Linux for zSeries and S/390

How to Improve the Performance of Linux on z/VM with Execute-in-Place Technology July 30, 2004

Linux Kernel 2.6 (April 2004 stream)

LNUX-H6DS-00

Linux for zSeries and S/390

How to Improve the Performance of Linux on z/VM with Execute-in-Place Technology July 30, 2004

Linux Kernel 2.6 (April 2004 stream)

LNUX-H6DS-00

Note

Before using this information and the product it supports, read the information in “Notices” on page 19.

First Edition (July 2004)

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

LNUX-H6DS is the Linux for zSeries and S/390® kernel 2.6 (April 2004 stream) equivalent to LNUX-HTDS which applies to Linux for zSeries and S/390 kernel 2.4, June 2003 stream.

Contents

How to improve the performance of Linux on z/VM with execute-in-place technology

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

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

Important: DCSSs occupy spool space. Ensure that your available spool space is at least a multiple of the DCSS size.

What is a DCSS?

This section gives an introduction to z/VM DCSS and how it 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 can be shared

The shared data must be identical across all sharing Linux instances. This document describes how to share data at the granularity of file system directories.

- 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.
- Directories with applications and libraries that are frequently used by numerous Linux instances are good candidates for sharing.

The following items are not suitable for sharing:

- Directories that are being written to.

Note: If you set up a DCSS according to this document, 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.

How does it work?

z/VM allows you to define DCSSs that can be filled 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.

All guests access the DCSS with the same real addresses. To avoid conflicts with the guest's addressing the DCSS addresses are set to be above the virtual storage limit of all individual guests. The guest virtual storage is the amount of memory that z/VM presents to a guest as the guest's real memory.

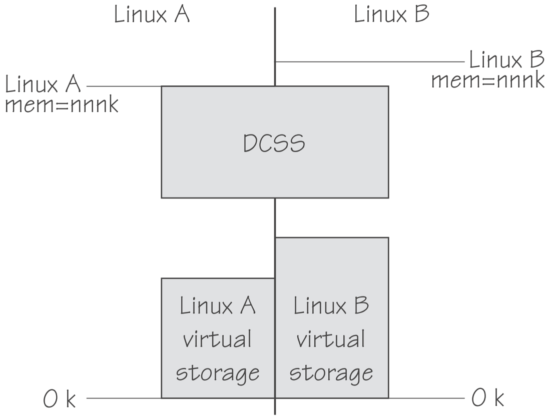

Figure 1. Linux guest virtual storage and DCSS

Figure 1 shows two Linux guests with their virtual 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.

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

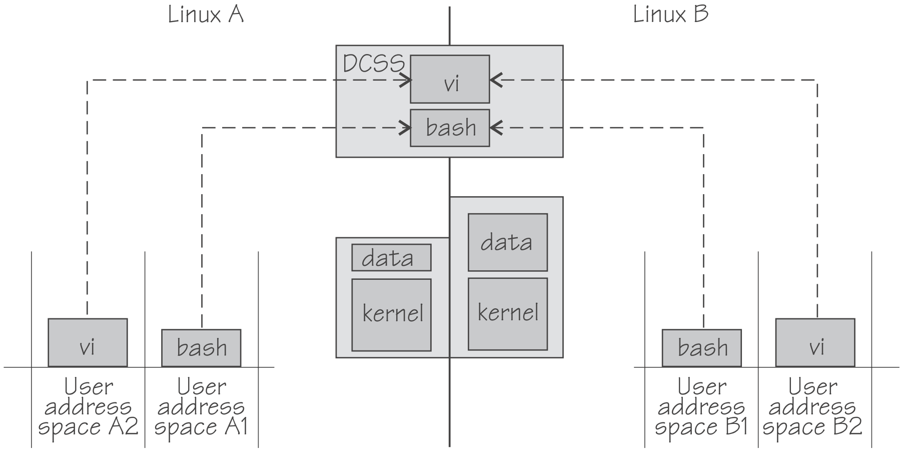

Figure 2. Shared programs run directly from the DCSS

Figure 2 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.

