Note that running DHCP does not have to mean you hand out different IP addresses for your systems (which is what DHCP normally is used for). You can manually bind IP addresses to the physical interfaces and keep the same IP address each time the server starts. DHCP would only be the easy way to configure the IP stack of each system.

You could even do dynamic IP addresses for your servers. This would be an option when combined with Dynamic Domain Name Service (DDNS). In that case, each system would register through DDNS, for example, as the next system offering a particular service. Round-robin DNS would then cause the traffic to be spread over multiple systems.

10.4.5 When to apply the configuration changes

There are at least two different ways to apply the changes to the copied disks for the new Linux image.

- Store a small script in the boot process to apply these changes the first time the cloned image is executed. Instead of using a custom script with hardcoded values, you can write a generic script that obtains the IP address and hostname through some other mechanism. The `hcp` command from the cpint package can be used to obtain the user ID of the virtual machine running the image.
- Mount the file system into an existing running Linux image and run a script to apply the changes. This is a more flexible choice, because you can do different repair actions that may be necessary (like installing additional packages). Unfortunately, it is currently difficult to access the file system of a Linux image that is not running (also known as a "dead penguin"), or one that does not have a network connection.

Which method will work best in your situation depends on various aspects of your installation (such as network topology, naming convention, etc).

10.5 Sharing code among images

The process outlined in 10.4, "Copying disks instead of doing a full install" on page 213 can simplify the install process, but it does not yet exploit z/VM facilities for sharing resources. For those who are used to working with VM, it appears obvious that you want to share the common code between your Linux images. For a "default install" of SuSE, about 85% of the disk space is used for the /usr directory of the file system. Most packages have their binaries installed in /usr, so it seems obvious that this also should be shared.

The Linux Standard Base (LSB) defines the file system hierarchy such that /usr can be shared among images (outside the S/390 world, over NFS). Unfortunately, it also defines that application code should be installed in /opt, which makes sharing /usr less attractive.

The file system can be split over two separate minidisks after installation, but it's much easier to let YaST do that job; during installation, you define one disk to hold the /usr directory and YaST will copy all those files to that minidisk. This process is apparently so obvious that many people have built Linux images that had a R/O link to a common minidisk holding the /usr subtree of the file system. This results in a significant reduction in disk space requirements for running a large number of Linux images.

However, as with many cheap solutions, this one also comes with a few drawbacks.

- Once a Linux image is using the shared /usr disk, it is no longer possible to make changes to the contents of the disk.

  In CMS, to share R/O minidisks, people have developed tricks to deal with the "one writer - many readers" situation, but for Linux this does not work because every Linux image will buffer parts of that disk in its buffer cache.
- Portions of applications live outside the /usr directory, for example in /bin and /sbin. When those other portions are part of the private disk space of the Linux image, it will be difficult to maintain consistency when upgrading packages. This means there is no easy way to upgrade the systems afterward.
- Many applications and users need to write into portions of /usr for their function. An example of this is the package manager rpm that keeps its database (the software inventory) in /usr, as well.
- Many applications do not separate code and data (like WebSphere, writing the log file in the bin directory by default), which makes it very difficult to share code.
- The recent standards define /opt to hold packages that are not part of the code system. This would make sharing /usr less effective.

We believe a realistic answer to this could be a new device driver as described in "Shadowed disk support" on page 413. This code is not available yet.

Despite these drawbacks, this simple way of sharing the disk may be attractive for special situations (for example, "disposable penguins" only needed for a limited period). If an application has a specific requirement to write in an otherwise R/O directory, a private writable directory can be provided through a symlink to another part of the file system, or by "overmounting" with another block device (via the loop driver, for example).

10.6 Breeding a colony of penguins

For a demonstration of the cloning process, we had to use a configuration with /usr on a separate R/O linked minidisk, despite the drawbacks illustrated in 10.5, "Sharing code among images" on page 221. Trying the cloning process with private disk space for each cloned penguin would have been too expensive, both in disk space and in time to create the images.

Unfortunately, YaST runs SuSEconfig after the first reboot, which turns out to write into the /usr directory. This means we need to finish the install process completely before we have a file system that can be copied and used by others. Since other installations may have similar restrictions, we decided to take a more generic approach and complete the install before copying the disks.

The cloning process is demonstrated with each of the steps invoked by hand. This does not mean the process could not be automated, but it's probably easier to follow this way than with a single program that does all.

10.6.1 Images used in the cloning process

The images used in the cloning process will have a few minidisks:

01A0	A 100-cylinder private disk to hold the root file system
02A0	R/O link to the original root file system for copying
01A1	A 2000-cylinder disk linked R/O by all images
00CD	R/O link to a starter disk with initrd

Each of the cloned images will have an IUCV connection to a single common VM TCP/IP stack.

10.6.2 Create a patch file for cloning

We create a patch for the cloning process as described in "Using diff to find the changes" on page 216. Doing this after finishing the install is not much different from doing it earlier in the process.

Obviously, `diff` will also find a lot of differences in log files and other things that you do not want to end up in the patch, so you can get those out of the process as soon as possible with, for example, a few `grep` commands before sorting the list of filenames.

To create the two different disks to compare, we copied the 01A0 disk of the install system to a new minidisk and then ran YaST in the installation system to change the IP address and hostname.

Although the chosen network setup resulted in identical gateway addresses and default routes for each of the cloned images, we decided to put that in the patch anyway to have a more generic patch.

Example 10-10 Patch generated for the demo setup

```
--- a/etc/HOSTNAME      Wed Aug  8 02:54:04 2001
+++ b/etc/HOSTNAME      Wed Aug  8 03:04:17 2001
@@ -1 +1 @@
-tux8mstr
+:hostname:
--- a/etc/hosts Wed Aug  8 02:54:04 2001
+++ b/etc/hosts Wed Aug  8 03:04:17 2001
@@ -21,4 +21,4 @@
 ff02::2         ipv6-allrouters
 ff02::3         ipv6-allhosts

-192.168.6.254          tux8mstr.hub6.itso.ibm.com      tux8mstr
+:myip:         :hostname:.hub6.itso.ibm.com :hostname:
--- a/etc/rc.config     Wed Aug  8 02:54:02 2001
+++ b/etc/rc.config     Wed Aug  8 03:04:16 2001
@@ -132,7 +132,7 @@
 #
 # IP Adresses
 #
-IPADDR_0="192.168.6.254"
+IPADDR_0=":myip:"
 IPADDR_1=""
 IPADDR_2=""
 IPADDR_3=""
@@ -151,7 +151,7 @@
 # sample entry for ethernet:
 # IFCONFIG_0="192.168.81.38 broadcast 192.168.81.63 netmask 255.255.255.224"
 #
-IFCONFIG_0="192.168.6.254 pointopoint 192.168.6.1 mtu 8188 up"
+IFCONFIG_0=":myip: pointopoint :gwip: mtu :gwmtu: up"
 IFCONFIG_1=""
 IFCONFIG_2=""
 IFCONFIG_3=""
@@ -207,7 +207,7 @@
 # (e.g. "riemann.suse.de" or "hugo.linux.de")
 # don't forget to also edit /etc/hosts for your system
 #
-FQHOSTNAME="tux8mstr.hub6.itso.ibm.com"
+FQHOSTNAME=":hostname:.hub6.itso.ibm.com"

 #
 # Shall SuSEconfig maintain /etc/resolv.conf (needed for DNS) ?
--- a/etc/route.conf    Thu Aug  9 16:21:37 2001
+++ b/etc/route.conf    Thu Aug  9 16:19:37 2001
```

```
@@ -34,4 +34,4 @@
 # 192.168.0.1            0.0.0.0         255.255.255.255      ippp0
 # default                192.168.0.1
 192.168.6.1            0.0.0.0            255.255.255.255        iucv0
-default                       192.168.6.1
+default                       :gwip:
```

The resulting patch is shown in Example 10-10. It turns out to be very small. It touches only four files in the system (those marked with +++ characters). This patch was stored in the root directory of the installation system so that it would be available in each cloned image as well.

Copy the root file system and IPL starter system

The root file system is copied using the COPYDISK program shown in Example 10-11. Exploiting the fact that this root file system is filled for 70%, this turned out to be the fastest solution.

Example 10-11 The COPYDISK program

```
/* Copy a reserved disk to a formatted disk */
signal on error
arg cuu1 cuu2 .

'ACCESS' cuu1 'J'
'PIPE COMMAND LISTFILE * * J | var infile'
parse var infile fn .
'ACCESS' cuu2 'K'
queue '1'; 'RESERVE' fn userid() 'K'
'PIPE <' infile,
    '| spec number 1.10 ri 1-* n',
    '| not verify 11-* x00',
    '| fileupdate' fn userid() 'K'
'PIPE mdiskblk number J 1.2 | mdiskblk write K'
return rc
```

After the root file system is copied, the RAMdisk system is started.

Example 10-12 Copying the root file system and IPL with RAMdisk

```
copydisk 2a0 1a0
DMSACC724I 2A0 replaces J (2A0)
DMSACP723I J (2A0) R/O
DMSACC724I 1A0 replaces K (1A0)
DMSRSV603R RESERVE will erase all files on disk K(1A0). Do you wish to
continue? Enter 1 (YES) or 0 (NO).
1
DMSRSV733I Reserving disk K
```

```
Ready; T=0.54/0.80 20:43:15
ipl 0cd clear
Linux version 2.2.16 (root@ikr_tape.suse.de) (gcc version 2.95.2 19991024
(release)) #1 SMP Tue May 1 11:47:13 GMT 2001
Command line is: ro ramdisk_size=32768 root=/dev/ram0 ro
```

Load the DASD driver and mount the disks

Now the DASD driver is loaded and the new disks are mounted.

Example 10-13 Accessing the disks from the RAMdisk system

```
# insmod dasd dasd=1a0,1a1
Using /lib/modules/2.2.16/block/dasd.o
dasd:initializing...
dasd:Registered successfully to major no 94
dasd(eckd):ECKD discipline initializing
dasd:Registered ECKD discipline successfully
dasd(fba):FBA  discipline initializing
dasd:Registered FBA discipline successfully
dasd(eckd):01A0 on sch 0: 3390/0C(CU:3990/04) Cyl:100 Head:15 Sec:224
dasd(eckd):01A0 on sch 0: 3390/0C (CU: 3990/04): Configuration data read
dasd: devno 0x01A0 on subchannel 0 (ECKD) is /dev/dasda (94:0)
dasd(eckd):01A1 on sch 9: 3390/0C(CU:3990/04) Cyl:1500 Head:15 Sec:224
dasd(eckd):01A1 on sch 9: 3390/0C (CU: 3990/04): Configuration data read
dasd: devno 0x01A1 on subchannel 9 (ECKD) is /dev/dasdb (94:4)
dasd:waiting for responses...
dasd(eckd):/dev/dasda (01A0): capacity (4kB blks): 72000kB at 48kB/trk
 dasda:(CMS1)/TUX1A0:(MDSK) dasda dasda1
dasd(eckd):/dev/dasdb (01A1): capacity (4kB blks): 1080000kB at 48kB/trk
 dasdb:(CMS1)/TUX1A1:(MDSK) dasdb dasdb1
dasd:initialization finished
# mount /dev/dasda1 /mnt
# mount /dev/dasdb1 /mnt/usr -r
```

Apply the patch to the copied file system

The patch is applied to the copied file system by running the updclone.sh program. By running the program with **chroot**, it sees the file system mounted at /mnt as its root file system for the duration of the program.

```
# chroot /mnt /updclone.sh tux80000 192.168.6.2 192.168.6.1 8188
patching file etc/HOSTNAME
patching file etc/hosts
patching file etc/rc.config
patching file etc/route.conf
```

The **updclone.sh** script reads the generic patch and transforms that into a patch specific for this image using the host name and IP address specified. For a less error-prone implementation, you should consider storing the host names and IP addresses in a table on that disk. That way, a **grep** could be used to get the arguments for the **updclone.sh** script.

Example 10-14 The updclone.sh program to apply the patch

```
#! /bin/sh

if [ -z $4 ]; then
  echo "Need hostname IP-address gateway-ip gateway-mtu"
  exit
fi
cat generic.diff             \
   | sed "s/:hostname:/$1/  " \
   | sed "s/:myip:/$2/   " \
   | sed "s/:gwip:/$3/ " \
   | sed "s/:gwmtu:/$4/ " \
   | patch -p1 $5
```

If you register the Linux images in DNS, you could even consider getting the arguments for updclone.sh from that. The **hcp** command could be used to get the user ID of the virtual machine. Given a practical naming convention, **nslookup** could get you the IP address of the Linux image and the gateway.

Shut down the system

The system shutdown should not only unmount the file systems cleanly, but also leave network connections in a better state to be restarted when the image is rebooted. Unfortunately, the starter system does not load a disabled-wait PSW, but loops after a shutdown. You get out of this using the #CP command to do the IPL.

Example 10-15 Shutting down the starter system

```
# shutdown -h now
Syncing all buffers...
Sending KILL signal to linuxrc for shutdown...
Sending all processes the TERM signal...
Aug  9 22:05:27 suse exiting on signal 15
Sending all processes the KILL signal...
Syncing all buffers...
Turning off swap...
Unmounting file systems...
/dev/dasdb1 umounted
/dev/dasda1 umounted
/dev/ram2 umounted
```

IPL from the patched root file system

With the patches applied, the Linux image can now be booted from the new disk.

Example 10-16 IPL from the root file system after the patch was applied

```
IPL 1A0 CLEAR
Linux version 2.2.16 (root@Tape.suse.de) (gcc version 2.95.2 19991024
(release))
 #1 SMP Sun May 6 06:15:49 GMT 2001
Command line is: ro dasd=0200,01A0,01A1,0cd,0100,0101 root=/dev/dasdb1 noinitrd
iucv=$TCPIP

We are running under VM
This machine has an IEEE fpu
Initial ramdisk at: 0x02000000 (16777216 bytes)
Detected device 01A0 on subchannel 0000 - PIM = F0, PAM = F0, POM = FF
Detected device 0009 on subchannel 0001 - PIM = 80, PAM = 80, POM = FF
Detected device 000C on subchannel 0002 - PIM = 80, PAM = 80, POM = FF
```

10.7 Linux IPL from NSS

A Named Saved System (NSS) is like a snapshot copy of part of the memory of a virtual machine. In addition to simulating the normal S/390 IPL of a device, VM can also IPL a virtual machine from a NSS. This is especially efficient for CMS, because the NSS for CMS is defined such that large portions of it can be shared between virtual machines. This reduces the storage requirements for running a large number of CMS virtual machines.

An NSS can have shared and non-shared pages. The S/390 architecture requires all pages in a single 1 MB segment to be either shared or non-shared. The shared pages will be shared among all virtual machines that IPL the NSS. For the non-shared pages, each virtual machine will get its own copy, initialized from the NSS.

10.7.1 Using an NSS with just the kernel

Unfortunately,the Linux for S/390 kernel is not designed to be shared among images. The writable portions of the kernel are mixed with the read-only portions. While the non-shared pages do not reduce overall memory requirements, having memory initialized at startup should at least speed up the boot process.

One of the complications with running the kernel from an NSS is that the kernel parameters (e.g. disk addresses for the DASD driver) need to be the same for each Linux image using this NSS. Fortunately, VM allows us to tailor the virtual machine to fit on the addresses defined for the kernel that was saved in the NSS.

While you can create the NSS by IPL from the virtual reader, it is easier to do when booting from disk because `silo` gives you the information you need to define your NSS.

Example 10-17 Output of silo to compute NSS addresses

```
original parameterfile is: '/boot/parmfile'...ok...
final parameterfile is: '/boot/parmfile.map'...ok...
ix 0: offset: 002592 count: 0c address: 0x00000000
ix 1: offset: 00259f count: 80 address: 0x0000c000
ix 2: offset: 00261f count: 80 address: 0x0008c000
ix 3: offset: 00269f count: 76 address: 0x0010c000
ix 4: offset: 001609 count: 01 address: 0x00008000
Bootmap is in block no: 0x0000160a
```

When you run `silo` to make a disk bootable, it displays the memory address where the kernel is going to be loaded. The addresses in Example 10-17 show that different portions of the kernel use the memory from 0 to 0x00182000. This means that the NSS should at least contain the pages 0-181[2] in exclusive write (EW) mode (and there is no reason to do more than that).

To freeze the Linux image at the correct point during the IPL process, you can use the CP TRACE command. The current Linux for S/390 kernel starts execution at address 0x010000, so the following TRACE command will cause the Linux image to stop at that point.

When execution is stopped at that point, the TRACE command causes the segment to be saved and also ends the trace.

Example 10-18 Defining the NSS and saving it

```
DEFSYS SUSE 0-181 EW MINSIZE=40M
HCPNSD440I The Named Saved System (NSS) SUSE was successfully defined in fileid
0089.
TRACE INST RANGE 10000.2 CMD SAVESYS SUSE "#TRACE END ALL
IPL 1B0 CLEAR
Tracing active at IPL
 -> 00010000  BASR  0DD0        CC 2
HCPNSS440I Named Saved System (NSS) SUSE was successfully saved in fileid 0089.
Trace ended
```

[2] You can use the "Scientific" mode of the calculator in your MS Windows accessories to do the computations.

After the NSS has been saved as shown in Example 10-18, the Linux images can boot this kernel with a simple IPL SUSE command.

```
CP IPL SUSE
Linux version 2.2.18 (root@vmlinux6) (gcc version 2.95.2 19991024
SMP Thu Jul 19 10:07:30 EST 2001
```

Because the NSS is defined as exclusive write (EW), the kernel pages are not shared by the Linux images and using the NSS does not reduce overall storage requirements as it does with CMS. Work is in progress to change the layout of the kernel such that significant portions of the code can be shared.

10.7.2 Using an NSS as a starter system

To simplify the boot process, we packaged the kernel with the RAMdisk image in a single NSS that was completely defined as EW. We compared an NSS with a compressed RAMdisk image to one with an uncompressed RAMdisk image, expecting an impressive performance boost when skipping the RAMdisk uncompress at each IPL. However, Table 10-2 shows that the opposite was true in our case.

Table 10-2 Elapsed time to boot from NSS

Compressed RAMdisk image	12 s
Uncompressed RAMdisk image	19 s

A more detailed comparison of the two scenarios showed us that the CPU usage for an IPL from the compressed RAMdisk is indeed significantly higher (as expected), but the 10 seconds of elapsed time missing in the case of the uncompressed RAMdisk turn out to be spooling I/O for CP loading the pages of the NSS in (one at a time).

