Redpaper

Steve Womer
Rick Troth
Mike MacIsaac

Sharing and maintaining Linux under z/VM

Large operating systems, such as z/OS®, have, for several decades, taken advantage of *shared file structures*. The benefits of a shared file structure are reduced disk space, simplified maintenance, and simplified systems management. This IBM® Redpaper describes how to create a Linux® solution with shared file systems on IBM System z™ hardware (the mainframe) running under z/VM®. It also describes a maintenance system where the same Linux image exists on a test, maintenance and *gold* virtual servers. The benefits of such a system are:

- Extremely efficient resource sharing, which maximizes the business value of running Linux on System z
- Staff productivity because fewer people are needed to manage a large-scale virtual server environment running on z/VM
- Operational flexibility because companies can take advantage of and use their IT infrastructure to enhance business results

Note: A word of caution and a disclaimer are necessary. The techniques that we describe in this paper are not simple to implement and require both z/VM and Linux on System z skills. Further, it is not guaranteed that such a system will be supported. Therefore, you need to check with your Linux distributor and your support organization to verify that the changes that we describe in this paper will be supported. This being said, this paper is based on a system that was implemented and is in production at Nationwide Mutual Insurance Company.

This paper is divided into the following parts:

- "Read-only root Linux" on page 2 describes the shared root file structure and the maintenance system.
- "Building a read-write maintenance system" on page 18 describes how to create the maintenance system using conventional Linux images with read-write directories.
- "Building a read-only root system" on page 35 describes how to create Linux systems with only certain file systems read-write. Most are read-only, including the root file system.
- "Contents of tar file" on page 46 lists all the Linux scripts, z/VM REXX™ EXECs, and configuration files that are available in the tar file that is associated with this paper.

This paper is based on z/VM Version 5.3 and Novell SUSE Linux Enterprise Server 10.

Read-only root Linux

The system is called *read-only root* because the root file system (/) is mounted with read permission, not read-write.

Why a read-only root file system

By creating a read-only root file structure, the basic Linux system code can be shared among many virtual Linux servers, which helps with Linux standardization. Many Linux servers will share exactly the same version of the Linux operating system. This environment makes maintenance much simpler. With this environment, it becomes possible to roll out a new version of Linux by updating the master copy of the shared root instead of updating each system that uses it.

Overview of the system

The root file system is read-only because it does not have to be read-write. Actually, four directories are chosen to be read-write:

- /etc/
- /var/
- /srv/
- /root/

In addition, the /tmp/ directory is read-write, but it is in-memory and is built at boot time. The /proc/ and /sys/ psuedo-directories are abstractions of kernel control blocks, and the permissions are not under the control of the systems administrator.

During the boot process, read-write copies of the directories /etc/, /var, and /srv/ are bind-mounted from /local/ over the read-only copies, while /var/ is its own minidisk.

Figure 1 shows a block diagram of the entire system. The boxes above the dashed line represent a maintenance system for conventional Linux servers with most file systems being read-write. The boxes below the line represent the read-only root portion of the solution.

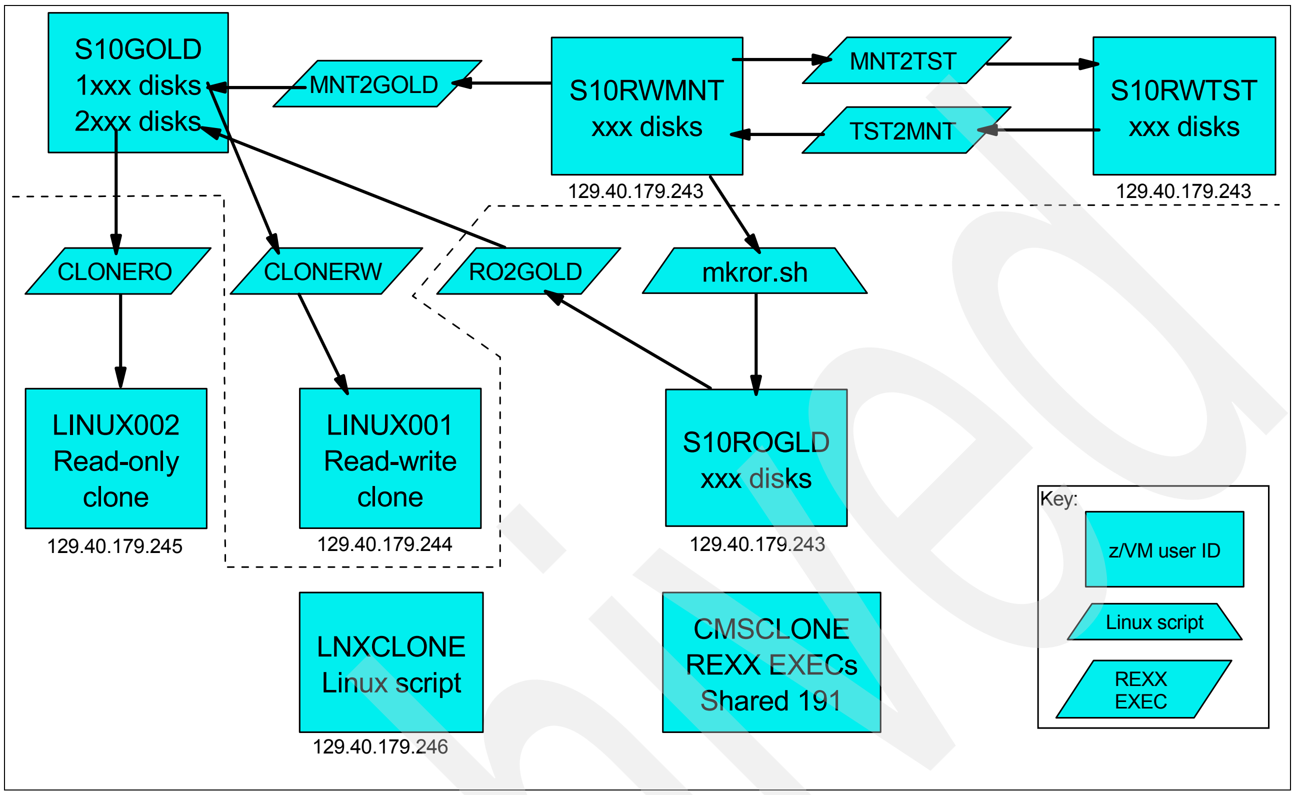


Figure 1 Block diagram of read-only root system

The names of the z/VM user IDs convey their function:

S10GOLD	The SLES 10 golden system
S10RWMNT	The SLES 10 read-write system on which maintenance is done
S10RWTST	The SLES 10 read-write system on which testing is done
S10ROGLD	The user ID that will have a read-only root
LNXCLONE	A user ID to run a Linux script to clone a read-only root system
CMSCLONE	A user ID to run REXX cloning EXECs (CMS files)
LINUX001	The first cloned Linux
LINUX002	The second cloned Linux

Similarly, the REXX EXEC and script names are named intuitively.

High-level approach of read-only root system

In this section, we describe the read-only root solution at a high level. In addition to the read-only root environment, we also establish a maintenance process. You can find details about implementing the solution in “Building a read-write maintenance system” on page 18.

SLES 10 Linux is installed on an initial virtual server with the z/VM user ID *S10RWMNT* (as shown in Figure 2). Packages to create a minimal Linux system are selected, security hardening done, and other configuration changes are applied. Table 1 on page 15 describes the minidisks for this server.

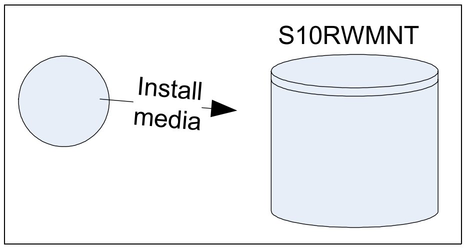


Figure 2 Installing the first Linux system

After Linux is installed on the initial server (S10RWMNT), it is shut down and copied to a test server, *S10RWTST* (see Figure 3). Linux is configured on this user ID where all features and functions are tested and validated. After the system is validated, it is copied in the reverse direction.

Note: The IP address and host name are not changed. Therefore, only one user ID can have Linux running at any time.

If changes on the test system are not valid, they can be discarded by recopying the maintenance system.

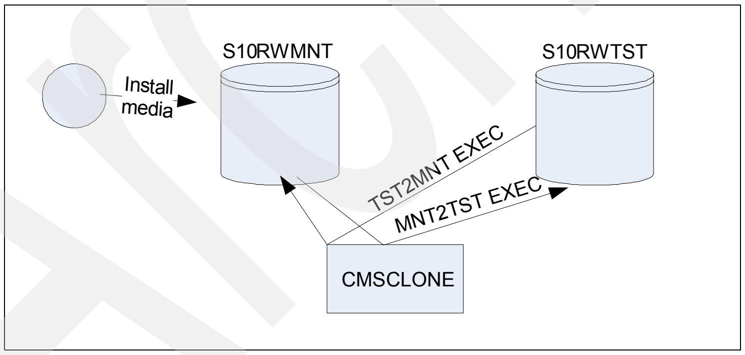


Figure 3 Cloning the first Linux system

When configuration and testing are completed, the read-write Linux system is frozen to become the source disks for Linux cloning. This copy is called the *gold* version of the Linux minidisks. This whole system is shown in Figure 4. A Linux script (**`cloneprep.sh`**) is written to clean up the system before cloning. After running the cleanup script, the system is shut down, and then the disks are copied to the gold user ID using a REXX EXEC.

At Nationwide, three versions of the gold minidisks are maintained:

- Old
- Current
- Next

This paper only addresses the current copy. To add additional versions, such as old and next, you would need to create more sets of minidisks on the gold user ID, with an agreed upon device numbering convention.

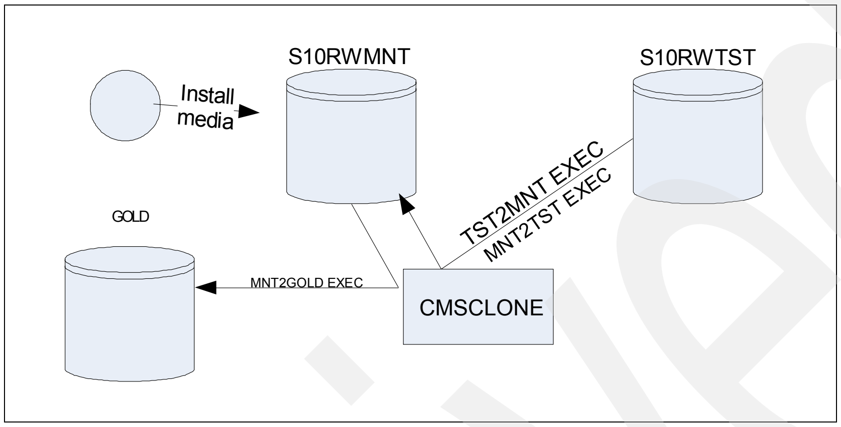


Figure 4 Freezing a golden copy of a Linux system

When a new Linux server needs to be cloned or provisioned with conventional read-write disks, it is copied from the gold minidisks. An REXX EXEC is run to copy from the gold minidisks to a target Linux user ID (Figure 5).

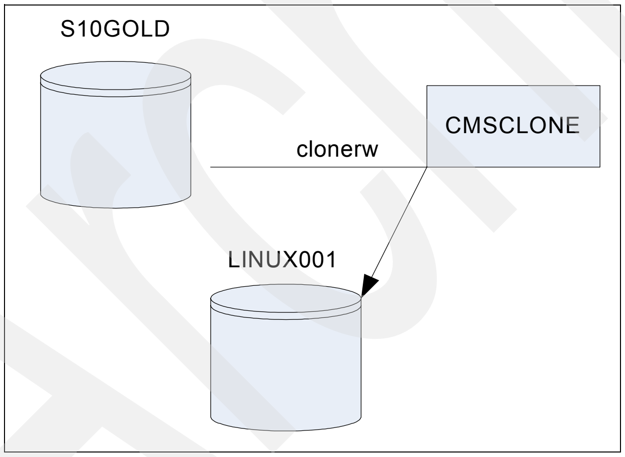


Figure 5 Cloning a read-write virtual server

A script (**`boot.findself`**) is also copied that is designed to run once at boot time to uniquely configure the new Linux server with the correct IP address and host name. This script will not modify the IP address and host name of the predefined S10xxxxx user IDs. They, they all share the same values. As such, only one of these user IDs can have Linux IPLed at any given time.

After a read-write system has been created on S10RWMNT, a read-only root system can be created from it. Another Linux system running on CMSCLONE is used to build the read-only root system. A Linux script (**`mkror.sh`**) is provided to create the first read-only root system (Figure 6). The read-only root server, S10ROGLD, is used to test and create the gold version of minidisks, which is used for read-only root provisioning.

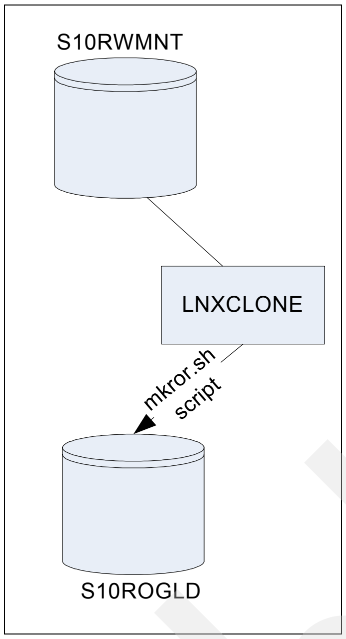


Figure 6 Creating a read-only root system with the mkror.sh script

Then, the cleanup script is run on the first read-only root Linux system, and the system is shut down. Now, a REXX EXEC (RO2GOLD) is executed to copy from S10ROGLD to the 2xxx minidisks on S10GOLD. These are now the gold read-only root disks.

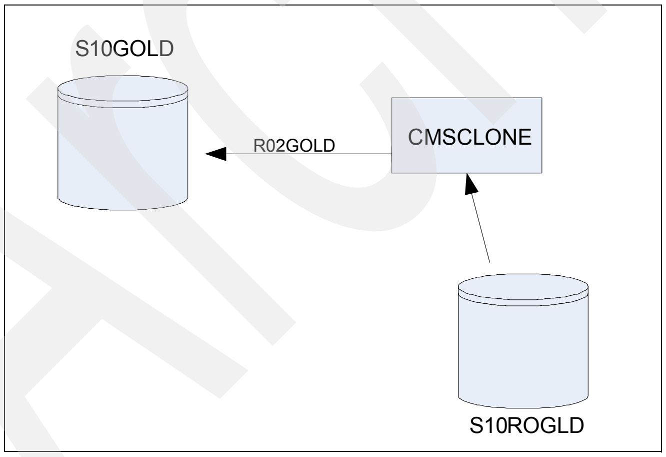


Figure 7 Freezing the read-only root system to S10GOLD

To clone a read-only root system, the process is similar, except that the target user ID is defined to have three read-write minidisks and four read-only links to the gold disks. Then, the CLONERO EXEC only has to copy three minidisks, not seven.

Directory structure of the read-only root system

Figure 8 shows the directory structure for the Linux shared read-only root file system.

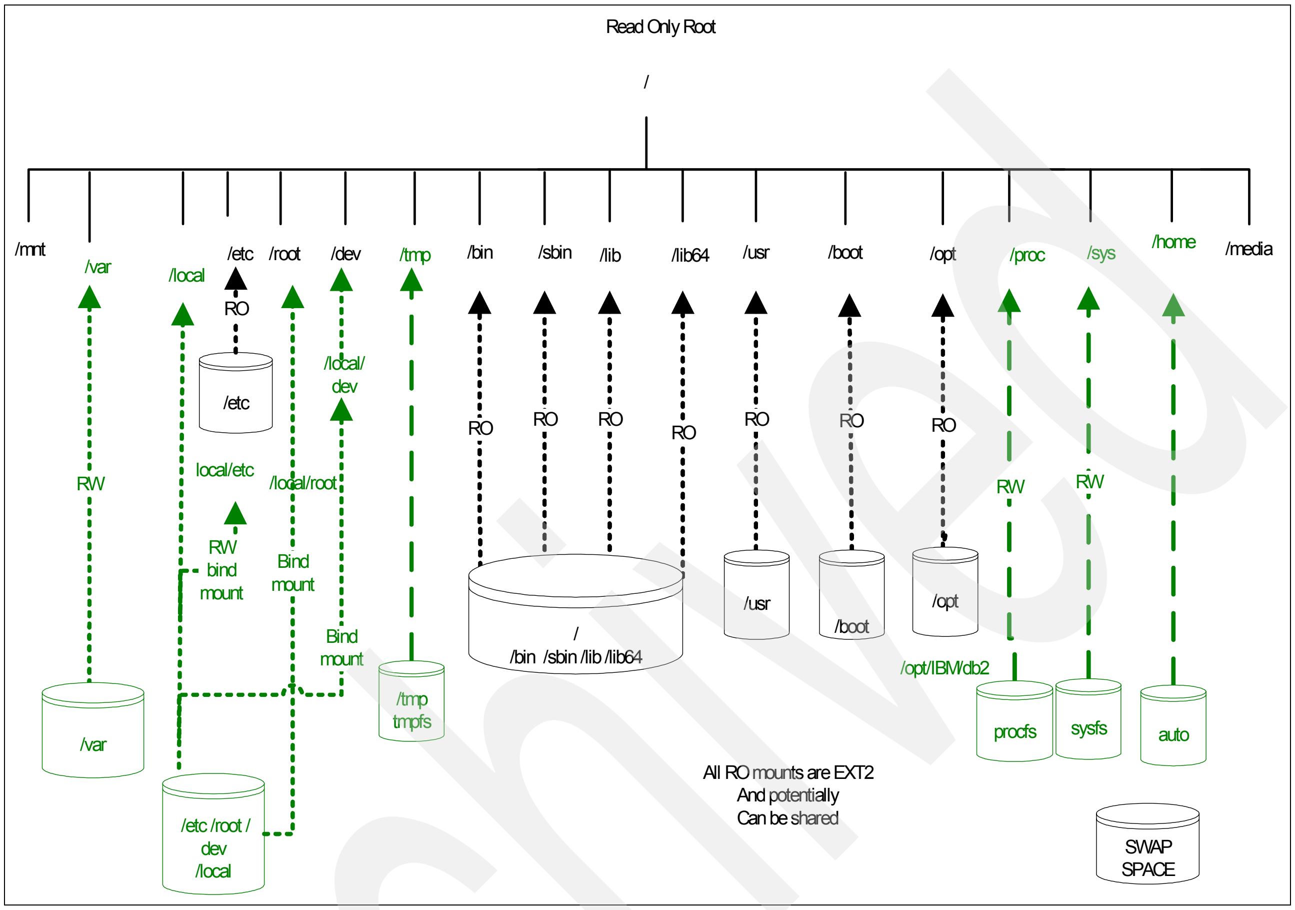


Figure 8 Read-only root directory structure

This section describes the directory structure that we used when developing this paper. We adapted the information from the following resources:

- The Linux File Hierarchy Standard (FHS)[1]

  http://tldp.org/LDP/Linux-Filesystem-Hierarchy/html/Linux-Filesystem-Hierarchy.html

- The Linux File System Standard (FSSTND):

  http://www.pathname.com/fhs/

  The sections that we include in this paper are copied verbatim with permission.

- The Linux Standards Base (LSB). See the following Web site:

  http://www.linux-foundation.org/en/LSB

[1] Copyright 2003, Binh Nguyen

To comply with the FHS, the following directories, or symbolic links to directories, are required in the read-only root file system (/).

/bin	Essential command binaries
/boot	Static files of the boot loader
/dev	Device files
/etc	Host-specific system configuration
/lib	Essential shared libraries and kernel modules
/media	Mount point for removable media
/mnt	Mount point for mounting a file system temporarily
/opt	Add-on application software packages
/sbin	Essential system binaries
/srv	Data for services provided by this system
/tmp	Temporary files
/usr	Secondary hierarchy
/var	Variable data

The following directories, or symbolic links to directories, must be in /, if the corresponding subsystem is installed:

/home	User home directories (optional)
/lib<qual>	Alternate format essential shared libraries (optional)
/root	Home directory for the root user (optional)

We address each of these directories in the following sections.

/sbin

The /sbin directory is used primarily for privileged commands such as **`shutdown`**, **`ifconfig`**, **`reboot`**, and others. The commands in this directory are mostly intended for system administrators and usually require root access. The /sbin/ directory is in root's PATH but is not in a regular user's PATH.

/bin

The FHS states:

> "Unlike /sbin, the /bin directory contains several useful commands that are of use to both the system administrator as well as non-privileged users. It usually contains the shells like **`bash`**, **`csh`**, etc.... and commonly used commands like **`cp`**, **`mv`**, **`rm`**, **`cat`**, **`ls`**. For this reason and in contrast to /usr/bin, the binaries in this directory are considered to be essential. The reason for this is that it contains essential system programs that must be available even if only the partition containing the root directory is mounted. This situation may arise should you need to repair other partitions but have no access to shared directories (i.e. you are in single user mode and hence have no network access). It also contains programs which boot scripts may require to be present."

