0" ]; do
 dir="${TEMPVAR0%%,*}"
 TEMPVAR0="${TEMPVAR0#*,}"
 if [ ! -n "$TEMPVAR0" ]; then
     continue
 fi
 echo "copying directory" /mnt/$dir
 mkdir -p /mnt/$dir
 if [ ! -d "/mnt/$dir" ]; then
     echo "error:cannot create directory" /mnt/$dir
            unmount_segment
     delete_devnode
     unload_segment
     echo "cannot recover from earlier error - exit"
     exit 1
 fi
 cp -a /$dir/* /mnt/$dir
 if [ $? -ne 0 ]; then
     echo "error:cannot copy directory" /mnt/$dir
            unmount_segment
     delete_devnode
     unload_segment
     echo "cannot recover from earlier error - exit"
     exit 1
  fi
    done
}

save_update () {
    if [ ! -f /sys/devices/dcssblk/$MEMAREA/save ]; then
 echo "error:segment or block device driver not loaded"
 exit 1
    fi

    echo 1 >/sys/devices/dcssblk/$MEMAREA/save
    TEMPVAR0=$(cat /sys/devices/dcssblk/$MEMAREA/save)
    if [ $TEMPVAR0 -eq 0 ]; then
 echo "success: segment saved"
    fi
    if [ $TEMPVAR0 -eq 1 ]; then
 echo "warning: segment scheduled to be saved when it becomes idle, check why it is busy!"
    fi
}


parse_xipinit
check_blockdev
load_segment
create_devnode
mount_segment
update_segment
unmount_segment
delete_devnode
save_update
unload_segment
```

Figure 13. update.sh (continued 3/3)

xipinit.sh

You need xipinit.sh for "Task 4: Providing a script to over-mount shared data on startup" on page 15 if you want to share entire directories. If you want to share individual files use "xipinit-fw.sh" on page 41 instead.

```
00001 #!/bin/sh
00002 #
00003 # xipinit.sh, Version 1
00004 #
00005 # (C) Copyright IBM Corp. 2002,2005
00006 #
00007 # /sbin/xipinit: use read only files from another file system
00008 #
00009 # default options
00010 RODEV=none
00011 ROFS=xip2
00012 ROOPTIONS="ro,memarea=XIP2502S"
00013 RODIRS=/lib,/usr/lib,/usr/X11R6/lib,/bin,/sbin,/usr/X11R6/bin,/usr/bin,
00014 # internals
00015 #set -v
00016 #FAKE=echo
00017 CONFIGFILE=/etc/roinitrc
00018 ROMOUNT=/mnt
00019 CHROOT=/usr/sbin/chroot
00020 INIT=/sbin/init
00021 # save kernel parameters from environment
00022 _RODEV="$rodev"
00023 _ROFS="$rofs"
00024 _ROOPTIONS="$rooptions"
00025 _RODIRS="$rodirs"
00026 # read config file, if any
00027 test -f "$CONFIGFILE" && . "$CONFIGFILE"
00028 # override parameters with values from config file
00029 RODEV="${rodev:-$RODEV}"
00030 ROFS="${rofs:-$ROFS}"
00031 ROOPTIONS="${rooptions:-$ROOPTIONS}"
00032 RODIRS=${rodirs:-$RODIRS}
00033 # override parameters with kernel parameters
00034 RODEV="${_RODEV:-$RODEV}"
00035 ROFS="${_ROFS:-$ROFS}"
00036 ROOPTIONS="${_ROOPTIONS:-$ROOPTIONS}"
00037 RODIRS=${_RODIRS:-$RODIRS}
00038 # make sure it ends with ,
00039 RODIRS="$RODIRS",
00040 # mount ro file system to its mount point
00041 echo "mounting read-only segment"
00042 $FAKE mount "$RODEV" "$ROMOUNT" -o "$ROOPTIONS" -t "$ROFS"
00043 # bind mount all ro dirs into rw filesystem
00044 while [ -n "$RODIRS" ] ; do
00045 dir="${RODIRS%%,*}"
00046 RODIRS="${RODIRS#*,}"
00047 test -d "$dir" || continue
00048 echo "binding directory" $dir
00049 $FAKE mount --bind "$ROMOUNT/$dir" "$dir"
00050 done
00051 # run real init
00052 $FAKE exec "$INIT" "$@"
```

Figure 14. xipinit sample script

xipinit-fw.sh

You need xipinit-fw.sh for “Task 4: Providing a script to over-mount shared data on startup” on page 15 if you want to share the read-only parts of applications and individual files. If you want to share entire directories use “xipinit.sh” on page 40 instead.

```
#!/bin/bash
#
# xipinit-fw.sh, Version 1
#
# (C) Copyright IBM Corp. 2005

#mount point of xipimage
MPXIPIMAGE=""
#name of xipimage
XIPIMAGE=""

/bin/mount -t xip2 -o ro,memarea=$XIPIMAGE none $MPXIPIMAGE
for i in $(/usr/bin/find $MPXIPIMAGE/*)
do
  if [[ -f $i ]]
    then
       /bin/mount --bind $i /$i
  fi
done
exec /sbin/init $@
```

Figure 15. xipinit-fw.sh sample script

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.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

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.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

Trademarks

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

@server
IBM
S/390
z/VM
zSeries

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

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

Readers' Comments — We'd Like to Hear from You

Linux on zSeries
How to use Execute-in-Place Technology
with Linux on z/VM
March 23, 2005
Linux Kernel 2.6 (April 2004 stream)

Publication No. SC33-8283-00

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	☐	☐	☐	☐	☐

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	☐	☐	☐	☐	☐
Complete	☐	☐	☐	☐	☐
Easy to find	☐	☐	☐	☐	☐
Easy to understand	☐	☐	☐	☐	☐
Well organized	☐	☐	☐	☐	☐
Applicable to your tasks	☐	☐	☐	☐	☐

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? ☐ Yes ☐ No

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Name

Address

Company or Organization

Phone No.

Readers' Comments — We'd Like to Hear from You
SC33-8283-00

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape

PLACE
POSTAGE
STAMP
HERE

IBM Deutschland Entwicklung GmbH
Information Development
Department 3248
Schoenaicher Strasse 220
71032 Boeblingen
Germany

Fold and Tape | **Please do not staple** | Fold and Tape

SC33-8283-00

Cut or Fold
Along Line

SC33-8283-00