However, when a Linux image was already running from its non-shared copy of the NSS, the next image could IPL from NSS in 3 seconds—this suggests that CP incorrectly considers all EW pages from a NSS as if they were shared pages (where it is safe to say EW is the most obvious indication of not sharing the page).

For an NSS like CMS, this is not a significant issue since there are just a few EW pages in the segment. However, we can assume that excessive use of an NSS with a lot of EW pages should be avoided.

10.7.3 Picking up IPL parameters

One of the problems with an IPL from NSS is that the kernel parameters are defined in the NSS and will be the same for each image that IPLs the NSS.

The CP IPL command in z/VM has a PARM option that allows the user to pass parameters to the system that is being IPLed. Traditionally, this is used by CMS to control some options in the IPL process (like bypass execution of the system profile). A logical extension of this is to use that option to tailor the command line parameters for Linux as well (for example, to specify what disks to use).

In order to experiment with this, a "quick and dirty" patch was written against the Linux kernel to pick up the argument from the IPL command.

Example 10-19 Patch to pass IPL parameters to the kernel

```
--- boelinux-2.2.16/arch/s390/bootipleckd.S   Mon Apr 30 17:22:41 2001
+++ linux/arch/s390/boot/ipleckd.S     Wed Aug  8 18:20:12 2001
@@ -41,6 +41,7 @@

        .org 0xf0                        # Lets start now...
 _start: .globl _start
+       stm     %r0,%r15,0x0fc0          # save the registers for later
        l       %r1,__LC_SUBCHANNEL_ID   # get IPL-subchannel from lowcore
        st      %r1,__LC_IPLDEV          # keep it for reipl
        stsch   .Lrdcdata
@@ -112,7 +113,23 @@
        mvc     0x500(256,%r3),0x80(%r4)
        mvc     0x600(256,%r3),0x180(%r4)
        mvc     0x700(256,%r3),0x280(%r4)
-.Lrunkern:
+.Lrunkern:                               # We align here to 0x0200
+       j       .stopnss                 # because that is easy to
+       .org    0x0200                   # remember for the trace
+.stopnss:
+       lm      %r0,%r15,0x0fc0          # last instr when not NSS
+       stm     %r0,%r15,0x0fc0          # first instr from NSS
+       tr      0x0fc0(64,0),.ebcasc     # translate saved registers
+       lm      %r3,%r4,.Lstart
+.find00:
+       cli     0x480(%r3),0x00          # end of string?
+       la      %r3,1(%r3)
+       jnz     .find00
+       mvi     0x47f(%r3),0x020         # put a blank instead of 0x00
+       mvc     0x480(64,%r3),0x0fc0
+       mvi     0x4c0(%r3),0x00          # and a 0x00 in case all 64 used
+       lm      %r3,%r4,.Lstart
+.notnss:
 #      lhi     %r2,17
 #      sll     %r2,12
 #      st      %r1,0xc6c(%r2)           # store iplsubchannel to lowcore
@@ -296,7 +313,43 @@
        .long 0x47400010,0x00000000+.Llodata
 .Lrdccw:
        .long 0x86400000,0x00000000
-       .org 0x800
+
+
+       .org 0xe00      # EBCDIC to lowercase ASCII table
+.ebcasc:
+       .byte   0x00,0x01,0x02,0x03,0x07,0x09,0x07,0x7F
+       .byte   0x07,0x07,0x07,0x0B,0x0C,0x0D,0x0E,0x0F
+       .byte   0x10,0x11,0x12,0x13,0x07,0x0A,0x08,0x07
+       .byte   0x18,0x19,0x07,0x07,0x07,0x07,0x07,0x07
+       .byte   0x07,0x07,0x1C,0x07,0x07,0x0A,0x17,0x1B
+       .byte   0x07,0x07,0x07,0x07,0x07,0x05,0x06,0x07
+       .byte   0x07,0x07,0x16,0x07,0x07,0x07,0x07,0x04
+       .byte   0x07,0x07,0x07,0x07,0x14,0x15,0x07,0x1A
+       .byte   0x20,0xFF,0x83,0x84,0x85,0xA0,0x07,0x86
+       .byte   0x87,0xA4,0x9B,0x2E,0x3C,0x28,0x2B,0x7C
+       .byte   0x26,0x82,0x88,0x89,0x8A,0xA1,0x8C,0x07
+       .byte   0x8D,0xE1,0x21,0x24,0x2A,0x29,0x3B,0xAA
+       .byte   0x2D,0x2F,0x07,0x8E,0x07,0x07,0x07,0x8F
+       .byte   0x80,0xA5,0x07,0x2C,0x25,0x5F,0x3E,0x3F
+       .byte   0x07,0x90,0x07,0x07,0x07,0x07,0x07,0x07
+       .byte   0x70,0x60,0x3A,0x23,0x40,0x27,0x3D,0x22
+       .byte   0x07,0x61,0x62,0x63,0x64,0x65,0x66,0x67
+       .byte   0x68,0x69,0xAE,0xAF,0x07,0x07,0x07,0xF1
+       .byte   0xF8,0x6A,0x6B,0x6C,0x6D,0x6E,0x6F,0x70
+       .byte   0x71,0x72,0xA6,0xA7,0x91,0x07,0x92,0x07
+       .byte   0xE6,0x7E,0x73,0x74,0x75,0x76,0x77,0x78
+       .byte   0x79,0x7A,0xAD,0xAB,0x07,0x07,0x07,0x07
+       .byte   0x5E,0x9C,0x9D,0xFA,0x07,0x07,0x07,0xAC
+       .byte   0xAB,0x07,0x5B,0x5D,0x07,0x07,0x07,0x07
+       .byte   0x7B,0x61,0x62,0x63,0x64,0x65,0x66,0x67
+       .byte   0x68,0x69,0x07,0x93,0x94,0x95,0xA2,0x07
+       .byte   0x7D,0x6A,0x6B,0x6C,0x6D,0x6E,0x6F,0x70
+       .byte   0x71,0x72,0x07,0x96,0x81,0x97,0xA3,0x98
+       .byte   0x5C,0xF6,0x73,0x74,0x75,0x76,0x77,0x78
+       .byte   0x79,0x7A,0xFD,0x07,0x99,0x07,0x07,0x07
+       .byte   0x30,0x31,0x32,0x33,0x34,0x35,0x36,0x37
+       .byte   0x38,0x39,0x07,0x07,0x9A,0x07,0x07,0x07
+
 # end of pre initialized data is here CCWarea follows
 # from here we load 1k blocklist
 # end of function
```

Note that the patch shown in Example 10-19 is not a production-strength solution, and it has not yet been submitted to the IBM team in Boeblingen who maintain the S/390-specific portions of the kernel source.

Note: The IPL parameters are not restricted to the IPL from NSS. You can also use them when you IPL from disk. With a little more code in the kernel, it's possible to make the code pick up a SAVESYS option to save the NSS, as is done for CMS.

The length of the parameters passed on the IPL command is restricted to 64 characters. This may not be sufficient in many cases, so we wrote the code such that the parameters are appended to whatever is already in the kernel (i.e. the parameters in the parameter file as it was used in silo).

Tip: Additional parameters specified in the kernel command line (not used by the kernel or device drivers) end up as environment variables to the boot scripts. This can be used to pass options to the boot scripts (e.g. IP address).

To exploit this patch, a trace should be set up to stop execution at 0x0200, at which point the SAVESYS command can be issued, as shown in Example 10-20.

Example 10-20 Defining and saving the NSS

```
CP DEFSYS SUSE216A 0-181 EW MINSIZE=40M PARMREGS=0-15
The Named Saved System (NSS) SUSE216A was successfully defined in fileid 0110.
TRACE I R 200.2
IPL 1A0 CLEAR
Tracing active at IPL
 -> 00000200  LM    980F0FC0    00000FC0    CC 2
SAVESYS SUSE216A
Named Saved System (NSS) SUSE216A was successfully saved in fileid 0110.
TRACE END ALL
Trace ended
```

This is very similar to what is shown in 10.7.1, "Using an NSS with just the kernel" on page 228, except for the different address to freeze the IPL.

Example 10-21 Demonstrate the modified kernel command line

```
IPL SUSE216A PARM TEST=EXAMPLE
Linux version 2.2.16 (root@tux60000) (gcc version 2.95.2 19991024 (release)) #1
SMP Wed Aug 1 18:21:27 EST 2001
Command line is: ro dasd=0200,01A0,01A1,0cd root=/dev/dasdb1 noinitrd
 test=example
We are running under VM
This machine has an IEEE fpu
Initial ramdisk at: 0x02000000 (16777216 bytes)
Detected device 01A0 on subchannel 0000 - PIM = F0, PAM = F0, POM = FF
Detected device 01A1 on subchannel 0001 - PIM = F0, PAM = F0, POM = FF
Detected device 0009 on subchannel 0002 - PIM = 80, PAM = 80, POM = FF
```

The IPL in Example 10-21 shows the `test=example` option added to the IPL command. This shows up again at the end of the command line echoed by the kernel.

Note: Because CP will uppercase the command when typed in on the console, the patch was made to translate it to lowercase. This way we can specify the options for Linux that need to be lowercase. This obviously causes problems when you want to do things in uppercase. Doing it right will take more thinking and more coding.

The current implementation of the DASD driver requires all disks to be specified in a single dasd= parameter. If we want to tailor the list of disks at each IPL, we need to specify them all on the IPL command. When the dasd= parameter is specified more than once, the last one is used. This is useful because the patch allows us to override the list of disks defined in the parameter file.

You can do more with this than override the dasd= parameter. Parameters that are not processed by the built-in driver during the boot process will be made available as environment variables to the init process. This means you can pick up values in the boot scripts (e.g. to configure your IP address). If you need the values after the init process has completed, you can take the contents of the /proc/cmdline pseudo file to retrieve the parameters.

11 Network infrastructure design

This chapter uses the networking theory introduced in Chapter 4, “Networking a penguin colony” on page 73 to help you design your virtual networking environment.

11.1 Virtual IP addressing

In this section we describe how to set up our virtual IP address solution using the Linux dummy interface. In 4.2.3, "Virtual IP addressing" on page 79, we introduce the concept of virtual IP addressing using dummy, and this method becomes part of the solution presented in the remainder of this chapter.

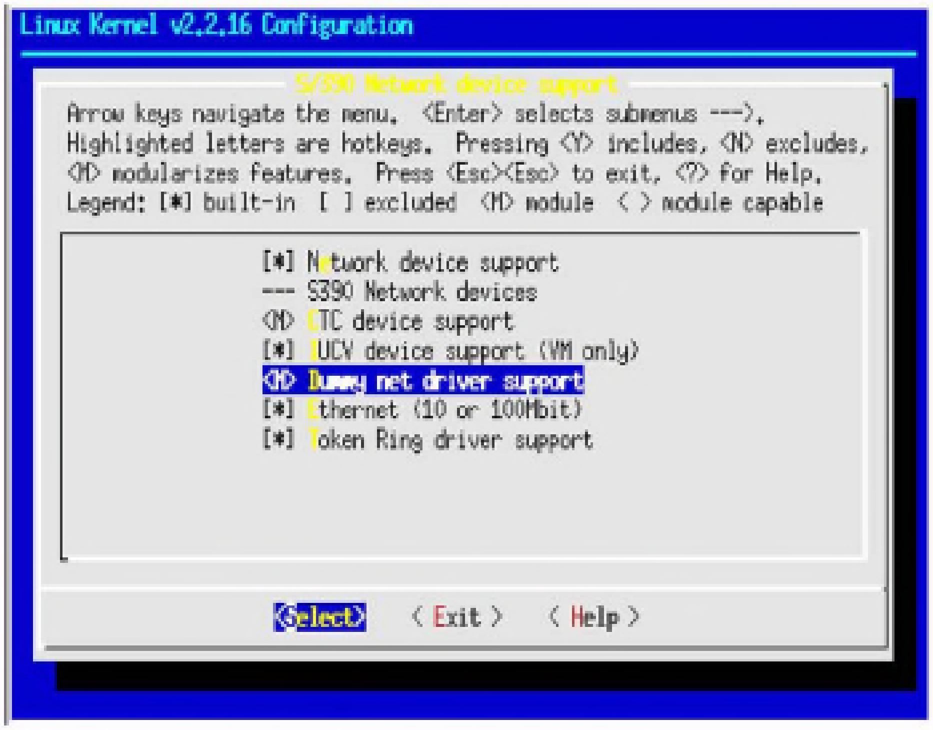
11.1.1 Sample configuration

In our example scenario, we have a single Linux instance which uses private IP addressing on the CTC device to the router, but requires a public IP address for network connectivity.

11.1.2 Compiling dummy.o

If dummy interface support is not present in your kernel, you'll have to build it. Although this is not a major task, instructions on how to build a complete new kernel are beyond the scope of this book. Instead, you can refer to the Linux HOWTOs to find one on kernel complication that describes the basics.

In your kernel configuration file, dummy support is controlled by the CONFIG_DUMMY variable. Make this value `Y` or `M` to include dummy support. Figure 11-1 on page 237 shows a **`make menuconfig`** session at the S/390 Network device support panel. Here, we have selected to add the dummy support as a module. We recommend that the dummy support be built as a module, as this provides the least intrusive way of adding the support to a running system.



```
Linux Kernel v2.2.16 Configuration

                    S/390 Network device support
  Arrow keys navigate the menu.  <Enter> selects submenus --->.
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
  <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help.
  Legend: [*] built-in  [ ] excluded  <M> module  < > module capable

              [*] Network device support
              --- S390 Network devices
              <M> CTC device support
              [*] IUCV device support (VM only)
              <M> Dummy net driver support
              [*] Ethernet (10 or 100Mbit)
              [*] Token Ring driver support

              <Select>    < Exit >    < Help >
```

Figure 11-1   Compiling dummy net driver support into kernel

Once you have updated the kernel configuration, you can make just the network module directory with the following command (issued from the root of the Linux source tree):

```
make modules SUBDIRS=drivers/net
```

In our case, we received the following output from the compilation:

```
# make modules SUBDIRS=drivers/net
make -C  drivers/net CFLAGS="-Wall -Wstrict-prototypes -O2
-fomit-frame-pointer -fno-strict-aliasing -D__SMP__ -pipe
-fno-strength-reduce -DMODULE" MAKING_MODULES=1 modules
make[1]: Entering directory `/usr/src/linux-2.2.16.SuSE/drivers/net'
gcc -D__KERNEL__ -I/usr/src/linux/include -Wall -Wstrict-prototypes -O2
-fomit-frame-pointer -fno-strict-aliasing -D__SMP__ -pipe
-fno-strength-reduce -DMODULE   -c -o dummy.o dummy.c
rm -f $TOPDIR/modules/NET_MODULES
echo   dummy.o >> $TOPDIR/modules/NET_MODULES
echo drivers/net/
drivers/net/
cd $TOPDIR/modules; for i in   dummy.o; do \
    ln -sf ../drivers/net//$i $i; done
make[1]: Leaving directory `/usr/src/linux-2.2.16.SuSE/drivers/net'
```

Once the module has been built, it must be copied to the /lib/modules tree so that the kernel utilities can find the module when required. The easiest way is to simply copy the module using the following command:

```
# cp drivers/net/dummy.o /lib/modules/2.2.16/net/
```

This is obviously for a 2.2.16 kernel; you will need to confirm the correct directory under /lib/modules/ for your system.

Attention: Usually, the command `make modules_install` is used to copy newly-compiled modules into the right /lib/modules directory. However, if you used the `make` command shown above and have not previously built a kernel and full modules in this source tree, do not use the `make modules_install` command to install the new module. Doing so would delete all of your existing modules.

After copying the module, run the command `depmod -a` to recreate the module dependency file. Once this is complete, you can issue `modprobe` or `insmod` to install the dummy module:

```
# insmod dummy
Using /lib/modules/2.2.16/net/dummy.o
```

You are now ready to configure a dummy interface.

11.1.3 Configuring dummy0

In our example, our Linux instance has a CTC device configured with an IP address of 192.168.11.1. We want to add a dummy interface with the IP address 9.12.6.99, which is visible to outside our network. The following command will do this:

```
# ifconfig dummy0 9.12.6.99 broadcast 9.12.6.99 netmask 255.255.255.255 mtu 1500
```

Now, we can ping our interface to see that it is active:

```
# ping 9.12.6.99
PING 9.12.6.99 (9.12.6.99): 56 data bytes
64 bytes from 9.12.6.99: icmp_seq=0 ttl=255 time=10.469 ms
64 bytes from 9.12.6.99: icmp_seq=1 ttl=255 time=5.903 ms
--- 9.12.6.99 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 5.903/8.186/10.469 ms
```

Our dummy interface is ready to receive traffic. To complete the configuration, the rest of the network will have to be configured to direct traffic for the virtual IP address to the correct Linux instance. In our test case, we entered static routes in the intervening routers. This allowed us to connect to services from Windows machines on the network to our Linux instance using the virtual IP address; see Figure 11-2:

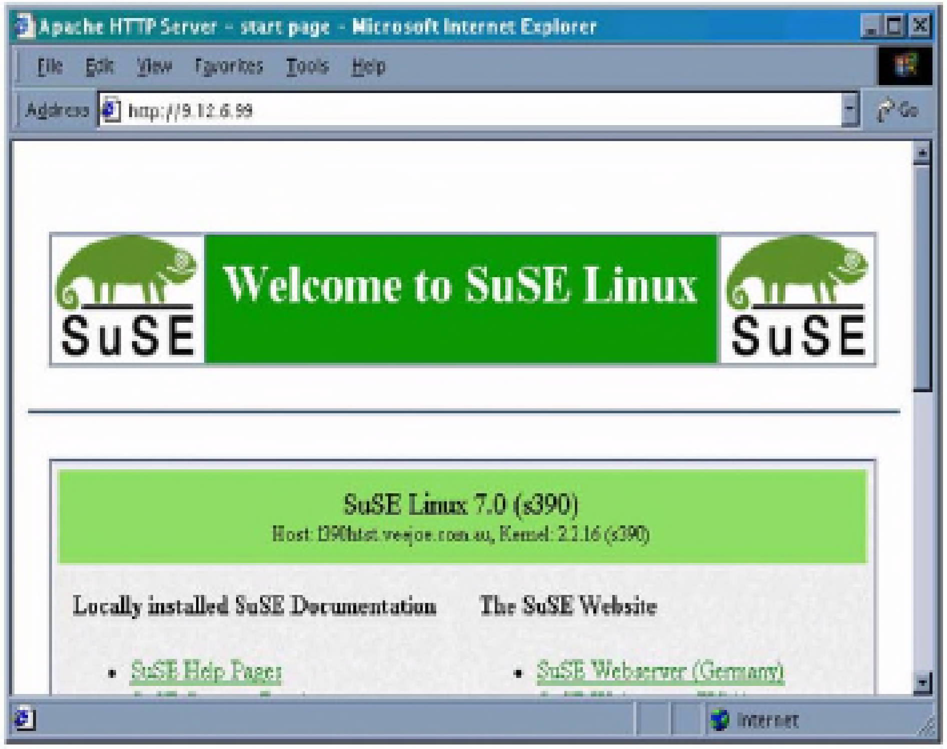

Apache HTTP Server - start page - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://9.12.6.99
Go
Welcome to SuSE Linux
SuSE
SuSE Linux 7.0 (s390)
Host: t390htst.vejoe.com.au, Kernel: 2.2.16 (s390)
Locally installed SuSE Documentation
The SuSE Website
SuSE Help Pages
SuSE Webserver (Germany)
Internet

Figure 11-2 Accessing server using virtual IP address

The next step would be to add the virtual IP address to name resolution services such as DNS or NIS. When clients request the address for our server, these services should respond with the virtual IP address rather than an interface address. The interface addresses can remain in the name server, but with a different name that identifies which *interface* in the server the address belongs to (such as hub port or device name). This can aid problem resolution when tracing connectivity faults.