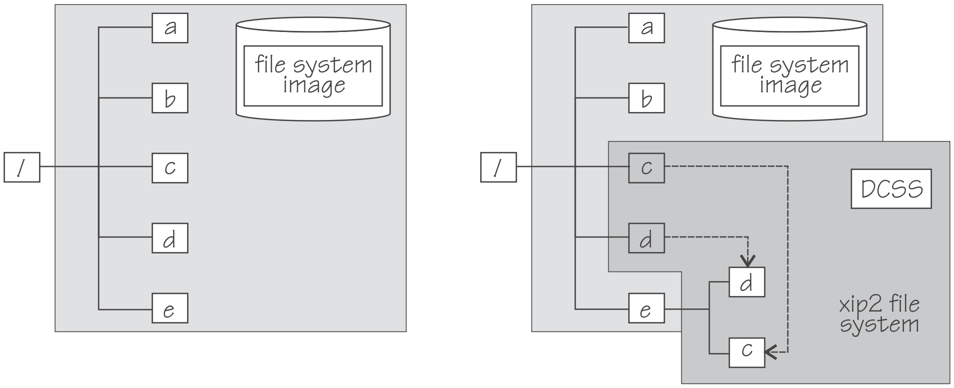

Figure 3. xip2 over-mounting directories

On the left side, Figure 3 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. This increases performance. Running applications directly in a shared memory segment saves memory.

Limitations and trade-offs you must be aware of

- The DCSS 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 virtual guest storage (perceived as physical memory by Linux) of any operating system instance that accesses it.
- The ending address of the DCSS has an upper limit. It must not exceed:
  - 1960 MB for 31-bit Linux kernels. Linux needs the address range from 1960 MB to 2 GB for virtual memory allocations.
  - 2 GB for 64-bit Linux kernels. This limit is a z/VM restriction.
- 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 directories that contain rarely used data can degrade the performance and scalability of a server farm.

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.
- The Linux kernels require the execute-in-place file system xip2, either built-in or as a module.

To be able to update the contents of an existing DCSS your Linux kernel also needs:

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

Which special skills do you need?

- If you cannot find a Linux distribution that satisfies the requirements, you need the skill to build and maintain your own Linux kernel.
- If you want to share directories that are not on the root file system, you need to be able to modify the startup scripts of your Linux distribution.

Creating 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. Build a Linux kernel with the required modules.
2. Plan the layout of your file system.
3. Provide a script to over-mount shared directories on startup.
4. Create a file system image.
5. Create a DCSS from the image file.
6. Change the kernel parameter line.
7. Test the DCSS.
8. Activate execute-in-place.

Task 1: Building a Linux kernel with the required modules

This task describes how you can build a Linux kernel that is suitable for working with z/VM DCSS. You need to build your own kernel only if you cannot find a suitable distribution that includes the modules you need. Linux kernels need the xip2 file system to use a z/VM DCSS. Linux kernels need the DCSS block device driver to update an existing DCSS.

Prerequisites: The following description assumes that you know how to build and install a Linux kernel. If you have little experience with building and installing kernels, refer to the general Linux documentation and user mailing lists and do a dry-run with a standard kernel first.

If you cannot find a suitable distribution with the modules you need for working with DCSSs, perform these steps to build your own kernel:

1. Download the latest 2.6.x. kernel patches from: `http://oss.software.ibm.com/linux390/index.shtml`
2. Download the vanilla kernel with the corresponding version from: `http://www.kernel.org`
3. Unpack the vanilla kernel and apply the patches.
4. Download the 2.6.x. xip2 patches from: `http://linuxvm.org/Patches/`.
5. Apply the xip2 patches to the kernel.
6. If you want to be able to update existing DCSS file systems from Linux, include the DCSS block device driver. Select this item in the kernel configuration menu options:

   ```
   Block device drivers
       z/VM discontiguos saved segments (DCSS) block device driver
   ```
7. For all other options, select the values according to your system setup.
8. Build and install your kernel as usual.