/boot

The FHS states:

> "This directory contains everything required for the boot process except for configuration files not needed at boot time (the most notable of those being those that belong to the GRUB boot-loader, not used in System z Linux) and the map installer. Thus, the /boot directory stores data that is used before the kernel begins executing user-mode programs. This may include redundant (back-up) master boot records, sector/system map files, the kernel and other important boot files and data that is not directly edited by hand. Programs necessary to arrange for the boot loader to be able to boot a file are placed in /sbin. Configuration files for boot loaders are placed in /etc. The system kernel is located in either / or /boot (or as under Debian in /boot but is actually a symbolically linked at / in

accordance with the FHS). The boot loader for IBM System z also resides in this directory."

Note: In this paper, /boot/ is read-only; however, there might be times when a read-write copy is necessary.

/dev

This directory highlights one important characteristic of the Linux file system—almost everything is a file or a directory. If you look at this directory and see *dasda1*, *dasda2*, and so forth, these represent the various partitions on the disk drive of the system. The entries beginning with *sda** are SCSI devices. Each logical minidisk is represented as *dasd**a*** for the first, *dasd**b*** for the second, and so forth. This directory must remain read-write.

/etc

The FHS states:

> "This is the nerve center of your system, it contains all system related configuration files (or in subdirectories). A *configuration file* is defined as a local file used to control the operation of a program; it must be static and cannot be an executable binary. For this reason, it is a good idea to backup this directory regularly. Normally, no binaries should be or are located here."

In the read-only root file system, this directory contains both the boot configuration (/etc/init.d) and the local server configuration. Thus, two copies are necessary. The first copy is used during the boot process and is located in /sbin/etc/init.d/. The second copy is located in /local/etc/ and will be bind mounted to /etc/ during the boot process.

/home

The LSB states that:

> "Linux is a multi-user environment so each user is also assigned a specific directory which is accessible only to them and the system administrator. These are the user home directories, which can be found under /home/<*username*>. This directory also contains the user specific settings for programs like IRC, X, etc."

The FSSTND states that:

> "/home is a fairly standard concept, but it is clearly a site-specific file system. Different people prefer to place user accounts in a variety of places. This section describes only a suggested placement for user home directories; nevertheless we recommend that all FHS-compliant distributions use this as the default location for home directories. On small systems, each user's directory is typically one of the many subdirectories of /home such as /home/smith, /home/torvalds, /home/operator, etc. On large systems (especially when the /home directories are shared amongst many hosts using NFS) it is useful to subdivide user home directories. Subdivision may be accomplished by using subdirectories such as /home/staff, /home/guests, /home/students, etc.
>
> The setup will differ from host to host. Therefore, no program should rely on this location.
>
> If you want to obtain a user's home directory, you should use the getpwent(3) library function rather than relying on /etc/passwd because user information may be stored remotely using systems such as NIS.
>
> User specific configuration files for applications are stored in the user's home directory in a file that starts with the '.' character (a "dot file"). If an application needs to create more than one dot file then they should be placed in a subdirectory with a name starting with a '.'

character, (a “dot directory”). In this case the configuration files should not start with the '.' character.

It is recommended that apart from autosave and lock files programs should refrain from creating non dot files or directories in a home directory without user intervention.”

In this paper, we do not address the /home/ directory in detail. However, we include a short discussion of using automount, NFS, and LDAP in “Implementing /home/ with automount, NFS, and LDAP” on page 44.

/lib

The LSB states that:

“The /lib directory contains kernel modules and those shared library images (the C programming code library) needed to boot the system and run the commands in the root file system, i.e. by binaries in /bin and /sbin. Libraries are readily identifiable through their filename extension of *.so. Windows® equivalent to a shared library would be a DLL (dynamically linked library) file. They are essential for basic system functionality. Kernel modules (drivers) are in the subdirectory /lib/modules/'kernel-version'. To ensure proper module compilation you should ensure that /lib/modules/'kernel-version'/kernel/build points to /usr/src/'kernel-version' or ensure that the Makefile knows where the kernel source itself are located.”

/lost+found

The LSB states that:

“Linux should always go through a proper shutdown. Sometimes your system might crash or a power failure might take the machine down. Either way, at the next boot, a lengthy file system check using **fsck** will be done. **fsck** will go through the system and try to recover any corrupt files that it finds. The result of this recovery operation will be placed in this directory. The files recovered are not likely to be complete or make much sense but there always is a chance that something worthwhile is recovered.”

/media

The LSB states that:

“Amid much controversy and consternation on the part of system and network administrators a directory containing mount points for removable media has now been created. Funnily enough, it has been named /media.”

/mnt

The LSB states that:

“This is a generic mount point under which you mount your file systems or devices. Mounting is the process by which you make a file system available to the system. After mounting your files will be accessible under the mount-point.”

The FSSTND v2.3 has changed the purpose of this directory:

“This directory is provided so that the system administrator may temporarily mount a file system as needed. The content of this directory is a local issue and should not affect the manner in which any program is run.

This directory must not be used by installation programs: a suitable temporary directory not in use by the system must be used instead.”

For this paper, the root directory (/) is read-only and /mnt/ is not a separate file system. Therefore, mount points under /mnt/ can only be created by file system design.

/opt

The LSB states that:

> “This directory is reserved for all the software and add-on packages that are not part of the default installation. To comply with the FSSTND, all third party applications should be installed in this directory. Any package to be installed here must locate its static files (i.e. extra fonts, clipart, database files) in a separate /opt/*package* or /opt/*provider* directory tree (similar to the way in which Windows will install new software to its own directory tree C:\Windows\Progam Files*Program Name*), where *package* is a name that describes the software package and *provider* is the provider’s LANANA registered name.
>
> Although most distributions neglect to create the directories /opt/bin, /opt/doc, /opt/include, /opt/info, /opt/lib, and /opt/man they are reserved for local system administrator use. Packages may provide *front-end* files intended to be placed in (by linking or copying) these reserved directories by the system administrator, but must function normally in the absence of these reserved directories. Programs to be invoked by users are located in the directory /opt/*package*/bin. If the package includes UNIX® manual pages, they are located in /opt/*package*/man and the same substructure as /usr/share/man must be used. Package files that are variable must be installed in /var/opt. Host-specific configuration files are installed in /etc/opt.”

In the read-only root system at Nationwide, DB2® is installed under /opt/IBM/db2, MQ Series is installed under /opt/mqm with links into /usr/bin/ and /usr/lib/. So, these mount points are created early while /opt/ is still read-write.

/proc

The LSB states that:

> “/proc is very special in that it is also a virtual file system. It’s sometimes referred to as a process information pseudo-file system. It doesn’t contain “real” files but runtime system information (e.g. system memory, devices mounted, hardware configuration, etc). For this reason it can be regarded as a control and information center for the kernel. In fact, quite a lot of system utilities are simply calls to files in this directory.”

/root

The LSB states that:

> “This is the home directory of the System Administrator, “root”. This may be somewhat confusing (root on root) but in former days, “/” was root’s home directory (hence the name of the Administrator account). To keep things tidier, “root” got his own home directory. Why not in “/home”? Because “/home” is often located on a different partition or even on another system and would thus be inaccessible to “root” when—for some reason—only “/” is mounted.”

The FSSTND states that:

> “If the home directory of the root account is not stored on the root partition it will be necessary to make certain it will default to / if it cannot be located.
>
> We recommend against using the root account for tasks that can be performed as an unprivileged user, and that it be used solely for system administration. For this reason, we recommend that subdirectories for mail and other applications not appear in the root account's home directory, and that mail for administration roles such as root, postmaster, and web master be forwarded to an appropriate user.”

/tmp

The LSB states that:

> "This directory contains mostly files that are required temporarily. Many programs use this to create lock files and for temporary storage of data. Do not remove files from this directory unless you know exactly what you are doing! Many of these files are important for currently running programs and deleting them may result in a system crash. Usually it will not contain more than a few KB anyway. On most systems, this directory is cleared out at boot or at shutdown by the local system. The basis for this was historical precedent and common practice. However, it was not made a requirement because system administration is not within the scope of the FSSTND. For this reason people and programs must not assume that any files or directories in /tmp are preserved between invocations of the program. The reasoning behind this is for compliance with IEEE standard P1003.2 (POSIX, part 2)."

In the read-only root system, /tmp is actually an in-memory only virtual file system. Temporary file storage facility (TPFS), also known as *shared memory file system* (SHMFS) is a standard temporary file system as well. For more information, see:

`http://en.wikipedia.org/wiki/TMPFS`

/usr

The LSB states that:

> "/usr usually contains by far the largest share of data on a system. Hence, this is one of the most important directories in the system as it contains all the user binaries, their documentation, libraries, header files, etc. X and its supporting libraries can be found here. User programs like telnet, ftp, etc. are also placed here. In the original Unix implementations, /usr was where the home directories of the users were placed (that is to say, /usr/someone was then the directory now known as /home/someone). In current Unices, /usr is where user-land programs and data (as opposed to "system land" programs and data) are. The name hasn't changed, but its meaning has narrowed and lengthened from "everything user related" to "user usable programs and data". As such, some people may now refer to this directory as meaning "User System Resources" and not "user" as was originally intended."

The FSSTND states:

> "/usr is shareable, read-only data. That means that /usr should be shareable between various FHS-compliant hosts and must not be written to. Any information that is host-specific or varies with time is stored elsewhere.
>
> Large software packages must not use a direct subdirectory under the /usr hierarchy."

/var

The LSB states that:

> "(/var) Contains variable data like system logging files, mail and printer spool directories, and transient and temporary files. Some portions of /var are not shareable between different systems. For instance, /var/log, /var/lock, and /var/run. Other portions may be shared, notably /var/mail, /var/cache/man, /var/cache/fonts, and /var/spool/news. "/var" contains variable data, i.e. files and directories the system must be able to write to during operation."

The FSSTND states:

> "If /var cannot be made a separate partition, it is often preferable to move /var out of the root partition and into the /usr partition. (This is sometimes done to reduce the size of the root partition or when space runs low in the root partition.) However, /var must not be

linked to /usr because this makes separation of /usr and /var more difficult and is likely to create a naming conflict. Instead, link /var to /usr/var.

Applications must generally not add directories to the top level of /var. Such directories should only be added if they have some system-wide implication, and in consultation with the FHS mailing list."

/srv

The FSSTD states:

"/srv contains site-specific data which is served by this system. This main purpose of specifying this is so that users may find the location of the data files for particular service, and so that services which require a single tree for read-only data, writable data and scripts (such as cgi scripts) can be reasonably placed. Data that is only of interest to a specific user should go in that users' home directory.

The methodology used to name subdirectories of /srv is unspecified as there is currently no consensus on how this should be done. One method for structuring data under /srv is by protocol, eg. ftp, rsync, www, and cvs.

On large systems it can be useful to structure /srv by administrative context, such as /srv/physics/www, /srv/compsci/cvs, etc. This setup will differ from host to host. Therefore, no program should rely on a specific subdirectory structure of /srv existing or data necessarily being stored in /srv. However /srv should always exist on FHS compliant systems and should be used as the default location for such data.

Distributions must take care not to remove locally placed files in these directories without administrator permission. This is particularly important as these areas will often contain both files initially installed by the distributor, and those added by the administrator."

Other directories: /local

In addition to the directories that are defined by the Linux Standard Base, Nationwide has included the /local directory.

This directory contains local content specific to this server. The subdirectories for local copies of /etc/, /dev/, and /root/ are located in this directory with bind mounts to the root directory.

Overview of bind mounts

The /etc/ and /root directories are implemented on the read-only root systems through bind mounts. Here, we provide a background on these types of mounts to explain how the solution implemented at Nationwide works.

Note: We got much of this information from *Linux on IBM eServer zSeries and S/390: Large Scale Linux Deployment*, SG24-6824. You can find the entire contents on this book online at:

http://w3.itso.ibm.com/abstracts/sg246824.html?Open

A bind mount expands the functionality of the device file system mount. Using bind mounts, it is possible to graft a directory sub-tree from one part of the global file system to another. Bind mounts differ from device mounts in that the source is the global file system itself, not a block device.

As an example, consider a directory /guestvol/there that contains a file named goo.bar. An additional directory /mnt/here exists in the global name space. Issue the following command:

```
# mount --bind /guestvol/there /mnt/here
```

Now the same file, goo.abc, can be referenced by two path names (Figure 9):

- /guestvol/there/goo.abc, the original path name
- /mnt/here/goo.abc, the bind mount path name

Both names refer to the same underlying file. The following bind mounts are added to the root file system to support a read-only root:

```
# mount -n -o bind /local/etc /etc
# mount -n -o bind /local/root /root
# mount -n -o bind /local/srv /srv
```

The Linux kernel maintains coherence and consistency regardless of the name that is used. For more details about bind mounts, see *Linux on IBM eServer zSeries and S/390: Large Scale Linux Deployment*, SG24-6824.

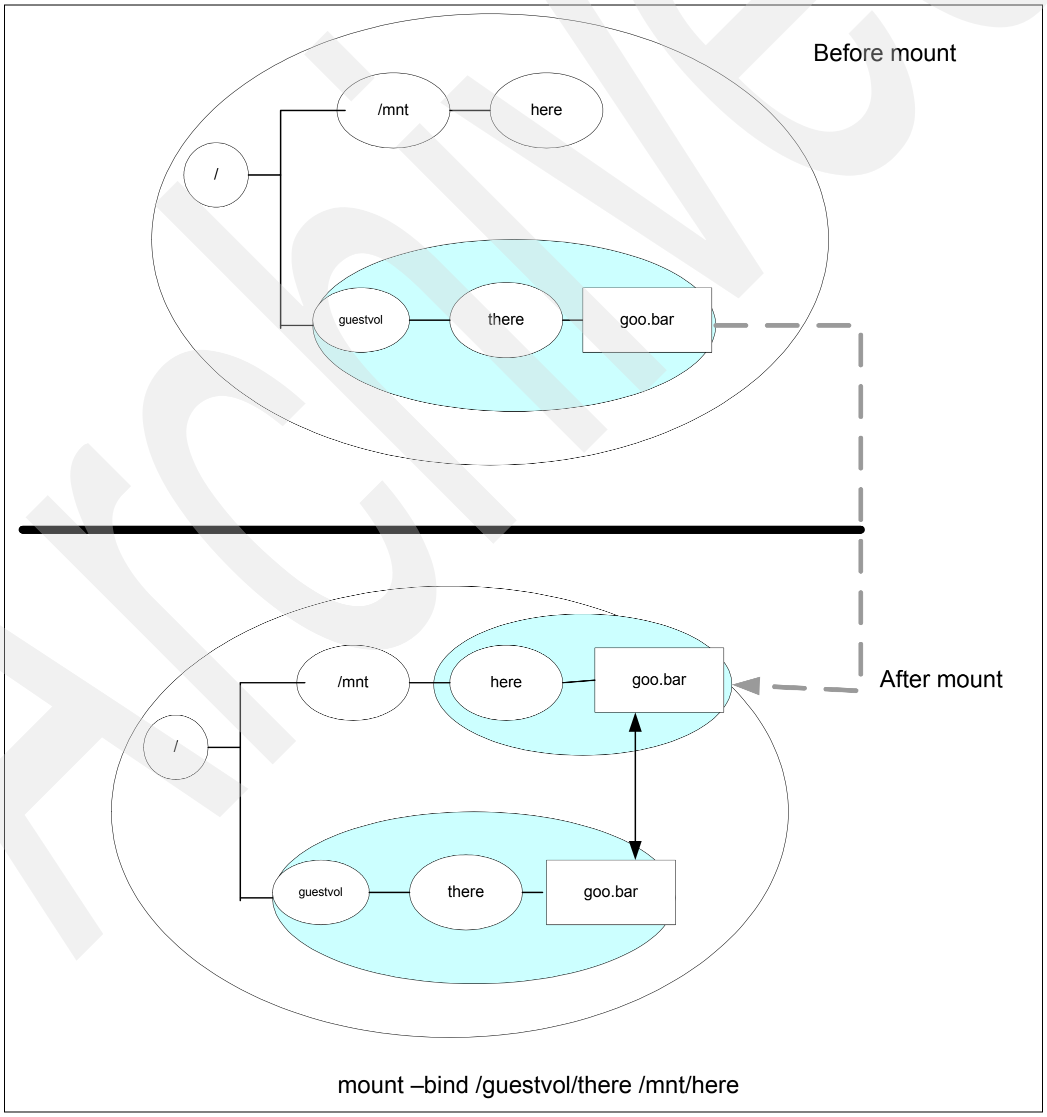


Figure 9 Overview of bind mounts

Overview of read-only root

Table 1 summarize the file systems in the read-only root system. Note that file systems that are read-only use a type *ext2*, because a journal cannot be written to a read-only file system. Read-write file systems use a type of *ext3*, which is more conventional.

Table 1 Summary of file systems and swap spaces

Directory	FS type	Attributes	Device	Vaddr	Notes
/	ext2	R/O	/dev/dasdb1	1B1	read-only root, 400 cylinder minidisk
`/bin/`	ext2	R/O			Part of root file system
`/boot/`	ext2	R/O	/dev/dasda1	1B0	30 cylinder minidisk
`/dev/`	udev	R/W			Different for SLES10 and SLES9
`/etc/`	bind mount	R/W			Stored in /local/etc/; bind mounted to /etc/
`/home/`	auto mount	R/W			Not addressed currently. See "Implementing /home/ with automount, NFS, and LDAP" on page 44
`/lib/`, `/lib64/`	ext2	R/O			Part of root file system
`/local`	ext3	R/W	/dev/dasdf1	1B5	100 cylinder minidisk (~69 MB), where /etc/, /root/ and /srv/ are stored
`/media/`	ext2	R/O			Not used on s390x architecture
`/mnt/`	ext2	R/O			R/W directory can be mounted over R/O
`/opt/`	ext2	R/O	/dev/dasdi1	1B8	108 cylinder minidisk (~76 MB)
`/proc/`	procfs	R/W			In memory kernel file system
`/root/`	bind	R/W			Stored in /local/root/ - bind-mounted at boot time
`/sbin/`	ext2	R/O			Part of root file system
`/sys/`	sysfs	R/W			In memory file system
`/tmp/`	tmpfs	R/W			In memory file system; contents are lost at shutdown
`/usr/`	ext2	R/O	/dev/dasdh1	1B7	1750 cylinder minidisk (~1.2 GB)
`/var/`	ext3	R/W	/dev/dasdg1	1B6	400 cylinder minidisks (~273 MB)
`/srv/`	ext3	R/W			Stored in /local/srv/; bind mounted at boot time
swap 1	swap	R/W	/dev/dasdc1	1B2	64 MB in memory VDISK
swap 2	swap	R/W	/dev/dasdd1	1B3	64 MB in memory VDISK
swap 3	swap	R/W	/dev/dasde1	1B4	550 cylinder minidisk (~384 MB)

The modified boot process

During the normal Linux boot process, the root file system is initially mounted read-only and then later mounted read-write. In the read-only root system, it is not remounted read-write. Figure 10 shows the System z Linux boot process.

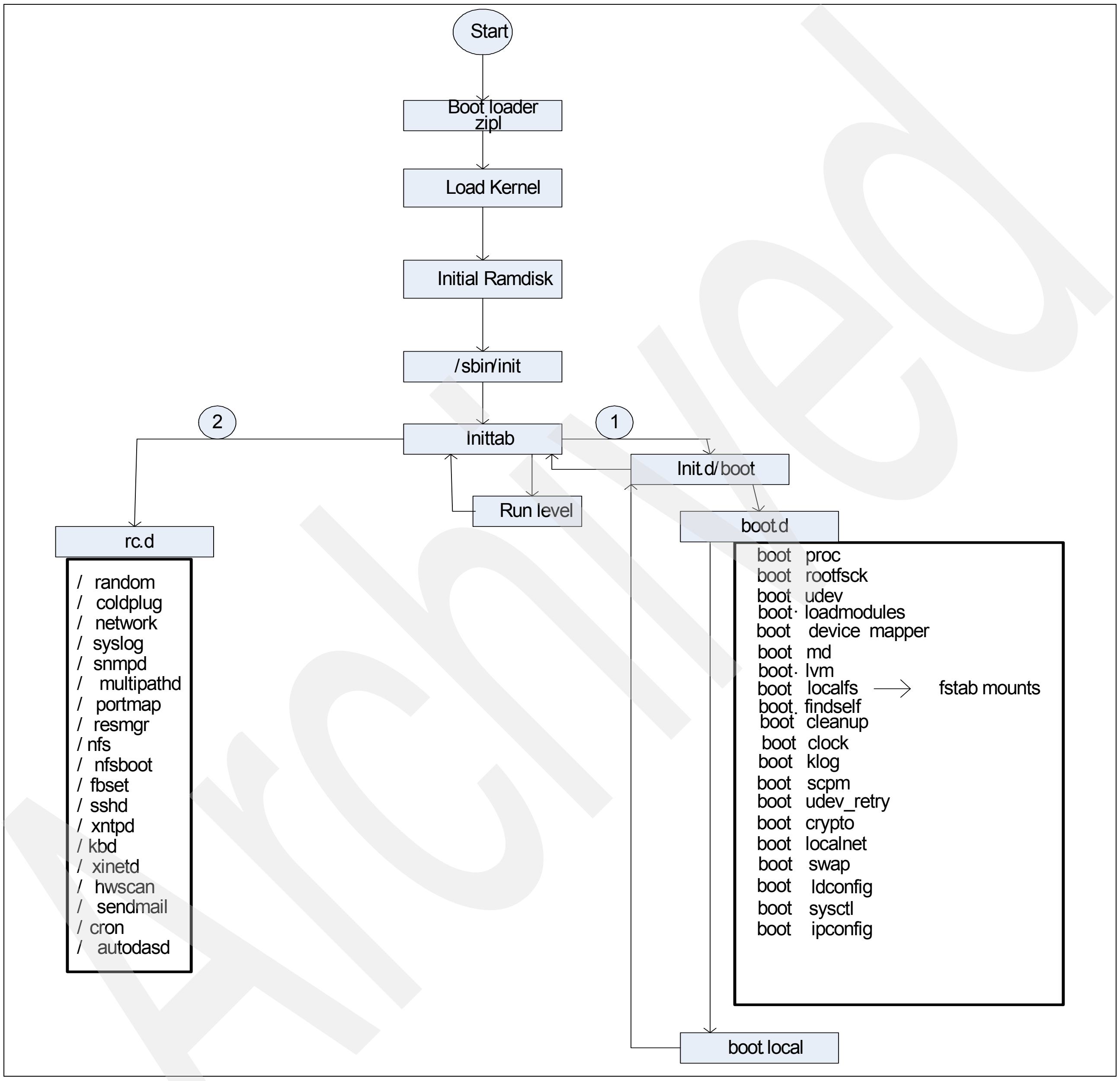


Figure 10 Modified boot process

In preparation for a read-only root, the directory /etc/init.d is moved to /sbin/etc/init.d. The boot script is modified to use /sbin/etc/init.d as the boot.rc variable. This change allows the boot.d scripts to be executed from /sbin/etc/init.d/boot.d. The initialization script /etc/init.d/boot.rootfsck is modified during the creation of the first read-only root system.

