Table of Contents

1. Dynamic management of DASD devices in Linux running on zSeries
2. Formatting and labeling a DASD volume for Linux guests running under z/VM
3. Linux on IBM zSeries: Configuring gcc as a cross-compiler
4. Partitioning DASD for Linux guests running under z/VM
5. Configuring Logical Volume Management (LVM) on Linux for zSeries
6. Avoiding the PROFILE EXEC when IPLing CMS
7. Creating a multi-call Linux binary
8. Creating a user home directory when you use LDAP authentication
9. The CP SET LOADDEV command
1. Dynamic management of DASD devices in Linux running on zSeries

Abstract

The DASD device driver for Linux on zSeries® uses a /proc file system interface to dynamically configure DASD devices. To simplify the interface and save typing keystrokes, try this script.

For related information about this topic, refer to the following IBM Redbooks publication: "Linux on IBM eServer zSeries and S/390: Large Scale Linux Deployment,” SG24-6824-00.

Contents

The DASD script is a wrapper for the /proc file system interface to the DASD device driver. It can help systems administrators dynamically manage DASD devices for Linux running on zSeries. Add the script to a directory in the PATH (e.g., /usr/local/bin).

The DASD command syntax is as follows.

To list all attached DASD devices:

dasd list

To add DASD device number devno, or devices in the range start to end:

dasd add devno | start-end

To disable DASD device number devno, or devices in the range start to end:

dasd off devno | start-end

To enable DASD device number devno, or devices in the range start to end:

dasd on devno | start-end

The following code is the DASD script:

#!/bin/sh
# dasd - simple utility for dynamic DASD management

if [ "$1" = "add" -a "$2" != "" ]; then
    echo "add range=$2" > /proc/dasd/devices
elif [ "$1" = "on" -a "$2" != "" ]; then
    echo "set device range=$2 on" > /proc/dasd/devices
elif [ "$1" = "off" -a "$2" != "" ]; then
    echo "set device range=$2 off" > /proc/dasd/devices
elif [ "$1" = "list" ]; then
    cat /proc/dasd/devices
else
    echo "Usage: dasd add|on|off vdev_or_range" 1>&2
    echo " dasd list" 1>&2
    exit 2
fi
2. Formatting and labeling a DASD volume for Linux guests running under z/VM

Abstract

z/VM® employs the concept of “minidisks,” whereby physical DASD can be split into smaller sizes, or minidisks, and assigned to individual VM guest systems. In this tip, we describe an approach for dedicating whole DASD volumes for Linux systems. These volumes are then split into minidisks owned by individual Linux guests.

Contents

If DASD has been “genned shared” in the zSeries IOCDS, it is visible to multiple logical partitions (LPARs) on the machine.

To distinguish Linux volumes from DASD on other zSeries operating systems, we recommend initializing and labeling all the Linux DASD with a volume serial that makes it obvious these are Linux disks. That way, if one or more of the disks is inadvertently varied online to another LPAR on the machine, it should be obvious that the disk belongs to the Linux environment.

In our examples, we use the following volume serial naming convention: V2LXnn. V2LX identifies these volumes as Linux volumes belonging to the VM2 LPAR, while nn is the sequence number (in hexadecimal) of the volume.

To format and label a DASD volume, follow the sequence of steps outlined below. User input is denoted in bold.

attach 3101 *
DASD 3101 ATTACHED TO MAINT 3101 WITH DEVCTL

cpfmtx
ENTER FORMAT, ALLOCATE, LABEL, OR QUIT:

format
ENTER THE VDEV TO BE PROCESSED OR QUIT:
3101
ENTER THE CYLINDER RANGE TO BE FORMATTED ON DISK 3101 OR QUIT:
0 0
ENTER THE VOLUME LABEL FOR DISK 3101:

v2lx01
CPFMTXA:
FORMAT WILL ERASE CYLINDERS 00000-00000 ON DISK 3101
DO YOU WANT TO CONTINUE? (YES | NO)

yes
HCPCCF6209I INVOKING ICKDSF.