You can now safely boot your new configuration. xip2 and the DCSS block device driver only have an effect when you use them explicitly.

Task 2: Planning the DCSS content

This task describes how to determine the maximum size of a DCSS and helps you to decide what to include in a DCSS.

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.

1. Determine the maximum size of the DCSS.
   a. Determine the lower limit of the DCSS address range. The DCSS address range must not overlap with the virtual storage of the operating system images that use it. The lower limit is, therefore, equal to the highest amount of virtual storage that has been allocated to any of your images.
   b. Determine the upper limit of the DCSS address range. The upper limit is 1960 MB for 31-bit Linux kernels or 2 GB for 64-bit Linux kernels.
   c. Subtract the lower limit from the upper limit to obtain the maximum possible DCSS size.

   Example: Consider a number of Web servers with 31-bit Linux kernels where some have been assigned 256 MB virtual storage and others have been assigned 512 MB virtual storage.

Because the highest amount of virtual storage is 512 MB, the lower limit of the DCSS is 512 MB. The upper limit for 31-bit Linux is 1960 MB. The maximum size of the DCSS is therefore:

1960 MB - 512 MB = 1448 MB.

2. Identify the directories to be shared.

   The following list helps you to find 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.

3. 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 which is below the 1448 MB we had calculated as the maximum for our example.

4. Ensure that the required size does not exceed the maximum DCSS size for your environment. If necessary, remove some directories from the list of directories you want to share. Remove directories where you expect the least benefit, that is, where the data is used least frequently.
5. 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.

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

   Example: Continuing the example of the previous steps:

   Start address: 512 MB => 0x20000000
   End address: 512 MB + 1024 MB - 1 B => 0x5FFFFFFF
6. 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: For a start address of 0x20000000 and an end address of 0x5FFFFFFF the start and end frame numbers are 0x20000 and 0x5FFFF.

Task 3: Providing a script to over-mount shared directories on startup

This task provides a sample script that can be used to over-mount directories with the content of a DCSS before Linux accesses them. It also provides steps to tailor the script to your environment.

In this context, over-mounting a directory means replacing it with the contents of the DCSS at system startup. Shared directories must be over-mounted before any data on them is accessed. If a directory is accessed before being over-mounted, accessed libraries and binaries might remain in memory, and so waste memory, after the corresponding directory is over-mounted.

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 following xipinit script to over-mount directories on the root file system. On system startup, xipinit runs instead of `/sbin/init`. First xipinit over-mounts the shared directories and then it runs `/sbin/init`.

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

Figure 4. xipinit sample script

Perform these steps to tailor the script to your environment:

1. Specify your DCSS. In line 8, `ROOPTIONS="ro,memarea=XIP2502S"`, replace `XIP2502S` with the name of your DCSS.
2. Specify the directories you want to be shared. In line 9,

   ```
   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 14 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.

You can now proceed to create a file system with the shared directories. The script you have tailored in this task will be needed:

- In the next task, "Task 4: Creating a file system image with the content of the DCSS," if you create the file system image with the provided script.
- In "Task 8: Activating execute-in-place" on page 15.

Task 4: 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.

Prerequisites: 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.

A DCSS holds shared Linux code in form of a Linux file system. Before you can save a file system as a DCSS, you need to IPL from a disk that contains the file system image. Create the file system image on a separate disk. This disk is needed temporarily only, for initializing the DCSS.

1. Prepare the disk as a normal Linux volume using the **dasdfmt** command.
2. Create one single large partition on the disk using the **fdasd** command.
3. Create a file system in the partition and mount it on a spare mount point, for example, /mnt.

   Tip: 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 10).
4. 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
   ```
5. 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".
6. Create a mount point for the file system.