The changes to this script are as follows:

- Does not check (`fsck`) root (1B1 disk)
- Checks and mounts /local/ (1B5 disk)
- Bind mounts /etc/, /srv/, and /root/

This script leaves the root file system in read-only mode after performing a file check on the /local/ directory. You can find a copy of the modified script in “Modified boot.rootfsck file” on page 58.

The **`boot.findself`** script runs on the first boot to update the IP address for the new virtual Linux server. This allows the servers to be cloned with identical IP addresses then updated on first boot.

Performance

During testing at Nationwide, the changes to read-only root were found to be insignificant to the overall performance of the Linux systems. The boot process might be slightly faster, because the I/O request is cached. Thus the booting of hundreds of servers will only require one physical disk read.

Testing multiple server simultaneous starts

Testing at Nationwide entailed starting multiple servers simultaneously (8 to 12 servers at a time), without any identified waits or delays.

Product installations

Installations for the following products were tested at Nationwide:

- IBM WebSphere® 6.1
- IBM WebSphere 5.1
- IBM DB2 UDB Version 8
- IBM WebSphere IHS server
- IBM WebSphere MQ series

Peer review

The Linux systems administrators at Nationwide reviewed the read-only root environment to validate that a consistent environment was retained.

Building a read-write maintenance system

In this section, we describe the implementation of the read-only root system in detail. Before building a read-only root system, we address a system for maintaining and cloning a conventional read-write Linux system.

Creating the system

The read-only root system is created in two phases. In the first phase, a read-write system is created with a maintenance plan in mind. These are the boxes shown above the dashed line in Figure 1 on page 3.

To create the system, follow these steps:

1. Create the first z/VM user IDs.
2. Populate the CMS minidisks on CMSCLONE.
3. Install SLES 10 onto S10RWMNT.
4. Install and customize SLES 10 onto LNXCLONE.
5. Copy Linux from S10RWMNT to S10RWTST.
6. Customize and test Linux on S10RWTST.
7. Copy Linux back from S10RWTST to S10RWMNT.
8. Copy Linux from S10RWMNT to gold disks.
9. Clone a read-write Linux.

When the read-write system is tested and working, you can start the second phase to create a read-only root system. Follow these steps:

1. Define a user ID, S10ROGLD, for the first read-only root system.
2. Create a prototype system and test.
3. Copy read-only Linux to gold disks.
4. Clone a read-only Linux.

Creating the first z/VM user IDs

In this paper, the USER DIRECT file is used. If you are using a directory maintenance product, you need to adjust these steps accordingly.

A user ID, *CMSCLONE*, is defined and is given two minidisks:

191	For storing the cloning EXECs.
192	A common disk that will become the other ID's read-only 191 disk. It will store kernels, RAM disks, parameter files, the SWAPGEN EXEC, install EXECs, and so forth.

Note: The ALL keyword is used for the read password so that any user ID can link to the disk without a password.

Note:

- The option LNKNOPAS is included so that the user ID can link to all minidisks without the need for a password.
- Class B permission is needed so that the FLASHCOPY command can be invoked.

Authorization is needed to run the FLASHCOPY command, which is part of CP privilege class B. It is possible to allow the FLASHCOPY command to be executed without class B permission by creating a locally-defined privilege class that gives that authority and nothing else. Details are outside the scope of this paper.

Example 1 shows the directory definition for the CMSCLONE user ID.

Example 1 The directory definition for CMSCLONE

```
USER CMSCLONE PASSWD 256M 1G BG
 INCLUDE IBMDFLT
 OPTION APPLMON LNKNOPAS
 IPL CMS
 MACHINE ESA 4
 MDISK 0191 3390 0001 0030 MM912F MR PASSWD PASSWD PASSWD
 MDISK 0192 3390 0031 0100 MM912F MR ALL PASSWD PASSWD
```

A profile, *RORDFLT*, is added at the top of the USER DIRECT file that will be common to all Linux user IDs. It has the NICDEF statement that creates a virtual OSA connection to the system VSWITCH named VSW1. Think of this statement as creating a virtual Network Interface Card (NIC) that is created and *plugged in* to the system VSWITCH when the user ID is logged on.

The LINK statement to the CMSCLONE 192 disk enables the other user IDs to share a common read-only 191 disk, as shown in Example 2.

Example 2 LINK statement to the CMSCLONE 192 disk

```
******************************
PROFILE RORDFLT
  IPL CMS
  MACHINE ESA 4
  CPU 00 BASE
  CPU 01
  NICDEF 600 TYPE QDIO LAN SYSTEM VSW1
  SPOOL 000C 2540 READER *
  SPOOL 000D 2540 PUNCH A
  SPOOL 000E 1403 A
  CONSOLE 009 3215 T
  LINK MAINT 0190 0190 RR
  LINK MAINT 019D 019D RR
  LINK MAINT 019E 019E RR
  LINK CMSCLONE 192 191 RR
  LINK TCPMAINT 592 592 RR
******************************
```

A user ID S10RWMNT is created. The first Linux image is installed into this ID. Refer to Table 1 on page 15 to see which minidisks are for which file systems. This definition uses only a single 3390-3 for all disks. You might want to use more disk space for each read-write Linux system, such as a 3390-9, or you might want to implement logical volumes on some file systems, such as /var/, to allow for growth. Logical volumes have not been implemented in this paper; however, there is a brief discussion on the topic in “Utilizing logical volumes” on page 43.

Example 3 shows the directory definition for user ID S10RWMNT.

Example 3 The directory definition

```
USER S10RWMNT PASSWD 256M 1G G
 INCLUDE RORDFLT
 OPTION APPLMON
 MDISK 01B0 3390 0001 0030 MM9131 MR PASSWD PASSWD PASSWD
 MDISK 01B1 3390 0031 0400 MM9131 MR PASSWD PASSWD PASSWD
 MDISK 01B4 3390 0431 0550 MM9131 MR PASSWD PASSWD PASSWD
 MDISK 01B5 3390 0981 0100 MM9131 MR PASSWD PASSWD PASSWD
 MDISK 01B6 3390 1081 0400 MM9131 MR PASSWD PASSWD PASSWD
 MDISK 01B7 3390 1481 1750 MM9131 MR PASSWD PASSWD PASSWD
 MDISK 01B8 3390 3231 0108 MM9131 MR PASSWD PASSWD PASSWD
```

A user ID, S10GOLD, is created with the directory definition shown in Example 4. This ID should never be logged on to, so the password is set to NOLOG. The minidisks 1xxx are the read-write gold disks, and 2xxx are the read-only gold disks.

Example 4 User ID S10GOLD with the directory definition

```
USER S10GOLD NOLOG 256M 1G G
 INCLUDE RORDFLT
 OPTION APPLMON
* Gold disks for a read-write system
 MDISK 11B0 3390 0001 0030 MM912E MR PASSWD PASSWD PASSWD
 MDISK 11B1 3390 0031 0400 MM912E MR PASSWD PASSWD PASSWD
 MDISK 11B4 3390 0431 0550 MM912E MR PASSWD PASSWD PASSWD
 MDISK 11B5 3390 0981 0100 MM912E MR PASSWD PASSWD PASSWD
 MDISK 11B6 3390 1081 0400 MM912E MR PASSWD PASSWD PASSWD
 MDISK 11B7 3390 1481 1750 MM912E MR PASSWD PASSWD PASSWD
 MDISK 11B8 3390 3231 0108 MM912E MR PASSWD PASSWD PASSWD
* Gold disks for read-only system
 MDISK 21B0 3390 0001 0030 MM9130 MR PASSWD PASSWD PASSWD
 MDISK 21B1 3390 0031 0400 MM9130 MR PASSWD PASSWD PASSWD
 MDISK 21B4 3390 0431 0550 MM9130 MR PASSWD PASSWD PASSWD
 MDISK 21B5 3390 0981 0100 MM9130 MR PASSWD PASSWD PASSWD
 MDISK 21B6 3390 1081 0400 MM9130 MR PASSWD PASSWD PASSWD
 MDISK 21B7 3390 1481 1750 MM9130 MR PASSWD PASSWD PASSWD
 MDISK 21B8 3390 3231 0108 MM9130 MR PASSWD PASSWD PASSWD
```

A user ID, LNXCLONE, is created with the directory definition shown in Example 5. A minimal Linux image is installed onto the 1B0 disk so that the **mkror.sh** script can be run from /usr/local/sbin/.

Important: The LNKNOPAS option is included so that all minidisks can be linked with no password. Use caution when selecting this option.

Example 5 User ID LNXCLONE with directory definition

```
USER LNXCLONE   PASSWD 256M 1G BG
 INCLUDE RORDFLT
 OPTION APPLMON LNKNOPAS
 MDISK 01B0 3390 0001 3338 MM9132 MR PASSWD PASSWD PASSWD
```

A user ID, S10RWTST, is created with the directory definition shown in Example 6. This is where changes to the gold image are tested.

Example 6 User ID S10RWTST with directory definition

```
USER S10RWTST PASSWD 256M 1G G
 INCLUDE RORDFLT
 OPTION APPLMON
 MDISK 01B0 3390 0001 0030 MM9133 MR PASSWD PASSWD PASSWD
 MDISK 01B1 3390 0031 0400 MM9133 MR PASSWD PASSWD PASSWD
 MDISK 01B4 3390 0431 0550 MM9133 MR PASSWD PASSWD PASSWD
 MDISK 01B5 3390 0981 0100 MM9133 MR PASSWD PASSWD PASSWD
 MDISK 01B6 3390 1081 0400 MM9133 MR PASSWD PASSWD PASSWD
 MDISK 01B7 3390 1481 1750 MM9133 MR PASSWD PASSWD PASSWD
 MDISK 01B8 3390 3231 0108 MM9133 MR PASSWD PASSWD PASSWD
```

The minidisk layout is verified using the DISKMAP command, and the changes are brought online using the DIRECTXA command.

Allowing access to the system VSWITCH

The S10RWMNT, S10RWTST, S10ROGLD, and LNXCLONE user IDs are given access to system's VSWITCH. The S10GOLD user ID does not need access to the VSWITCH because no Linux system will ever be IPLed from it.

The following statements are put in AUTOLOG1's PROFILE EXEC:

```
'cp set vswitch vsw1 grant s10rwmnt'
'cp set vswitch vsw1 grant s10rwtst'
'cp set vswitch vsw1 grant s10rogld'
'cp set vswitch vsw1 grant lnxclone'
```

These commands are also run interactively from the command line for the current IPL so that the system does not have to be shut down and re-IPLed.

Populating CMS disks on CMSCLONE

The CMSCLONE user ID has two CMS disks:

191	To store the CLONE EXECs. These are part of the tar file that are associated with this paper.
192	A common CMS disk that is linked by other Linux systems can share that disk read-only as their 191 disk.

First the CMS disks are formatted using the FORMAT command. Then, the following files are copied to the CMSCLONE 192 disk:

S10RWTST PARM-S10	The parameter file for user ID S10RWTST. A sample parameter file is included in the tar file that is associated with this paper, under the vm/ subdirectory. If you have not yet untarred that file, you can information about it in "Sample PARM-S10 file" on page 70.
SLES10 EXEC	The EXEC to invoke the SLES 10 installation. It is included in the tar file. You can find information about it in "SLES10 EXEC" on page 70.
SWAPGEN EXEC	The EXEC to create VDISK swap spaces. At the time of writing, the latest copy of this EXEC is named SWAPGEN5.EXEC. Download it and rename it. It is available from the Sine Nomine Associates Web page: `http://www.sinenomine.net/vm/swapgen`
PROFILE EXEC	The EXEC that will always IPL Linux from 1B0 on a disconnected machine or, if desired, in an interactive session. It is included in the tar file. You can find information about it in "PROFILE EXEC" on page 67.
PROFILE XEDIT	This is the configuration file that is read when XEDIT is invoked. It is copied from the MAINT 191 disk. Not having a copy of this file can lead to problems, especially because the default value for CASE is UPPER. As a result, modified lines are folded to upper case. Having this happen when editing a parameter file is a real problem.
SLES10 KERNEL	The SLES 10 kernel. The SLES 10 kernel. This is available from the /boot/s390x/ directory of the SLES 10 install media, where it is named vmrdr.ikr.
SLES10 INITRD	The SLES 10 initial RAMdisk. This is also available from the /boot/s390x/ directory of the SLES 10 install media, where it is named *initrd*.
<userid> PARM-S10	Parameter files for other user IDs. You create these by copying and modifying the sample parameter file.

Installing SLES 10 onto S10RWMNT

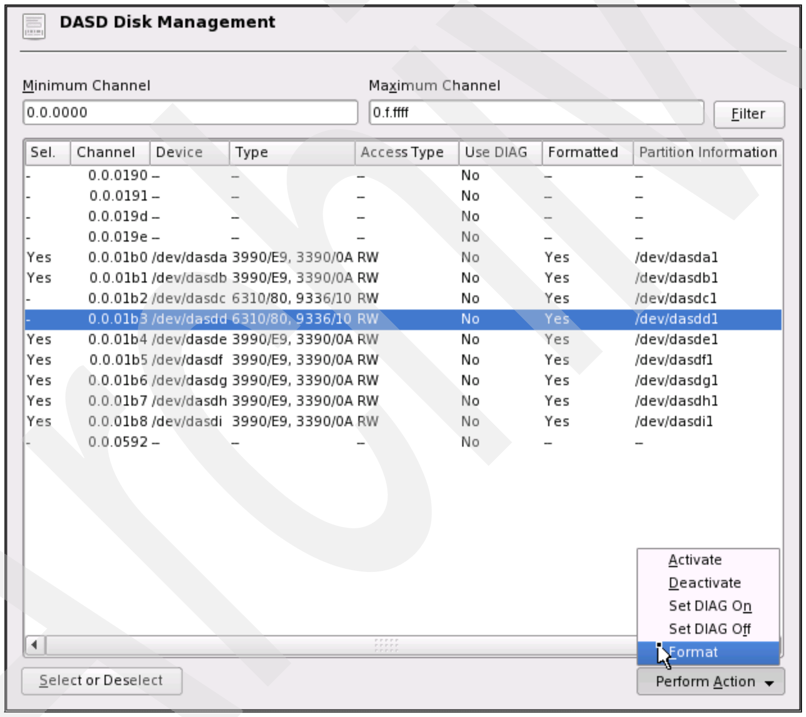
This section does not supply every detail on installing Linux. For more details, see the *z/VM and Linux on IBM System z: The Virtualization Cookbook for SLES 10*, which is available online at:

```
http://linuxvm.org/present
```

The kernel and RAMdisk are copied using FTP to the CMSCLONE 192 disk. Do not forget to transfer them in binary mode with fixed-record 80 byte blocks. If you are FTPing between CMS and a Linux server, this can be accomplished using the FTP subcommand **`bin fix 80`** or **`site bin fix 80`**, depending on the direction that you are copying the files.

The S10RWMNT PARM-S10 file is configured with the correct IP and DNS information. Because 256 MB is not sufficient memory with which to install SLES, the machine size is increased to 512 MB using the CP command DEF STOR 512M. Then CMS is reIPLed.

Now a minimal SLES 10 system is installed onto S10RWMNT. The install is invoked using the SLES10 EXEC. The minidisks are formatted, by first selecting and activating devices 1B0 to 1B8. Then 1B2 and 1B3 are deselected, so that the VDISK swap spaces that are created by SWAPGEN EXEC are not reformatted. Figure 11 shows the remaining seven disks.

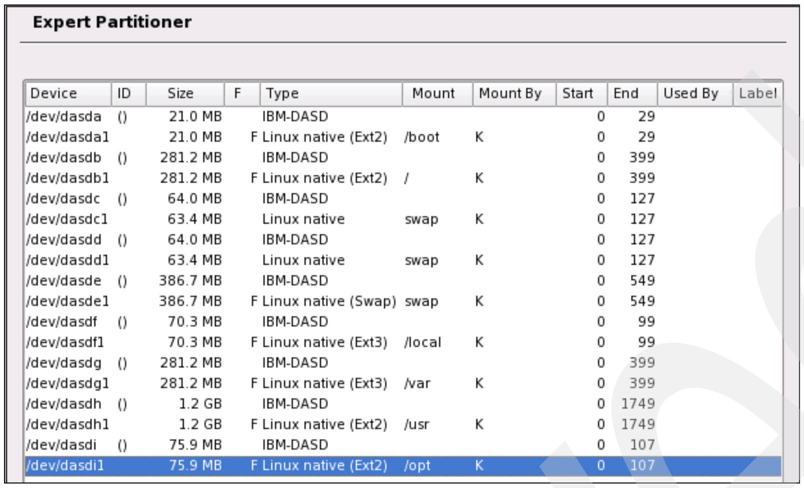

DASD Disk Management

Minimum Channel: 0.0.0000
Maximum Channel: 0.f.ffff

Sel.	Channel	Device	Type	Access Type	Use DIAG	Formatted	Partition Information
-	0.0.0190	–	–	–	No	–	–
-	0.0.0191	–	–	–	No	–	–
-	0.0.019d	–	–	–	No	–	–
-	0.0.019e	–	–	–	No	–	–
Yes	0.0.01b0	/dev/dasda	3990/E9, 3390/0A	RW	No	Yes	/dev/dasda1
Yes	0.0.01b1	/dev/dasdb	3990/E9, 3390/0A	RW	No	Yes	/dev/dasdb1
-	0.0.01b2	/dev/dasdc	6310/80, 9336/10	RW	No	Yes	/dev/dasdc1
-	0.0.01b3	/dev/dasdd	6310/80, 9336/10	RW	No	Yes	/dev/dasdd1
Yes	0.0.01b4	/dev/dasde	3990/E9, 3390/0A	RW	No	Yes	/dev/dasde1
Yes	0.0.01b5	/dev/dasdf	3990/E9, 3390/0A	RW	No	Yes	/dev/dasdf1
Yes	0.0.01b6	/dev/dasdg	3990/E9, 3390/0A	RW	No	Yes	/dev/dasdg1
Yes	0.0.01b7	/dev/dasdh	3990/E9, 3390/0A	RW	No	Yes	/dev/dasdh1
Yes	0.0.01b8	/dev/dasdi	3990/E9, 3390/0A	RW	No	Yes	/dev/dasdi1
-	0.0.0592	–	–	–	No	–	–

Figure 11   Formatting seven minidisks

For partitioning the DASD, select the devices, mount points, and file system types as shown in the Directory, FS Type, and Device columns of Table 1 on page 15. Figure 12 shows a summary.