........
ICK00002I ICKDSF PROCESSING COMPLETE. MAXIMUM CONDITION CODE WAS 0
ENTER ALLOCATION DATA
TYPE CYLINDERS

............... perm 0-end
end

........
ICK00001I FUNCTION COMPLETED, HIGHEST CONDITION CODE WAS 0

Begin by attaching the DASD device to the VM guest. Next, invoke the cpfmtx utility to format the volume. The utility prompts for the device number to format (3101, in our case). Specifying a cylinder range of 0 0 will format cylinder 0. The remaining cylinders will be formatted by Linux using the dasdfmt
Upon completion, the DASD volume is available to be partitioned and formatted by Linux guests.

3. Linux on IBM zSeries: Configuring gcc as a cross-compiler

Abstract

This tip details how to configure gcc on Linux for IBM zSeries to act as a cross-compiler.

For related information about this topic, refer to the following IBM Redbooks publication: "Linux on IBM eServer zSeries and S/390: Application Development," SG24-6807-00.

Contents

By default, gcc compiles code for the same type of machine that it runs on. However, it may also be configured as a "cross-compiler". A cross-compiler allows you to create programs that can be run on architectures different than the one used for a compilation. Configuring a cross-compiler consists of the following steps:

- Downloading and installing prerequisites
- Compiling binutils
- Installing libraries
- Compiling the cross-compiler

The compiler and utilities are in the /usr/cross-devel/ directory. To use the compiler, you need a full development environment on your machine; make sure at least the following packages are present:

- gcc-compiler and standard C/C++ development libraries
- binutils package
- autoconf (provides autoheader utility)
- yacc and flex packages

Before you compile the binutils package, you must obtain the source code, so install the binutils.spm package provided by SuSE distribution. The source ball will be extracted to the /usr/src/packages/SOURCE directory. Unpack this to a temporary directory:

bunzip2 < /usr/src/packages/SOURCES/binutils-2.11.90.0.27.tar.bz2 |tar -C /tmp -xf -

Change your working directory to the binutils source and run configure:

cd /tmp/binutils-2.11.90.0.27
./configure --prefix=/usr/cross-devel --target=i386-pc-linux s390-ibm-linux

Start the compilation and install the binaries to the /usr/cross-devel directory:

make; mkdir /usr/cross-devel; make install
Installing libraries

To produce executable files, you need to install libraries precompiled for a particular architecture. Compiling glibc from source is possible, but we suggest using the precompiled version that is commonly used on Linux for i386. You can download glibc from the following site:

http://www.rpmfind.net

Look for the version closest to that installed on your platform:

```
rpm -q glibc
glibc-2.2.2-25
rpm -q glibc-devel
glibc-devel-2.2.2-25
```

In this example, we used version 2.2.2-38 (part of the SuSE distribution for Linux on i386). We needed both glibc.rpm and glibc-devel.rpm. Include files and libraries should be installed in the directories with names that begin with `compiler_prefix/architecture`. In this case, we used:

```
/usr/cross-devel/i386-pc-linux/include
```

and

```
/usr/cross-devel/i386-pc-linux/lib
```

These library packages cannot be installed using the rpm tool (this would conflict with existing libraries). Instead, use the following procedure.