Important: This method of implementing a virtual IP address only works for connections inbound to the Linux instance. For outbound connections, Linux will use the address of the network interface. We can use NAT to make the traffic appear to originate at the dummy interface, and this is explained in 11.2.12, "Examples for using NAT in the enterprise and ISP/ASP" on page 265.

11.1.4 Using virtual IP addressing in penguin colonies

It may seem intensive to use host routes to access virtual IP addresses on numerous Linux instances. However, when combined with a hierarchical routing model, the routes required to reach the Linux servers can be aggregated into small network routes.

For example, if you have a number of second-level Linux routers with worker penguins behind them, you would allocate a small subnet of IP addresses to each Linux router. The addresses used by the workers would be allocated from this subnet. This enables you to have only a single subnet route, for the addresses behind each Linux router. This process could be continued back up the chain of Linux routers, "supernetting" the addresses from each lower level.

Attention: This process is known as *route summarization*, and is often performed in TCP/IP networks to control and minimize the amount of routing data transferred between routers.

Normally, route summarization is performed automatically by dynamic routing protocols in routers, but the method shown here reduces (or even eliminates) the need to run a dynamic routing protocol within the penguin colony.

In this example, worker penguins reside at the third level of a penguin colony design. Two levels of routing are being performed, with the first level done by a VM TCP/IP guest.

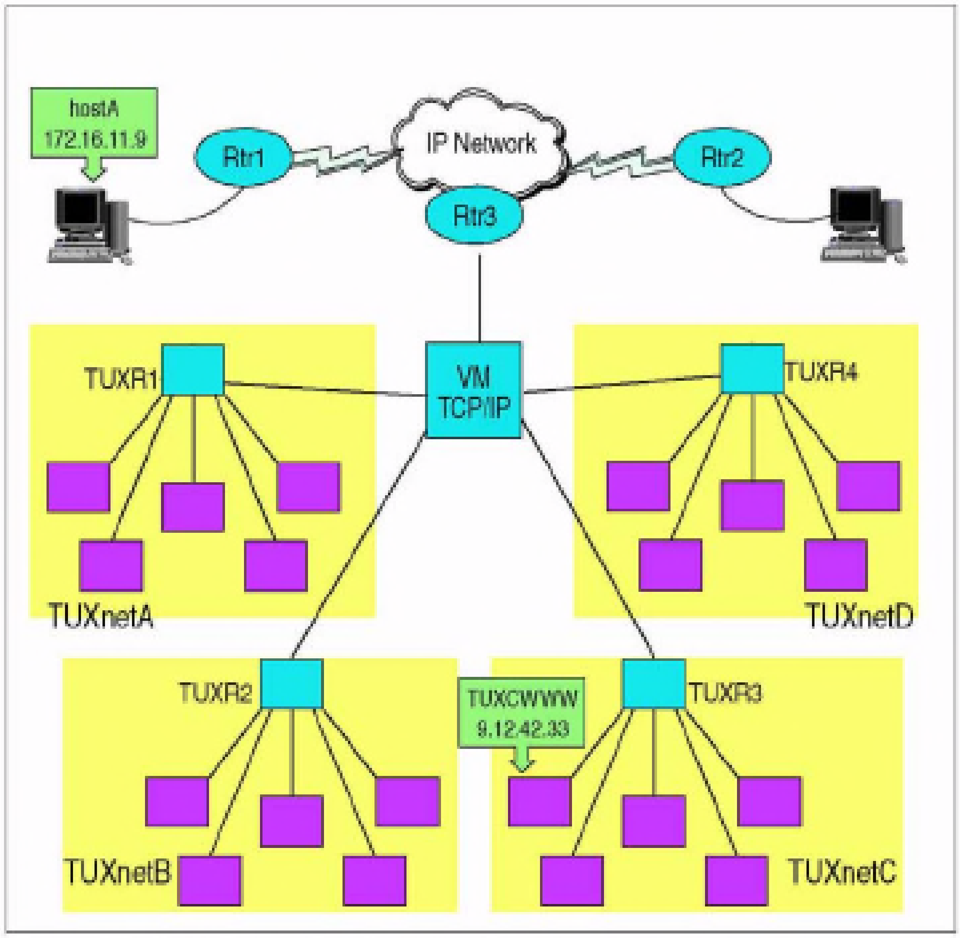


Figure 11-3 Example penguin colony network

We are using IUCV connections inside the penguin colony shown in Figure 11-3, so we work within the limitations of the IUCV driver in having a maximum of 10 IUCV links defined. This means that each router serves nine workers. We also must work within TCP/IP subnetting rules, which dictate that to provide 9 IP addresses in a subnet we must use a '/28' mask, which provides 14 addresses for use (this gives us a small amount of room for expansion).

Virtual Subnet	**Router name**	**Virtual IP hosts**	**Route served**
TUXnetA	TUXR1	9.12.42.1 - 9.12.42.14	9.12.42.0/28
TUXnetB	TUXR2	9.12.42.17 - 9.12.42.30	9.12.42.16/28
TUXnetC	TUXR3	9.12.42.33 - 9.12.42.46	9.12.42.32/28
TUXnetD	TUXR4	9.12.42.49 - 9.12.42.62	9.12.42.48/28

Each router penguin must have the host routes to all the worker penguins it manages (these would usually be added automatically when the interfaces are configured, but because we are using virtual addresses on dummy interfaces, they would have to be configured in advance). In the VM TCP/IP stack, however, only the four subnet routes shown in the previous table are needed. The relevant section of the GATEWAY statement from the TCP/IP PROFILE of the TCP/IP guest is shown here:

```
GATEWAY
; IP Network First        Link     Max. Packet  Subnet          Subnet
; Address    Hop          Name     Size (MTU)   Mask            Value
; ---------- ------------ ------   -----------  ------------    -----------
  9          192.168.1.1  TUXR1    1500         0.255.255.240   0.12.42.0
  9          192.168.1.2  TUXR2    1500         0.255.255.240   0.12.42.16
  9          192.168.1.3  TUXR3    1500         0.255.255.240   0.12.42.32
  9          192.168.1.4  TUXR4    1500         0.255.255.240   0.12.42.48
```

Routers in the enterprise network that need to direct traffic to VM TCP/IP for forwarding now only require a single route that covers all of our worker penguins. In this case, the total range of host addresses goes from 9.12.42.1 to 9.12.42.62. This is the 9.12.42.0 network with a 255.255.255.192 mask.

As an example, if the network uses Cisco routers, the IOS configuration command to add this network route is:

```
Rtr3(config)# ip route 9.12.42.0  255.255.255.192 192.168.0.1
```

where 192.168.0.1 is the address of the OSA adapter used by VM TCP/IP to connect to the IP network. This command would be entered at the router adjacent to the OSA.

Note: It should not be necessary to manually enter this route into all routers in the network. A dynamic routing protocol in the IP network would usually take care of propagating the route to the rest of the network. Also, by using a dynamic routing protocol in VM TCP/IP, the route could be advertised directly from VM.

Alternatively, to allow a Linux host on the same network as the OSA to connect to the penguin colony, the route command would look like this:

```
# route add -net 9.12.42.0 netmask 255.255.255.192 gw 192.168.0.1
```

To verify this, we can follow the path of a packet through the network. In our diagram, hostA starts a Web browser and requests a document from TUXCWWW. The IP address of TUXCWWW is 9.12.42.33, which is within the network 9.12.42.0/26.

The routers in the network have learned that the bottom router in the diagram is the path to the 9.12.42.0/26 network, so the packet is forwarded there, and that router forwards it to VM TCP/IP. When the VM TCP/IP stack receives the packet, it recognizes that 9.12.42.33 is part of the 9.12.42.32/28 network, and forwards the packet to TUXR3.

Finally the packet arrives at TUXR3, and it knows that 9.12.42.33 is a host route accessible via the IUCV connection to the TUXCWWW guest. The packet is forwarded to TUXCWWW, the destination.

The path for return traffic, while inside the penguin colony, would simply use default routes. The worker penguins nominate the router penguin as default, and the router penguins nominate VM TCP/IP as default. VM TCP/IP, in turn, would use the appropriate network router as default. Once into the IP network, normal IP routing would return the traffic to hostA.

If we need to add more routers for more penguins, changing the mask from 255.255.255.192 to 255.255.255.128 would change the definition to the 9.12.42.0/25 network, which covers host addresses from 9.12.42.1 to 9.12.42.126. This means that we can add another four routers to the configuration, carrying on the same pattern for subnet allocation as the original four routers.

Attention: Consult your networking team, or a reference on TCP/IP routing, before you decide on an addressing scheme. There are dependencies on subnetting and supernetting that can become complex and are not easily changed after the network is in production, so planning for expansion is very important.

Also, the details of ensuring that the routes to the worker penguins are properly distributed to the IP network must be analyzed. As host configurators, don't assume that just because the routing network uses a dynamic routing protocol, that the right thing will happen "automagically". Discuss your requirements with the network team, so that a solution which is mutually agreeable can be achieved.

11.2 Packet filtering and NAT with IPTables

In this section we present an example configuration using the designs discussed in 4.3, "Packet filtering and Network Address Translation" on page 81.

11.2.1 What you need to run packet filtering

To set up a packet filter server with IPTables, your Linux installation needs to meet the following requirements:

1. You need kernel Version 2.4.5 or higher with the latest S/390 patches from the Linux for S/390 developerWorks page at:

   `http://www10.software.ibm.com/developerworks/opensource/linux390/index.shtml`

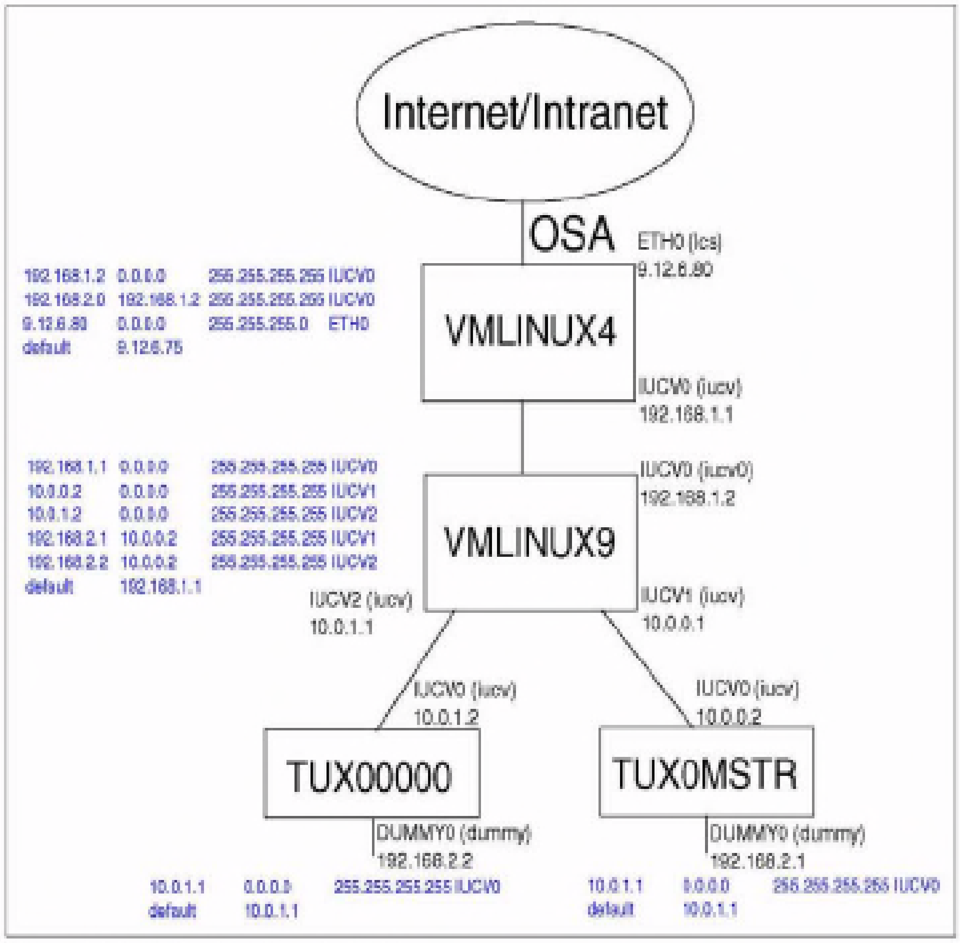
   We recommend that you use the latest available stable version. The kernel has to be compiled with support for netfilters. This means that you have to select **CONFIG_NETFILTER** in the kernel configuration. Depending of your packet filtering configuration, you also need to select the proper configuration options under **IP: Netfilter Configuration**. We recommend that you compile all your networking options and available modules. If you want to use your Linux server as a router, choose **IP - advanced router** under **TCP/IP networking**. This will also increase the routing performance.
2. Loadable kernel modules version 2.4.6 or newer with the latest S/390 patches from the Linux for S/390 developerWorks page.
3. IPTables 1.2.2 or newer.

Important: IPTables is the follow-on to IPChains, which was used in the 2.2 kernel.

If your Linux installation does not include the required software levels, you should download all available patches and compile new versions of the software yourself. Usually the kernel that comes with a distribution is not enabled to support netfilters, so you need to recompile the kernel with the required support.

11.2.2 Network configuration for a packet filtering implementation

In this section we describe our lab network setup for implementing a packet filtering solution.



```
192.168.1.2  0.0.0.0      255.255.255.255 IUCV0
192.168.2.0  192.168.1.2  255.255.255.255 IUCV0
9.12.6.80    0.0.0.0      255.255.255.0   ETH0
default      9.12.6.75
```

```
192.168.1.1  0.0.0.0      255.255.255.255 IUCV0
10.0.0.2     0.0.0.0      255.255.255.255 IUCV1
10.0.1.2     0.0.0.0      255.255.255.255 IUCV2
192.168.2.1  10.0.0.2     255.255.255.255 IUCV1
192.168.2.2  10.0.0.2     255.255.255.255 IUCV2
default      192.168.1.1
```

```
10.0.1.1     0.0.0.0      255.255.255.255 IUCV0
default      10.0.1.1
```

```
10.0.1.1     0.0.0.0      255.255.255.255 IUCV0
default      10.0.1.1
```

Figure 11-4 Lab network setup No.1

As you can see in Figure 11-4, we set up a two-layered router/firewall implementation. All our Linux images are running inside VM. In the example we have the following setup:

1. vmlinux4

   vmlinux4 is acting as a main router for our penguin colony. It is connected to the outside world via an Ethernet connection using an lcs driver. The setup for the eth0 interface is as follows:

   ETH0 9.12.6.80 255.255.255.0

   vmlinux4 is then connected to the second-layer router/firewall VM Linux9 over the IUCV connection:

   IUCV0 192.168.1.1 Pointopoint 192.168.1.2

   In our example, we used SuSE 7.2 31-bit distribution; the entries in the /etc/rc.config file for this setup are shown in Example 11-1 on page 246.

Example 11-1 /etc/rc.config file entries for vmlinux4

```
#
# Networking
#
# Number of network cards: "_0" for one, "_0 _1 _2 _3" for four cards
#
NETCONFIG="_0 _1"

#
# IP Adresses
#
IPADDR_0="9.12.6.80"
IPADDR_1="192.168.1.1"
IPADDR_2="192.168.2.1"
IPADDR_3="10.0.0.1"

#
# Network device names (e.g. "eth0")
#
NETDEV_0="eth0"
NETDEV_1="iucv0"
NETDEV_2="NONE"
NETDEV_3="NONE

#
# Parameters for ifconfig, simply enter "bootp" or "dhcpclient" to use the
# respective service for configuration.
# Sample entry for ethernet:
# IFCONFIG_0="192.168.81.38 broadcast 192.168.81.63 netmask 255.255.255.224"
#
IFCONFIG_0="9.12.6.80 broadcast 9.12.6.255 netmask 255.255.255.0 mtu 1492 up"
IFCONFIG_1="192.168.1.1 pointopoint 192.168.1.2 mtu 1492 up"
IFCONFIG_2=""
IFCONFIG_3=""

#
# Runtime-configurable parameter: forward IP packets.
# Is this host a router? (yes/no)
#
IP_FORWARD="yes"
```

We also have to set up the correct routing tables so that our second-level router/firewall and the servers connected to it will be accessible from the outside world. In our example, all the servers connected to the second-layer router/firewall have IP addresses in the 192.168.2.0 subnet.

In our example we have the following routes:

– Default route 9.12.6.75
– Route to the 192.168.2.0 subnet via the second-layer router/firewall

This means that we forward all the traffic to the subnet 192.168.2.0 to the second-level router/firewall.