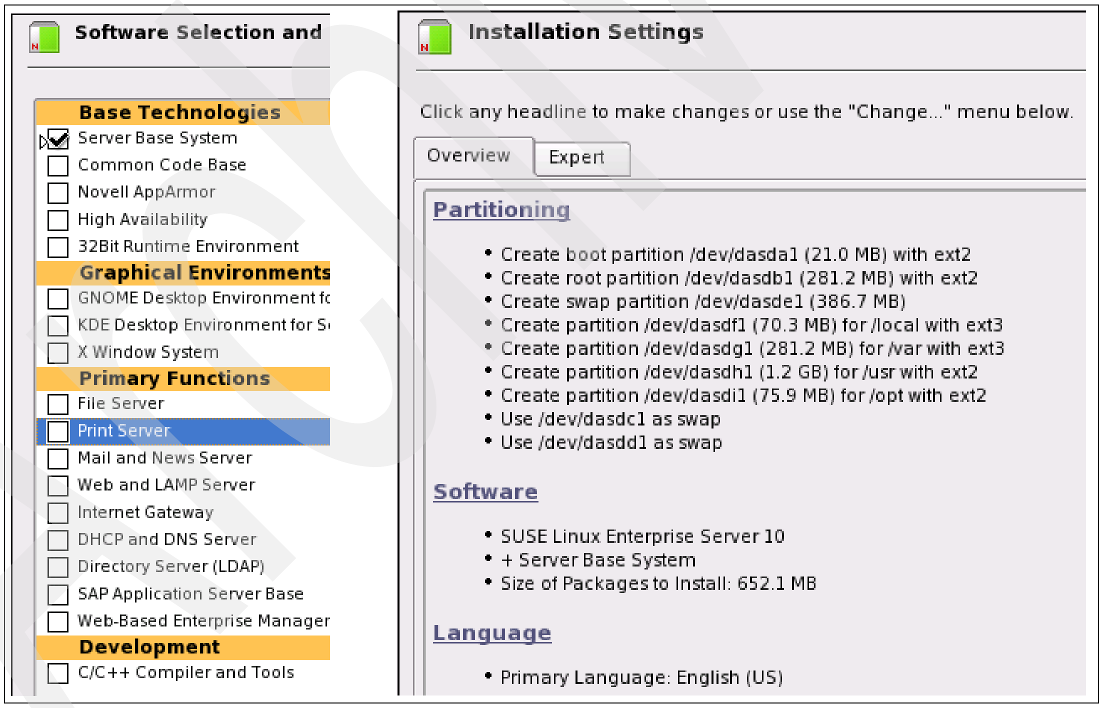


Expert Partitioner

Device	ID	Size	F	Type	Mount	Mount By	Start	End	Used By	Label
/dev/dasda	()	21.0 MB		IBM-DASD			0	29		
/dev/dasda1		21.0 MB	F	Linux native (Ext2)	/boot	K	0	29		
/dev/dasdb	()	281.2 MB		IBM-DASD			0	399		
/dev/dasdb1		281.2 MB	F	Linux native (Ext2)	/	K	0	399		
/dev/dasdc	()	64.0 MB		IBM-DASD			0	127		
/dev/dasdc1		63.4 MB		Linux native	swap	K	0	127		
/dev/dasdd	()	64.0 MB		IBM-DASD			0	127		
/dev/dasdd1		63.4 MB		Linux native	swap	K	0	127		
/dev/dasde	()	386.7 MB		IBM-DASD			0	549		
/dev/dasde1		386.7 MB	F	Linux native (Swap)	swap	K	0	549		
/dev/dasdf	()	70.3 MB		IBM-DASD			0	99		
/dev/dasdf1		70.3 MB	F	Linux native (Ext3)	/local	K	0	99		
/dev/dasdg	()	281.2 MB		IBM-DASD			0	399		
/dev/dasdg1		281.2 MB	F	Linux native (Ext3)	/var	K	0	399		
/dev/dasdh	()	1.2 GB		IBM-DASD			0	1749		
/dev/dasdh1		1.2 GB	F	Linux native (Ext2)	/usr	K	0	1749		
/dev/dasdi	()	75.9 MB		IBM-DASD			0	107		
/dev/dasdi1		75.9 MB	F	Linux native (Ext2)	/opt	K	0	107		

Figure 12 Partitioning 1B0 (/dev/dasda) to 1B8 (/dev/dasdi)

For Software Selection, all package groups are deselected except for *Server Base System*. Figure 13 shows the software packages groups and disk partitioning information.



Click any headline to make changes or use the "Change..." menu below.

Partitioning

- Create boot partition /dev/dasda1 (21.0 MB) with ext2
- Create root partition /dev/dasdb1 (281.2 MB) with ext2
- Create swap partition /dev/dasde1 (386.7 MB)
- Create partition /dev/dasdf1 (70.3 MB) for /local with ext3
- Create partition /dev/dasdg1 (281.2 MB) for /var with ext3
- Create partition /dev/dasdh1 (1.2 GB) for /usr with ext2
- Create partition /dev/dasdi1 (75.9 MB) for /opt with ext2
- Use /dev/dasdc1 as swap
- Use /dev/dasdd1 as swap

Software

- SUSE Linux Enterprise Server 10
- + Server Base System
- Size of Packages to Install: 652.1 MB

Language

- Primary Language: English (US)

Figure 13 Software selection and disk partitioning settings

The first half of the installation completes, and the new system is IPLed from 1B0. The second half of the installation is completed, and an SSH session is started.

The file systems should be similar to that shown in Example 7.

Example 7 File systems after installation

```
ntc242:~ # df -h
Filesystem           Size  Used Avail Use% Mounted on
/dev/dasdb1          273M   91M  169M  35% /
udev                 247M  124K  247M   1% /dev
/dev/dasda1           21M   16M  3.5M  83% /boot
/dev/dasdf1           69M  4.1M   61M   7% /local
/dev/dasdi1           74M   23M   47M  33% /opt
/dev/dasdh1          1.2G  672M  479M  59% /usr
/dev/dasdg1          273M   62M  197M  24% /var
```

This Linux system will be the basis for both the read-write and the read-only systems. You can now shut down the Linux system running on S10RWMNT.

Installing and customizing SLES 10 onto LNXCLONE

A minimal system is also installed onto the LNXCLONE user ID. This system has just a single 3390-3 minidisk at 1B0 for a root file system. The main function of this system is to be able to create a read-only root system, using the **mkror.sh** script, with both the source and target systems shutdown and their user IDs logged off from z/VM.

To customize the system, make the following modifications:

- Download and untar the tar file that is associated with this paper.
- Set the **vmcp** module to load at boot time so that CP commands can be issued.

Downloading the files that are associated with this paper

Copy the tar file redp4322.tgz to /usr/local/ and then untar the file (Example 8).

Example 8 Downloading and untarring redp4322.tgz

```
# cd /usr/local
...Copy the tar file ...
# tar xzf 4322.tgz
README.txt
sbin/
sbin/boot.findself
sbin/boot.rootfsck.ror
sbin/fstab.ror
sbin/mkror.sh
sbin/cloneprep.sh
vm/
vm/CLONERO.EXEC
vm/CLONERW.EXEC
vm/MNT2GOLD.EXEC
vm/MNT2TST.EXEC
vm/PROFILE.EXEC
vm/RO2GOLD.EXEC
vm/TST2MNT.EXEC
vm/COPYMDSK.EXEC
vm/SAMPLE.PARM-S10
vm/SLES10.EXEC
```

These files are used in the construction of the read-only root system.

Setting the vmcp module to be loaded

The vmcp module is added to the file /etc/sysconfig/kernel to allow the system to issue CP commands using vmcp:

```
MODULES_LOADED_ON_BOOT="vmcp"
```

The system can now either be rebooted to effect the change or the module can be loaded for the current session with the command **`modprobe vmcp`**.

Copying Linux from S10RWMNT to S10RWTST

Linux is shutdown and S10RWMNT is logged off. The minidisks 1B0 to 1B8 are copied from the newly installed S10RWMNT to the corresponding minidisks on S10RWTST. Linux is IPLed on S10RWTST. Then modifications are made to the Linux on S10RWTST. In this fashion, there is a backup copy of Linux. If tests are not successful on the test system, a fresh copy of Linux can be copied quickly from the maintenance system.

There are a number of ways to copy the disks. It is important that before copying that both the source and target systems are shut down and that their IDs are logged off z/VM.

The MNT2TST EXEC run from the CMSCLONE user ID tries to use FLASHCOPY to copy the minidisks quickly. If this command is not supported or fails, it falls back to using the DDR command. See “MNT2TST EXEC” on page 66 for a complete listing of the source code.

Example 9 The MNT2TST EXEC from CMSCLONE

```
==> mnt2tst
Do you want to copy R/W disks from S10RWMNT to S10RWTST? y/n
y
Copying minidisk 01B0 to 11B0 ...
00: Command complete: FLASHCOPY 01B0 0 END TO 11B0 0 END
Return value = 0

Copying minidisk 01B1 to 11B1 ...
00: Command complete: FLASHCOPY 01B1 0 END TO 11B1 0 END
Return value = 0

Copying minidisk 01B4 to 11B4 ...
00: Command complete: FLASHCOPY 01B4 0 END TO 11B4 0 END
Return value = 0

Copying minidisk 01B5 to 11B5 ...
00: Command complete: FLASHCOPY 01B5 0 END TO 11B5 0 END
Return value = 0

Copying minidisk 01B6 to 11B6 ...
00: Command complete: FLASHCOPY 01B6 0 END TO 11B6 0 END
Return value = 0

Copying minidisk 01B7 to 11B7 ...
00: Command complete: FLASHCOPY 01B7 0 END TO 11B7 0 END
Return value = 0

Copying minidisk 01B8 to 11B8 ...
```

```
00: Command complete: FLASHCOPY 01B8 0 END TO 11B8 0 END
Return value = 0

Cleaning up ...
00: 01B0 01B1 01B4 01B5 01B6 01B7 01B8 11B0 DETACHED
00: 11B1 11B4 11B5 11B6 11B7 11B8 DETACHED
```

Customizing and testing Linux on S10RWTST

Now that the fresh installation has been copied to S10RWTST, log on to it. You see a virtual Network Interface Card (NIC) defined at virtual addresses 600-602. Using the sample PROFILE EXEC and SWAPGEN EXEC, you see the two VDISK swap spaces created at virtual addresses 1B2 and 1B3. You are then prompted to IPL Linux from 1B0 as shown in Example 10.

Example 10 IPL Linux from 1B0

```
00: NIC 0600 is created; devices 0600-0602 defined
00: z/VM Version 5 Release 3.0, Service Level 0701 (64-bit),
00: built on IBM Virtualization Technology
00: There is no logmsg data
00: FILES:   NO RDR,   NO PRT,   NO PUN
00: LOGON AT 10:17:18 EDT FRIDAY 08/31/07
z/VM V5.3.0    2007-06-19 08:41

DMSACP723I A (191) R/O
DMSACP723I C (592) R/O
DIAG swap disk defined at virtual address 1B2 (16245 4K pages of swap space)
DIAG swap disk defined at virtual address 1B3 (16245 4K pages of swap space)
Do you want to IPL Linux from DASD 1B0? y/n
y
```

Linux then boots, and you see the prompts shown in Example 11.

Example 11 Linux boots

```
00: zIPL v1.5.3 interactive boot menu
00:
00:  0. default (ipl)
00:
00:  1. ipl
00:  2. failsafe
00:
00: Note: VM users please use '#cp vi vmsg <number> <kernel-parameters>'
00:
00: Please choose (default will boot in 10 seconds):
00: Booting default (ipl)...
Linux version 2.6.16.21-0.8-default (geeko@buildhost) (gcc version 4.1.0 (SUSE
Linux)) #1 SMP Mon Jul 3 18:25:39 UTC 2006
We are running under VM (64 bit mode)
...
```

The following configuration changes are recommended:

- The parameter line and menu delay are changed in /etc/zipl.conf.
- The “timer pop” is turned off.
- The CMS file system (cmsfs) package is installed.
- The cmm module is set to be loaded at boot time.
- The system is set to halt, not reboot, when z/VM is shut down.
- The script **`boot.findself`** is copied for newly cloned systems to find their IP and DNS information.
- Empty mount points are created under /opt/ for the possibility of mounting middleware
- The script **`cloneprep.sh`** is copied to aid in cleanup before cloning.

Modifying zipl.conf

You need to modify the parameter line in /etc/zipl.conf as follows:

- Reduce the menu time-out from 10 seconds to 3 so that Linux IPLs more quickly with no user input.
- Add the `dasd=` parameter so that there that are well-known *slots* for additional disks at addresses 1B9 to 1BF (that is /dev/dasdi to /dev/dasdp), 2B0 to 2BF, and 320 to 33F. The additional slots from 1B9 to 1BF can be used for adding devices for file systems (for example, adding a minidisk at 1B9 for /sbin/ and one at 1BA for /bin/). The additional slots from 2B0 to 2BF can be used for adding devices for logical volumes, and the additional slots from 320 to 32F can be considered reserved for future growth.
- Add the `vmpoff=LOGOFF` parameter so that VM user IDs are logged off after Linux is shut down.

Back up the original zipl.conf, and then make the changes shown in Example 12.

Example 12 Changes to zipl.conf

```
# cp zipl.conf zipl.conf.orig
# vi zipl.conf
# Modified by YaST2. Last modification on Wed Aug 29 19:48:46 UTC 2007
[defaultboot]
defaultmenu = menu

:menu
target = /boot/zipl
timeout = 3
prompt = 1
1 = ipl
2 = failsafe
default = 1

###Don't change this comment - YaST2 identifier: Original name: ipl###
[ipl]
    target = /boot/zipl
    image = /boot/image
    ramdisk = /boot/initrd,0x1000000
    parameters = "dasd=1b0-1bf,2b0-2bf,320-33f root=/dev/dasdb1 TERM=dumb vmpoff=LOGOFF"
...
```

Run the **`zipl`** command to write the changes to /boot/:

```
# zipl
...
```

Turning off the timer pop

The *timer pop* is turned off by adding the following line to the bottom of /etc/sysctl.conf:

```
kernel.hz_timer = 0
```

Adding the CMS file system package

The **cmsfs** RPM is added using the **yast -i** command. It is needed by the **boot.findself** script to obtain the correct IP address and host name from the 191 (CMSCLONE 192) disk at first boot.

```
# yast -i cmsfs
...
```

Inserting the cmm module

The cmm module is added to the file /etc/sysconfig/kernel. This will cause the module to be loaded at IPL time. When loaded it allows z/VM to communicate with Linux virtual servers and to possibly increase performance via collaborative memory management and VMRM. This topic is outside the scope of this paper, but the modification is still recommended:

```
MODULES_LOADED_ON_BOOT="cmm"
```

Setting the system to halt on SIGNAL SHUTDOWN

Rather than rebooting, you might want your system to halt (shutdown). Change this setting by changing **shutdown -r** to **shutdown -h** in the /etc/inittab file, as shown in Example 13.

Example 13 Setting the system to halt on shutdown

```
# cd /etc
# vi inittab // change shutdown -r to shutdown -h
...
# what to do when CTRL-ALT-DEL is pressed
ca::ctrlaltdel:/sbin/shutdown -h -t 4 now
...
```

Copying the cloneprep.sh and boot.findself scripts

The script **cloneprep.sh** is copied to /usr/local/sbin/ so it can be used to clean up files just before cloning. The script **boot.findself** is copied to /etc/init.d/ so it can be run one time at first boot.

The scripts are copied using the **scp** command. In Example 14, the IP address of the Linux running on LNXCLONE is *129.40.179.246*.

Example 14 Copying the scripts

```
# cd /usr/local/sbin
# scp 129.40.179.246:/usr/local/sbin/cloneprep.sh .
Password:
cloneprep.sh                                  100% 2280     2.2KB/s   00:00
# cd /etc/init.d
# scp 129.40.179.246:/usr/local/sbin/boot.findself .
Password:
boot.findself                                 100% 5409     5.3KB/s   00:00
```

The **boot.findself** script is set to run at boot time with the **chkconfig** command:

```
# chkconfig boot.findself on
```

See “The boot.findself script” on page 46 for a listing of the code and a brief description of the logic.

Creating empty mount points under /opt/

Mounting middleware binaries read-only is beyond the scope of this paper. However, if there is a possibility that you may run WebSphere Application Server, DB2 UDB, or MQ Series, you might want to create the following, or other, empty mount points:

- /opt/IBM/WebSphere/
- /opt/mqm/
- /opt/IBM/db2/

In this fashion, all cloned servers will have empty mount points for possibly mounting software.

```
# cd /opt
# mkdir mqm IBM
# cd IBM
# mkdir WebSphere db2
```

Other modifications

These are only a few modifications to the base system. You can choose many other modifications such as adding more RPMs, removing unnecessary RPMS, setting the software clock correctly using NTP and so forth.

Testing your system

The next step is to perform all tests that are necessary and that are appropriate for your environment.

The last step

Now, run the **cloneprep.sh** script. The output should be similar to the following:

```
# cloneprep.sh
rm: cannot remove `/var/log/YaST2/y2log-*': No such file or directory
rm: cannot remove `/var/log/*.gz': No such file or directory
System should be ready for shutdown and cloning
```

The system is now ready to clone. Shut down the test system, and log off the z/VM user ID.

```
# shutdown -h now
...
```

Log off the S10RWTST user ID now.

Copying Linux from S10RWTST to S10RWMNT

After the test system has been modified and thoroughly tested, it can be copied back to the maintenance system using the TST2MNT EXEC from the CMSCLONE user ID, as shown in Example 15.

Example 15 Copying Linux using TST2MNT EXEC from the CMSCLONE user ID

```
==> tst2mnt
Do you want to copy R/W disks from S10RWTST to S10RWMNT? y/n
y
Copying minidisk 01B0 to 11B0 ...
00: Command complete: FLASHCOPY 01B0 0 END TO 11B0 0 END
Return value = 0

Copying minidisk 01B1 to 11B1 ...
00: Command complete: FLASHCOPY 01B1 0 END TO 11B1 0 END
Return value = 0

Copying minidisk 01B4 to 11B4 ...
00: Command complete: FLASHCOPY 01B4 0 END TO 11B4 0 END
Return value = 0

Copying minidisk 01B5 to 11B5 ...
00: Command complete: FLASHCOPY 01B5 0 END TO 11B5 0 END
Return value = 0

Copying minidisk 01B6 to 11B6 ...
00: Command complete: FLASHCOPY 01B6 0 END TO 11B6 0 END
Return value = 0

Copying minidisk 01B7 to 11B7 ...
00: Command complete: FLASHCOPY 01B7 0 END TO 11B7 0 END
Return value = 0

Copying minidisk 01B8 to 11B8 ...
00: Command complete: FLASHCOPY 01B8 0 END TO 11B8 0 END
Return value = 0

Cleaning up ...
00: 01B0 01B1 01B4 01B5 01B6 01B7 01B8 11B0 DETACHED
00: 11B1 11B4 11B5 11B6 11B7 11B8 DETACHED
```

The test and the maintenance systems are now the same. The next step is to create the first gold read-write copy.

Copying read-write Linux to the gold disks

Again from CMSCLONE, copy the maintenance system to the 1xxx disks on the gold system, S10GOLD using MNT2GOLD EXEC, as shown in Example 16.

Example 16   Copying Linux using MNT2GOLD EXEC

```
==> mnt2gold
Do you want to copy R/W disks from S10RWMNT to S10GOLD? y/n
y

Copying minidisk 01B0 to 11B0 ...
00: Command complete: FLASHCOPY 01B0 0 END TO 11B0 0 END
Return value = 0

Copying minidisk 01B1 to 11B1 ...
00: HCPCMM296E Status is not as required - 01B1; an unexpected conditio
00: HCPCMM296E occurred while executing a FLASHCOPY command, code = A7.
FLASHCOPY failed, falling back to DDR ...
z/VM DASD DUMP/RESTORE PROGRAM
HCPDDR696I VOLID READ IS 0X01B1
HCPDDR696I VOLID READ IS 0X01B1
COPYING   0X01B1
COPYING DATA  10/12/07 AT 16.47.31  GMT FROM 0X01B1 TO 0X01B1
INPUT CYLINDER EXTENTS      OUTPUT CYLINDER EXTENTS
      START       STOP         START        STOP
          0        399             0         399
END OF COPY
END OF JOB
Return value = 0

Copying minidisk 01B4 to 11B4 ...
...
```

Note that FLASHCOPY did not succeed in every case, and the EXEC falls back to DDR. This is not unexpected as the copying of the data from the previous EXEC had probably not completed in the background of the disk subsystem.

Now the same system exists on S10RWMNT, S10RWTST, and S10GOLD.

Cloning a read-write Linux system

You are now ready to clone a read-write system. The CLONERW EXEC is simply another EXEC that copies the 1B0, 1B1, and 1B4-1B8 minidisks from one system to another. Here the source is the S10GOLD system and the target user ID must be specified.

A new user ID, LINUX001, is created with the same size and numbered minidisks as the other IDs (Example 17).

Example 17   New user ID, LINUX001

```
USER LINUX001 PASSWD 256M 1G G
 INCLUDE RORDFLT
 OPTION APPLMON
 MDISK 01B0 3390 0001 0030 MM9136 MR PASSWD PASSWD PASSWD
 MDISK 01B1 3390 0031 0400 MM9136 MR PASSWD PASSWD PASSWD
 MDISK 01B4 3390 0431 0550 MM9136 MR PASSWD PASSWD PASSWD
 MDISK 01B5 3390 0981 0100 MM9136 MR PASSWD PASSWD PASSWD
 MDISK 01B6 3390 1081 0400 MM9136 MR PASSWD PASSWD PASSWD
 MDISK 01B7 3390 1481 1750 MM9136 MR PASSWD PASSWD PASSWD
 MDISK 01B8 3390 3231 0108 MM9136 MR PASSWD PASSWD PASSWD
```

The changes are brought online using DIRECTXA. The user ID is given access to system's VSWITCH as shown previously.