Create a temporary directory:

```
mkdir /tmp/i386
```

Unpack libraries with cpio:

```
 cd /tmp/i386
 rpm2cpio ../glibc.rpm | cpio -id
 rpm2cpio ../glibc-devel.rpm | cpio -id
```

Move the include part to `/usr/cross-devel/i386-pc-linux`. *(Note: You don’t have to create `/usr/cross-devel/i386-pc-linux`; it is created during installation of the binutils package).*

```
 cd /tmp/i386/usr
 cp -a include /usr/cross-devel/i386-pc-linux
```

Move the contents of lib and usr/lib directories to the `/usr/cross-devel/i386-pc-linux/lib` directory:

```
 cd /tmp/i386
 cp -a lib /usr/cross-devel/i386-pc-linux
 cd usr
 cp -a lib /usr/cross-devel/i386-pc-linux
```

We merged files from two directories. Now we have to fix some symbolic links in `/usr/cross-devel/i386-pc-linux/lib`:

```
 cd /usr/cross-devel/i386-pc-linux/lib
```
We need to remove the prefix `../../lib/` from symbolic links. *(Note: Type in this long command carefully; it is a single command).*

```bash
ls -l | grep ../../lib | cut -c57- | cut -d' ' -f1,3 | \
  sed 's^../../lib/^\n' | \n  while read a b ; do rm -f $a ; ln -s $b $a ; done
```

Correct `libc.so` accordingly (this is a plain text file):

```c
/* GNU ld script
Use the shared library. Some functions are only in
the static library, so try that secondarily.
*/
```

**Compiling the cross-compiler**

The `gcc.spm` package that comes with SuSE Linux 7.2 for zSeries contains the main source `gcc-2.95.3.tar.gz` and patches specific for S/390. Look for the following:

- `gcc-2.95.3-s390.tar.gz`
- `gcc-2.95.3-s390-1.tar.gz`
- `gcc-2.95.3-s390-2.tar.gz`

Next is how to prepare the source code. Unpack the main source package:

```bash
tar -C /tmp -zxf /usr/src/packages/SOURCES/gcc-2.95.3.tar.gz
```

Unpack patches in the temporary directory:

```bash
mkdir /tmp/gcc-patches
cd /tmp/gcc-patches
tar -zxf /usr/src/packages/SOURCES/gcc-2.95.3-s390.tar.gz

tar -zxf /usr/src/packages/SOURCES/gcc-2.95.3-s390-1.tar.gz

tar -zxf /usr/src/packages/SOURCES/gcc-2.95.3-s390-2.tar.gz
```

Apply the patches:

```bash
cd /tmp/gcc-2.95.3
patch -p1 < /tmp/gcc-patches/gcc-2.95.3-s390.diff
patch -p1 < /tmp/gcc-patches/gcc-2.95.3-s390-1.diff
patch -p1 < /tmp/gcc-patches/gcc-2.95.3-s390-2.diff
```

Configure `gcc` as a cross-compiler:

```bash
./configure --prefix=/usr/cross-devel \ 
  --with-headers=/usr/cross-devel/i386-pc-linux/include/ \ 
  --target=i386-pc-linux s390-ibm-linux
```

*(Note: We found that the `--with-headers` option is essential to avoid specifying the `-I` option when working with the cross-compiler later on).*
Compile:

make

(Note: In some configurations, the PATH_MAX constant is not correctly defined properly). If you encounter this problem, add the following lines after the include section:

```c
#ifndef _POSIX_PATH_MAX
#define _POSIX_PATH_MAX 255
#endif
```

Install package:

make install

Using the cross-compiler

Once the compiler has been installed, update your PATH environment variable:

```bash
export PATH=/usr/cross-devel/bin:$PATH
```

Executable files for the i386 platform are created when `gcc` is invoked with a proper prefix. In our example, we used the `i386-pc-linux-gcc` command. We suggest setting the CC variable to this command to instruct `make` which compiler to use. For example, to make an executable test from source file `test.c`, issue the following:

```bash
$ export CC=i386-pc-linux-gcc
$ make test
i386-pc-linux-gcc t.c -o t
$ file test
test: ELF 32-bit LSB executable, Intel 80386, version 1, dynamically linked (uses shared libs), not stripped
```

4. Partitioning DASD for Linux guests running under z/VM

Abstract

Before DASD can be used by Linux on zSeries, the DASD volume must be partitioned and formatted by Linux. This tip describes the necessary commands to partition and format DASD.