Example 11-2 shows the /etc/route.conf file used in our installation:

Example 11-2 /etc/route.conf file for vmlinux4

```
9.12.6.0            0.0.0.0             255.255.255.0       eth0
192.168.1.2         0.0.0.0             255.255.255.255     iucv0
default             9.12.6.75           0.0.0.0             eth0
192.168.2.0         192.168.1.2         255.255.255.0       iucv0
```

2. vmlinux9

   vmlinux9 is our second layer router/firewall. vmlinux9 is connected to the vmlinux4 over the IUCV connection IUCV0:

   ```
   IUCV0 192.168.1.2 Pointopoint 192.168.1.1
   ```

   Each server connected to the router/firewall on the second layer uses its own IUCV interface. In our example, we have two servers connected with IUCV1 and IUCV2 connections:

   ```
   IUCV0 10.0.0.1 Pointopoint 10.0.0.2
   IUCV0 10.0.1.1 Pointopoint 10.0.1.2
   ```

   Example 11-3 shows the /etc/rc.config file entries for this setup:

Example 11-3 /etc/rc.config file entries for vmlinux9

```
#
# Networking
#
# Number of network cards: "_0" for one, "_0 _1 _2 _3" for four cards
#
NETCONFIG="_0 _1 _2"

#
# IP Adresses
#
IPADDR_0="192.168.1.2"
IPADDR_1="10.0.0.1"
IPADDR_2="10.0.1.1"
IPADDR_3=""

#
# Network device names (e.g. "eth0")
#
NETDEV_0="iucv0"
```

```
NETDEV_1="iucv1"
NETDEV_2="iucv2"
NETDEV_3=""

#
# Parameters for ifconfig, simply enter "bootp" or "dhcpclient" to use the
# respective service for configuration.
# Sample entry for ethernet:
# IFCONFIG_0="192.168.81.38 broadcast 192.168.81.63 netmask 255.255.255.224"
#
IFCONFIG_0="192.168.1.2 pointopoint 192.168.1.1 mtu 1492 up"
IFCONFIG_1="10.0.0.1 pointopoint 10.0.0.2 mtu 1492 up"
IFCONFIG_2="10.0.1.1 pointopoint 10.0.1.2 mtu 1492 up"
IFCONFIG_3=""

#
# Runtime-configurable parameter: forward IP packets.
# Is this host a router? (yes/no)
#
IP_FORWARD="yes"
```

We also have to set up the correct routing tables so that the servers connected to this router will be accessible from outside world. In our example, all servers have IP addresses in the 192.168.2.0 subnet and we have the following routes:

- Default route 192.168.1.1
- Route to the 192.168.2.1 address via the IUCV1 connection
- Route to the 192.168.2.2 address via the IUCV2 connection

The traffic for each individual server is forwarded to the unique connection used just for this server. In this case, only the packets with the address of the destination server will go to this server.

Example 11-4 shows the /etc/route.conf file used in our configuration:

Example 11-4 /etc/route.conf file for vmlinux9

```
192.168.1.1        0.0.0.0            255.255.255.255      iucv0
10.0.0.2           0.0.0.0            255.255.255.255      iucv1
10.0.1.2           0.0.0.0            255.255.255.255      iucv2
default            192.168.1.1        0.0.0.0              iucv0
192.168.2.1        10.0.0.2           255.255.255.255      iucv1
192.168.2.2        10.0.1.2           255.255.255.255      iucv2
```

3. tux0mstr

   tux0mstr is a first server in our colony. It has an IUCV connection IUCV0 to the second-layer router/firewall vmlinux9:

   ```
   IUCV0 10.0.0.2 Pointopoint 10.0.0.1
   ```

The real address of the server is implemented on the DUMMY0 interface. We used this approach because we wanted to have the real subnet in the connection to the router/firewall:

```
DUMMY0 192.168.2.1 255.255.255.255
```

Example 11-5 shows the corresponding entries in the /etc/rc.config file used in our configuration:

Example 11-5 /etc/rc.config file entries for tux0mstr

```
#
# Networking
#
# Number of network cards: "_0" for one, "_0 _1 _2 _3" for four cards
#
NETCONFIG="_0 _1"

#
# IP Adresses
#
IPADDR_0="10.0.0.2"
IPADDR_1="192.168.2.1"
IPADDR_2=""
IPADDR_3=""

#
# Network device names (e.g. "eth0")
#
NETDEV_0="iucv0"
NETDEV_1="dummy0"
NETDEV_2=""
NETDEV_3=""

#
# Parameters for ifconfig, simply enter "bootp" or "dhcpclient" to use the
# respective service for configuration.
# Sample entry for ethernet:
# IFCONFIG_0="192.168.81.38 broadcast 192.168.81.63 netmask 255.255.255.224"
#
IFCONFIG_0="10.0.0.2 pointopoint 10.0.0.1 mtu 1492 up"
IFCONFIG_1="192.168.2.1 broadcast 192.168.2.1 netmask 255.255.255.255 mtu 1492
up"
IFCONFIG_2=""
IFCONFIG_3=""

#
# Runtime-configurable parameter: forward IP packets.
# Is this host a router? (yes/no)
#
```

```
IP_FORWARD="yes"
```

For the routing, we just need to set up the default route to the second-layer router/firewall. This means that all the packets are traveling back to the router.

Example 11-6 shows the /etc/route.conf file used in our configuration:

Example 11-6 tux0mstr /etc/route.conf file

```
10.0.0.1                  0.0.0.0                   255.255.255.255         iucv0
default                   10.0.0.1                  0.0.0.0                 iucv0
```

4. TUX00000

   TUX00000 is our second server in the colony and uses the same approach for the network setup as tux0mstr. Example 11-7 shows the /etc/rc.config file entries:

Example 11-7 /etc/rc.config file entries for TUX00000

```
#
# Networking
#
# Number of network cards: "_0" for one, "_0 _1 _2 _3" for four cards
#
NETCONFIG="_0 _1"

#
# IP Adresses
#
IPADDR_0="10.0.1.2"
IPADDR_1="192.168.2.2"
IPADDR_2=""
IPADDR_3=""

#
# Network device names (e.g. "eth0")
#
NETDEV_0="iucv0"
NETDEV_1="dummy0"
NETDEV_2=""
NETDEV_3=""

#
# Parameters for ifconfig, simply enter "bootp" or "dhcpclient" to use the
# respective service for configuration.
# Sample entry for ethernet:
# IFCONFIG_0="192.168.81.38 broadcast 192.168.81.63 netmask 255.255.255.224"
#
IFCONFIG_0="10.0.1.2 pointopoint 10.0.1.1 mtu 1492 up"
```

```
IFCONFIG_1="192.168.2.2 broadcast 192.168.2.2 netmask 255.255.255.255 mtu 1492
up"
IFCONFIG_2=""
IFCONFIG_3=""

#
# Runtime-configurable parameter: forward IP packets.
# Is this host a router? (yes/no)
#
IP_FORWARD="yes"
```

Example 11-8 shows the /etc/route.conf file used in our configuration:

Example 11-8 TUX00000 /etc/route.conf file

```
10.0.1.1              0.0.0.0               255.255.255.255       iucv0
default               10.0.1.1              0.0.0.0               iucv0
```

As you can see from the /etc/rc.config file entries, we enabled IP forwarding on both of our routers/firewalls and servers. With IP forwarding enabled, Linux can act as a router. In our example we did not use any router daemons for the routing purposes, but instead simply defined static routes.

11.2.3 How to permanently enable IP forwarding

By default, IP Forwarding is not enabled. In order to enable it, edit the /etc/rc.config file and make sure IP_FORWARD is set to `yes`. You can check the status of IP_FORWARD setting with the following command:

```
# grep IP_FORWARD /etc/rc.config
```

The output should be similar to:

```
IP_FORWARD="yes"
```

Run SuSEconfig to commit the changes, and restart the network by typing the following:

```
# rcnetwork restart
```

Alternatively, you can execute the following command:

```
# /etc/init.d/network restart
```

You can check if your IP forwarding is enabled by executing the following command:

```
# cat /proc/sys/net/ipv4/ip_forward
```

If IP forwarding is enabled, the output of this command will be `1`.

Now your server is ready to act as a router. You can verify this by pinging the eth0 interface with IP address 9.12.6.80 from the tux0mstr Linux on 192.168.2.1 IP address; you are pinging the external interface in our main router from the Linux server on the 10.0.0.2 IP.

As you'll notice, there will be no reply to this ping because the **ping** command will use the IP address of the IUCV0 connection as the source address—not the address of the DUMMY0 adapter that is used to assign the external address of the server. When the packet is received by the main router/firewall, this will reply to the address 10.0.0.2, because this address was specified as the source for the ping.

Since we have not have defined any special route to the 10.0.0.0 subnet, the packet will go the default gateway 9.12.6.75, and of course the gateway will drop the packet because it does not have the route definition for the 10.0.0.0 subnet. To resolve this issue, we have to add the following route to the /etc/route.conf file on the vmlinux4 Linux system:

– Route to the 10.0.0.0 subnet via the second layer router/firewall

Example 11-9 shows the modified /etc/route.conf file:

Example 11-9 Modified /etc/rc.config file

```
9.12.6.0        0.0.0.0         255.255.255.0       eth0
192.168.1.2     0.0.0.0         255.255.255.255     iucv0
default         9.12.6.75       0.0.0.0             eth0
192.168.2.0     192.168.1.2     255.255.255.0       iucv0
10.0.0.0        192.168.1.2     255.255.0.0         iucv0
```

After modifying this file, you should execute the following command:

```
# /etc/initd.d/route restart
```

Now you should try to execute the **ping** command on the tux0mstr Linux again:

```
# ping 9.12.6.80
```

If the ping is successful, both of your routers are working correctly. You will see output similar to Example 11-10:

Example 11-10 Pinging the main router external interface

```
tux0mstr:~ # ping 9.12.6.80
PING 9.12.6.80 (9.12.6.80): 56 data bytes
64 bytes from 9.12.6.80: icmp_seq=0 ttl=254 time=0.405 ms
64 bytes from 9.12.6.80: icmp_seq=1 ttl=254 time=0.425 ms
64 bytes from 9.12.6.80: icmp_seq=2 ttl=254 time=0.430 ms
64 bytes from 9.12.6.80: icmp_seq=3 ttl=254 time=0.427 ms
64 bytes from 9.12.6.80: icmp_seq=4 ttl=254 time=0.403 ms
```

```
--- 9.12.6.80 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max = 0.403/0.418/0.430 ms
```

To access the servers on 192.168.2.0 subnet in our colony from the outside world, the routers on the network have to be updated to reflect this configuration. This means that all the traffic for the 192.168.2.0 subnet has to go to the IP address 9.12.6.80 in our example.

In our test environment, we had the computers on the same subnet as our main router external interface. So we simply added the static route with the command from the Windows 2000 command prompt, and we were able to access the servers in our colony:

```
C:\> route add 192.168.2.0 mask 255.255.255.0 9.12.6.80
```

At this point, following our process, the routers for your Linux colony inside the zSeries should also be successfully set up.

11.2.4 The first IP Tables rules

Now we can deploy the routers. We want to limit the access to our servers in the colony, as shown in Figure 11-4 on page 245, so in our example, we'll allow only HTTP protocol to our servers. Before we implement our protection, however, we try to ping our server with the command:

```
# ping 192.168.2.1
```

Example 11-11 shows the result:

Example 11-11 Pinging the server

```
C:\>ping 192.168.2.1

Pinging 192.168.2.1 with 32 bytes of data:

Reply from 192.168.2.1: bytes=32 time=10ms TTL=253
Reply from 192.168.2.1: bytes=32 time<10ms TTL=253
Reply from 192.168.2.1: bytes=32 time<10ms TTL=253
Reply from 192.168.2.1: bytes=32 time<10ms TTL=253

Ping statistics for 192.168.2.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum =  10ms, Average =  2ms
```

The following steps will implement security rules for the servers.

Note: We discuss the meaning of the iptables commands in "Using IP Tables" on page 256.

1. We created a new chain for our rules with the command:

   ```
   # iptables -N httpallow
   ```

2. We defined the rule, in our httpallow chain, which allows all connections from the computer we're using for remote management of the router/firewall with the command:

   ```
   # iptables -A httpallow -s 9.12.6.133 -j ACCEPT
   ```

3. We defined the rules, in our httpallow chain, which allows building a new connection only for the HTTP protocol with the command:

   ```
   # iptables -A httpallow -m state --state NEW -p TCP --dport www -j ACCEPT
   # iptables -A httpallow -m state --state NEW -p UDP--dport www -j ACCEPT
   ```

4. We defined the rule, in our httpallow chain, which keeps alive the established and related connections with the command:

   ```
   # iptables -A httpallow -m state --state ESTABLISHED,RELATED -j ACCEPT
   ```

5. With the following command we defined the rule, in our httpallow chain, which drops all other incoming packets:

   ```
   # iptables -A httpallow -j DROP
   ```

6. Finally, with the following commands, we need to define that all the packets from the INPUT and FORWARD chain will jump into our httpallow chain:

   ```
   # iptables -A INPUT -j httpallow
   # iptables -A FORWARD -j httpallow
   ```

Now we were ready to test our security implementation. We pinged the server with the command:

```
# ping 192.168.2.1
```

Example 11-12 shows the output:

Example 11-12 Pinging after applying security rules

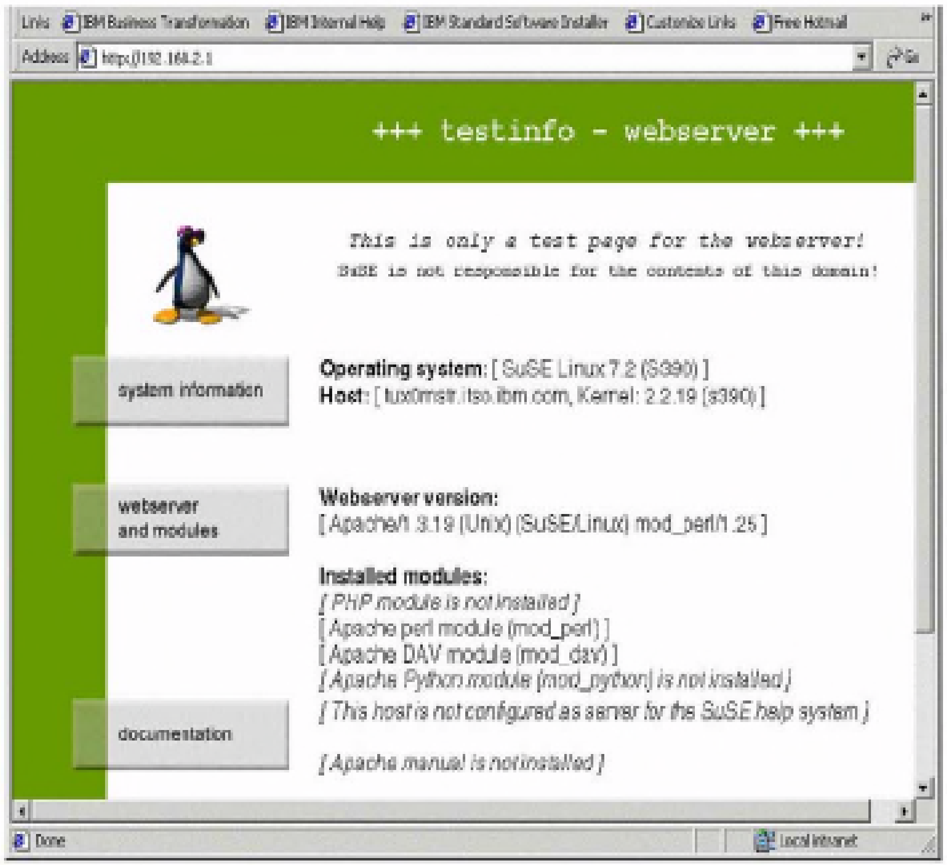
```
C:\>ping 192.168.2.1

Pinging 192.168.2.1 with 32 bytes of data:

Reply from 9.32.44.3: Destination host unreachable.
Reply from 9.32.44.3: Destination host unreachable.
Reply from 9.32.44.3: Destination host unreachable.
Request timed out.
```

```
Ping statistics for 192.168.8.1:
    Packets: Sent = 4, Received = 3, Lost = 1 (25% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum =  0ms, Average =  0ms
```

As you can see, **`ping`** is not working. This means our security was successful! Next we tried to connect to the server with a browser; Figure 11-5 shows the screen we received:



+++ testinfo - webserver +++

This is only a test page for the webserver!
SuSE is not responsible for the contents of this domain!

Operating system: [SuSE Linux 7.2 (S390)]
Host: [tux0mstr.itso.ibm.com, Kernel: 2.2.19 (s390)]

Webserver version:
[Apache/1.3.19 (Unix) (SuSE/Linux) mod_perl/1.25]

Installed modules:
[PHP module is not installed]
[Apache perl module (mod_perl)]
[Apache DAV module (mod_dav)]
[Apache Python module (mod_python) is not installed]
[This host is not configured as server for the SuSE help system]

[Apache manual is not installed]

Figure 11-5 Accessing the Web server after applying security rules

11.2.5 Checking your filter

With the `/usr/sbin/iptables` command you can set up your rules for packet checking.

Note: By default, all checking policies are set to Accept. This means that all packets can come in, go through, or go out from your server without any restrictions.