Now a parameter file must be created on the CMSCLONE 192 disk (which is the Linux user ID's read-only 191 disk). By default CMS accesses the 192 disk as D. So the S10RWMNT parameter file on CMSCLONE's D disk is copied as a template, and the host name and IP address are modified, as shown in Example 18.

Example 18 Creating a parameter file on the CMSCLONE 192 disk

```
===> copy s10rwmnt parm-s10 d linux001 = =
===> x linux001 parm-s10 d
ramdisk_size=65536 root=/dev/ram1 ro init=/linuxrc TERM=dumb
HostIP=129.40.179.244 Hostname=ntc244.pbm.ihost.com
Gateway=129.40.179.254 Netmask=255.255.255.0
Broadcast=129.40.179.255 Layer2=0
ReadChannel=0.0.0600  WriteChannel=0.0.0601  DataChannel=0.0.0602
Nameserver=129.40.106.1 Portname=dontcare
Install=nfs://129.40.179.200/nfs/sles10/SLES-10-CD-s390x-GMC-CD1.iso
UseVNC=1  VNCPassword=123456
InstNetDev=osa OsaInterface=qdio OsaMedium=eth Manual=0
```

By using the SLES 10 parameter file to maintain the IP address and host name, there is a side effect. If for some reason you need to install Linux manually, or even use the installation process as a rescue system, this file will be available and the same IP/DNS information will be used.

The read-write Linux system is cloned using the CLONERW EXEC, as shown in Example 19.

Example 19 Cloning the read-write Linus system using CLONERW EXEC

```
==> clonerw linux001
Do you want to copy R/W system from S10GOLD to linux001 y/n
y

Copying minidisk 11B0 to 01B0 ...
Command complete: FLASHCOPY 11B0 0 END TO 01B0 0 END
Return value = 0

Copying minidisk 11B1 to 01B1 ...
Command complete: FLASHCOPY 11B1 0 END TO 01B1 0 END
Return value = 0

Copying minidisk 11B4 to 01B4 ...
Command complete: FLASHCOPY 11B4 0 END TO 01B4 0 END
Return value = 0

Copying minidisk 11B5 to 01B5 ...
Command complete: FLASHCOPY 11B5 0 END TO 01B5 0 END
Return value = 0
```

```
Copying minidisk 11B6 to 01B6 ...
Command complete: FLASHCOPY 11B6 0 END TO 01B6 0 END
Return value = 0

Copying minidisk 11B7 to 01B7 ...
Command complete: FLASHCOPY 11B7 0 END TO 01B7 0 END
Return value = 0

Copying minidisk 11B8 to 01B8 ...
Command complete: FLASHCOPY 11B8 0 END TO 01B8 0 END
Return value = 0

01B0-01B1 DETACHED
01B4-01B8 DETACHED
11B0-11B1 DETACHED
11B4-11B8 DETACHED
```

You now have four identical systems, three of which can be IPLed: S10RWMNT, S10RWTST, and LINUX001. All of these systems have identical IP and DNS information. For the first three system user IDs, this identical information is acceptable knowing that only one will be booted at a time. However, the target systems will naturally need unique IP and DNS values.

You can now log on to LINUX001 and IPL from 1B0. You should notice the modified default boot time of 3 seconds and the modified parameter line, as shown in Example 20.

Example 20 Log on to LINUX001 and IPL from 1B0

```
Do you want to IPL Linux from DASD 1B0? y/n
y
00: zIPL v1.5.3 interactive boot menu
00:
00:  0. default (ipl)
00:
00:  1. ipl
00:  2. failsafe
00:
00: Note: VM users please use '#cp vi vmsg <number> <kernel-parameters>'
00:
00: Please choose (default will boot in 3 seconds):
00: Booting default (ipl)...
Linux version 2.6.16.21-0.8-default (geeko@buildhost) (gcc version 4.1.0 (SUSE L
inux)) #1 SMP Mon Jul 3 18:25:39 UTC 2006
We are running under VM (64 bit mode)
Detected 2 CPU's
Boot cpu address  0
Built 1 zonelists
Kernel command line: dasd=1b0-1bf,2b0-2bf,320-33f root=/dev/dasdb1 TERM=dumb
vmpoff=LOGOFF BOOT_IMAGE=0
...
```

The script /etc/init.d/boot.findself runs later in the boot sequence. It accesses the 191 disk (CMSCLONE 192), read the parameter file, and set the TCP/IP address and host name. It does this by modifying the files /etc/HOSTS, /etc/hostname, and the eth0 configuration file under /etc/sysconfig/network/. Note that the gateway, DNS server, and broadcast information are not modified. The script assumes this information is the same for all Linux virtual servers. If this information is different, you need to modify the script /etc/init.d/boot.findself. See "The boot.findself script" on page 46 for a complete listing of the source code.

You should see informational messages similar to that shown in Example 21.

Example 21   Informational messages

```
...
/etc/init.d/boot.findself: changing (escaped) ntc243\.pbm\.ihost\.com to
ntc245.pbm.ihost.com in /etc/HOSTNAME
/etc/init.d/boot.findself: changing ntc243 to ntc245 and IP address in /etc/hosts
/etc/init.d/boot.findself: changing (escaped) 129\.40\.179\.243 to 129.40.179.245
in /etc/sysconfig/network/ifcfg-qeth-bus-ccw-0.0.0600
```

Building a read-only root system

You now have a read-write Linux system with tools to maintain copies of Linux for test, maintenance, and cloning. It is now time to tackle creating a read-only root system.

As summary, the steps are as follows:

1. Define a user ID, S10ROGLD, for the first read-only root system.
2. Create a prototype system and test.
3. Copy read-only Linux to gold disks.
4. Clone a read-only Linux.

Defining a user ID for first read-only root system

A user ID, S10ROGLD, is created where the read-only system will be created (Example 22). Note that all minidisks are read-write because the process of moving from a read-write to a read-only system requires access to various disks as read-write for certain periods of time.

Example 22   Creating user ID S10ROGLD

```
USER S10ROGLD PASSWD 256M 1G G
 INCLUDE RORDFLT
 OPTION APPLMON
 MDISK 01B0 3390 0001 0030 MM9134 MR PASSWD PASSWD PASSWD
 MDISK 01B1 3390 0031 0400 MM9134 MR PASSWD PASSWD PASSWD
 MDISK 01B4 3390 0431 0550 MM9134 MR PASSWD PASSWD PASSWD
 MDISK 01B5 3390 0981 0100 MM9134 MR PASSWD PASSWD PASSWD
 MDISK 01B6 3390 1081 0400 MM9134 MR PASSWD PASSWD PASSWD
 MDISK 01B7 3390 1481 1750 MM9134 MR PASSWD PASSWD PASSWD
 MDISK 01B8 3390 3231 0108 MM9134 MR PASSWD PASSWD PASSWD
```

The directory changes are brought online using the DIRECTXA command. The user ID is given access to systems VSWITCH.

Creating a prototype read-only root Linux

The script mkror.sh and some modified files were composed to help facilitate creating a read-only root system from a conventional read-write Linux system. Understand that these are neither supported, nor well-tested. Again, check with your Linux distributor or support company to be sure that such a system will be supported. If you implement it, test everything well. Remember, you are on the *bleeding* edge.

The global variables and functions calls are at the bottom of the script. Here are the global variables:

```
sourceID="S10RWMNT"
targetID="S10ROGLD"
rorDiffs="/usr/local/sbin/boot.rootfsck.diffs"
fstabFile="/usr/local/sbin/fstab.ror"
```

The source user ID is *S10WRMNT*, and the target user ID is *S10ROGLD*. The global variable `rordiffs` specifies the diff file that is shipped with the tar file that is used to modify the /etc/init.d/boot.rootfsck script. The variable `fstabFile` specifies the modified /etc/fstab file also shipped with the tar file. Because of these user ID names, boot script, and file system configuration file are *hard-coded* into the script, you must perform all the steps in this section identically, or the script could easily fail.

Example 23 lists the function calls.

Example 23 The function calls

```
checkID $sourceID
checkID $targetID
linkSourceDisks
linkTargetDisks
copySystem
enableSourceDisks
enableTargetDisks
mountSourceRoot
mountTargetDisks
modifySystem
cleanUp
echo "Exiting with rc 0!"
exit 0
```

See “The mkror.sh script” on page 50 for a complete listing of script. Here is a high-level description of the functions:

- The first two function calls to `checkID` verify that the source and target user IDs are logged off.
- The next two function calls, `linkSourceDisks` and `linkTargetDisks` utilize **`vmcp`** to link to the S10RWMNT and S10ROGLD minidisks.
- The next function, `copySystem`, calls `copyDisk` to copy all minidisks. The `copyDisk` function first tries to use the z/VM FLASHCOPY command. If that fails, it falls back to the Linux **`dasdfmt`** and **`dd`** commands.
- The next two function calls, `enableSourceDisks` and `enableTargetDisks`, utilize the Linux **`chccwdev`** command to enable the source and target disks.

- The next two function calls, `mountSourceRoot` and `mountTargetDisks`, mount the source and target disks following the Linux file system hierarchy over the directories /mnt/source/ and /mnt/target/.
- The `modifySystem` function is where some a lot occurs, so we analyze it in detail here. Example 24 shows the code.

Example 24 The modifySystem() function

```
function modifySystem()
 {
  TGT="/mnt/target"
  echo ""  1
  echo "Backing up and modifying /etc/init.d/boot script ..."
  cp $TGT/etc/init.d/boot $TGT/etc/init.d/boot.orig
  if [ "$?" != 0 ]; then exit 47; fi
  cat $TGT/etc/init.d/boot.orig | \
    sed -e 's:bootrc=/etc/init.d/boot.d:bootrc=/sbin/etc/init.d/boot.d:g' > \
    $TGT/etc/init.d/boot
  if [ "$?" != 0 ]; then exit 48; fi

  echo ""  2
  echo "Backing up and patching /etc/init.d/boot.rootfsck script ..."
  cp $TGT/etc/init.d/boot.rootfsck $TGT/etc/init.d/boot.rootfsck.orig
  if [ "$?" != 0 ]; then exit 49; fi
  patch $TGT/etc/init.d/boot.rootfsck < $rorDiffs
  if [ "$?" != 0 ]; then exit 50; fi

  echo ""  3
  echo "Backing up and copying modified /etc/fstab file ..."
  cp $TGT/etc/fstab $TGT/etc/fstab.orig
  if [ "$?" != 0 ]; then exit 51; fi
  cp $fstabFile $TGT/etc/fstab
  if [ "$?" != 0 ]; then exit 52; fi

  echo ""  4
  echo "Backing up and modifying /etc/zipl.conf file ..."
  cp $TGT/etc/zipl.conf $TGT/etc/zipl.conf.orig
  if [ "$?" != 0 ]; then exit 53; fi
  cat $TGT/etc/zipl.conf.orig | \
    sed -e 's/1b0-1bf/1b0(ro),1b1(ro),1b2-1b7,1b8(ro),1b9-1bf/g' > \
    $TGT/etc/zipl.conf
  cp $fstabFile $TGT/etc/fstab
  if [ "$?" != 0 ]; then exit 54; fi
```

In Example 24, the three files that are modified are backed up: /etc/init.d/boot, /etc/init.d/boot.rootfsck and /etc/fstab. In 1, the first script requires only a change to point to /sbin/etc/init.d/boot rather than /etc/init.d/boot. Therefore, the **sed** command is used.

In 2, the second script, /etc/init.d/boot.rootfsck is backed up and then modified using the **patch** command and the supplied diff file. Example 37 on page 58 shows the resulting modified code.

In 3, the file /etc/fstab is backed up and a modified copy is put in its place. Example 38 shows the contents of this file.

Finally, in **4** the parameter line in the file /etc/zipl.conf is modified so that the 1B0 (/boot/), 1B1 (root) and 1B8 (/usr/) disks get the **ro** parameter so the Linux kernel knows that they are read-only.

Then, the code continues as shown in Example 25.

Example 25 Continuation of code

```
  echo ""
  echo "Copying source /etc/, /root/ and /srv/ to target /local/ ..."
  cp -a $TGT/etc $TGT/local
  if [ "$?" != 0 ]; then exit 54; fi
  cp -a $TGT/root $TGT/local
  if [ "$?" != 0 ]; then exit 55; fi
  cp -a $TGT/srv $TGT/local
  if [ "$?" != 0 ]; then exit 56; fi
```

The lines in Example 25 make copies of /etc/, /root/, and /srv/ to /local/. These directories are three of the four directories that will be read-write in the read-only root system. (The fourth directory, /var/, is its own minidisk.) These three directories will be bind-mounted later from /local/ to their original slot by the modified **boot.rootfsck** script.

The code continues as shown in Example 26.

Example 26 Continuation of code

```
echo ""
  echo "Manipulating /etc/init.d and /sbin on $TGT ..."
  chroot $TGT mv /etc/init.d /sbin
  chroot $TGT ln -s /sbin/init.d /etc
 }
```

The lines in Example 26 move /etc/init.d/ to /sbin/init.d and then create a symbolic link from /etc/init.d/ back to /sbin/init.d/ so that /etc/init.d/ is available at boot time and before the correct contents of /etc/ are bind-mounted from /local/etc/.

- Finally, the `cleanup` function is called to unmount all mounted file systems, disable all devices, and then DETACH the minidisks.

Now it is time to run the **mkror.sh** script from the Linux system running on the LNXCLONE user ID. Your output should look similar to that shown in Example 27.

Example 27 Output of mkror.sh

```
# mkror.sh
Invoking CP command: QUERY S10RWMNT
HCPCQU045E S10RWMNT not logged on
Error: non-zero CP response for command 'QUERY S10RWMNT': #45
Invoking CP command: QUERY S10ROGLD
HCPCQU045E S10ROGLD not logged on
Error: non-zero CP response for command 'QUERY S10ROGLD': #45

Linking source disks ...
Invoking CP command: link S10RWMNT 1b1 11b1 rr
Invoking CP command: link S10RWMNT 1b0 11b0 rr
Invoking CP command: link S10RWMNT 1b4 11b4 rr
Invoking CP command: link S10RWMNT 1b5 11b5 rr
Invoking CP command: link S10RWMNT 1b6 11b6 rr
Invoking CP command: link S10RWMNT 1b7 11b7 rr
Invoking CP command: link S10RWMNT 1b8 11b8 rr

Linking target disks ...
Invoking CP command: link S10ROGLD 1b1 21b1 mr
Invoking CP command: link S10ROGLD 1b0 21b0 mr
Invoking CP command: link S10ROGLD 1b4 21b4 mr
Invoking CP command: link S10ROGLD 1b5 21b5 mr
Invoking CP command: link S10ROGLD 1b6 21b6 mr
Invoking CP command: link S10ROGLD 1b7 21b7 mr
Invoking CP command: link S10ROGLD 1b8 21b8 mr

Copying root file system ...

FLASHCOPYing 11b1 to 21b1 ...
Invoking CP command: FLASHCOPY 11b1 0 end to 21b1 0 end
Command complete: FLASHCOPY 11B1 0 END TO 21B1 0 END

Copying /boot/ ...
...
Copying minidisk swap space ...
...
Copying /local/ ...
...
Copying /var/ ...
...
Copying /usr/ ...
...
Copying /opt/ ...
...
Enabling source disks ...
Setting device 0.0.11b1 online
Done
WARNING: Device[0.0.11b0] is already  online
Done
WARNING: Device[0.0.11b5] is already  online
Done
WARNING: Device[0.0.11b6] is already  online
Done
WARNING: Device[0.0.11b7] is already  online
Done
WARNING: Device[0.0.11b8] is already  online
Done

Enabling target disks ...
Setting device 0.0.21b1 online
Done
Setting device 0.0.21b0 online
Done
Setting device 0.0.21b5 online
Done
Setting device 0.0.21b6 online
Done
Setting device 0.0.21b7 online
Done
Setting device 0.0.21b8 online
```

```
Done

Making source mount point ...

Mounting source root file system over /mnt/source ...

Mounting target file systems over /mnt/target ...

Making target mount points ...

Mounting memory file systems

Copying modified boot scripts and /etc/fstab to target ...

Copying source /etc/, /root/ and /srv/ to target /local/ ...

Manipulating /etc/init.d and /sbin on /mnt/target ...

Cleaning up ...
Setting device 0.0.21b0 offline
Done
Setting device 0.0.21b1 offline
Done
Setting device 0.0.21b5 offline
Done
Setting device 0.0.21b6 offline
Done
Setting device 0.0.21b7 offline
Done
Setting device 0.0.21b8 offline
Done
DASD 21B0 DETACHED
DASD 21B1 DETACHED
DASD 21B4 DETACHED
DASD 21B5 DETACHED
DASD 21B6 DETACHED
DASD 21B7 DETACHED
DASD 21B8 DETACHED
Setting device 0.0.11b1 offline
Done
DASD 11B1 DETACHED
Exiting with rc 0!
```

Because the target file systems are unmounted and minidisks are detached, you are now able to log on to S10ROGLD and try the new system.

Tip: If the script fails, you can immediately issue the command **`echo $?`** to get the return code. That value isolates the line in the script that is failing.

Also, if you run the script a second time after failing, you might get an error that a file system is already mounted over /mnt/source/ or /mnt/target/. You can workaround this error by unmounting the directory before running the script again. (You will need to code a cleanup function rather than just exiting.)

Testing the newly created system

You might receive error messages relating to the disk, such as a dasd(eckd) I/O status report for device 0.0.xxx. These message are due to a mismatch between VM having the disks as Read-Write and Linux as Read-Only. The messages are benign but can be eliminated by setting the guest directory to the right access (RW or RO).

Even though it has seven read-write minidisks, you configure it with most of the directories linked read-only. You can choose to verify that certain disks are read-only while others are read-write. Try the following commands:

```
# touch /etc/goo
# touch /var/goo
# touch /root/goo
# touch /srv/goo
# touch /tmp/goo
# touch /goo
touch: cannot touch `/goo': Read-only file system
# touch /opt/goo
touch: cannot touch `/opt/goo': Read-only file system
```

The first five commands succeed because those are the read-write directories. The last two commands fail because those directories are accessed read-only. You can choose to leave the empty goo files and verify that they remain in place across a reboot. After that is verified, you might want to delete the empty files so they do not get cloned.

Copying read-only Linux to gold disks

Shut down the read-only root system on S10ROGLD and log off. Log on to CMSCLONE and use the RO2GOLD EXEC to copy the contents to the 1xxx minidisks on S10GOLD, as shown in Example 28.

Example 28 Using RO2GOLD EXEC

```
==> ro2gold
Do you want to copy disks from S10ROGLD to S10GOLD? y/n
y

Copying minidisk 01B0 to 21B0 ...
Command complete: FLASHCOPY 01B0 0 END TO 21B0 0 END
Return value = 0

Copying minidisk 01B1 to 21B1 ...
Command complete: FLASHCOPY 01B1 0 END TO 21B1 0 END
Return value = 0

Copying minidisk 01B4 to 21B4 ...
Command complete: FLASHCOPY 01B4 0 END TO 21B4 0 END
Return value = 0

Copying minidisk 01B5 to 21B5 ...
Command complete: FLASHCOPY 01B5 0 END TO 21B5 0 END
Return value = 0

Copying minidisk 01B6 to 21B6 ...
Command complete: FLASHCOPY 01B6 0 END TO 21B6 0 END
Return value = 0
```

```
Copying minidisk 01B7 to 21B7 ...
Command complete: FLASHCOPY 01B7 0 END TO 21B7 0 END
Return value = 0

Copying minidisk 01B8 to 21B8 ...
Command complete: FLASHCOPY 01B8 0 END TO 21B8 0 END
Return value = 0
01B0-01B1 DETACHED
01B4-01B8 DETACHED
21B0-21B1 DETACHED
21B4-21B8 DETACHED
```

The modified read-only version of the gold read-write system is now “alongside” the read-write version on S10GOLD.

Cloning a read-only Linux

You can now clone a read-only Linux system. A user ID, LINUX002, is created to which to clone the system (Example 29). Note that only the 1B4 (swap), 1B5 (/local/), and 1B6 (/var/) minidisks are read-write. The other disks are read-only links to the corresponding S10GOLD 1xxx minidisks.

Example 29 Creating user ID LINUX002

```
USER LINUX002 PASSWD 256M 1G G
 INCLUDE RORDFLT
 OPTION APPLMON
 LINK S10GOLD 11B0 01B0 RR
 LINK S10GOLD 11B1 01B1 RR
 MDISK 01B4 3390 0001 0550 MM9135 MR PASSWD PASSWD PASSWD
 MDISK 01B5 3390 0551 0100 MM9135 MR PASSWD PASSWD PASSWD
 MDISK 01B6 3390 0651 0400 MM9135 MR PASSWD PASSWD PASSWD
 LINK S10GOLD 11B7 01B7 RR
 LINK S10GOLD 11B8 01B8 RR
```

You need to create a parameter file on the CMSCLONE 192 disk. The S10RWMNT parameter file is copied as a template, and the host name and IP address are modified, as shown in Example 30.