Contents

Before a DASD volume can be accessed from Linux on zSeries, the volume must be formatted and partitioned by Linux. Use the `dasdfmt` command to format the DASD; use the `fasd` command to partition the DASD device.

In the following example, we first format DASD device `/dev/dasda`:

```bash
# dasdfmt -f /dev/dasda -b 4096 -p
Drive Geometry: 3338 Cylinders * 15 Heads = 50070 Tracks
```

Then we format the device `/dev/dasda` in the following way:
Device number of device : 0x201
Labelling device : yes
Disk label : VOL1
Disk identifier : 0X0201
Extent start (trk no) : 0
Extent end (trk no) : 48419
Compatible Disk Layout : yes
Blocksize : 4096

--- ATTENTION! ---
All data of that device will be lost.
Type "yes" to continue, no will leave the disk untouched: yes
Formatting the device. This may take a while (get yourself a coffee).

cyl 3338 of 3338 |##################################################| 100%

Finished formatting the device.
Rereading the partition table... ok

The -f /dev/dasda parameter specifies that the /dev/dasda device is to be formatted. A block size of 4096 bytes is used when formatting with the -b 4096 parameter. The -p specifies that a progress bar is to be displayed.

Once formatted, the DASD device can be partitioned using the fasd command, as shown:

# fasd -a /dev/dasda
auto-creating one partition for the whole disk...
writing volume label...
writing VTOC...
rereading partition table...

This causes the command to operate non-interactively; a single partition will be created for the entire device.

5. Configuring Logical Volume Management (LVM) on Linux for zSeries

Abstract

Minidisks on z/VM cannot span more than one physical DASD volume. Without a volume management system, this limits the size of a file system on Linux for zSeries to the size of a DASD volume. For a 3390 model 3, this translates to 2.3GB of available space for a Linux file system.

Volume management systems allow multiple DASD devices to be combined into a single Linux file system (and thereby overcome the single DASD volume size limit). This tip describes how to configure Logical Volume Management (LVM) for Linux on zSeries.

Contents

LVM concepts

LVM has its own terminology, which we briefly describe here:
A DASD volume is called a physical volume (PV), because that’s the volume where the data is physically stored. The PV is divided into several physical extents (PE) of the same size. The PEs are like blocks on the PV. Several PVs make up a volume group (VG), which becomes a pool of PEs available for the logical volume (LV). The LVs appear as normal devices in /dev/ directory. You can add PVs to or delete PVs from a VG, and increase or decrease your LVs.

LVM can be exercised either from the command line or by using the setup tool YaST. Note, however, that if the YaST implementation does not complete successfully, your new logical volume could be left in an inconsistent state that may be difficult to correct. Also note that YaST with SLES-7 does not recognize DASD device nodes beyond /dev/dasdz1. Therefore, we will describe the LVM line commands to use.

Preparing DASD devices for LVM

Before the physical volumes can be created, they must be formatted and partitioned for zSeries Linux. This is done using the dasdfmt and fdasd commands.

To see the DASD devices recognized by Linux, issue the following command:

```
# cat /proc/dasd/devices
```

```
0201(ECKD) at ( 94: 0) is dasda : active at blocksize: 4096, 36000 blocks, 140 MB
0202(ECKD) at ( 94: 4) is dasdb : active at blocksize: 4096, 564840 blocks, 2206 MB
0203(ECKD) at ( 94: 8) is dasdc : active at blocksize: 4096, 36000 blocks, 140 MB
0204(ECKD) at ( 94: 12) is dasdd : active at blocksize: 4096, 36000 blocks, 140 MB
```

To format a single DASD device, a block size of 4 KB is recommended. Use the following command:

```
# dasdfmt -b 4096 -f /dev/dasdc
```

Drive Geometry: 200 Cylinders * 15 Heads = 3000 Tracks

We will format the device /dev/dasdc in the following way:

```
Device number of device : 0x203
Labelling device : yes
Disk label : VOL1
Disk identifier : 0X0203
Extent start (trk no) : 0
Extent end (trk no) : 2999
Compatible Disk Layout : yes
Blocksize : 4096
---> ATTENTION! <<<
All data of that device will be lost.
Type "yes" to continue, no will leave the disk untouched: yes
Formatting the device. This may take a while (get yourself a coffee).
Finished formatting the device.
Rereading the partition table... ok
```

Answer yes to the query.