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

   ```
   # mkdir /xipimage
   ```

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

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

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

8. Copy all directories you want to be shared to the file 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
   ```

9. Unmount the file system.

   Example:

   ```
   # unmount /xipimage
   ```

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

Using a script to create a file system image

This section provides an example script for creating a file system image. Using the script is an alternative to performing the steps 4 through 9 of “Task 4: Creating a file system image with the content of the DCSS” on page 9.

Prerequisites:

- 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 3: Providing a script to over-mount shared directories on startup” on page 7)
- You need to know the name of the file system image and its size in MB.

```
#!/bin/sh
# 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 5. mkxipimage.sh sample script

Perform the following steps to use the script for creating a file system image on a disk:

1. Name the script `mkxipimage.sh` and permit user root to run it.
2. 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.
3. When prompted, enter the file name for the image file to be created.
4. When prompted, enter the size of the image file to be created.
5. 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.
6. 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
   ```
7. Check that everything is properly copied to the image file.
8. Unmount the file system image.

Task 5: Creating a DCSS from the image file

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

Prerequisite:

- You need your file system image on a dedicated DASD (see “Task 4: Creating a file system image with the content of the DCSS” on page 9).
- You need privilege class E on z/VM.

Perform these steps to create the DCSS:

1. 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
   ```

2. Take down Linux and IPL CMS in your guest machine.
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. Find out how many CPUs are defined for your z/VM guest. Issue:

   ```
   # query cpus
   ```

   There must only be one CPU. If there is more than one CPU, issue commands like this to detach excess CPUs:

   ```
   # detach cpu <number of cpu>
   ```

6. 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 of 1536 MB issue:

```
# define stor 1536M
```

7. 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.
8. Save the DCSS by issuing a command like this:

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

   This command can take several minutes to complete.

   Example: To save a DCSS named LNXSHARE issue:

   ```
   # saveseg LNXSHARE
   ```

9. 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).
10. Log off from your z/VM guest. Next time you IPL an operating system in the guest the virtual storage size and the number of CPUs will be reset to their normal settings.

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: Changing the kernel parameter line

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

Before your Linux kernel can use the DCSS, 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, 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`.

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

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

Task 7: Testing the DCSS

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

Before you can test the DCSS, you must change the kernel parameter line of your Linux as described in “Task 6: Changing the kernel parameter line” on page 13.

Perform the following steps to test the DCSS:

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

   ```
   # 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, verify that:

- The mem parameter is active and covers the entire DCSS. Issue the following command from CP to show the DCSS memory range:

  ```
  # query nss
  ```

- The virtual machine memory is smaller than the start address of the DCSS.
- You are running the correct kernel (see “Task 1: Building a Linux kernel with the required modules” on page 5)
- If you have opted to build the xip2 file system as a module also confirm that you have loaded the file system module.
- The file system size exceeds 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: Building a Linux kernel with the required modules” on page 5 through “Task 7: Testing the DCSS.”

Task 8: Activating execute-in-place

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

If all directories to be shared are located on the root file system, you just need the `xipinit` script in place as described in "Task 3: Providing a script to over-mount shared directories on startup" on page 7.

If you also want to share directories that are not on your root file system, you have to ensure that all directories 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. Test your `xipinit` script. As user root, issue: `/sbin/xipinit`. The output depends on the directories you are sharing. It looks similar to this 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 to be shared have been over-mounted with the xip2 file system. Depending on the directories you are sharing, the output should look similar to this example:

```
# 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` does not add the mounted directories to `/etc/fstab`, the **mount** command does not reflect these mounts properly!

   If your `xipinit` script does not work as intended, revisit "Task 3: Providing a script to over-mount shared directories on startup" on page 7.

3. Add `init=/sbin/xipinit` 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.

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 you cannot find a distribution that includes the xip2 file system support, build a Linux kernel that includes the support (see “Task 1: Building a Linux kernel with the required modules” on page 5).
2. Change the kernel parameter line (see “Task 6: Changing the kernel parameter line” on page 13).
3. Get a copy of `/sbin/xipinit` and activate it on your Linux (“Task 8: Activating execute-in-place” on page 15).