You can examine the current checking policies by using the `-L` flag with the `iptables` command. You should see output similar to that shown in Example 11-13:

Example 11-13 Listing of the default IP Tables policies

```
# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

11.2.6 Using IP Tables

With the `/usr/sbin/iptables` command, you can create, change or delete your own policies for checking packets or you can modify built-in policies. You cannot delete the built-in chains, but you can append your rules to the existing chains or even create your own chains.

To manage whole chains you can use the parameters described in Table 11-1.

Table 11-1 Parameters for managing the whole chains

Parameter	Description
-N	Create a new chain
-X	Delete an empty chain
-P	Change the policy for a built-in chain
-L	List the rules in a chain
-F	Flush the rules out of a chain
-Z	Zero the packets and the byte counters on all rules in a chain
-E	Rename the chain

For manipulating rules inside the chain, you can use the parameters explained in Table 11-2.

Table 11-2 Parameters for managing rules in the chain

Parameter	Description
-A	Append new rule to a chain
-I	Insert a new rule in a chain at some position
-R	Replace a rule at some position in a chain
-D	Delete a rule at some position in a chain
-D	Delete the first rule that matches in a chain

11.2.7 How to create a rule

Each rule specifies a set of conditions the packet must meet, and what to do if those conditions are met.

The most common syntax for creating a new rule is as follows:

```
# /usr/sbin/iptables -t table -A INPUT -s source -d destination \
# -p protocol -i input_interface -o output_interface -f -p extension \
# -m match_extension -j target
```

The parameters are described in the Table 11-3.

Table 11-3 IP Tables parameters

Parameter	Description
-t table	The table to manipulate. If you omit this parameter, the table "filter" will be used. The "filter" table holds the rules for packet filtering.
-A INPUT	Append a new rule to an INPUT chain.
-s source	IP address or host name of the source.
-d destination	IP address or host name of the destination.
-p protocol	Type of the protocol to which a rule is applied.
-i input_interface	The input network interface to match.
-o output_interface	The output network interface to match.
-f	Fragments flag - if this is used, the rule is only valid for second and further fragments through.

Parameter	Description
-p extension	With this you invoke extension. The following "new match" extensions are available (you can also use a custom-supplied extension): 1. TCP (-p tcp), the parameters are: --tcp-flags (ALL,SYN,ACK,FIN,RST,URG,PSH,NONE) --syn (short for --tcp-flags SYN,RST,ACK,SYN) --sport (source port) --dport (destination port) --tcp-option (examine TCP options) 2. UDP (-p udp), the parameters are: --sport (source port) --dport (destination port) 3. ICMP (-p icmp), the parameters are: --icmp-type (icmp type)
-m match_extension	With this option you load "other match" extensions: 1. mac (-m mac), the parameters are: --mac-source (source MAC address) 2. limit (-m limit), the parameters are: --limit (maximum average number of matches per second) --limit-burst (maximum burst, before --limit takes over) 3. owner (-m owner), the parameters are: --uid-owner (user ID) --gid-owner (group ID) --pid-owner (process ID) --sid-owner (session ID) 4. state (-m state), the parameters are: --state (NEW - new packets, ESTABLISHED - packets which belong to the established connection, RELATED - related packets, INVALID - unidentified packets)

Parameter	Description
-j target	What we do with the packet that matches the rule. The built-in targets are: 1. ACCEPT - packet will be accepted 2. DROP - packet will be dropped For the target, you can also use a user-defined chain. By providing a kernel module or iptables extension, you can have additional targets. The default extension in the iptables distribution are: 1. LOG - kernel logging of matching packets; the parameters are: --log-level (log level) --log-prefix (the string up to 29 characters, which will show at the beginning of the message) 2. REJECT - the same as DROP, but sender is sent the ICMP port unreachable error message. In some cases the message is not sent - RFC 1122, the parameters are: --reject-with (you can alter the replay packet used) There are also two special built-in targets: 1. RETURN - for a built-in chain, the policy of the chain is executed; for a user-defined chain, the traversal continues at the previous chain, just after the rule which jumped to this chain 2. QUEUE - the packet is queued to the userspace processing

For example, if you want to create a rule for denying the ICMP protocol packets, which are used when you execute the ping command, for a specific IP address you will do this by executing the command:

```
# /usr/sbin/iptables -A input -s IP_address -p icmp -j DROP
```

If you omit the protocol definition, *all* packets will be denied. So for example, to block access to your machine from network 172.168.1.0 with subnet mask 255.255.255.0, execute the following command:

```
# /usr/sbin/iptables -A input -s 172.168.1.0/255.255.255.0 -j DROP
```

Or you can use:

```
# /usr/sbin/iptables -A input -s 172.168.1.0/24 -j DROP
```

As you can see, the subnet mask can be specified with the number of used bits for that mask.

To disallow *any* traffic from your server to network 172.168.1.0 with subnet mask 255.255.255.0, use this command:

```
# /usr/sbin/iptables -A output -d 172.168.1.0/24 -j DROP
```

Here we used the -d parameter to specify the destination address.

11.2.8 Using the inversion ! option

With some parameters, you can use the inversion option !, which means that the rule will be applied to everything *except* the parameters specified after !. For example, if you want to deny packets that come from all IP addresses except from network 192.168.1.0 with subnet mask 255.255.255.0, you can do this by executing the command:

```
# /usr/sbin/iptables -A input -s ! 192.168.1.0/24 -j DROP
```

Note: The rules you make are not permanent, so next time you restart the server they will be lost.

11.2.9 Making the rules permanent

To make rules permanent, you have to create the script with the IPTables commands and integrate this script into your boot process. Using the example from 11.2.4, "The first IP Tables rules" on page 253, we created a script similar to that shown in Example 11-14:

Example 11-14 Script for setting up rules

```
#! /bin/sh
/usr/sbin/iptables -N httpallow
/usr/sbin/iptables -A httpallow -s 9.12.6.133 -j ACCEPT
/usr/sbin/iptables -A httpallow -m state --state NEW -p TCP --dport www -j ACCEPT
/usr/sbin/iptables -A httpallow -m state --state ESTABLISHED,RELATED -j ACCEPT
/usr/sbin/iptables -A httpallow -j DROP
/usr/sbin/iptables -A INPUT -j httpallow
/usr/sbin/iptables -A FORWARD -j httpallow
```

In our example we named this script /etc/init.d/filters. Because the default run level for our system is 3, we positioned this script to execute at the end of the run level 3 with the following command:

```
# ln -s /etc/init.d/rc3.d/S99filters /etc/init.d/filters
```

After rebooting, we verified that the rules were loaded by using the following command:

```
# /usr/sbin/iptables -L
```

Example 11-15 shows our output:

Example 11-15 Current rules

```
# /usr/sbin/iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
httpallow  all  --  anywhere             anywhere

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination
httpallow  all  --  anywhere             anywhere

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination

Chain httpallow (2 references)
target     prot opt source               destination
ACCEPT     all  --  tot67.itso.ibm.com   anywhere
ACCEPT     tcp  --  anywhere             anywhere           state NEW tcp
dpt:http
ACCEPT     all  --  anywhere             anywhere           state
RELATED,ESTABLISHED
DROP       all  --  anywhere             anywhere
```

11.2.10 Sample packet filtering configuration for ISP/ASP

In this section we show how to create packet filtering rules to protect the ISP/ASP environment. For this environment, we assume that the following services will be offered on the servers:

1. Web service - HTTP, HTTPS
2. FTP service - FTP
3. SMTP service - SMTP
4. POP3 service - POP3
5. Secure shell service - SSH

For the rules for each service, we follow the approach described in 11.2.4, "The first IP Tables rules" on page 253. For each type of service, we will allow NEW TCP and UDP packets, and packets from ESTABLISHED and RELATED connections.

In our example, we will allow all packets from the administration computer with IP address 9.12.6.133; all other packets will be dropped. Example 11-16 on page 262 shows the script we used to set these rules; we named this script /etc/init.d/filters.

Example 11-16 ISP/ASP packet filtering security

```
#! /bin/sh
/usr/sbin/iptables -N httpallow
/usr/sbin/iptables -A httpallow -s 9.12.6.133 -j ACCEPT
/usr/sbin/iptables -A httpallow -m state --state ESTABLISHED,RELATED -j ACCEPT
/usr/sbin/iptables -A httpallow -m state --state NEW -p TCP --dport www -j ACCEPT
/usr/sbin/iptables -A httpallow -m state --state NEW -p UDP --dport www -j ACCEPT
/usr/sbin/iptables -A httpallow -m state --state NEW -p TCP --dport https -j ACCEPT
/usr/sbin/iptables -A httpallow -m state --state NEW -p UDP --dport https -j ACCEPT
/usr/sbin/iptables -A httpallow -m state --state NEW -p TCP --dport ftp-data -j ACCEPT
/usr/sbin/iptables -A httpallow -m state --state NEW -p UDP --dport ftp-data -j ACCEPT
/usr/sbin/iptables -A httpallow -m state --state NEW -p TCP --dport ftp -j ACCEPT
/usr/sbin/iptables -A httpallow -m state --state NEW -p TCP --dport smtp -j ACCEPT
/usr/sbin/iptables -A httpallow -m state --state NEW -p UDP --dport smtp -j ACCEPT
/usr/sbin/iptables -A httpallow -m state --state NEW -p TCP --dport pop3 -j ACCEPT
/usr/sbin/iptables -A httpallow -m state --state NEW -p UDP --dport pop3 -j ACCEPT
/usr/sbin/iptables -A httpallow -m state --state NEW -p TCP --dport ssh -j ACCEPT
/usr/sbin/iptables -A httpallow -m state --state NEW -p UDP --dport ssh -j ACCEPT
/usr/sbin/iptables -A httpallow -m state --state NEW -p TCP --dport domain -j ACCEPT
/usr/sbin/iptables -A httpallow -m state --state NEW -p UDP --dport domain -j ACCEPT
/usr/sbin/iptables -A httpallow -j DROP
/usr/sbin/iptables -A INPUT -j httpallow
/usr/sbin/iptables -A FORWARD -j httpallow
```

Note: This script should be started each time the server is restarted, as described in 11.2.9, "Making the rules permanent" on page 260.

We also created a script for deleting all rules from the /etc/init.d/filters script. This script is called /etc/init.d/filters-down and is shown in Example 11-17.

Example 11-17 Script for deleting rules

```
#! /bin/sh
/usr/sbin/iptables -F httpallow
/usr/sbin/iptables -F INPUT
/usr/sbin/iptables -F FORWARD
/usr/sbin/iptables -X httpallow
```

To delete *all* rules, execute the following command:

```
# /etc/init.d/filters-down
```

To re-enable the rules, execute the following command:

```
# /etc/init.d/filters
```

Tip: If for some reason your network is not working after executing the script /etc/init.d/filters-down, you should execute the following commands:

```
# /etc/init.d/network restart
# /etc/init.d/route restart
```

11.2.11 Using IPTables for NAT

In 4.3.5, "Network Address Translation (NAT)" on page 88, we introduced the concept of NAT. In this section we discuss the types of NAT and how they are set up using IPtables.

To manipulate NAT chains we use IP Tables, the same tool used for packet filtering—but instead of using the default table "filter", we use table "nat".

Note: In the Linux 2.2 kernel, different tools were used to set up filtering and to set up NAT, so this is an improvement.

Figure 11-6 shows where NAT takes place in the packet's journey through the router/firewall.

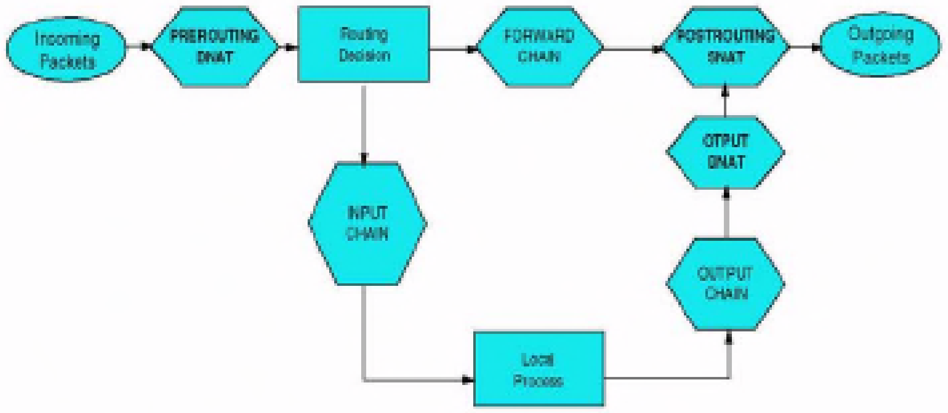


Figure 11-6 NAT role in IP packet travel

Basically, NAT is implemented using the same principle as packet filtering, using three built-in chains to control address translation. As you can see in Figure 11-6 those chains are:

1. PREROUTING - for DNAT, when packets first come in
2. POSTROUTING - for SNAT, when packets leave
3. OUTPUT - for DNAT of locally generated packets

Source NAT

Source NAT is specified with rule `-j SNAT` and the `--to-source` option, which specifies the IP address, a range of the IP addresses, and optional port of range of ports. You can also use the `-o` (outgoing interface) option to specify that the rule only applies to traffic on a particular interface.

For example, to change the source address of your packet to 172.168.2.1, execute the command:

```
# iptables -t nat -A POSTROUTING -o eth0 -j SNAT --to-source 172.168.2.1
```

To change the source address to 172.168.2.1, ports 1-1023, use this command:

```
# iptables -t nat -A POSTROUTING -p tcp -o eth0 -j SNAT \
--to-source 172.168.2.1:1-1023
```

Masquerading

See 4.3.5, "Network Address Translation (NAT)" on page 88 for a discussion of masquerading.

Masquerading is specified using rule `-j MASQUERADE`. For example, to masquerade everything leaving our router on eth0, we used:

```
# iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

Destination NAT

Destination NAT is specified with the rule `-j DNAT` and the `--to-destination` option, which specifies the IP address, a range of the IP addresses, and optional port of range of ports. You can also use the `-i` (incoming interface) option to specify that the rule is to apply only to a particular interface.

To alter the destination of locally generated packets, use the OUTPUT chain.

For example, to change the destination address to 192.168.10.1, execute the following command:

```
# iptables -t nat -A PREROUTING -i eth0 -j DNAT \
--to-destination 192.168.10.1
```

To change the destination address of Web traffic to 192.168.10.1, port 8080, use this command:

```
# iptables -t nat -A PREROUTING -p tcp --dport 80 -i eth0 -j DNAT \
--to-source 192.168.2.1:8080
```

Redirection

This is a specialized version of DNAT, a convenient equivalent of doing DNAT to the address of the incoming interface.

For example, to send the incoming port 80 Web traffic to our squid (transparent) proxy, we used the following command:

```
# iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 80 \
-j REDIRECT --to-port 3128
```

Special protocols

Some protocols do not work well with NAT, so for each of these protocols we need two extensions—one for connection tracking of the protocol, and one for the actual NAT.

Inside the netfilter distribution, there are currently modules for FTP:

1. ip_conntrack_ftp.o
2. ip_nat_ftp.o

If you plan to use NAT for FTP transfers, insert the following modules as shown:

```
# insmod ip_conntrack_ftp
# insmod ip_table_nat
# insmod ip_nat_ftp
```

Attention: If you are doing source NAT, you must ensure that the routing is set up correctly. That means if you change the source address so it is different from your external interface (for example, if you use unused IP addresses in your subnet), you need to tell your router to respond to ARP requests for that address as well. This can be done by creating an IP alias:

```
# ip address add IP_address dev eth0
```

11.2.12 Examples for using NAT in the enterprise and ISP/ASP

In this section, we show you how to use NAT in the enterprise and in ISP/ASP environments.

Changing the source address

In 11.2.3, "How to permanently enable IP forwarding" on page 251, we showed how to add a special route to the 10.0.0.0/255.255.0.0 subnet, because some of the packets coming from our server had the source address from this subnet.

However, instead of providing this special route, we can use the SNAT translation on the servers:

```
# iptables -t nat -A POSTROUTING -o iucv0 -j SNAT --to-source dummy0_IPaddr
```

This command will always change the source IP address of the packet to the IP_address of the DUMMY0 interface, which in our example is used as the server's external interface. Because the routers/firewalls are already aware of the routes to this external address, we do not need to provide an additional route to the 10.0.0.0/255.255.0.0 subnet.

Providing access to the Internet from an intranet

In this example, we show how to provide transparent access for an internal computer on the private subnet via the VM Linux router connected to the Internet with the public IP address. The sample configuration is shown in Figure 11-7.

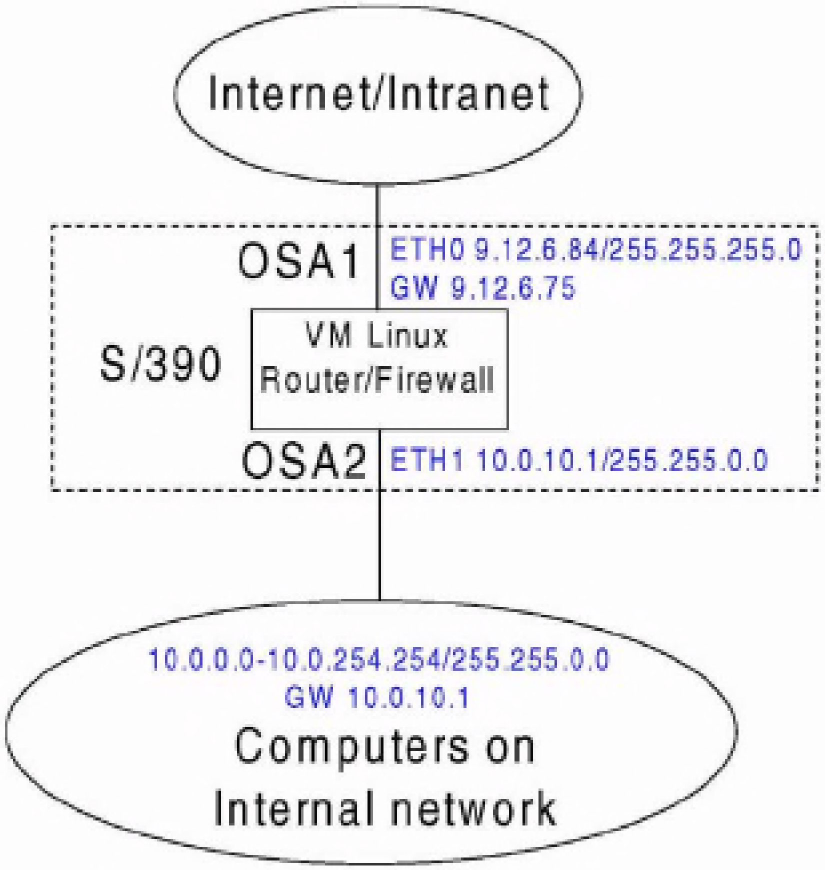


Figure 11-7 Internet access for local LAN

Without any settings, the packet traveling from the internal network via the VM Linux router/firewall will reach its Internet destination. But the source address of this package is from our *internal* subnet—and the server we are talking to does not know how to send the packet back.

To resolve this, we have to enable the source network address translation (SNAT) for each packet going out of the router. This can be done with the following command:

```
# iptables -t nat -A POSTROUTING -s 10.0.0.0/255.255.0.0 -o eth0 \
-j SNAT --to-source 9.12.6.84
```

With this rule, we are configuring the kernel to change the source address of each packet coming from the 10.0.0.0/255.255.0.0 subnet to the IP address of our external interface 9.12.6.84.

When the packet from the internal subnet reaches the server on the Internet, this server will respond to the router; and when the packet comes back to the router, it will send that packet back to the computer on the internal subnet, with the destination address of this computer.

Tip: If you plan to also provide FTP access to the Internet, insert the following two modules into the kernel:

```
# insmod ip_conntrack_ftp
# insmod ip_nat_ftp
```

Port forwarding

If you use the DeMilitarized Zone (DMZ) approach for your servers, this can be done using port forwarding. In the DMZ setup, you separate your Web/mail server from the server that is connected to the router/firewall over a private subnet.