To create a single partition from a formatted DASD, use the -a flag of the fdasd command. If the -a flag is omitted, an interactive formatting session similar to the PC fdisk command is invoked, as follows:

```
# fdasd -a /dev/dasdc
```
auto-creating one partition for the whole disk...
writing volume label...
writing VTOC...
rereading partition table…

Repeat the **dasdfmt** and **fdasd** commands for each DASD device to be used by LVM.

### Setting up LVM

Once the DASD devices have been formatted and partitioned, LVM can be installed on them. Use the following commands to create a logical volume consisting of the `/dev/dasdc1` and `/dev/dasdd1` devices.

Create physical volumes for each DASD that will be included in the logical volume. This can be done for all DASD by using one command, or one at a time by using multiple **pvcreate** commands, as follows:

```
# pvcreate /dev/dasdc1 /dev/dasdd1
pvcreate -- physical volume "/dev/dasdc1" successfully created
pvcreate -- physical volume "/dev/dasdd1" successfully create
```

Show the currently defined physical volumes by using the **pvscan** command:

```
# pvscan
pvscan -- reading all physical volumes (this may take a while...)
pvscan -- inactive PV "/dev/dasdc1" is in no VG [140.53 MB]
pvscan -- inactive PV "/dev/dasdd1" is in no VG [140.53 MB]
pvscan -- total: 2 [281.06 MB] / in use: 0 [0] / in no VG: 2 [281.06 MB]
```

Create the volume group by using the **vgcreate** command. All physical volumes to be included in the group must be specified in the command, as follows:

```
# vgcreate lvmdata /dev/dasdc1 /dev/dasdd1
vgcreate -- INFO: using default physical extent size 4 MB
vgcreate -- INFO: maximum logical volume size is 255.99 Gigabyte
vgcreate -- doing automatic backup of volume group "lvmdata"
vgcreate -- volume group "lvmdata" successfully created and activated
```

Display the volume group just created by using the **vgdisplay** command, as follows:

```
# vgdisplay /dev/lvmdata
--- Volume group ---
VG Name lvmdata
VG Access read/write
VG Status available/resizable
VG # 0
MAX LV 256
Cur LV 0
Open LV 0
MAX LV Size 255.99 GB
Max PV 256
Cur PV 2
Act PV 2
VG Size 272 MB
PE Size 4 MB
Total PE 68
Alloc PE / Size 0 / 0
Free PE / Size 68 / 272 MB
VG UUID fxozbU-5VqJ-v5FG-LfIV-JcyK-1Jbh-KHXd2N
```
The reported VG SIZE is used in the next step. Create a logical volume using the VG SIZE noted above:

```
# lvcreate -L 272M -n vol1 lvmdata
lvcreate -- doing automatic backup of "lvmdata"
lvcreate -- logical volume "/dev/lvmdata/vol1" successfully created
```

Display the logical volume just created by using the `lvdisplay` command, as follows:

```
# lvdisplay /dev/lvmdata/vol1
--- Logical volume ---
LV Name /dev/lvmdata/vol1
VG Name lvmdata
LV Write Access read/write
LV Status available
LV # 1
# open 0
LV Size 272 MB
Current LE 68
Allocated LE 68
Allocation next free
Read ahead sectors 1024
Block device 58:00
```

**Creating a file system on the logical volume**

After the logical volume is created, a file system can be installed on it. For the Reiser file system, use the `mkreiserfs` command:

```
# mkreiserfs /dev/lvmdata/vol1
```

The file system can now be mounted, as follows:

```
# mkdir /data
# mount /dev/lvmdata/vol1 /data
```

### 6. Avoiding the PROFILE EXEC when IPLing CMS

**Abstract**

When logging on to CMS, the PROFILE EXEC found on the user's 191 A-disk is executed on CMS IPL. If a statement in the PROFILE EXEC causes a fatal CMS error, the user will be thrown into CP.