Updating the software on a DCSS

This task describes how you can perform minor software updates by writing to files on an existing DCSS file system.

Prerequisites:

- You need the DCSS block device driver, either as a module or built into the kernel.
- You need Class E user privileges in z/VM to save changes to a segment.

Restriction: The updated file system image must fit into the existing DCSS.

Tip: Instead of updating an existing DCSS by following the steps in this task, you can also replace your DCSS with an new DCSS.

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) instead. ext2 uses the DCSS block driver to access the file system in memory.

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

1. Ensure that the DCSS block device driver is available. Unless you have built the DCSS block device driver into your kernel, load the dcssblk module. Issue:

   ```
   # modprobe dcssblk
   ```

   Result: The driver has been loaded successfully if `cat /proc/devices` shows an entry “dcssblk”. The number preceding the 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 16 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
   # mknod /dev/dcssblk4 b <major number> 4
   # mknod /dev/dcssblk5 b <major number> 5
   # mknod /dev/dcssblk6 b <major number> 6
   # mknod /dev/dcssblk7 b <major number> 7
   # mknod /dev/dcssblk8 b <major number> 8
   # mknod /dev/dcssblk9 b <major number> 9
   # mknod /dev/dcssblk10 b <major number> 10
   # mknod /dev/dcssblk11 b <major number> 11
   # mknod /dev/dcssblk12 b <major number> 12
   # mknod /dev/dcssblk13 b <major number> 13
   # mknod /dev/dcssblk14 b <major number> 14
   # mknod /dev/dcssblk15 b <major number> 15
   ```

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

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

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

   The first DCSS you add becomes `/dev/dcssblk0`, the second `/dev/dcssblk1`, and so on. To view a list with the device number for each active segment issue:

   ```
   # cat /proc/dcssblk/list
   ```

   Adding a segment creates two entries in the /proc file system:

   - `/sys/devices/dcssblk/`<*name of DCSS*>`/shared`
   - `/sys/devices/dcssblk/`<*name of DCSS*>`/save`

   Example:

   ```
   # echo "LNXSHARE" > /sys/devices/dcssblk/add
   ```

   creates the two entries:

   - `/sys/devices/dcssblk/LNXSHARE/shared` and
   - `/sys/devices/dcssblk/LNXSHARE/save`

4. Set the appropriate access mode for the DCSS. There are two modes:

   shared

   If no Linux instance accesses the DCSS you can use the shared mode.

   In shared mode, the memory for the DCSS is allocated only once and changes are visible to guests as soon as they access it. Shared is the default.

   To switch to shared from exclusive-writable issue:

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

   exclusive-writable

   If there are Linux instances that use the DCSS, use the exclusive-writable mode.

   z/VM creates a private copy of the DCSS that is writable. Before changes can become effective they must be saved.

   To switch to exclusive-writable from shared issue:

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

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

   Example:

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

6. Update the data in the DCSS.
7. If you have made your updates in exclusive-writable mode, save the updates.

   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
```

Result: If no Linux instance is accessing the DCSS, the existing copy of the DCSS is discarded and replaced with the updated copy. If there are Linux instances that are accessing the existing copy, z/VM retains two copies of the DCSS until the last Linux instance has stopped using the older copy. Linux instances stop using the older DCSS when they unmount the xip2 file system. When they mount it again (for example, when rebooting Linux) they use the updated copy of the DCSS.

8. Unmount the file system.

   Example:

   ```
   # unmount /mnt
   ```

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

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

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 for zSeries and S/390
How to Improve the Performance of
Linux on z/VM
with Execute-in-Place Technology
July 30, 2004
Linux Kernel 2.6 (April 2004 stream)

Publication No. LNUX-H6DS-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

Company or Organization

Phone No.

Address

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

LNUX-H6DS-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

Cut or Fold
Along Line