This server's private subnet is separated from the local subnet used for computers accessing the Internet. The example of such a setup is shown in Figure 11-8 on page 268.

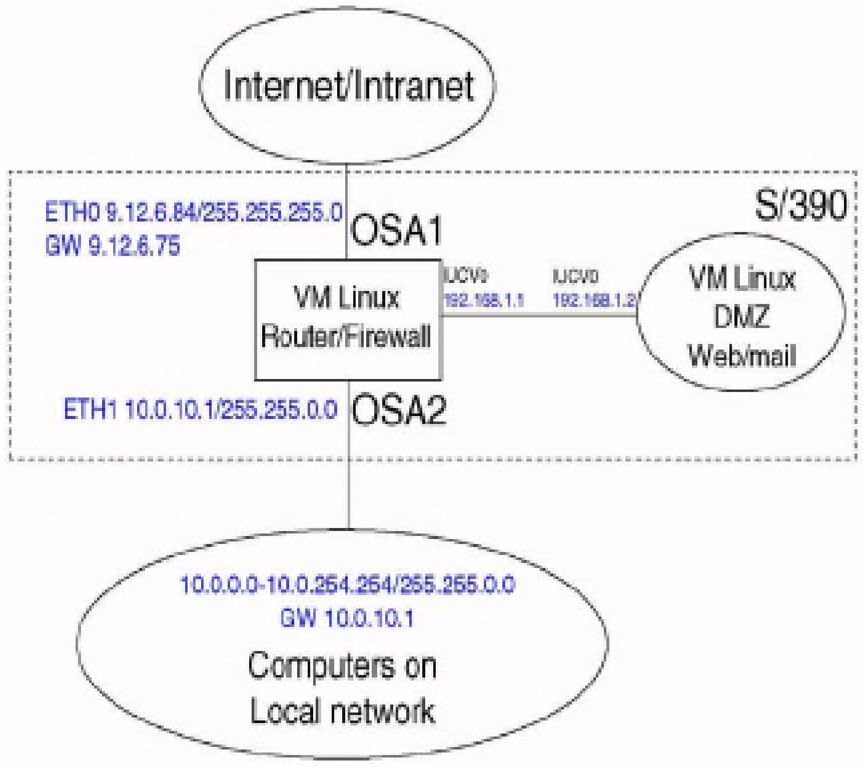


Figure 11-8 Port forwarding

In this scenario, the HTTP requests, for example, are coming to our router, which means that our www name is associated with the router external interface in the DNS. But because we do not have the Web server running on the router, we forward those packets to the Web server, which is in the DMZ. You can achieve this with the following command:

```
# iptables -t nat -A PREROUTING -p tcp -d 9.12.6.84 --dport 80 \
-j DNAT --to-destination 192.168.1.2:80
```

As you can see, we forward all TCP packets coming to address 9.12.6.84 port 80 to the address 192.168.1.2 port 80. You can also do port forwarding for other services.

11.2.13 Additional information

You can find more information on the official Linux IP Tables on the Linux Documentation Project home page:

```
http://www.linuxdoc.org
```

12

Backup using Amanda

In this chapter we discuss the Advanced Maryland Automatic Network Disk Archiver, known as Amanda.

Since Amanda has not previously been written about specifically for the Linux for zSeries and S/390 environment, we provide basic information about its operation and use in this environment. The Amanda package is included in the SuSE distribution.

For more information about general Amanda usage, refer to the Amanda Web page:

    http://www.amanda.org

Along with useful information, this site contains page links to Amanda archives and a number of mailing lists on Amanda (you can also join the mailing lists from here).

12.1 About Amanda

Amanda is an open source backup scheduler, originally developed at the University of Maryland for scheduling the backup of computing facilities there.

Amanda uses a client-server arrangement to facilitate the backup of network-attached servers. Using Amanda, it is possible to have a single tape-equipped server backing up an entire network of servers and desktops.

Note: Like many other Open Source Software projects, Amanda comes with no warranty, has no formal support, and is developed in people's spare time.

This consideration should be kept in mind when choosing a backup strategy.

12.1.1 How Amanda works

Backups are scheduled on one or more servers equipped with offline storage devices such as tape drives. At the scheduled time, the Amanda server contacts the client machine to be backed up, retrieving data over the network and writing it to tape. The data from the client can be stored in a staging area on disk, which improves the performance of the tape writing process (or provides a fallback, in case of tape problems).

Amanda can perform compression of the data being backed up, using standard Linux compression utilities (**gzip**, **bzip**). If network utilization is high, the compression can be done on the client to reduce the network load and potentially reduce backup times. This also lightens the load on the backup server, which may be processing many simultaneous backups.

Figure 12-1 shows Amanda's client-server architecture.

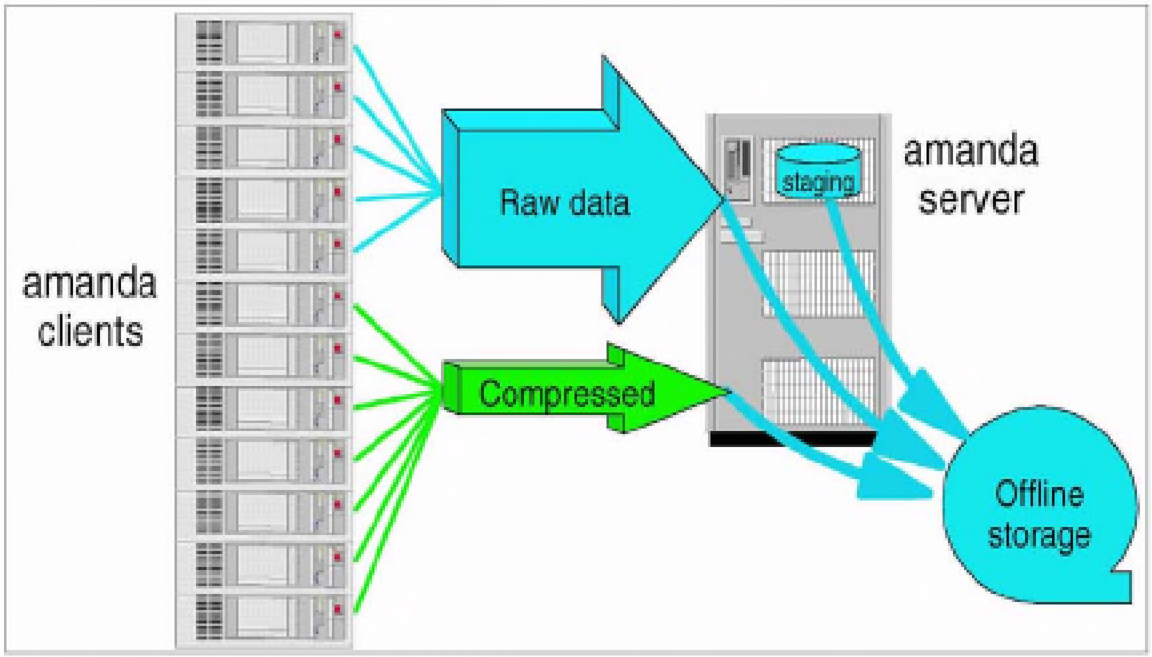


Figure 12-1 Amanda client-server architecture

Amanda uses its own authentication and access protocol, and it can also use Kerberos for authentication. It can be used to back up SMB/CIFS servers directly, which provides an alternative to running the **amandad** process on the clients[1].

Configuring an Amanda server involves five tasks:

1. Identifying the clients, and the devices on those clients, to be backed up
2. Establishing access controls to allow the server to read the clients' data
3. Configuring the disk staging areas on the server
4. Configuring the tapes and tape schedule
5. Setting up and activating the backup schedule

The backup schedule is usually invoked by entries in the /etc/crontab file to invoke the **amdump** program at the correct intervals. **amdump** initiates the connection to the client machine to read the data and output it to the staging disk. When this is done, **amdump** calls standard Linux tape management programs to write the data to tape. Amanda also provides support programs to recover from errors caused by tapes filling, incorrect tapes being loaded, and other possible failures.

Naturally, there is also an **amrestore** program to facilitate recovery of data from a backup.

[1] For Linux servers, this only makes sense if Samba was being set up on the server anyway, since it is more difficult to set up Samba than amandad.

12.2 Using Amanda in a penguin colony

Use of Amanda can be helpful in a penguin colony because of its low overhead and native Linux operation.

Note: Amanda is a backup scheduler, and does not actually perform the backups itself. The Amanda programs launch utilities like tar, dump and smbtar to perform the backup.

12.2.1 Planning for Amanda

The Amanda configuration process involves firstly determining the backups to be done, and creating configuration directories for these. The backup is then referred to by that directory name in all Amanda commands.

Important: Amanda is a file-level backup system. This means, for example, that it is not aware of the internal structure of databases (refer to "Complex application backup" on page 129 for more information on this).

At the time of writing, there is no plan to extend Amanda to provide awareness of file internals.

Amanda differs from other backup utilities in regard to the backup cycle. Other systems have a set process for the time that a full backup is done in relation to incremental backups (such as full backups on the weekend, and incremental backups overnight during the week).

Amanda does not work this way. It will switch between *level-0* (full) and *level-1* (incremental) backups during the cycle specified in the amanda.conf file, dependent upon the number of tapes available, the time since the last full backup, and so on. It will make sure that at least one level-0 backup is available at all times, and that a backup image is not overwritten if it would be required to form part of a backup.

Tip: The Amanda FAQ-O-Matic, available at the following site, contains information as to how a "traditional" backup cycle can be set up.

http://www.amanda.org/cgi-bin/fom?

12.2.2 Configuring Amanda

Amanda is configured using two major configuration files in the backup set configuration directory:

- amanda.conf
- disklist

amanda.conf

The file amanda.conf contains the following data for each backup:

- Specifications of tape drives to be used for this backup set
- Maximum network bandwidth to be used for this backup
- Definition of the backup cycle (number of tapes, duration, backup frequency)
- Other attributes of the backup set.

Our sample amanda.conf file is shown in Example 12-1.

Tip: The amanda.conf file provided with the Amanda product (in a backup set called "example") contains a great deal of information about configuring Amanda backups.

Example 12-1 Example `amanda.conf` *file*

```
#
# amanda.conf - sample Amanda configuration file

org "ITSOArchive"      # your organization name for reports
mailto "amanda"        # space separated list of operators at your site
dumpuser "amanda"      # the user to run dumps under
inparallel 2           # maximum dumpers that will run in parallel (max 63)
netusage  800 Kbps     # maximum net bandwidth for Amanda, in KB per sec
dumpcycle 4 weeks      # the number of days in the normal dump cycle
runspercycle 20        # the number of amdump runs in dumpcycle days
tapecycle 25 tapes     # the number of tapes in rotation
bumpsize 20 Mb         # minimum savings (threshold) to bump level 1 -> 2
bumpdays 1             # minimum days at each level
bumpmult 4             # threshold = bumpsize * bumpmult^(level-1)
etimeout 300           # number of seconds per filesystem for estimates.
dtimeout 1800          # number of idle seconds before a dump is aborted.
ctimeout 30            # max. number of seconds amcheck waits for client host
tapebufs 20            # tells taper how many 32k buffers to allocate.
runtapes 1             # number of tapes to be used in a single run of amdump
tapedev "/dev/ntibm0"  # the no-rewind tape device to be used
rawtapedev "/dev/ntibm0"  # the raw device to be used (ftape only)
tapetype IBM-3480-B40  # what kind of tape it is (see tapetypes below)
labelstr "^ITSO[0-9][0-9]*$" # label constraint regex: all tapes must match

# Specify holding disks.
holdingdisk hd1 {
   comment "main holding disk"
   directory "/dumps/amanda"# where the holding disk is
   use 290 Mb          # how much space can we use on it
   chunksize 1Gb       # size of chunk if you want big dump to be
                       # dumped on multiple files on holding disks
   }

infofile "/var/lib/amanda/ITSOArchive/curinfo"  # database DIRECTORY
logdir   "/var/lib/amanda/ITSOArchive"          # log directory
indexdir "/var/lib/amanda/ITSOArchive/index"    # index directory
# tapelist is stored, by default, in the directory that contains amanda.conf

define tapetype IBM-3490E-B40 {
   comment "IBM 3490E-B40"
   length 913152 kbytes         # these numbers generated by the
   filemark 32 kbytes           # Amanda tapetype program
   speed 2439 kps               # (not supplied with SuSE)
}

# dumptypes
define dumptype global {
   comment "Global definitions"
   index yes
}
define dumptype always-full {
   global
   comment "Full dump of this filesystem always"
   compress none
   priority high
   dumpcycle 0
}
define dumptype root-tar {
   global
   program "GNUTAR"
   comment "root partitions dumped with tar"
   compress none
   index
   exclude list "/usr/local/lib/amanda/exclude.gtar"
   priority low
}
define dumptype user-tar {
   root-tar
   comment "user partitions dumped with tar"
   priority medium
}
define dumptype high-tar {
   root-tar
   comment "partitions dumped with tar"
   priority high
}
define dumptype comp-root-tar {
   root-tar
   comment "Root partitions with compression"
   compress client fast
}
define dumptype comp-user-tar {
   user-tar
   compress client fast
}
define dumptype holding-disk {
   global
   comment "The master-host holding disk itself"
   holdingdisk no # do not use the holding disk
   priority medium
}
define dumptype comp-user {
   global
   comment "Non-root partitions on reasonably fast machines"
   compress client fast
   priority medium
}
define dumptype nocomp-user {
   comp-user
   comment "Non-root partitions on slow machines"
   compress none
}
define dumptype comp-root {
   global
   comment "Root partitions with compression"
   compress client fast
   priority low
}
define dumptype nocomp-root {
   comp-root
   comment "Root partitions without compression"
   compress none
}
define dumptype comp-high {
   global
   comment "very important partitions on fast machines"
   compress client best
   priority high
}
define dumptype nocomp-high {
   comp-high
   comment "very important partitions on slow machines"
   compress none
}
define dumptype nocomp-test {
   global
   comment "test dump without compression, no /etc/dumpdates recording"
   compress none
   record no
   priority medium
}
define dumptype comp-test {
   nocomp-test
   comment "test dump with compression, no /etc/dumpdates recording"
   compress client fast
}

# network interfaces
define interface local {
   comment "a local disk"
   use 1000 kbps
}
define interface eth1 {
   comment "100 Mbps ethernet"
   use 800 kbps
}
```

Note: Amanda does not provide a tape definition for the IBM mainframe tape devices supported by the tape390 driver. Therefore, we had to follow instructions contained in the example amanda.conf file in order to create a tapetype entry. To do this, however, we had to obtain the source package for Amanda, because the tapetype program is not supplied with the SuSE binaries of Amanda.

Once the tapetype program was available, we ran it against our 3490E-B40 tape drive to produce the tapetype entry shown in Example 12-1 on page 273.

disklist

The other configuration file you have to create is named disklist, which tells the **amdump** program which disks (or directories) on which hosts to back up.

```
#
# File format is:
#
#       hostname diskdev dumptype [spindle [interface]]
#
# ITSO machines.
#
vmlinux2        dasdb1          comp-root 1 local
vmlinux2        dasdc1          comp-user 2 local
vmlinux2        //tot12/vjc     smb-user  1 eth1
vmlinux7        dasda1          comp-root 1 eth1
vmlinux7        dasdb1          comp-user 2 eth1
```

As you can see, it is fairly easy to specify the servers and devices to be backed up. The dumptype (third field) must be given for each entry. This value chooses the type of dump from the amanda.conf file.

The spindle attribute refers to disk configurations where different physical file systems may exist as partitions of a single physical disk (which is not usually an issue for zSeries). The attribute can be used to increase performance by ensuring that Amanda does not simultaneously back up different file systems that share the same physical disk.

Tip: Unless you specifically want to have backups operating sequentially, specify each file system on a particular host with a different spindle value to make sure that you get the maximum simultaneous operation.

Amanda can also back up using SMB, allowing Windows machines to be included in your Amanda backup sets. An example of how this is configured appears in the third line of our example disklist.

From the perspective of the Amanda server, the SMB share to be backed up is part of the file system of an Amanda client. On that Amanda client, however, the SMB code in Amanda uses smbclient to access the SMB share on the Windows host. Files are retrieved from the source using SMB, then sent using the normal Amanda protocols from the client to the server.

Restriction: Using Amanda to back up Windows shares does not retain the Access Control List (ACL) information from the Windows file system. If you have complex ACLs in your Windows servers, Amanda is not the best backup solution. It is more suitable for lightweight backups of data directories on desktop computers, for example.

Also, the binaries of the Amanda package as distributed with SuSE do not have SMB support enabled in the amandad client. To test this function, we had to rebuild amandad from source.

Other files are created in the configuration directory, but are maintained by Amanda. These include tapelist, which is a list of the tapes that belong to a particular backup set and is updated by the `amlabel` program.

Tape changer

Amanda can utilize a tape changer if one is installed. It does this using a shell script identified in the tpchanger entry in amanda.conf. As long as your tape changer provides a program-level interface, Amanda can make use of it. Sample scripts for popular tape changers are supplied with Amanda.

Without the **tpchanger** parameter set, Amanda automatically switches off any multitape capability. So in order to use the automatic tape loader (ATL) on our 3490E-B40 (which automatically loads the next tape in the rack when the current tape is ejected), we had to either find a suitable script, or write our own.

We used the **chg-multi** script provided with Amanda to drive our ATL. The script provides enough basic function to support our autoloader, but also can be used as a template for writing your own scripts. The **chg-multi** script is generic, which saved us from having to write specific commands in the script to drive our ATL.

Amanda drives the tape changer during a backup process. For example, since it knows all of the labelled tapes in a backup set and keeps track of which tape can be used next, it can skip through the tapes in the rack until the required tape is loaded. It can also load another tape if the backup requires it.

Activating Amanda

The following line needs to be added (or un-commented) to your /etc/inetd.conf configuration file to enable the Amanda client.

```
amanda   dgram   udp     wait    amanda   /usr/lib/amanda/amandad amandad
```

In this example, /usr/lib/amanda/amandad is the path to the **amandad** executable. This line was already present on our SuSE installation (with the amanda package installed), and simply had to be un-commented.