To correct the error, the PROFILE EXEC will need to be edited, typically using XEDIT. However, XEDIT only runs under CMS, leading to a dilemma: the PROFILE EXEC cannot be edited because CMS cannot be IPLed.

To resolve this problem, it is possible to IPL CMS without executing the user PROFILE EXEC. This tip shows how to do this.

For related information about this topic, refer to the following IBM Redbooks publication: "Linux on IBM eServer zSeries and S/390: Large Scale Linux Deployment," SG24-6824-00.
Most VM user IDs are set up to automatically start CMS when the user ID logs on to VM. Any time CMS is loaded (via the IPL CMS command), a special routine called PROFILE EXEC is automatically run. The PROFILE EXEC can contain a set of CP and CMS commands that define the current virtual machine and CMS environments. Many Linux guest virtual machines have a PROFILE EXEC that defines a working environment and then issues a CP IPL of the Linux machine. This makes the boot of a Linux guest nearly invisible to the end user.

There may be times, however, when you do not want the PROFILE EXEC and simply want to run in CMS. If you want to suppress the automatic invocation of the PROFILE EXEC, the first command you enter after the IPL CMS command is the CMS ACCESS command with the NOPROF option. For example, enter:

ipl cms

The system response may be:

z/VM V4.4.0 2002-07-11 10:58

To suppress the execution of your PROFILE EXEC, enter:

access (noprof

When the system responds with:

Ready;

...you have loaded CMS and accessed filemode A without running your PROFILE EXEC.

7. Creating a multi-call Linux binary

Abstract

A multi-call binary is an executable, written in C, that performs the action of more than one utility. A prime example of a multi-call binary is the BusyBox package. BusyBox implements a large number of standard Linux utilities (such as the ls and ln commands) in a single executable. This enables a reduced size for specialized Linux distributions. This tip describes how multi-call binaries are written.

The BusyBox package is one of the best examples of a multi-call binary. This concept allows a single executable file to perform the function of dozens of different utilities usually packaged as separate files. Multi-call binaries exploit a number of operating system features so that a system user is not even aware that the programs they are running are all the same file. There are two ways to invoke BusyBox functions.

In the first method, you issue the BusyBox command followed by the name of the function you want to issue. For example, the command BusyBox ls would perform the directory list function (equivalent to the usual ls command). This method requires no administration, but users of the program need to remember that they cannot perform a function simply by issuing the name of a command.
The second method is to create a set of symbolic links to the BusyBox executable, each with the name of a function implemented by BusyBox. When BusyBox is run, it checks the name by which it was invoked, and uses that name as the function to be executed. This method does require some administration, as the symbolic links must be maintained, but system users can follow the normal practice of performing a function by issuing the name of the command.

To illustrate, the following sequence shows the content of a BusyBox /bin directory and the effect of issuing the `ls` command:

```
# pwd
/bin
# ls -l
lrwxrwxrwx 1 root root 12 Oct 2 00:11 ln -> /bin/busybox
lrwxrwxrwx 1 root root 14 Oct 2 00:11 login -> /bin/tinylogin
lrwxrwxrwx 1 root root 12 Oct 2 00:11 ls -> /bin/busybox
# ls -G
ls: invalid option -- G
BusyBox v0.60.3 (2002.09.26-00:58+0000) multi-call binary
Usage: ls [-AacCdEfnlpRStuvwxXhk] [filenames...]
```

List directory contents

In the first output, you can also see `login`, which is a symbolic link to `/bin/tinylogin`. TinyLogin is a partner program to BusyBox and performs program functions such as `login` and `sulogin`. These functions can be implemented in BusyBox, but for security reasons it is preferred to have a separate executable for login processing.