Example 30 Creating a parameter file on the CMSCLONE 192 disk

```
===> copy s10rwmnt parm-s10 d linux002 = =
===> x linux002 parm-s10 d
ramdisk_size=65536 root=/dev/ram1 ro init=/linuxrc TERM=dumb
HostIP=129.40.179.245 Hostname=ntc245.pbm.ihost.com
Gateway=129.40.179.254 Netmask=255.255.255.0
Broadcast=129.40.179.255 Layer2=0
ReadChannel=0.0.0600  WriteChannel=0.0.0601  DataChannel=0.0.0602
Nameserver=129.40.106.1 Portname=dontcare
Install=nfs://129.40.179.200/nfs/sles10/SLES-10-CD-s390x-GMC-CD1.iso
UseVNC=1  VNCPassword=123456
InstNetDev=osa OsaInterface=qdio OsaMedium=eth Manual=0
```

Be sure to give the new user ID access to the system VSWITCH. You are now ready to clone a read-only Linux system using the CLONERO EXEC, as shown in Example 31.

Example 31   Cloning a read-only Linux system using CLONERO EXEC

```
==> clonero linux002
Do you want to copy R/O system from S10GOLD to linux002 y/n
y
...
00: 21B4 21B5 21B6 01B4 01B5 01B6 DETACHED
```

When complete, you can log on to LINUX002 and boot from 1B0. A read-only root system will IPL. Again, you might see I/O errors early in the boot process similar to those shown in Example 32.

Example 32   I/O error messages

```
...
Mounting root /dev/dasdb1
dasd_erp(3990):  0.0.01b1: EXAMINE 24: Command Reject detected - fatal error
dasd(eckd): I/O status report for device 0.0.01b1:
dasd(eckd): in req: 000000000f6e9ee8 CS: 0x00 DS: 0x02
dasd(eckd): device 0.0.01b1: Failing CCW: 000000000f6e9fc8
dasd(eckd): Sense(hex)  0- 7: 80 02 00 00 00 00 00 00
...
```

These error messages appear to be benign. Determining the reason for these errors are left is an exercise for those readers with Linux expertise. If you solve this problem or have other problems, you can issue a posting on the linux-390 list server. For more information, see:

http://www2.marist.edu/htbin/wlvindex?linux-390

Other considerations

This paper describes a single solution. You can consider other changes such as:

- Utilizing logical volumes
- Implementing /home/ with automount, NFS, and LDAP
- Maintenance options

Utilizing logical volumes

Using Linux Logical Volume Manager (LVM) makes a lot of sense for file systems that are likely to grow large, such as /var/ and perhaps /srv/. We did not implement these file systems for this paper for simplicity and due to time constraints.

If you decide to implement your solution using logical volumes and if you plan to use the **`mkror.sh`** script, you need to modify the script accordingly.

Example 33 is a sample function to create logical volumes on the target system that was lightly tested. It targets device 2B0, which is linked read-write at virtual address 22B0, and the device file is set in the variable dev22b0.

Example 33 Sample function to create logical volumes

```
function makeLVs
 {
  echo ""
  echo "Making logical volumes ..."
  fdasd -a $dev22b0
  pvcreate "$dev22b0"1
  vgcreate rwVG "$dev22b0"1
  lvcreate -L 200M -n varLV rwVG
  lvcreate -L 1.2G -n srvLV rwVG
  mke2fs /dev/rwVG/varLV > /dev/null
  mke2fs /dev/rwVG/srvLV > /dev/null
  echo ""
  echo "Mounting logical volumes ..."
  vgscan
  vgchange -ay
  mount /dev/rw_vg/srv_lv /mnt/target/srv
  mount /dev/rw_vg/var_lv /mnt/target/var
 }
```

Implementing /home/ with automount, NFS, and LDAP

It is convenient for system administrators and developers to keep work in their own home directory under /home/. With tens or even hundreds of Linux systems, you would normally have tens or even hundreds of home directories. The size requirement for this file system varies widely from server to server.

It is easier to manage a single large /home/ directory *that follows users around*. You can implement this directory with the automount service, NFS, and LDAP. LDAP allows for central authentication and authorization of users and resources. Figure 14 shows a block diagram.

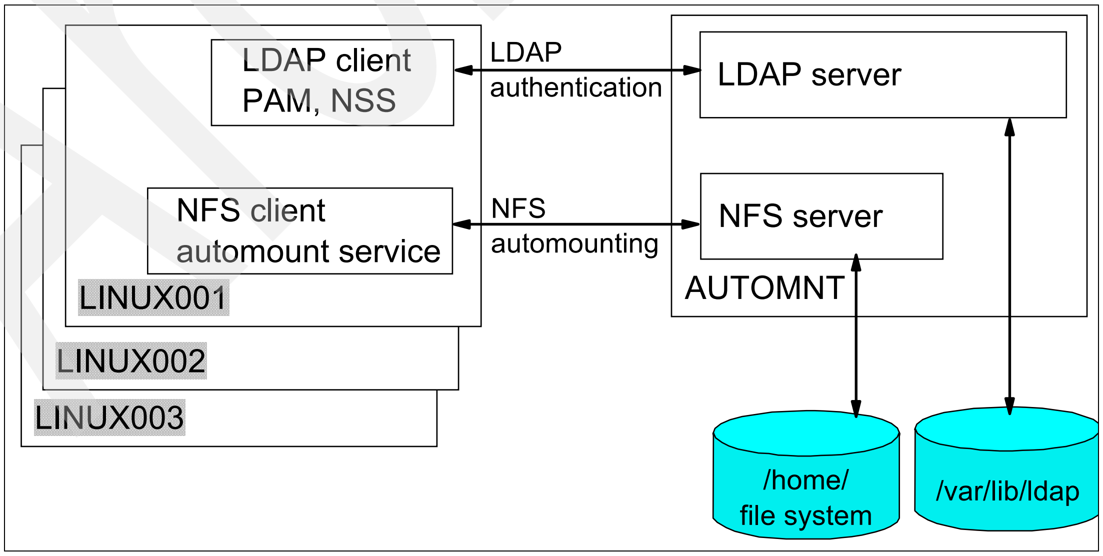


Figure 14 Automounted /home/ block diagram

For details on how to implement this directory, see *z/VM and Linux on IBM System z: The Virtualization Cookbook for SLES 10*, which is available at the following Web site:

`http://linuxvm.org/present`

Maintenance options

In this section, we discuss the following two aspects of maintenance:

- Updating the gold disks
- Updating the cloned servers

Updating the gold disks

If you need to modify the systems on the gold disks, you need to modify them in tandem (that is, modify both the read-write and read-only systems). In between times of maintenance, the read-write system on S10RWMNT, S10RWTST, and the 1xxx disks of S10GOLD all need to be identical. Similarly, the read-only system on S10ROGLD and the 2xxx disks of S10GOLD need to be the same.

To make any change, for example to add an RPM, we recommend the following steps:

1. Copy S10RWMNT to S10RWTST using the MNT2TST EXEC.
2. IPL the read-write system on S10RWTST.
3. Make the change (add the RPM) and test.
4. When the system is completely tested, copy the system to S10RWMNT using TST2MNT.
5. Recreate a read-only root system from S10RWMNT to S10ROGLD using **`mkror.sh`**.
6. Test the read-only root system.
7. When both systems are fully tested, copy them to the gold read-write disks using RW2GOLD and RO2GOLD.

Now, the change is reflected in any new Linux system that is cloned, whether it is read-write or read-only. Of course, this change does not affect any of the Linux systems that are already in existence.

Updating the cloned servers

Performing maintenance on existing Linux servers is a little more involved.

Note: The following approach is just one option and is very similar to the approach being used at Nationwide.

Assume that the gold disks are now at version 2, and assume that the Linux clones are still at version 1 and that the read-write directories have changed. You need to account for these changes.

Here are the steps:

1. Make a copy of each cloned server that is to be updated, allowing for *fallback* if necessary. After the update, you need to preserve these disks for a suitable period of time.
2. Next, create duplicate minidisks for the read-write data. In this paper, this step means creating duplicates of the minidisks 1B5 and 1B6. Temporarily, put these duplicates at different addresses, for example 5B5 and 5B6. Copy the contents of the *version 2* gold disks to these new disks.

3. Then, merge the contents of the cloned server's read-write disks (1B5 copy to 5B5, 1B6 copy to 5B6). New files that did not exist in version 1 of the gold disks are now on the new disks. You need to copy files on the old minidisks (1B1,1B6) but not on the new version (5B1,5B6) onto the new.
4. Finally, review the files that are in common between the two read write volumes but that have been updated by maintenance using the **`rsync`** and **`diff`** commands.

After you have made these changes, the server can use the updated read-only root minidisks. If there is any problem, you can roll back the system to the previous version of gold minidisks.

Contents of tar file

The redp4322.tgz file is a compressed tar file that contains the following files and directories:

README.TXT	The readme file
sbin/	The Linux code. See "z/VM code" (which is the next section). The subdirectory is named sbin/ so that you can untar it from /usr/local/ and have the scripts in your default PATH.
vm/	The z/VM code. See "z/VM code" on page 62

Linux code

The section describes the code that runs on Linux and one modified configuration file:

boot.findself	Script to obtain IP information on first boot of a cloned system
boot.rootfsck.diffs	Differences to apply to /etc/init.d/boot.rootfsck
fstab.ror	Modified /etc/fstab file for read-only root systems
mkror.sh	Script to create a read-only root system from a read-write system

The boot.findself script

This script is invoked once, the first time a clone Linux system is booted. It does not take any action on the user ID's S10RWMNT, S10RWTST, or S10ROGLD.

Note: Note that these names are hard-coded in the script, so if you choose different names for the ID's you will have to modify the code accordingly.

The **`boot.findself`** script enables the ID's 191 disk, which is the CMSCLONE 192 disk and contains the source parameter file S10RWMNT PARM-S10 and the target parameter file where the file name is the same as the target user ID name.

Example 34 shows the **`boot.findself`** script.

Example 34 The boot.findself script

```
#!/bin/bash
#
# /etc/init.d/boot.findself
#
### BEGIN INIT INFO
```

```
# Provides:          boot.findself
# Required-Start:    boot.localfs
# Required-Start:
# Required-Stop:
# Default-Start:     B
# Default-Stop:
# Description:       upon first boot find/modify IP@ + hostname, gen SSH keys
### END INIT INFO
#
# This script requires two SLES 10 parameter files to exist on the user ID's
# 191 disk: (1) the file S10RWMNT PARM-S10 and (2) $userid PARM-S10 where
# $userid is the ID of the user that is running the script. It then modifies
# the IP address, Host name and fully qualified domain name in three
# configuration files that contain this info. It also regenerates SSH keys.
# The script then turns itself off via "chkconfig" so it only runs once.
#
# IBM DOES NOT WARRANT OR REPRESENT THAT THE CODE PROVIDED IS COMPLETE
# OR UP-TO-DATE.  IBM DOES NOT WARRANT,  REPRESENT OR IMPLY RELIABILITY,
# SERVICEABILITY OR FUNCTION OF THE CODE.  IBM IS UNDER NO OBLIGATION TO
# UPDATE  CONTENT NOR PROVIDE FURTHER SUPPORT.
# ALL CODE IS PROVIDED "AS IS," WITH NO WARRANTIES OR GUARANTEES WHATSOEVER.
# IBM EXPRESSLY DISCLAIMS TO THE  FULLEST EXTENT PERMITTED BY LAW ALL EXPRESS,
# IMPLIED,  STATUTORY AND OTHER WARRANTIES, GUARANTEES, OR  REPRESENTATIONS,
# INCLUDING, WITHOUT LIMITATION, THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR
# A PARTICULAR  PURPOSE, AND NON-INFRINGEMENT OF PROPRIETARY AND INTELLECTUAL
# PROPERTY RIGHTS.  YOU UNDERSTAND AND AGREE THAT  YOU USE THESE MATERIALS,
# INFORMATION, PRODUCTS, SOFTWARE, PROGRAMS, AND SERVICES, AT YOUR OWN
# DISCRETION AND  RISK AND THAT YOU WILL BE SOLELY RESPONSIBLE FOR ANY DAMAGES
# THAT MAY RESULT, INCLUDING LOSS OF DATA OR DAMAGE TO YOUR COMPUTER SYSTEM.
# IN NO EVENT WILL IBM BE LIABLE TO ANY PARTY FOR ANY DIRECT, INDIRECT,
# INCIDENTAL, SPECIAL, EXEMPLARY OR  CONSEQUENTIAL DAMAGES OF ANY TYPE
# WHATSOEVER RELATED TO OR ARISING FROM USE OF THE CODE FOUND HEREIN, WITHOUT
# LIMITATION, ANY LOST PROFITS, BUSINESS INTERRUPTION, LOST SAVINGS, LOSS OF
# PROGRAMS OR OTHER DATA, EVEN IF IBM IS EXPRESSLY ADVISED OF THE POSSIBILITY
# OF SUCH DAMAGES. THIS EXCLUSION AND WAIVER OF LIABILITY APPLIES TO ALL
# CAUSES OF ACTION, WHETHER BASED ON CONTRACT, WARRANTY, TORT OR ANY OTHER
# LEGAL THEORIES.
#
#+--------------------------------------------------------------------------+
function findID()
# Get my VM user ID - don't find self on S10RWMNT, S10RWTST or S10RWGLD
#+--------------------------------------------------------------------------+
 {
  targetID=$(cat /proc/sysinfo | grep "VM00 Name" | awk '{print $3}')
  if [ $targetID = "S10RWMNT" ] || [ $targetID = "S10RWTST" ] || \
     [ $targetID = "S10ROGLD" ]; then # don't do anything on these three IDs
    exit
  fi
 }

#+--------------------------------------------------------------------------+
function enableAdisk()
# Enable my 191 (A) disk
#+--------------------------------------------------------------------------+
 {
  chccwdev -e 191 > /dev/null 2>&1
  rc=$?
  if [ $rc != 0 ]; then # unable to enable 191 disk
    echo "$0: Unable to enable 191, rc from chccwdev = $rc"
    exit 1
  fi
  sleep 1# wait a sec to be sure disk is ready
  Adisk=/dev/$(egrep '^0.0.0191' /proc/dasd/devices | awk '{print $7}')
 }

#+---------------------------------------------------------------------------+
function findSourceIP()
# Get the source IP address and hostName
#+---------------------------------------------------------------------------+
 {
  sourceParm="$sourceID.$parmType"
  cmsfslst -d $Adisk | grep $sourceID | grep $parmType > /dev/null
  rc=$?
  if [ $rc != 0 ]; then
    echo "$0: $sourceParm not found on 191 minidisk. Exiting"
    exit 2
  fi
  export local $(cmsfscat -a -d $Adisk $sourceParm)
  # set global variable names escaping any dots (.) in the strings
  sourceName=$(echo "$Hostname" | sed -e 's:\.:\\\.:g')
  sourceHost=${Hostname%%.*}  # Chop domain name off to leave host name
  sourceIP=$(echo "$HostIP" | sed -e 's:\.:\\\.:g')
  sourceOSA=$(echo "$ReadChannel " | sed -e 's:\.:\\\.:g')
 }

#+---------------------------------------------------------------------------+
function findTargetIP()
# Get my new IP address and hostname
#+---------------------------------------------------------------------------+
 {
  targetParm="$targetID.$parmType"
  cmsfslst -d $Adisk | grep $targetID | grep $parmType > /dev/null
  rc=$?
  if [ $rc != 0 ]; then
    echo "$0: $targetParm not found on 191 minidisk. Exiting"
    exit 3
  fi
  export local $(cmsfscat -a -d $Adisk $targetParm)
  targetName=$Hostname
  targetHost=${Hostname%%.*}  # Chop domain name off to leave host name
  targetIP=$HostIP
 }

#+---------------------------------------------------------------------------+
function modifyIP()
# Modify IP address and host name in /etc/HOSTNAME, /etc/hosts and
#  /etc/sysconfig/network/ifcfg-qeth-bus-ccw-$ReadChannel
#+---------------------------------------------------------------------------+
 {
  # TODO: this function should also modify, DNS, Gateway, broadcast, etc.
  eth0file="/etc/sysconfig/network/ifcfg-qeth-bus-ccw-$ReadChannel"
  echo "$0: changing (escaped) $sourceName to $targetName in /etc/HOSTNAME"
  sed --in-place -e "s/$sourceName/$targetName/g" /etc/HOSTNAME
  echo "$0: changing $sourceHost to $targetHost and IP address in /etc/hosts"
  sed --in-place -e "s/$sourceHost/$targetHost/g" \
                 -e "s/$sourceIP/$targetIP/g" /etc/hosts
  echo "$0: changing (escaped) $sourceIP to $targetIP in $eth0file"
  sed --in-place -e "s/$sourceIP/$targetIP/g" $eth0file
 }

#+--------------------------------------------------------------------------+
function genSSHkeys()
# Regenerate a set of SSH keys
#+--------------------------------------------------------------------------+
 {
  rm /etc/ssh/ssh_host_*
  ssh-keygen -t rsa -N "" -q -f /etc/ssh/ssh_host_rsa_key
  ssh-keygen -t dsa -N "" -q -f /etc/ssh/ssh_host_dsa_key
  ssh-keygen -t rsa1 -N "" -q -f /etc/ssh/ssh_host_key
 }

# main()
# global variables
sourceID="S10RWMNT"       # VM user ID where first Linux was installed
parmType="PARM-S10"       # File type of parameter file on 191 disk

# function calls
findID
enableAdisk
findSourceIP
findTargetIP
modifyIP
genSSHkeys
chkconfig boot.findself off  # run only once => turn self off
```

The cloneprep script

This script prepares the Linux system on S10RWMNT to be cloned. Example 35 shows the **cloneprep** script.

Example 35 The cloneprep script.

```
#!/bin/bash
# script to clean up files before cloning
#+--------------------------------------------------------------------------+
function cleanFile()
# delete file, create empty file and set permission mode
# arg 1: file to delete and create
# arg 2: mode to set empty file to
#+--------------------------------------------------------------------------+
 {
  if [ -f $1 ]; then # the file exists
    rm $1
  fi
  touch $1
  chmod $2 $1
 }

# main()
# clean up certain files in /var/log
rm /var/log/YaST2/y2log-*
rm /var/log/*.gz
cleanFile /var/log/authlog 600
cleanFile /var/log/faillog 600
cleanFile /var/log/lastlog 644
cleanFile /var/log/secure 600
cleanFile /var/log/secure 600
cleanFile /root/.bash_history 600

echo "System should be ready for shutdown and cloning"
```

The mkror.sh script

This script clones from S10RWMNT to S10ROGLD. Example 36 shows the **mkror.sh** script.