Important: Any time the inetd configuration is changed, you must signal inetd to initialize. You can use the following command:

```
killall -HUP inetd
```

For the Amanda server, two more lines must be added to the inetd configuration to support the index service. Again, these lines were already in the /etc/inetd.conf file on our system and just had to be un-commented.

```
amandaidx   stream   tcp   nowait   root   /usr/lib/amanda/amindexd amindexd
amidxtape   stream   tcp   nowait   root   /usr/lib/amanda/amidxtaped amidxtaped
```

12.2.3 Backing up with Amanda

Prior to making any backups, you must plan and set up your amanda.conf and disklist files. This is because all operations in Amanda are performed with respect to the backup set being used. All of the **am** commands require the backup set name (the configuration directory name) as a parameter.

Tip: When creating your configuration, it is a good idea to have your holding area on a separate file system from the data being backed up. Otherwise, staging files will become part of your backup, and your incremental backups for that file system will be huge.

You can also experiment with using the **exclude** parameter in amanda.conf to exclude the holding area from being backed up.

Once you've created your configuration, you can then label your tapes using the **amlabel** command. This command creates the label that Amanda uses to identify the tape. Various information is kept here, including the name of the backup set the tape belongs to and the date the tape was used. The **amlabel** command also adds the tape to the tapelist file.

Note: You'll need to label all tapes in your backup set prior to using them for backups.

When we ran amlabel on our first tape, we received this output.

```
# amlabel normal ITSODaily00
labeling tape in slot 0 (/dev/ntibm0):
rewinding, reading label, not an amanda tape
rewinding, writing label ITSODaily00, checking label, done.
```

Having labelled your tapes, you can now test your configuration using the amcheck program. This program will identify any problems with your configuration by doing the steps that Amanda would normally take in preparation for a backup.

Tip: Many Amanda users run amcheck *prior to* the backup run in their regular backup process. This is because if amcheck detects an error, it is easier to fix the problem and schedule the backup run later—rather than repair a backup that fails during execution.

A sample run of amcheck is shown here.

```
# amcheck normal
Amanda Tape Server Host Check
-----------------------------
WARNING: holding disk /dumps/amanda: only 294236 KB free
  (296960 KB requested)
amcheck-server: slot 0: date X     label ITSODaily00 (first labelstr match)
NOTE: skipping tape-writable test
Tape ITSODaily00 label ok
NOTE: info dir /var/lib/amanda/ITSODaily/curinfo: does not exist
NOTE: it will be created on the next run
```

```
NOTE: index dir /var/lib/amanda/ITSODaily/index: does not exist
Server check took 2.702 seconds

Amanda Backup Client Hosts Check
--------------------------------
Client check: 2 hosts checked in 0.343 seconds, 0 problems found

(brought to you by Amanda 2.4.2)
```

In this example, amcheck is informing us that we are slightly short of holding disk space. It also did a tape check, and the results are shown ('date X' on an Amanda tape indicates a tape that has been labelled but not used).

The next two messages are indications that we have not done a backup before. Amanda can keep two sets of information about backups:

- curinfo
  This is information about the current status of the backup set, including which disklist entries are backed up to what level, and so on.
- index
  Optional (you must select it in your dumptype), the index keeps track of all files backed up, and is used by the amrecover program to ease the task of restoring data.

Amanda will create the relevant directories as required.

Finally, amcheck contacts the clients listed in the disklist to verify that they are contactable, and that authorization has been given to the backup server to obtain files from them.

The next step is to test a backup. Normally you would have the command issued from cron, but it is a good idea to run backups manually until you are comfortable with the process. The following command will start the amdump program, commencing a run of the "normal" backup set:

```
# amdump normal
```

Tip: The amdump program does not execute in the background, by default, so if you want to issue commands in your terminal window while amdump is running, you'll need to force amdump to the background. Invoke amdump as follows:

```
amdump normal &
```

While the backup is running, the **amstatus** command can give you information about the progress of the backup, as shown in Example 12-2:

Example 12-2 Output from `amstatus`

```
# amstatus normal
Using /var/lib/amanda/ITSODaily/amdump from Mon Jul 30 16:58:38 EDT 2001

vmlinux2://tot12/vjc             0   82299k dumping    31872k ( 38.73%) (16:59:15)
vmlinux2:dasdb1                  0 [dumps too big, but cannot incremental dump new disk]
vmlinux2:dasdc1                  0   14944k finished (16:59:21)
vmlinux7:dasda1                  0  181822k dumping    84064k ( 46.23%) (16:58:53)
vmlinux7:dasdb1                  0  293784k wait for dumping

SUMMARY          part     real estimated
                          size      size
partition       :   5
estimated       :   5           1976075k
failed          :   1           1397244k           ( 70.71%)
wait for dumping:   1            293784k           ( 14.87%)
dumping to tape :   0                 0k           (  0.00%)
dumping         :   2  115936k   264121k ( 43.90%) (  5.87%)
dumped          :   1   14944k    20926k ( 71.41%) (  0.76%)
wait for writing:   0       0k        0k (  0.00%) (  0.00%)
writing to tape :   0       0k        0k (  0.00%) (  0.00%)
failed to tape  :   0       0k        0k (  0.00%) (  0.00%)
taped           :   1   14944k    20926k ( 71.41%) (  0.76%)
all dumpers active
taper idle
network free kps:     2540
holding space   :    29932k ( 10.17%)
 dumper0 busy   :  0:00:28  (100.00%)
 dumper1 busy   :  0:00:28  (100.00%)
   taper busy   :  0:00:05  ( 19.82%)
 0 dumpers busy :  0:00:00  (  0.00%)
 1 dumper busy  :  0:00:00  (  0.00%)
 2 dumpers busy :  0:00:28  (100.00%)           not-idle:  0:00:20  ( 73.56%)
                                              no-dumpers:  0:00:07  ( 26.44%)
```

In this case, the backup of dasdb1 on vmlinux2 has failed because it is too large for the tape. However, the next time **amdump** was run, the dump of dasdb1 on vmlinux2 was added to the tape. So why did this occur?

Referring to FAQ lists for Amanda, when it has a large number of level 0 backups to do (as would happen for the first backup in a set), it is sometimes unable to plan a backup run that would pick them all up. The next time the backup is run, the file system is correctly backed up.

Restriction: Amanda cannot currently write an image to tape if it must span more than one tape. Each entry in the disklist file (i.e. each disk to be backed up) creates a single image file to be written to tape, and while images for separate disks can be written across tapes in a single run, a single image that is larger than a tape cannot be split across tapes.

If you have large partitions to be backed up, it will be necessary to configure them as separate entries in your disklist file until Amanda supports images spanning tapes.

Once the amdump program is complete, a mail message is sent to the operators given in the amanda.conf file; see Example 12-3:

Example 12-3 Backup completion report

```
Date: Mon, 30 Jul 2001 17:10:46 -0400
From: Amanda Admin <amanda@vmlinux2.itso.ibm.com>
To: amanda@vmlinux2.itso.ibm.com
Subject: ITSODaily AMANDA MAIL REPORT FOR July 30, 2001

These dumps were to tape ITSODaily00.
The next tape Amanda expects to use is: a new tape.

FAILURE AND STRANGE DUMP SUMMARY:
  vmlinux2   dasdb1 lev 0 FAILED [dumps too big, but cannot incremental dump new disk]

STATISTICS:
                          Total       Full      Daily
                        --------   --------   --------
Estimate Time (hrs:min)    0:00
Run Time (hrs:min)         0:12
Dump Time (hrs:min)        0:12       0:12       0:00
Output Size (meg)         509.9      509.9        0.0
Original Size (meg)      1221.6     1221.6        0.0
Avg Compressed Size (%)    41.7       41.7        --
Filesystems Dumped            4          4          0
Avg Dump Rate (k/s)       713.4      713.4        --

Tape Time (hrs:min)        0:03       0:03       0:00
Tape Size (meg)           510.0      510.0        0.0
Tape Used (%)              57.2       57.2        0.0
Filesystems Taped             4          4          0
Avg Tp Write Rate (k/s)  2674.2     2674.2        --

?
NOTES:
  planner: Adding new disk vmlinux2:dasdb1.
  planner: Adding new disk vmlinux2:dasdc1.
  planner: Adding new disk vmlinux2://tot12/vjc.
  planner: Adding new disk vmlinux7:dasda1.
  planner: Adding new disk vmlinux7:dasdb1.
  driver: WARNING: /dumps/amanda: 296960 KB requested, but only 294188 KB available.
  taper: tape ITSODaily00 kb 522240 fm 4 [OK]

?
DUMP SUMMARY:
                                     DUMPER STATS            TAPER STATS
HOSTNAME     DISK        L ORIG-KB OUT-KB COMP% MMM:SS  KB/s MMM:SS  KB/s
-------------------------- --------------------------------- -------------
vmlinux2     //tot12/vjc 0  164614 146048  88.7   4:49 505.8   0:53 2763.8
vmlinux2     dasdb1      0 FAILED ---------------------------------------
vmlinux2     dasdc1      0   44931  14944  33.3   0:22 668.1   0:06 2685.1
vmlinux7     dasda1      0  393946 144544  36.7   2:44 879.0   0:56 2602.8
vmlinux7     dasdb1      0  647442 216576  33.5   4:16 844.9   1:21 2664.0

(brought to you by Amanda version 2.4.2)
```

Driving the tape changer

The amtape program provides the interface to the changer script identified in the amanda.conf file. Using amtape, you can load the next tape in the rack, load a tape with a particular label, eject a tape, and other operations.

Tip: Refer to the amtape man page for more information, and keep in mind that if your ATL is gravity-fed, you'll only be able to move forward through the slots in the changer.

The following example shows some amtape commands in use.

```
# amtape normal show
amtape: scanning all 6 slots in tape-changer rack:
slot 0: date 20010730 label ITSODaily00
slot 1: date 20010731 label ITSODaily01
slot 2: date 20010731 label ITSODaily02
slot 3: date 20010801 label ITSODaily03
slot 4: date 20010801 label ITSODaily04
slot 5: date X        label ITSODaily05
# amtape normal reset
amtape: changer is reset, slot 0 is loaded.
# amtape normal current
amtape: scanning current slot in tape-changer rack:
slot 0: date 20010731 label ITSODaily02
```

In this example, an operator checks which tapes are currently loaded in the ATL. Amanda scans each tape and outputs the label information it finds. After this the slots are empty, so the operator reloads the tapes and resets Amanda's status of the changer. Then, the operator rechecks the tape in the current slot.

Important: With this type of tape changer, Amanda does not keep track of when tapes have been changed or moved. The `amtape reset` command is an administrative command that advises Amanda that the status of the ATL has changed and reset to start.

amadmin

The amadmin program provides commands that allow you to control the backup process. A number of options are available, including:

- Force a full backup of a disklist entry at the next run
- Mark a tape to be reusable or non-reusable
- Find which tapes the backups for a particular host or disk are on
- Display information about the backups for certain hosts or disks

Note: There are many commands available with amadmin. We suggest you refer to the amadmin man page to get more information.

In "Reporting with amadmin" on page 289, we discuss the use of some amadmin commands for use with reporting.

Scheduling your backup

Once your testing has gone smoothly, add an entry to /etc/crontab which will start the backup automatically at regular times. An example is shown here:

```
22 2  *  *  *  amanda    amdump normal
```

This will instruct `cron` to issue the command `amdump normal` under the user amanda every day at 2:22am.

Important: Remember to restart cron after making a change to crontab.

It is also possible to use the `amcheck` command in the schedule, to provide a pre-test for the backup run.

```
22 1  *  *  *  amanda    amcheck -m normal
22 2  *  *  *  amanda    amdump normal
```

These lines in the crontab will schedule an amcheck prior to the scheduled time of the backup (in this case, at 1:22am, with the backup scheduled for 2:22am). The `-m` switch on the `amcheck` command instructs it to run silently, but to send an e-mail to the backup operators if any problems occur. This allows a problem that would cause the backup to fail (wrong tape loaded, network problem) to be rectified before the start of the backup.

12.2.4 Restoring

The amrecover program is the front-end to the Amanda recovery process, and it is invoked from the system you wish to restore files onto. It works similar to FTP, making the backup set appear like an FTP server.

Note: The amrestore program actually performs the restoration. If you know which backup file on the tape contains your required data, you can invoke amrestore directly. To make the restoration process easier, amrecover uses the backup indexes to feed the correct information to amrestore for you.

Amanda restores files relative to the root of the point the backup was taken from. For example, on our test system vmlinux2, /dev/dasdb1 is the root file system, and /dev/dasdc1 is mounted at /home. To restore files directly into the directory /home/tot12/testing, we would change to the /home directory and start amrecover from there. The amrestore program will expand the files into the correct directory.

You can choose to restore files into a different directory, so that you can migrate changes from a backup. For example, if you select /home/tot12/backup as the location to restore to, when you restore /home/tot12/testing, you'll find the restored files in the directory /home/tot12/backup/tot12/testing.

Example: An example of a single file recovery session is shown in "Single file or directory restoration with Amanda" on page 292.

12.2.5 Reporting

Amanda keeps extensive logs of the backup process, and comes with utilities to read the logs and report on attributes of the backup process.

amoverview

The amoverview program produces a summary of the history of the backup set.

```
# amoverview normal
```

```
         date              07 07 08 08
host     disk              30 31 01 02

vmlinux1 dasdb1                  0  1
vmlinux1 dasdc1                  0  1
vmlinux2 //tot12/vjc        0 11 10  1
vmlinux2 dasdb1             E  0 11  1
vmlinux2 dasdc1             0 11 01  1
vmlinux3 dasdb1                  E  0
vmlinux3 dasdc1                  0  1
vmlinux7 dasda1             0 11 11  1
vmlinux7 dasdb1             0 11 11  1
```

It doesn't look like much data, but there is a great deal of information in this output. The amoverview program prints the details of each disk in the backup set, when a backup was performed or attempted, and the backup level taken at that time. Let's look at the line for vmlinux2:dasdc1:

- A level 0 backup was taken on July 30.
- Two level 1 backups were taken on July 31.
- A level 0 backup, and then a level 1 backup, were taken on August 1. The level 0 taken on this day have made the previous backups redundant. These prior backups would not be required for a disaster recovery restoration, but may still be used if files from prior to August 1 were required.
- A level 1 backup was done on August 2.

An `E` indicates that a backup was attempted, but an error occurred. Both vmlinux2:dasdb1 and vmlinux3:dasdb1 experienced errors on their first attempt. A possible reason is that the size of the backup was too large to be taken with the other backups being done at the time. For both of these disks, you can see that the level 0 backup was done on the next run.

The **`amoverview`** tool gives you an easy way to check that your file systems are being backed up in a timely manner.

amplot

The amplot program analyses amdump files (produced during every amdump run) and produces a graphical analysis of the dump.

Note: amplot uses the gnuplot program, which in turn requires X. The amplot output displays in an X window. We had to download and compile gnuplot in order to use amplot. The gnuplot source can be obtained at:

```
http://www.gnuplot.org
```

The default configuration for gnuplot installs into /usr/local/bin, but amplot as packaged by SuSE expects gnuplot to be found in /usr/bin. You will need to take this into account if you build gnuplot for use with amplot.

If you do not have an X display, or if you prefer printed output, amplot can generate output as a Postscript file. Refer to the amplot man page for more information.

The graphs produced by amplot show statistics such as job queue length, network bandwidth utilization, holding disk utilization, tape idle time, and number of dumper tasks in use. A sample graph from amplot is shown in Figure 12-2 on page 288.

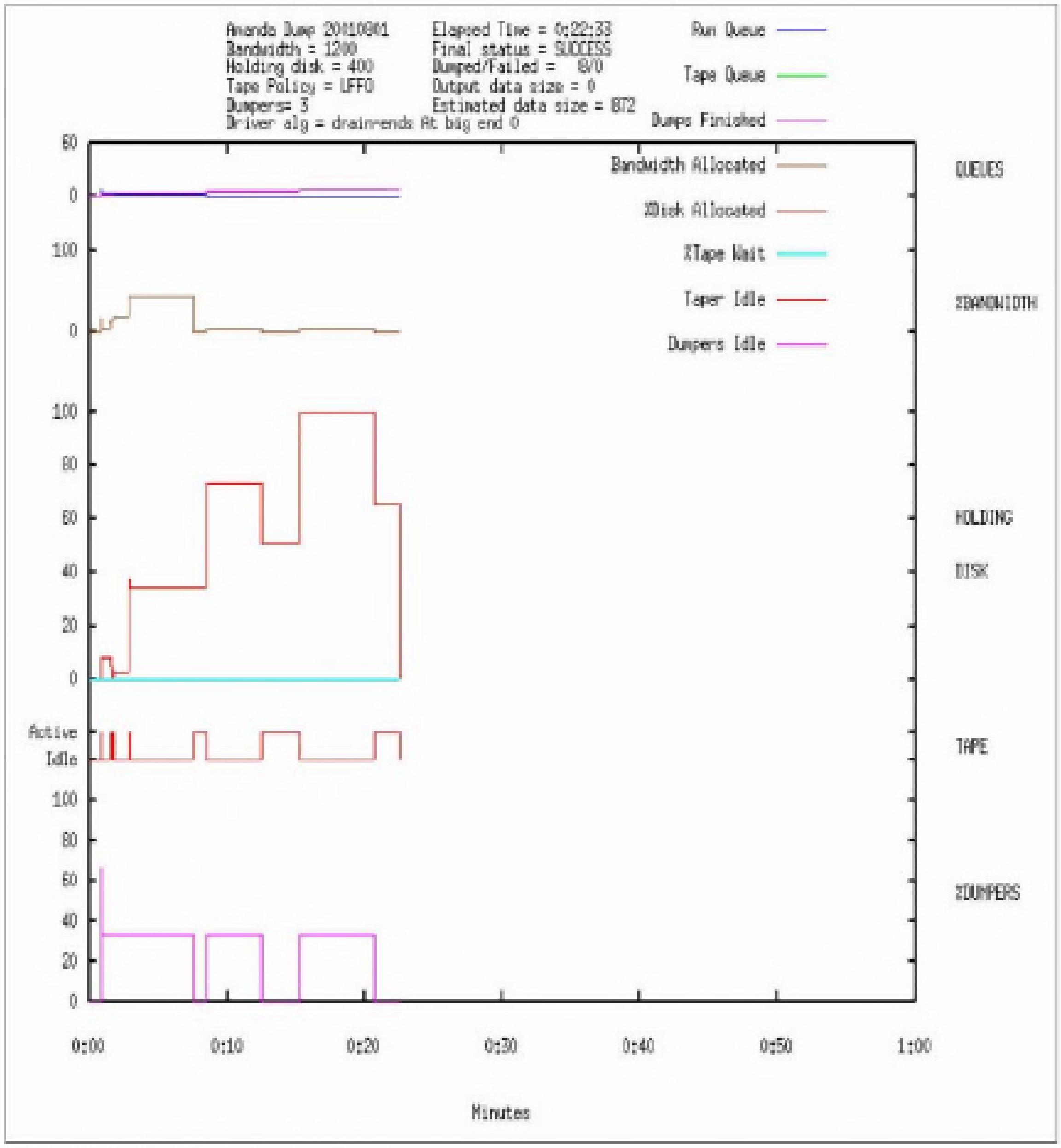


Figure 12-2 Sample `amplot` *graph*

Network bandwidth utilization is shown as a percentage of the bandwidth allowed in the amanda.conf file. On this graph, we see that network bandwidth does not appear to be a bottleneck. The graph also shows that the holding disk is full during the last part of the backup.