This example also shows us another feature of the BusyBox utility. In the full GNU implementation of `ls`, the `-G` option is valid (it suppresses the display of the group name from the directory list). In the interests of saving space, however, not all of the functions of the various utilities are provided. This is quite appropriate for BusyBox since the idea is to eliminate unused (or little used) functions in the interests of reducing the executable size.

So, how does a multi-call binary like BusyBox, when invoked using a symbolic link, know what function to perform? The answer is, a multi-call binary program is written differently than a normal program.

The C language is used for most systems programming on UNIX/POSIX systems. Programs written in C always have a `main()` function, which is the first part of the program to be executed. The main function is written in a particular way to allow the operating system to pass parameters to it. A typical `main()` function declaration appears here:

```
int main(int argc, *char argv[])  
```

The parameters passed to the `main()` function are `argc`, an integer containing the number of parameters passed by the system to the program, and `argv`, the list of the parameters passed. By convention (on UNIX/POSIX systems), there will always be at least one parameter passed to the program: the name used to invoke the program. This is usually the command typed by the user at the shell prompt to invoke the command, and will just about always be the name of the file that contains the program. In C notation, this value (the first item in the array called `argv`) is `argv[0]`.

Most single call binaries ignore the contents of `argv[0]`, as the program is designed to perform a single task and it is irrelevant what name the system used to invoke the program. For security reasons, some programs ensure that the command issued is correct. This can prevent a malicious user from executing a program they should not have access to.
A multi-call binary pays attention to this parameter and uses it to determine which function to execute. In the case of BusyBox, if argv[0] is the same as the executable file name, it will use the second item in the parameter list (argv[1]) as the name of the function to be executed. If argv[0] is not the same as the name of the BusyBox executable file, it will attempt to use the contents of argv[0] as the name of the requested function.

8. Creating a user home directory when you use LDAP authentication

Abstract

Lightweight Directory Access Protocol (LDAP) allows a system administrator to centrally define and manage Linux users. Using LDAP, an administrator can define a user to many Linux systems. User information, such as the user password and location of the user's home directory, is stored in the LDAP directory rather than on the local Linux system. This technote describes how to configure the Pluggable Authentication Module (PAM) to automatically create a user home directory the first time a user logs on.

For related information about this topic, refer to the following IBM Redbooks publication: “Linux on IBM eServer zSeries and S/390: Best Security Practices,” SG24-7023-00.

Contents

Linux users usually have their home directory (typically the /home/userid directory) created when the user ID is defined. When using LDAP authentication, users are remotely defined (and have no home directory created on the local host). Two possibilities exist to avoid manual creation of a home directory for each LDAP defined user:

- The user home directory can be located on a network file server (for example, an NFS-mounted file system)
- The home directory can be automatically created when a user first logs in

For login services, except SSH, add the pam_mkhomedir.so module to the PAM configuration file for the service, /etc/pam.d/login. For example:

```
session required pam_mkhomedir.so skel=/etc/skel/ umask=0077
```

If a user authenticates and no home directory exists, the home directory is created in /home. The umask=0077 parameter causes the directory permission to be set to 700. The home directory is constructed from the skeletal files found in the /etc/skel directory.

Beginning with OpenSSH Version 3.3, automatic creation of a user home directory using pam_mkhomedir.so is no longer supported due to a security modification in SSH. You can use the make_home_dir replacement for pam_mkhomedir.so. The make_home_dir package is available at:

http://www.trustsec.de/soft/oss
9. The CP SET LOADDEV command

Abstract

This technote describes the new CP SET LOADDEV command used to IPL a Linux guest from FCP-attached SCSI.