Example 36 The mkror.sh script

```
#!/bin/sh
# mkror.sh - script to create a read-only root system on target user ID
# Hard-coded virtual device addresses:
#  1b0 - /boot
#  1b1 - /
#  1b2 - swap (VDISK)
#  1b3 - swap (VDISK)
#  1b4 - swap
#  1b5 - /local
#  1b6 - /var
#  1b7 - /usr
#  1b8 - /opt
#
# Source disks are linked as 1xxx
# Target disks are linked as 2xxx
#
# IBM DOES NOT WARRANT OR REPRESENT THAT THE CODE PROVIDED IS COMPLETE
# OR UP-TO-DATE.  IBM DOES NOT WARRANT,  REPRESENT OR IMPLY RELIABILITY,
# SERVICEABILITY OR FUNCTION OF THE CODE.  IBM IS UNDER NO OBLIGATION TO
# UPDATE  CONTENT NOR PROVIDE FURTHER SUPPORT.
# ALL CODE IS PROVIDED "AS IS," WITH NO WARRANTIES OR GUARANTEES WHATSOEVER.
# IBM EXPRESSLY DISCLAIMS TO THE  FULLEST EXTENT PERMITTED BY LAW ALL EXPRESS,
# IMPLIED,  STATUTORY AND OTHER WARRANTIES, GUARANTEES, OR  REPRESENTATIONS,
# INCLUDING, WITHOUT LIMITATION, THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR
# A PARTICULAR  PURPOSE, AND NON-INFRINGEMENT OF PROPRIETARY AND INTELLECTUAL
# PROPERTY RIGHTS.  YOU UNDERSTAND AND AGREE THAT  YOU USE THESE MATERIALS,
# INFORMATION, PRODUCTS, SOFTWARE, PROGRAMS, AND SERVICES, AT YOUR OWN
# DISCRETION AND  RISK AND THAT YOU WILL BE SOLELY RESPONSIBLE FOR ANY DAMAGES
# THAT MAY RESULT, INCLUDING LOSS OF DATA OR DAMAGE TO YOUR COMPUTER SYSTEM.
# IN NO EVENT WILL IBM BE LIABLE TO ANY PARTY FOR ANY DIRECT, INDIRECT,
# INCIDENTAL, SPECIAL, EXEMPLARY OR  CONSEQUENTIAL DAMAGES OF ANY TYPE
# WHATSOEVER RELATED TO OR ARISING FROM USE OF THE CODE FOUND HEREIN, WITHOUT
# LIMITATION, ANY LOST PROFITS, BUSINESS INTERRUPTION, LOST SAVINGS, LOSS OF
# PROGRAMS OR OTHER DATA, EVEN IF IBM IS EXPRESSLY ADVISED OF THE POSSIBILITY
# OF SUCH DAMAGES. THIS EXCLUSION AND WAIVER OF LIABILITY APPLIES TO ALL
# CAUSES OF ACTION, WHETHER BASED ON CONTRACT, WARRANTY, TORT OR ANY OTHER
# LEGAL THEORIES.

#+--------------------------------------------------------------------------+
function CPcmd()
# Run a CP command and invoke it via the vmcp module/command
#   Arg1-n: the command to issue
#   Return: the command's return code
#+--------------------------------------------------------------------------+
 {
  echo "Invoking CP command: $@"
# parse output to get return code: awk -F# splits line at '#' with rc at end
  output=$(vmcp $@ 2>&1)
  if [ "X$output" != "X" ]; then # there is some output - echo it
    echo "$output"
  fi
  retVal=0
  retVal=$(echo $output | grep "Error: non-zero CP" | awk -F# '{print $2}')
  return $retVal
 }

#+--------------------------------------------------------------------------+
function checkID()
#   Arg 1: User ID to verify
# Verify user ID exists and is logged off
#+--------------------------------------------------------------------------+
 {
  userID=$1
  CPcmd QUERY $userID
  rc=$?
  case $rc in
    0)  # user ID is logged on or disconnected
      echo "Error: $userID user ID must be logged off"
      exit 2
      ;;
    3)  # user ID does not exist
      echo "Error: $ID user ID does not exist"
      exit 3
      ;;
   45) # user ID is logged off - this is correct - fall through
      ;;
   *) # unexpected
      echo "Unexpected rc from QUERY: $rc"
      echo "$targetID user ID must exist and be logged off"
      exit 4
  esac
 }

#+--------------------------------------------------------------------------+
function linkSourceDisks()
# Link R/O the source disks 1B0 - 1B8 as 1xxx
#+--------------------------------------------------------------------------+
 {
  echo ""
  echo "Linking source disks ..."
  CPcmd link $sourceID 1b1 11b1 rr
  if [ $? != 0 ]; then exit 5; fi
  CPcmd link $sourceID 1b0 11b0 rr
  if [ $? != 0 ]; then exit 6; fi
  CPcmd link $sourceID 1b4 11b4 rr
  if [ $? != 0 ]; then exit 7; fi
  CPcmd link $sourceID 1b5 11b5 rr
  if [ $? != 0 ]; then exit 8; fi
  CPcmd link $sourceID 1b6 11b6 rr
  if [ $? != 0 ]; then exit 9; fi
  CPcmd link $sourceID 1b7 11b7 rr
  if [ $? != 0 ]; then exit 10; fi
  CPcmd link $sourceID 1b8 11b8 rr
  if [ $? != 0 ]; then exit 11; fi
 }

#+--------------------------------------------------------------------------+
function linkTargetDisks()
# Link R/W the target disks 2b0-2b8 as 2xxx
#+--------------------------------------------------------------------------+
 {
  echo ""
  echo "Linking target disks ..."
  CPcmd link $targetID 1b1 21b1 mr
  if [ $? != 0 ]; then exit 12; fi
  CPcmd link $targetID 1b0 21b0 mr
  if [ $? != 0 ]; then exit 13; fi
  CPcmd link $targetID 1b4 21b4 mr
  if [ $? != 0 ]; then exit 14; fi
  CPcmd link $targetID 1b5 21b5 mr
  if [ $? != 0 ]; then exit 15; fi
  CPcmd link $targetID 1b6 21b6 mr
  if [ $? != 0 ]; then exit 16; fi
  CPcmd link $targetID 1b7 21b7 mr
  if [ $? != 0 ]; then exit 17; fi
  CPcmd link $targetID 1b8 21b8 mr
  if [ $? != 0 ]; then exit 18; fi
 }

#+--------------------------------------------------------------------------+
function enableSourceDisks()
# Enable the source and target disks (except swap disks x1b4)
#+--------------------------------------------------------------------------+
 {
  echo ""
  echo "Enabling source disks ..."
  chccwdev -e 11b1
  if [ $? != 0 ]; then exit 19; fi
  chccwdev -e 11b0
  if [ $? != 0 ]; then exit 20; fi
  chccwdev -e 11b5
  if [ $? != 0 ]; then exit 21; fi
  chccwdev -e 11b6
  if [ $? != 0 ]; then exit 22; fi
  chccwdev -e 11b7
  if [ $? != 0 ]; then exit 23; fi
  chccwdev -e 11b8
  if [ $? != 0 ]; then exit 24; fi
 }

#+--------------------------------------------------------------------------+
function enableTargetDisks()
# Enable the source and target disks (except swap disks x1b4)
#+--------------------------------------------------------------------------+
 {
  echo ""
  echo "Enabling target disks ..."
  chccwdev -e 21b1
  if [ $? != 0 ]; then exit 25; fi
  chccwdev -e 21b0
  if [ $? != 0 ]; then exit 26; fi
  chccwdev -e 21b5
  if [ $? != 0 ]; then exit 27; fi
  chccwdev -e 21b6
  if [ $? != 0 ]; then exit 28; fi
  chccwdev -e 21b7
  if [ $? != 0 ]; then exit 29; fi
  chccwdev -e 21b8
  if [ $? != 0 ]; then exit 30; fi
 }

#+--------------------------------------------------------------------------+
function copyDisk()
# Use FLASHCOPY to copy a disk, if it fails, fall back to dasdfmt then dd
#   Arg 1: Source vaddr
#   Arg 2: Target vaddr
#+--------------------------------------------------------------------------+
 {
  source=$1
  target=$2
  echo ""
  echo "FLASHCOPYing $source to $target ..."
  CPcmd FLASHCOPY $source 0 end to $target 0 end
  if [ $? != 0 ]; then
    echo "FLASHCOPY failed, falling back to dasdfmt and dd ..."
    sDev=/dev/$(egrep ^0.0.$source /proc/dasd/devices | awk '{ print $7 }')
    if [ "$?" != 0 ]; then exit 31; fi
    tDev=/dev/$(egrep ^0.0.$target /proc/dasd/devices | awk '{ print $7 }')
    if [ "$?" != 0 ]; then exit 32; fi
    echo ""
    echo "dasdfmt-ing $tDev ..."
    dasdfmt -y -b 4096 -f $tDev
    if [ "$?" != 0 ]; then exit 33; fi
    echo ""
    echo "dd-ing $sDev to $tDev ..."
    dd bs=4096 if=$sDev of=$tDev
    if [ "$?" != 0 ]; then exit 34; fi
    echo ""
    echo "disabling and re-enabling $target ..."
    chccwdev -d $target
    if [ $? != 0 ]; then exit 35; fi
    chccwdev -e $target
    if [ $? != 0 ]; then exit 36; fi
  fi
 }

#+--------------------------------------------------------------------------+
function copySystem()
# Copy /mnt/source to /mnt/target
#+--------------------------------------------------------------------------+
 {
  echo ""
  echo "Copying root file system ..."
  copyDisk 11b1 21b1

  echo ""
  echo "Copying /boot/ ..."
  copyDisk 11b0 21b0

  echo ""
  echo "Copying minidisk swap space ..."
  copyDisk 11b4 21b4

  echo ""
  echo "Copying /local/ ..."
  copyDisk 11b5 21b5

  echo ""
  echo "Copying /var/ ..."
  copyDisk 11b6 21b6

  echo ""
  echo "Copying /usr/ ..."
  copyDisk 11b7 21b7

  echo ""
  echo "Copying /opt/ ..."
  copyDisk 11b8 21b8
 }

#+--------------------------------------------------------------------------+
function mountSourceRoot()
# Mount disk at 11b1 over /mnt/source, then make mount points.
# Then mount disks at 11be over usr, 11bf over opt and 11b0 over boot
#+--------------------------------------------------------------------------+
 {
  echo ""
  echo "Making source mount point ..."
  if [ ! -d /mnt/source ]; then
    mkdir /mnt/source
    if [ "$?" != 0 ]; then exit 37; fi
  fi
  echo ""
  echo "Mounting source root file system over /mnt/source ..."
  dev11b1=/dev/$(egrep '^0.0.11b1'  /proc/dasd/devices | awk '{ print $7 }')1
  mount -o ro $dev11b1 /mnt/source
#  if [ "$?" != 0 ]; then exit 38; fi
 }

#+--------------------------------------------------------------------------+
function mountTargetDisks()
# Mount disk at 21b1 over /mnt/target, then make mount points.
# Then mount disks at 21be over usr, 21bf over opt and 21b0 over boot
#+--------------------------------------------------------------------------+
 {
  dev21b0=/dev/$(egrep '^0.0.21b0'  /proc/dasd/devices | awk '{ print $7 }')
  dev21b1=/dev/$(egrep '^0.0.21b1'  /proc/dasd/devices | awk '{ print $7 }')
  dev21b5=/dev/$(egrep '^0.0.21b5'  /proc/dasd/devices | awk '{ print $7 }')
  dev21b6=/dev/$(egrep '^0.0.21b6'  /proc/dasd/devices | awk '{ print $7 }')
  dev21b7=/dev/$(egrep '^0.0.21b7'  /proc/dasd/devices | awk '{ print $7 }')
  dev21b8=/dev/$(egrep '^0.0.21b8'  /proc/dasd/devices | awk '{ print $7 }')

  echo ""
  echo "Mounting target file systems over /mnt/target ..."
  mkdir /mnt/target
#  if [ "$?" != 0 ]; then exit 39; fi
  mount "$dev21b1"1 /mnt/target
  if [ "$?" != 0 ]; then exit 40; fi
  echo ""
  echo "Making target mount points ..."
  mkdir -p /mnt/target/{boot,usr,var,opt,local,sys,proc}
  mount "$dev21b0"1 /mnt/target/boot
  if [ "$?" != 0 ]; then exit 41; fi
  mount "$dev21b5"1 /mnt/target/local
  if [ "$?" != 0 ]; then exit 42; fi
  mount "$dev21b6"1 /mnt/target/var
  if [ "$?" != 0 ]; then exit 43; fi
  mount "$dev21b7"1 /mnt/target/usr
  if [ "$?" != 0 ]; then exit 44; fi
  mount "$dev21b8"1 /mnt/target/opt
  if [ "$?" != 0 ]; then exit 45; fi
  echo ""
  echo "Mounting memory file systems ..."
  mount -t sysfs sysfs /mnt/target/sys
  if [ "$?" != 0 ]; then exit 46; fi
  mount -t proc proc /mnt/target/proc
  if [ "$?" != 0 ]; then exit 47; fi
 }

#+--------------------------------------------------------------------------+
function modifySystem()
# 1) Copy modified /etc/init.d/boot, /etc/init.d/boot.rootfsck and /etc/fstab
# 2) Copy source /etc/ and /root/ directories to target /local/
# 3) Move /etc/init.d under /sbin/ and create symlink to point back
#+--------------------------------------------------------------------------+
 {
  TGT="/mnt/target"
  echo ""
  echo "Backing up and modifying /etc/init.d/boot script ..."
  cp $TGT/etc/init.d/boot $TGT/etc/init.d/boot.orig
  if [ "$?" != 0 ]; then exit 48; fi
  cat $TGT/etc/init.d/boot.orig | \
    sed -e 's:bootrc=/etc/init.d/boot.d:bootrc=/sbin/etc/init.d/boot.d:g' > \
    $TGT/etc/init.d/boot
  if [ "$?" != 0 ]; then exit 49; fi

  echo ""
  echo "Backing up and patching /etc/init.d/boot.rootfsck script ..."
  cp $TGT/etc/init.d/boot.rootfsck $TGT/etc/init.d/boot.rootfsck.orig
  if [ "$?" != 0 ]; then exit 50; fi
  patch $TGT/etc/init.d/boot.rootfsck < $rorDiffs
  if [ "$?" != 0 ]; then exit 51; fi

  echo ""
  echo "Backing up and copying modified /etc/fstab file ..."
  cp $TGT/etc/fstab $TGT/etc/fstab.orig
  if [ "$?" != 0 ]; then exit 52; fi
  cp $fstabFile $TGT/etc/fstab
  if [ "$?" != 0 ]; then exit 53; fi

  echo ""
  echo "Backing up and modifying /etc/zipl.conf file ..."
  cp $TGT/etc/zipl.conf $TGT/etc/zipl.conf.orig
  if [ "$?" != 0 ]; then exit 53; fi
  cat $TGT/etc/zipl.conf.orig | \
    sed -e 's/1b0-1bf/1b0(ro),1b1(ro),1b2-1b7,1b8(ro),1b9-1bf/g' > \
    $TGT/etc/zipl.conf
  cp $fstabFile $TGT/etc/fstab
  if [ "$?" != 0 ]; then exit 54; fi

  echo ""
  echo "Running zipl in target environment ..."
  chroot $TGT zipl
  if [ "$?" != 0 ]; then exit 55; fi

  echo ""
  echo "Copying source /etc/, /root/ and /srv/ to target /local/ ..."
  cp -a $TGT/etc $TGT/local
  if [ "$?" != 0 ]; then exit 56; fi
  cp -a $TGT/root $TGT/local
  if [ "$?" != 0 ]; then exit 57; fi
  cp -a $TGT/srv $TGT/local
  if [ "$?" != 0 ]; then exit 58; fi

  echo ""
  echo "Manipulating /etc/init.d and /sbin on $TGT ..."
  chroot $TGT mv /etc/init.d /sbin
  chroot $TGT ln -s /sbin/init.d /etc
 }

#+--------------------------------------------------------------------------+
function cleanUp()
# Unmount source and target file systems and detach minidisks
```

```
#+--------------------------------------------------------------------------+
 {
  echo ""
  echo "Cleaning up ..."
  # clean up target disks
  umount /mnt/target/opt
  umount /mnt/target/var
  umount /mnt/target/usr
  umount /mnt/target/local
  umount /mnt/target/boot
  umount /mnt/target/proc
  umount /mnt/target/sys
  umount /mnt/target
  chccwdev -d 21b0
  chccwdev -d 21b1
  chccwdev -d 21b5
  chccwdev -d 21b6
  chccwdev -d 21b7
  chccwdev -d 21b8
  vmcp det 21b0
  vmcp det 21b1
  vmcp det 21b4
  vmcp det 21b5
  vmcp det 21b6
  vmcp det 21b7
  vmcp det 21b8

  # clean up source disks
  umount /mnt/source
  chccwdev -d 11b1
  vmcp det 11b1
 }

# main()
# global variables
sourceID="S10RWMNT"
targetID="S10ROGLD"
rorDiffs="/usr/local/sbin/boot.rootfsck.diffs"
fstabFile="/usr/local/sbin/fstab.ror"

# function calls
checkID $sourceID
checkID $targetID
linkSourceDisks
linkTargetDisks
enableSourceDisks
enableTargetDisks
copySystem
mountSourceRoot
mountTargetDisks
modifySystem
cleanUp
echo "Exiting with rc 0!"
exit 0
```

Modified boot.rootfsck file

Only the differences to this file are shipped. The **mkror.sh** script applies these differences to the startup script **/etc/init.d/boot.rootfsck**. Example 37 shows the resulting script.

Example 37 The modified boot.rotfsck file

```
#! /bin/sh
#
# Copyright (c) 2001-2002 SuSE Linux AG, Nuernberg, Germany.
# All rights reserved.
#
# /etc/init.d/boot.readonlyroot
# derived from /etc/init.d/boot.rootfsck (SuSE)
#
### BEGIN INIT INFO
# Provides:          boot.readonlyroot
# Required-Start:
# Required-Stop:
# Default-Start:     B
# Default-Stop:
# Description:       check and mount /local filesystem
### END INIT INFO

. /etc/rc.status

# to get max number of parallel fsck processes
. /etc/sysconfig/boot

if [ -f /etc/sysconfig/dump ]; then
   . /etc/sysconfig/dump
fi

export FSCK_MAX_INST

rc_reset

case "$1" in
  start)
   # ROR: add 1
   echo "Read-only root: In modified $0"

   #
   # fsck may need a huge amount of memory, so make sure, it is there.
   #
   # However, in the case of an active LKCD configuration, we need to
   # recover the crash dump from the swap partition first, so we cannot
   # yet activate them.
   #
   if [ "$DUMP_ACTIVE" != "1" ]; then
      echo "Activating swap-devices in /etc/fstab..."
      swapon -ae &> /dev/null
      rc_status -v1 -r
   fi

   #
   # If we use a serial console, don't use the fsck progress bar
   #
   FSCK_PROGRESSBAR="-V"
        [ -x /sbin/showconsole ] && [ "`/sbin/showconsole`" = "/dev/tty1" ] &&
FSCK_PROGRESSBAR="-C"
   #
   # do fsck and start sulogin, if it fails.
   #
   FSCK_RETURN=0
   MAY_FSCK=1

   # we may get ROOTFS_BLKDEV passed from initrd, skip extra actions
   if [ -n "$ROOTFS_BLKDEV" ]  ; then
       if [ -n "$ROOTFS_REALDEV" ]  ; then
      ROOTFS_BLKDEV=$ROOTFS_REALDEV
       fi
   else
       # if not booted via initrd, /dev is empty.
       # use private devnode with proper permissions
       ROOTFS_BLKDEV="/dev/shm/root"

       rootcpio=`echo / | /bin/cpio --quiet -o -H newc`
       rootmajor=0x${rootcpio:62:8}
       rootminor=0x${rootcpio:70:8}
       if [ $((rootmajor)) -ne 0 ] ; then
       echo /bin/mknod -m600 $ROOTFS_BLKDEV b $((rootmajor)) $((rootminor))
       /bin/mknod -m600 $ROOTFS_BLKDEV b $((rootmajor)) $((rootminor))
       fi
       MAY_FSCK=0
       if test -n "$ROOTFS_BLKDEV" -a "$ROOTFS_BLKDEV" != "/" -a -b
"$ROOTFS_BLKDEV" ; then
      MAY_FSCK=1
       fi
        fi
   #
        FSCK_FORCE=""
   if test -f /forcefsck ; then
       FSCK_FORCE="-f"
       ROOTFS_FSCK=""
        fi
   if test "$ROOTFS_FSCK" = "0" ; then
       # already checked and ok, skip the rest
            MAY_FSCK=0
        fi
   if test ! -f /fastboot -a -z "$fastboot" -a $MAY_FSCK -eq 1 ; then
       # on an umsdos root fs this mount will fail,
       # so direct error messages to /dev/null.
       # this seems to be ugly, but should not really be a problem.
       mount -n -o remount,ro / 2> /dev/null
       if test $? = 0; then
      if test -n "$ROOTFS_FSCK" ; then
          FSCK_RETURN=$ROOTFS_FSCK
      else
          # ROR: chg 2
#         echo "Checking root file system..."
#         fsck $FSCK_PROGRESSBAR -a $FSCK_FORCE $ROOTFS_BLKDEV
                    echo "Checking /local filesystem..."
                    fsck $FSCK_PROGRESSBAR -a $FSCK_FORCE /local
          # A return code of 1 indicates that file system errors
          # were corrected, but that the boot may proceed.
          # A return code of 2 or larger indicates failure.
          FSCK_RETURN=$?
      fi
      test $FSCK_RETURN -lt 4
      rc_status -v1 -r
      if test $FSCK_RETURN -gt 1 -a $FSCK_RETURN -lt 4 ; then
          # if appropriate, switch bootsplash to verbose
          # mode to make text messages visible.
          test -f /proc/splash && echo "verbose" > /proc/splash
          echo
          echo "fsck succeed, but reboot is required."
          echo
          sleep 1
          sync
          reboot -f
      elif test $FSCK_RETURN -gt 3; then
          # if appropriate, switch bootsplash to verbose
          # mode to make text messages visible.
          test -f /proc/splash && echo "verbose" > /proc/splash
          # Stop blogd since we reboot after sulogin
          test -x /sbin/blogd && killproc -QUIT /sbin/blogd
          if test -x /etc/init.d/kbd ; then
         /etc/init.d/kbd start
          fi
               echo
               echo "fsck failed.  Please repair manually and reboot."
               echo "The /local file system is currently mounted read-only.
          echo To remount it read-write do:"
               echo
               echo "   bash# mount -n -o remount,rw /local"
               echo
               echo "Attention: Only CONTROL-D will reboot the system in this"
               echo "maintanance mode. shutdown or reboot will not work."
               echo
               PS1="(repair filesystem) # "
               export PS1
               /sbin/sulogin /dev/console

               # if the user has mounted something rw, this should be umounted
               echo "Unmounting file systems (ignore error messages)"
               umount -avn

               # on umsdos fs this would lead to an error message.
          # so direct errors to /dev/null
                    mount -no remount,ro / 2> /dev/null

               sync
               reboot -f
           fi
           sync
```

```
#     ROR: del 1, add 4
#          mount -n -o remount,rw /
               mount -n -o /local
               mount -n -o bind /local/etc /etc
               mount -n -o bind /local/root /root
               mount -n -o bind /local/srv /srv
               test $FSCK_RETURN -gt 0 && touch /fsck_corrected_errors
       else
      echo
#     ROR: chg 1
#     echo '*** ERROR!  Cannot fsck because root is not read-only!'
               echo '*** ERROR!  Cannot fsck because /local is not read-only!'
      echo
       fi
   else
       if test "$ROOTFS_FSCK" != "0" ; then
#     ROR: del 1, add 1
#     echo "root file system (/) is NOT being checked."
               echo "/local filesystem is NOT being checked."
#     ROR: del 1, add 4
#              mount -n -o remount,rw /local
               mount -n -o /local
               mount -n -o bind /local/etc /etc
               mount -n -o bind /local/root /root
               mount -n -o bind /local/srv /srv
       fi
       # remount in any case to apply additional options like xattr
#      ROR: del 1, add 3
#      mount -n -o remount,rw /
           mount -n /local
           mount -n -o bind /local/etc /etc
           mount -n -o bind /local/root /root
           mount -n -o bind /local/srv /srv
   fi
        # start with a clean mtab and enter root fs entry
   rm -f /etc/mtab*
   mount -f /
   ;;
    stop)
   ;;
#   ROR: add 3
    halt|reboot)
        mount -n -o remount,ro /local 2> /dev/null
        ;;
    restart)
   rc_failed 3
   rc_status -v
   ;;
    status)
   rc_failed 4
   rc_status -v
   ;;
    *)
   echo "Usage: $0 {start|stop|status|restart}"
   exit 1
   ;;
esac

rc_exit
```

Configuration file /etc/fstab

Example 38 shows the /etc/fstab configuration file.