For related information about this topic, refer to the following IBM Redbooks publication: "Linux on zSeries: Fibre Channel Protocol Implementation Guide," SG24-6344-00.

Contents

The new CP SET LOADDEV command is used to provide the parameters that are needed to access a SCSI disk to the machine loader. Parameters to SET LOADDEV include the WWPN and LUN number of the SCSI disk on which Linux resides. For example, to IPL Linux on the SCSI disk that is located at WWPN 5005076300CD9589 and LUN 5303000000000000, use the command:

```
SET LOADDEV PORTNAME 50050763 00CD9589 LUN 53030000 00000000
```

The CP QUERY LOADDEV command reports parameters set for the machine loader:

```
Q LOADDEV
PORTNAME 50050763 00CD9589 LUN 53030000 00000000 BOOTPROG 0
BR_LBA 00000000 00000000
```

For details on the SET LOADDEV and QUERY LOADDEV commands, consult "CP Command and Utility Reference," SC24-6008.

The LOADDEV user directory statement

It is possible to set the machine loader parameters in the user directory entry using the LOADDEV statement. Here, we set the LOADDEV WWPN and LUN parameters using DIRMAINT:

```
DIRM LOADDEV PORTNAME 5005076300CD9589
DVHXTMT1191I Your LOADDEV request has been sent for processing.
Ready; T=0.04/0.05 09:41:48
DVHREQ2288I Your LOADDEV request for LNXSU5 at * has been accepted.
DVHBIU3450I The source for directory entry LNXSU5 has been updated.
DVHBIU3424I The next ONLINE will take place immediately.
DVHBIU3428I Changes made to directory entry LNXSU5 have been placed
DVHBIU3428I online.
DVHREQ2289I Your LOADDEV request for LNXSU5 at * has completed; with RC =
DVHREQ2289I 0.
```

```
DIRM LOADDEV LUN 5303000000000000
DVHXTMT1191I Your LOADDEV request has been sent for processing.
Ready; T=0.04/0.05 09:42:14
DVHREQ2288I Your LOADDEV request for LNXSU5 at * has been accepted.
DVHBIU3450I The source for directory entry LNXSU5 has been updated.
DVHBIU3424I The next ONLINE will take place immediately.
DVHBIU3428I Changes made to directory entry LNXSU5 have been placed
DVHBIU3428I online.
DVHREQ2289I Your LOADDEV request for LNXSU5 at * has completed; with RC =
DVHREQ2289I 0.
```
The resulting directory entry is:

USER LNXSU5 LNXSU5 128M 1G G
INCLUDE IBMDFLT
IPL 190 PARM AUTOCR
LOADDEV PORTNAME 5005076300CD9589
LOADDEV LUN 5303000000000000
MACHINE XA
DEDICATE 7104 7138
DEDICATE 7105 7139
DEDICATE 7106 713A
DEDICATE 1605 1605
LINK TCPIP 0592 0592 RR
MDISK 0191 3390 175 25 440U1R

For details on the LOADDEV user directory statement, consult “CP Planning and Administration,” SC24-6043. For details on using DIRMAINT, see “Directory Maintenance Facility Command Reference,” SC24-6025.

Special Notices
This material has not been submitted to any formal IBM test and is published AS IS. It has not been the subject of rigorous review. IBM assumes no responsibility for its accuracy or completeness. The use of this information or the implementation of any of these techniques is a customer responsibility and depends upon the customer's ability to evaluate and integrate them into the customer's operational environment.

© Copyright 2004 IBM Corporation.

IBM Corporation, Software Group, 100 North Sepulveda Boulevard, El Segundo, California 90245 U.S.A.

All rights reserved.

IBM, the IBM logo, S/390, zSeries and z/VM are registered trademarks of International Business Machine Corporation in the United States, other countries or both. Other company, product and services names may be trademarks or service marks of others.