Example 38 The /etc/fstab configuration file

```
/dev/dasdb1         /                   ext2      ro,noatime,nodiratime 0 0
/dev/dasda1         /boot               ext2      ro,noatime,nodiratime 0 0
/dev/dasdf1         /local              ext3      noauto,acl,user_xattr 0 0
/dev/dasdi1         /opt                ext2      ro,noatime,nodiratime 0 0
/dev/dasdh1         /usr                ext2      ro,noatime,nodiratime 0 0
/dev/dasdg1         /var                ext3      acl,user_xattr        0 0
/dev/dasdc1         swap                swap      defaults              0 0
/dev/dasdd1         swap                swap      defaults              0 0
/dev/dasde1         swap                swap      defaults              0 0
proc                /proc               proc      defaults              0 0
sysfs               /sys                sysfs     noauto                0 0
debugfs             /sys/kernel/debug   debugfs   noauto                0 0
devpts              /dev/pts            devpts    mode=0620,gid=5       0 0
```

z/VM code

This section includes the following EXECs that run on z/VM:

COPYMDSK.EXEC	EXEC to copy a minidisk (used by clone EXECs)
CLONERO.EXEC	EXEC to clone from S10GOLD read-only disks to target Linux
CLONERW.EXEC	EXEC to clone from S10GOLD R/W disks to target Linux
MNT2GOLD.EXEC	EXEC to clone from S10RWMNT to S10GOLD R/W disks
MNT2TST.EXEC	EXEC to clone from to S10RWMNT to S10RWTST
PROFILE.EXEC	EXEC to IPL to create VDISK swap spaces and IPL from 1B0
RO2GOLD.EXEC	EXEC to clone from S10ROGLD to S10GOLD read-only disks
TST2MNT.EXEC	EXEC to clone from to S10RWTST to S10RWMNT
SAMPLE.PARM-S10	Sample SLES 10 parameter file
SLES10.EXEC	EXEC to kick off a SLES 10 installation

Because these files are not supported, all of them start with this disclaimer:

```
/*--------------------------------------------------------------------
THE PROGRAM IS PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR
CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT
LIMITATION, ANY WARRANTIES OR CONDITIONS OF TITLE, NON-INFRINGEMENT,
MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.
NEITHER RECIPIENT NOR ANY CONTRIBUTORS SHALL HAVE ANY LIABILITY FOR
ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING WITHOUT LIMITATION LOST PROFITS), HOWEVER CAUSED
AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF
THE USE OR DISTRIBUTION OF THE PROGRAM OR THE EXERCISE OF ANY RIGHTS
GRANTED HEREUNDER, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES
--------------------------------------------------------------------*/
```

COPYMDSK EXEC

The COPYMDSK EXEC tries to copy a file with FLASHCOPY. If that fails, it falls back to DDR. Example 39 shows the COPYMDSK EXEC.

Example 39   The COPYMDSK EXEC

```
-------------------------------------------------------------------*/
/* COPYMDSK EXEC - copy minidisk w/FLASHCOPY, if it fails, try DDR   */
/*  Parm 1: vaddr of source minidisk                                 */
/*  Parm 2: vaddr of target minidisk                                 */
Address 'COMMAND'
Parse Arg source target .
Say
Say 'Copying minidisk' source 'to' target '...'
'CP FLASHCOPY' source '0 END' target '0 END'
If (rc \= 0) Then Do                  /* Fallback to DDR             */
  Say 'FLASHCOPY failed, falling back to DDR ...'
                                      /* Queue up DDR commands       */
  Queue 'SYSPRINT CONS'               /* Don't print to file         */
  Queue 'PROMPTS OFF'                 /* Don't ask 'Are you sure?'   */
  Queue 'IN' source '3390'            /* Input minidisk              */
  Queue 'OUT' target '3390'           /* Output minidisk             */
  Queue 'COPY ALL'                    /* Copy all contents           */
  Queue ' '                           /* Empty record ends DDR       */
  'DDR'
  retVal = rc
  End
Else retVal = rc
Say 'Return value =' retVal
Return retVal
```

CLONERO EXEC

The CLONERO EXEC clones from the S10GOLD read-only 1xxx disks to a target Linux user ID. Example 40 shows the CLONERO EXEC.

Example 40   The CLONERO EXEC

```
/* Clone read-write Linux from gold ID to target ID                  */
Address 'COMMAND'
Parse Upper Arg target .
If (target = '') Then Do
  Say 'Target user ID is a required parameter'
  Say
  Say 'Syntax:  CLONERO  targetID'
  Exit 1
  End
Say 'Do you want to copy R/O system from S10GOLD to' target 'y/n'
Parse Upper Pull answer .
If (answer \= 'Y') Then Exit 1
source = 'S10GOLD'

rw_addrs = '01B4 01B5 01B6'
ro_addrs = '21B4 21B5 21B6'
```

```
/* link target disks R/W */
Do a = 1 to Words(rw_addrs)
  addr = Word(rw_addrs,a)
  'CP LINK' target addr addr 'MR'
  If (rc \= 0) Then Call CleanUp 100+a
  End

/* link source disks R/O */
Do a = 1 to Words(ro_addrs)
  addr = Word(ro_addrs,a)
  'CP LINK' source addr addr 'RR'
  If (rc \= 0) Then Call CleanUp 200+a
  End

/* copy disks */
Do a = 1 to Words(ro_addrs)
  source_addr = Word(ro_addrs,a)
  target_addr = Word(rw_addrs,a)
  'EXEC COPYMDSK' source_addr target_addr
  If (rc \= 0) Then Call CleanUp 300+a
  End

/* cleanup */
Call CleanUp 0

/*+------------------------------------------------------------------+*/
CleanUp: Procedure expose ro_addrs rw_addrs
/*| Detach all disks before exiting                                  |*/
/*|  parm 1: Exit code                                               |*/
/*+------------------------------------------------------------------+*/
Parse Arg retVal
Say
Say 'Cleaning up ...'
CP DETACH ro_addrs rw_addrs
Exit retVal
```

CLONERW EXEC

The CLONERW EXEC clones from the S10GOLD read-write 2xxx disks to a target Linux user ID. Example 41 shows the CLONERW EXEC.

Example 41 The CLONERW EXEC

```
/* Clone read-write Linux from gold ID to target ID                  */
Address 'COMMAND'
Parse Upper Arg target .
If (target = '') Then Do
  Say 'Target user ID is a required parameter'
  Say
  Say 'Syntax:  CLONERW  targetID'
  Exit 1
  End
Say 'Do you want to copy R/W system from S10GOLD to' target 'y/n'
Parse Upper Pull answer .
```

```
If (answer \= 'Y') Then Exit 1
source = 'S10GOLD'

rw_addrs = '01B0 01B1 01B4 01B5 01B6 01B7 01B8'
ro_addrs = '11B0 11B1 11B4 11B5 11B6 11B7 11B8'

/* link target disks R/W */
Do a = 1 to Words(rw_addrs)
  addr = Word(rw_addrs,a)
  'CP LINK' target addr addr 'MR'
  If (rc \= 0) Then Call CleanUp 100+a
  End

/* link source disks R/O */
Do a = 1 to Words(ro_addrs)
  addr = Word(ro_addrs,a)
  'CP LINK' source addr addr 'RR'
  If (rc \= 0) Then Call CleanUp 200+a
  End

/* copy disks */
Do a = 1 to Words(ro_addrs)
  source_addr = Word(ro_addrs,a)
  target_addr = Word(rw_addrs,a)
  'EXEC COPYMDSK' source_addr target_addr
  If (rc \= 0) Then Call CleanUp 300+a
  End

/* cleanup */
Call CleanUp 0

/*+------------------------------------------------------------------+*/
CleanUp: Procedure Expose ro_addrs rw_addrs
/*| Detach all disks before exiting                                  |*/
/*|  parm 1: Exit code                                               |*/
/*+------------------------------------------------------------------+*/
Parse Arg retVal
Say
Say 'Cleaning up ...'
'CP DETACH' ro_addrs rw_addrs
Exit retVal
```

MNT2GOLD EXEC

The MNT2GOLD EXEC copies the S10RWMNT xxx read-write disks to the S10GOLD read-write 2xxx disks. Example 42 shows the MNT2GOLD EXEC.

Example 42 The MNT2GOLD EXEC

```
/* Copy Linux from maintenance ID to gold ID's read-write disks  */
Address 'COMMAND'
Say 'Do you want to copy R/W disks from S10RWMNT to S10GOLD? y/n'
Parse Upper Pull answer .
If (answer \= 'Y') Then Exit 1
source = 'S10RWMNT'
```

```
target = 'S10GOLD'

ro_addrs = '01B0 01B1 01B4 01B5 01B6 01B7 01B8'
rw_addrs = '11B0 11B1 11B4 11B5 11B6 11B7 11B8'

/* link target disks R/W */
Do a = 1 to Words(rw_addrs)
  addr = Word(rw_addrs,a)
  'CP LINK' target addr addr 'MR'
  If (rc \= 0) Then Call CleanUp 100+a
  End

/* link source disks R/O */
Do a = 1 to Words(ro_addrs)
  addr = Word(ro_addrs,a)
  'CP LINK' source addr addr 'RR'
  If (rc \= 0) Then Call CleanUp 200+a
  End

/* copy disks */
Do a = 1 to Words(ro_addrs)
  source_addr = Word(ro_addrs,a)
  target_addr = Word(rw_addrs,a)
  'EXEC COPYMDSK' source_addr target_addr
  If (rc \= 0) Then Call CleanUp 300+a
  End

/* CleanUp */
Call CleanUp 0

/*+------------------------------------------------------------------+*/
CleanUp: Procedure Expose ro_addrs rw_addrs
/*| Detach all disks before exiting                                  |*/
/*|  parm 1: Exit code                                               |*/
/*+------------------------------------------------------------------+*/
Parse Arg retVal
Say
Say 'Cleaning up ...'
Call Diag 8,'DETACH' ro_addrs rw_addrs
Exit retVal
```

MNT2TST EXEC

The MNT2TST EXEC copies the S10RWMNT read-write disks to the S10RWTST read-write disks. Example 43 shows the MNT2TST EXEC.

Example 43 The MNT2TST EXEC

```
/* Copy Linux from maintenance ID to test ID */
Address 'COMMAND'
source = 'S10RWMNT'
target = 'S10RWTST'
Say 'Do you want to copy R/W disks from' source 'to' target'? y/n'
Parse Upper Pull answer .
If (answer \= 'Y') Then Exit 1
```

```
ro_addrs = '01B0 01B1 01B4 01B5 01B6 01B7 01B8'
rw_addrs = '11B0 11B1 11B4 11B5 11B6 11B7 11B8'

/* link target disks R/W */
Do a = 1 to Words(rw_addrs)
  addr = Word(rw_addrs,a)
  'CP LINK' target Right(addr,3) addr 'MR'
  If (rc \= 0) Then Call CleanUp 100+a
  End

/* link source disks R/O */
Do a = 1 to Words(ro_addrs)
  addr = Word(ro_addrs,a)
  'CP LINK' source addr addr 'RR'
  If (rc \= 0) Then Call CleanUp 200+a
  End

/* copy disks */
Do a = 1 to Words(ro_addrs)
  source_addr = Word(ro_addrs,a)
  target_addr = Word(rw_addrs,a)
  'EXEC COPYMDSK' source_addr target_addr
  If (rc \= 0) Then Call CleanUp 300+a
  End

/* cleanup */
Call CleanUp 0

/*+------------------------------------------------------------------+*/
CleanUp: Procedure Expose ro_addrs rw_addrs
/*| Detach all disks before exiting                                  |*/
/*|  parm 1: Exit code                                               |*/
/*+------------------------------------------------------------------+*/
Parse Arg retVal
Say
Say 'Cleaning up ...'
'CP DETACH' ro_addrs rw_addrs
Exit retVal
```

PROFILE EXEC

The PROFILE EXEC is a sample logon profile that is on the Linux user ID’s 191 disk. Example 44 shows the PROFILE EXEC.

Example 44 The PROFILE EXEC

```
/* PROFILE EXEC for Linux virtual servers                            */
Address 'COMMAND'
'CP SET RUN ON'
'CP SET PF11 RETRIEVE FORWARD'
'CP SET PF12 RETRIEVE'
'ACCESS 592 C'
'EXEC SWAPGEN 1B2 131072'      /* create a 64M VDISK disk swap space */
'EXEC SWAPGEN 1B3 131072'      /* create a 64M VDISK disk swap space */
```

```
'PIPE CP QUERY' Userid() '| VAR USER'
Parse Value user With id . dsc .
iplDisk = 1B0                        /* /boot/ is 1B0               */
If (dsc = 'DSC') Then                /* user is disconnected        */
  'CP IPL' iplDisk
Else Do                             /* user is interactive -> prompt */
  Say 'Do you want to IPL Linux from DASD' iplDisk'? y/n'
  Parse Upper Pull answer .
  If (answer = 'Y') Then
    'CP IPL' iplDisk
  End
Exit
```

RO2GLD EXEC

The RO2GLD EXEC copies the disks that will become the read-only root system from S10ROGLD to the 1xxx disks of S10GOLD. Example 45 shows the RO2GLD EXEC.

Example 45 The RO2GOLD EXEC

```
/* Copy Linux from read-only ID to Gold ID  */
Address 'COMMAND'
Say 'Do you want to copy disks from S10ROGLD to S10GOLD? y/n'
Parse Upper Pull answer .
If (answer \= 'Y') Then Exit 1
source = 'S10ROGLD'
target = 'S10GOLD'

ro_addrs = '01B0 01B1 01B4 01B5 01B6 01B7 01B8'
rw_addrs = '21B0 21B1 21B4 21B5 21B6 21B7 21B8'

/* link target disks R/W */
Do a = 1 to Words(rw_addrs)
  addr = Word(rw_addrs,a)
  'CP LINK' target addr addr 'MR'
  If (rc \= 0) Then Call CleanUp 100+a
  End

/* link source disks R/O */
Do a = 1 to Words(ro_addrs)
  addr = Word(ro_addrs,a)
  'CP LINK' source addr addr 'RR'
  If (rc \= 0) Then Call CleanUp 200+a
  End

/* copy disks */
Do a = 1 to Words(ro_addrs)
  source_addr = Word(ro_addrs,a)
  target_addr = Word(rw_addrs,a)
  'EXEC COPYMDSK' source_addr target_addr
  If (rc \= 0) Then Call CleanUp 300+a
  End

/* cleanup */
Call CleanUp 0
```

```
/*+------------------------------------------------------------------+*/
CleanUp: Procedure Expose ro_addrs rw_addrs
/*| Detach all disks before exiting                                  |*/
/*|  parm 1: Exit code                                               |*/
/*+------------------------------------------------------------------+*/
Parse Arg retVal
Say
Say 'Cleaning up ...'
'CP DETACH' ro_addrs rw_addrs
Exit retVal
```

TST2MNT EXEC

The TST2MNT EXEC copies the read-write disks from S10RWTST to S10RWMNT. Example 46 shows the TST2MNT EXEC.

Example 46 The TST2MNT EXEC

```
/* Copy Linux from maintenance ID to test ID */
Address 'COMMAND'
source = 'S10RWTST'
target = 'S10RWMNT'
Say 'Do you want to copy R/W disks from' source 'to' target'? y/n'
Parse Upper Pull answer .
If (answer \= 'Y') Then Exit 1

ro_addrs = '01B0 01B1 01B4 01B5 01B6 01B7 01B8'
rw_addrs = '11B0 11B1 11B4 11B5 11B6 11B7 11B8'

/* link target disks R/W */
Do a = 1 to Words(rw_addrs)
  addr = Word(rw_addrs,a)
  'CP LINK' target Right(addr,3) addr 'MR'
  If (rc \= 0) Then Call CleanUp 100+a
  End

/* link source disks R/O */
Do a = 1 to Words(ro_addrs)
  addr = Word(ro_addrs,a)
  'CP LINK' source addr addr 'RR'
  If (rc \= 0) Then Call CleanUp 200+a
  End

/* copy disks */
Do a = 1 to Words(ro_addrs)
  source_addr = Word(ro_addrs,a)
  target_addr = Word(rw_addrs,a)
  'EXEC COPYMDSK' source_addr target_addr
  If (rc \= 0) Then Call CleanUp 300+a
  End

/* CleanUp */
Call CleanUp 0
```

```
/*+------------------------------------------------------------------+*/
CleanUp: Procedure Expose ro_addrs rw_addrs
/*| Detach all disks before exiting                                  |*/
/*|  parm 1: Exit code                                               |*/
/*+------------------------------------------------------------------+*/
Parse Arg retVal
Say
Say 'Cleaning up ...'
'CP DETACH' ro_addrs rw_addrs
Exit retVal
```

Sample PARM-S10 file

This file is a sample parameter file for a SLES 10 installation. Example 47 shows the a sample PARM-S10 file.

Example 47   Sample parameter file

```
ramdisk_size=65536 root=/dev/ram1 ro init=/linuxrc TERM=dumb
HostIP=n.n.n.n       Hostname=name.example.com
Gateway=n.n.n.n       Netmask=255.255.255.0
Broadcast=n.n.n.n       Layer2=0
ReadChannel=0.0.0600  WriteChannel=0.0.0601  DataChannel=0.0.0602
Nameserver=n.n.n.n      Portname=dontcare
Install=nfs://n.n.n.n/nfs/sles10/SLES-10-CD-s390x-GMC-CD1.iso
UseVNC=1  VNCPassword=123456
InstNetDev=osa OsaInterface=qdio OsaMedium=eth Manual=0
```

SLES10 EXEC

The SLES10 EXEC will punch the kernel, parameter file, and RAMdisk to your reader and will invoke a SLES 10 install from there. Example 48 shows the SLES10 EXEC.

Example 48   The SLES10 EXEC

```
/* EXEC to punch SLES-10 install system to reader and IPL from it */
Address 'COMMAND'
'CP SPOOL PUN *'
'CP CLOSE RDR'
'CP PURGE RDR ALL'
'PUNCH SLES10 KERNEL * (NOHEADER'
'PUNCH' Userid() 'PARM-S10 * (NOHEADER'
'PUNCH SLES10 INITRD * (NOHEADER'
'CP CHANGE RDR ALL KEEP'
'CP IPL 00C CLEAR'
```

The team that wrote this IBM Redpaper

This IBM Redbooks® publication was produced by a team of specialists from Nationwide Mutual Insurance Company, working with the IBM International Technical Support Organization (ITSO).

Steve Womer is a Senior Consulting IT Architect at Nationwide in Columbus, Ohio. He has 28 years of experience in information systems. He holds a B.S. degree from Ohio State University. Steve's areas of expertise include z/OS systems management, systems automation, and Linux. He has written extensively about Linux on IBM System z.

Rick Troth works with Linux on z/VM at Nationwide. He has many years of experience working with VM systems and UNIX. He has a degree in Computer Science from Texas A&M University. Rick's areas of expertise include combining "open systems" (for example UNIX, POSIX, and especially Linux) with the incredible versatility of virtual machines. Rick is honored to be a Knight of VM (under the monicker Sir Santa). You can reach him by signing up to the IBMVM discussion list (hosted by the University of Arkansas) or to the LINUX-390 discussion list (hosted by Marist College) and hollering "Hey, Sir Santa!"

Michael MacIsaac is a professional working at IBM in Poughkeepsie, NY. He has worked on Linux and z/VM and was formerly an ITSO project leader.

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

Lydia Parziale
International Technical Support Organization, Poughkeepsie Center

Ernie Horn
IBM Poughkeepsie

Mike Nardone, Cortez Crosby
Nationwide

Mark Post
Novell, Inc.

A special thanks to Carlos Ordonez of IBM Poughkeepsie, for always being willing to sit through high-level architecture discussions.

A special thanks to Jim Vincent of Nationwide for converting raw REXX code into something more reputable.

Notices

This document REDP-4322-00 was created or updated on February 5, 2008.

Send us your comments in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:
  ibm.com/redbooks
- Send your comments in an email to:
  redbooks@us.ibm.com
- Mail your comments to:
  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400 U.S.A.

Redpaper™

Trademarks