The graphs can be used to point out ways to improve performance. In our example, even though we only have three dumpers allocated, they are never all active at once. In fact, except for an instant after the estimates are performed, only one dumper is active at a time for the duration of the backup. This would seem to indicate that Amanda is being prevented from dumping more than one image at a time, which might be a combination of insufficient holding disk space and large backup images.

The e-mail reports sent to the Amanda users at the end of a backup run also contain useful information about the backup process.

Tip: The report sent at the end of an Amanda backup run is generated by the amreport program. You can run amreport at any time to get the summary of a backup run. Refer to the amreport man page for instructions.

Reporting with amadmin

Apart from the operational aspects of amadmin that we cover in earlier sections, there are reporting sub-commands that are very useful.

One of these sub-commands, **info**, summarizes the data compression and speed statistics of the last three backups for a disk,or for all disks on a host, or for all disks in the backup set. It also shows on which tapes the most recent backup data can be found.

```
# amadmin normal info vmlinux2 dasdc1

Current info for vmlinux2 dasdc1:
  Stats: dump rates (kps), Full:  711.0, 679.0,  -1.0
                    Incremental:   16.0,  32.0,  16.0
          compressed size, Full:  33.3%, 33.3%,-100.0%
                    Incremental:  43.8%, 43.8%, 43.8%
  Dumps: lev datestmp  tape             file   origK   compK secs
          0  20010801  ITSODaily03         1   44931   14944   21
          1  20010802  ITSODaily06         4      73      32    2
```

This example shows information about our vmlinux2:dasdc1. The dump rates are shown, as well as the size of the data written to tape, for the last three incremental and full backups written (in our case, since there have been only two full backups, the last column of data for full backup is meaningless).

The **balance** sub-command gives an insight into the way that Amanda schedules full backups within a backup cycle. The purpose of the command is to view how balanced the tape runs have been during the backup cycle, but through interpreting the display, you can gain an understanding of part of Amanda's internal scheduling.

The following shows the output from **amadmin balance**, run early in the backup cycle.

```
# amadmin normal balance

 due-date  #fs   orig KB    out KB  balance
-------------------------------------------
 8/02 Thu    0         0         0     ---
 8/03 Fri    0         0         0     ---
 8/04 Sat    0         0         0     ---
 8/05 Sun    0         0         0     ---
 8/06 Mon    2   1041388    361120    -0.3%
 8/07 Tue    1   1297449    425824   +17.6%
 8/08 Wed    6   2922844   1023456  +182.7%
-------------------------------------------
TOTAL        9   5261681   1810400   362080  (estimated 5 runs per
dumpcycle)
```

This display tells us where full backups are currently due in the schedule. Amanda estimates the amount of data that will be backed up on these dates, based on previous backups, and uses these values to calculate the balance of the cycle.

According to the current plan, 6 out of the 9 file systems in the backup set are due on August 8. This means that the size of the backup on that date will be almost three times the average size of those backups. This creates a huge imbalance in the duration of the backup.

To minimize this, Amanda will promote some of these full backups to earlier in the cycle, in order to balance the workload more evenly throughout the backup cycle.

Important: Amanda will *never* postpone a full backup to balance the cycle.

When Amanda promotes a full backup, you will see messages like this in your backup report:

```
NOTES:
  planner: Full dump of vmlinux3:dasdc1 promoted from 2 days ahead.
  planner: Full dump of vmlinux7:dasda1 promoted from 4 days ahead.
```

In this case, Amanda decided to bring the full backups for these two file systems forward, in order to achieve a balanced backup cycle. Over the course of your backup cycle you may see these messages, especially if your file systems change in size over time.

Note: Refer to the amadmin man page for further information about the other sub-commands available.

12.2.6 Disaster recovery using Amanda

Amanda can be deployed in a disaster recovery role. With the aid of a small Linux system that loads using an initial root device (also known as a disaster recovery "bootstrap" system), Amanda can recover full systems up to the last incremental backup.

The system you use as a disaster recovery bootstrap would be like the installation starter system you first used to install Linux (unfortunately, you cannot use one of the installation systems because amandad is not present on these systems).

In 10.7, "Linux IPL from NSS" on page 228 we describe a way to build a VM NSS which can be used to IPL Linux images, and this is a useful way to implement a DR bootstrap. We describe other ways to build starter systems elsewhere in Chapter 10.

Important: Remember to install the amanda package as part of your disaster recovery bootstrap system, and to add the `amandad` line to inetd.conf.

The process would work as follows:

1. IPL the disaster recovery bootstrap image in your Linux guest.
2. Load the network driver and establish network connectivity.
3. Load the DASD driver, correctly mapping the DASDs as configured in the Linux instance to be restored (a standard disk configuration would help here).
4. Reformat the device which will contain your root file system.
5. Run **`amrecover`** to restore the root file system.
6. Execute step 4 for any "first level" file systems you have on separate devices (e.g. /usr, /home), and mount these empty file systems at their correct mount points.

Important: If you use LVM, this step will include a restoration of your LVM configuration using vgcfgrestore (assuming you backed-up your configuration using vgcfgbackup, and that the backup resides on a non-LVM file system!). Otherwise, manually recreate your LVM configuration.

7. Run **`amrecover`** on the "first level" file systems.

8. Repeat steps 6 and 7 for any remaining file systems you have, stepping through the organization of your physical devices as required.

While Amanda can be used in this way to provide disaster recovery capability, it is not the most efficient method of providing full volume backup for Linux instances under VM. A better way would be to have VM perform backups of the minidisks all at once, and use Amanda to provide incremental backups only. Restoration would then involve a VM-level full volume restoration of the Linux system's minidisks, followed by incremental restoration of changed files using Amanda.

12.3 Backup and recovery scenarios

This section illustrates scenarios using the concepts discussed in this chapter.

12.3.1 Single file or directory restoration with Amanda

In Example 12-4, we show a file recovery session using amrecover. The file mrtg_total.pl has been deleted from the root user's home directory, and we want to restore that file from our Amanda backup.

Example 12-4 A file restore session using amrecover

```
vmlinux7:/ # amrecover normal -s vmlinux2    1
AMRECOVER Version 2.4.2. Contacting server on vmlinux2 ...
220 vmlinux2 AMANDA index server (2.4.2) ready.
200 Access OK      2
Setting restore date to today (2001-08-01)
200 Working date set to 2001-08-01.
200 Config set to normal.
200 Dump host set to vmlinux7.
$CWD '/' is on disk 'dasda1' mounted at '/'.
200 Disk set to dasda1.
/
amrecover> history     3
200- Dump history for config "normal" host "vmlinux7" disk "dasda1"
201- 2001-07-31 1 ITSODaily01 4
201- 2001-07-31 1 ITSODaily02 3
201- 2001-07-30 0 ITSODaily00 2
200 Dump history for config "normal" host "vmlinux7" disk "dasda1"
amrecover> setdate --07-31      4
200 Working date set to 2001-07-31.
amrecover> settape vmlinux2:default       5
Using default tape from server vmlinux2.
amrecover> cd root     6
/root
```

```
amrecover> ls        7
2001-07-30 .
2001-07-30 .bash_history
2001-07-30 .exrc
2001-07-30 .gnupg/
2001-07-30 .gtkrc-kde
2001-07-30 .kde/
2001-07-30 .kde2/
2001-07-30 .kxmlrpcd
2001-07-30 .mcoprc
2001-07-30 .xinitrc
2001-07-30 KDesktop/
2001-07-30 bin/
2001-07-30 dead.letter
2001-07-30 gd/
2001-07-30 lcs-2.4.5-s390-2.tar.gz
2001-07-30 linux-2.2.19.tar.gz
2001-07-30 linux-2.4.5.tar.gz
2001-07-30 linux/
2001-07-30 mrtg_total.pl
2001-07-30 netsaint/
2001-07-30 sieve
2001-07-30 sieve.c
amrecover> add mrtg_total.pl     8
Added /root/mrtg_total.pl
amrecover> list        9
TAPE ITSODaily00 LEVEL 0 DATE 2001-07-30
        /root/mrtg_total.pl
amrecover> extract        10

Extracting files using tape drive /dev/ntibm0 on host vmlinux2.
The following tapes are needed: ITSODaily00

Restoring files into directory /
Continue? [Y/n]: y

Load tape ITSODaily00 now
Continue? [Y/n]: y
restore: ./root: File exists
set owner/mode for '.'? [yn] n
amrecover> quit        11
200 Good bye.
vmlinux7:/ # cd root
vmlinux7:~ # ls -l
total 45252
drwxr-xr-x   11 root     root         4096 Aug  1 03:31 .
drwxr-xr-x   18 root     root         4096 Jul 17 10:09 ..
-rw-------    1 root     root         8405 Jul 28 10:06 .bash_history
-rw-r--r--    1 root     root         1124 Feb 29  2000 .exrc
```

```
drwx--x--x    2 root     root         4096 Jul 17 10:07 .gnupg
-rw-r--r--    1 root     root         1105 Jul 18 04:47 .gtkrc-kde
drwx------    2 root     root         4096 Jul 18 03:27 .kde
drwx------    6 root     root         4096 Jul 18 03:58 .kde2
-r--------    1 root     root           21 Jul 18 04:47 .kxmlrpcd
-rw-------    1 root     root           31 Jul 18 04:47 .mcoprc
-rwxr-xr-x    1 root     root         2186 Apr 11 21:50 .xinitrc
drwx------    3 root     root         4096 Jul 18 08:21 KDesktop
drwxr-xr-x    2 root     root         4096 Jul 17 10:07 bin
-rw-------    1 root     netsaint   208645 Jul 27 02:42 dead.letter
drwxr-x---    6 root     root         4096 Jul 26 08:49 gd
-rw-r--r--    1 root     root        18690 Jul 21 10:17 lcs-2.4.5-s390-2.tar.gz
drwxr-xr-x   14 1046     netsaint     4096 May 26 11:12 linux
-rw-r--r--    1 root     root     19343412 Jul 20 00:40 linux-2.2.19.tar.gz
-rw-r-----    1 root     root     26534489 Jul 27 08:07 linux-2.4.5.tar.gz
-rwxr-xr-x    1 root     root        27675 Jul 26 08:44 mrtg_total.pl
drwxr-xr-x    4 root     root         4096 Jul 20 08:14 netsaint
-rwxr-xr-x    1 root     root        16481 Jul 18 04:12 sieve
-rw-r-----    1 root     root         1293 Jul 18 04:12 sieve.c
```

1. The amrecover program is invoked, specifying the name of the backup set (`normal`) and the Amanda server to be used (`-s vmlinux2`).
2. amrestore reports that it successfully contacted the index server on vmlinux2. It sets defaults for the amrecover session based on current directory, today's date, etc.
3. We request a backup history of the disk, to check that we can get the file we are looking for at the date we need.
4. We want the file as at 31 July, and the backup covers this. The **`setdate`** command is used to set the point-of-reference for the restore.
5. The **`settape`** command specifies where the backup tapes (and the tape drive) are located.
6. We can now look through the backup set to locate the file to be restored. First, we change to the directory the file was located.
7. After changing directory, we issue the **`ls`** command to list the files and directories in the backup. Notice that the file we want to restore, mrtg_total.pl, does appear in the list. The date beside the file tells us the most recent version of this file available. Since a level 0 backup was done on July 30, and incremental backups on July 31, it appears that the file did not change between the full backup and the incrementals.
8. Having located the file, we add it to our extraction list using the **`add`** command.
9. Using the **`list`** command, we can check the details of the recovery we are about to do. amrestore tells us which tape it will be using, and the path to the file being restored.

10. The `extract` command commences the restoration. amrecover prompts us for information to complete the restore, including when to load the tape. Since we are recovering into our existing directory (`/root`), the attempt to create the directory fails (`restore: ./root: File exists`), and this is normal. Again, since the directory already exists, we do not need to change permissions.

11. The file recovery is complete, and we can exit amrecover and check that the file is correct.

13

System monitoring

In this chapter, we review methods by which an enterprise running Linux guest machines under VM can record the computing resources consumed by those guest systems. This information could then form the basis of a charge back system.

We also discuss how to monitor the availability of Linux guest machines. This includes system availability, response times and the availability of services such as DNS, mail servers and Web servers.

13.1 Why measure resource consumption

In the context of this redbook, there are essentially two reasons to measure resources consumed in a computing environment. Firstly, a service provider (whether an ASP, ISP or traditional enterprise) will often want to bill its users for their use of computing resources such as CPU time, disk space and network I/O bandwidth.

Secondly, there is a need to ensure that Service Level Agreements are being adequately met.

13.2 Charge back method

The charge back or billing methodology you choose will depend on the type of services you're providing to your customers. For example, the billing requirements of an ASP or ISP will probably be quite different from those used by an enterprise using Linux on VM as a server consolidation platform.

13.2.1 Service Provider accounting and billing

For an ASP or ISP, services and rates are the basis for a charge back system. Each service provided by the service provider has a rate (or fee) that falls into one of two categories: sign-up fees or usage fees.

The *sign-up fee* is a one-time flat fee charged to set up the user account for the service. The *usage fee* is a predetermined, recurring charge that occurs during each billing cycle. The usage criteria may be based on several models, ranging from a simple scheme where a flat fee is charged for the use of the service, to sophisticated schemes where the exact usage of each resource (CPU, memory, disk, network bandwidth etc.) is metered and billed to the user. Promotions and discounts are frequently offered to encourage new users to sign up and current users to use more services.

At the end of the billing cycle, the billing software computes the total charge for each user and mails an invoice or debits a credit card account, depending on the user's payment model.

There are a number of open source ISP billing and account administration packages available for Linux. One example is Freeside, which is available at:

http://www.sisd.com/freeside

The service provider must have an accurate way of billing the customer for such things as application usage and system resource usage. *Application usage* is easily tracked by methods as simple as using timestamp checkpoints embedded in the application programs. With this method, the customer signs on to the application, and the time is recorded. When the customer signs off, the time is once again recorded. To generate a bill, the start and end times are used to calculate the charge for that particular user's session.

Billing for *system resource usage* is more complex, as it requires a greater level of measurement and recording. The first part of this chapter focuses on system resource measurement.

13.2.2 Enterprise accounting and billing

Enterprises are using Linux under VM as a server consolidation platform. For example, you can consolidate many disparate file and print servers or infrastructure servers (such as DNS, firewall, e-mail) onto a single S/390 or zSeries machine.

These functions may be purely internal within an organization and as such, an ASP or ISP billing model would probably not apply. However, it's often necessary to charge back individual departments within an organization for their use of computing resources. This requirement has existed since the earliest days of computing, when precious computing resource had to be shared among many groups.

13.3 What can we measure

There are many measurement metrics available. However, not all of these are necessarily useful for charge back purposes. For that reason, we'll focus on CPU consumption, DASD utilization and network bandwidth usage. Given that in the context of this redbook we ran multiple Linux guest systems under VM, we'll use a combination of VM and Linux tools to derive the resource measurements.

13.4 CPU time accounting

In the following sections, we detail the various aspects of CPU time accounting, including how to set up virtual machines for accounting purposes, and how to set up Linux process accounting.

13.4.1 VM accounting

The VM operating system has the capability to generate accounting records that can be used for charge back.

The VM Control Program (CP) creates and records accounting records when particular system events occur. Once accounting is running, CP creates an accounting record whenever one of the following events occurs:

- A virtual machine logs off, or detaches a virtual processor.
- A user detaches a dedicated device.
- A user releases temporary disk space.
- A virtual machine issues a DIAGNOSE code X'4C'
- A SNA/CCS terminal session ends.
- The system checkpoints (during shutdown, for example).
- You enter the `ACNT` command.

When one of these events occurs, CP creates and stores an accounting record that describes the event. Then CP notifies the accounting virtual machine of the new record. Accounting records remain in storage until the accounting virtual machine retrieves them. The default limit for accounting is 20 records. If the number of records in storage reaches that number, CP notifies the primary system operator. The buildup of records in storage indicates that retrieval is not active. You can change the limit with the RECORDING command.

13.4.2 Setting up virtual machines for accounting

Note: If VM accounting has not been enabled at your installation, this section will show you how to set up this process. For more detailed information on setting up a virtual machine for accounting, refer to the latest version of the VM Planning and Administration publication.

The VM installation media supplies a sample directory entry for an accounting virtual machine. This entry contains the required IUCV authorization for connecting to the CP accounting system service. Also supplied is a sample system configuration file that defines the user ID for the accounting virtual machine as DISKACNT.

The user ID for the accounting virtual machine is defined as part of the SYSTEM_USERIDS statement in the system configuration file so that it is automatically logged on by CP at IPL. A sample PROFILE EXEC for the accounting virtual machine is also supplied.

To set up a virtual machine to begin recording accounting information automatically, you must have the proper PROFILE EXEC and user directory set up. The following steps show this procedure:

1. Log on as MAINT.
2. Determine the write password of the accounting virtual machine's 191 disk.

Note: The accounting virtual machine has been specified in either the SYSTEM_USERIDS ACCOUNT1 or ACCOUNT2 system configuration file statement, or the SYSACNT macroinstruction. Before linking to the accounting virtual machine's 191 disk, find out its write password by examining its user directory entry.

If the accounting virtual machine's 191 disk does not have a write password, you must supply one and update the directory.

   Verify that the directory entry for this virtual machine contains the required IUCV authorization for connecting to the CP accounting system service (for example, IUCV *ACCOUNT).

3. Link to the accounting virtual machine's 191 disk by entering:

   ```
   link to diskacnt 191 as 391 wr
   ```

   When CP responds with `ENTER WRITE PASSWORD:` enter, for example, the following:

   ```
   wpass
   ```

   where *wpass* is the write password of the accounting virtual machine. The accounting virtual machine's 191 disk is now your 391 disk.

4. Access the 391 disk by entering:

   ```
   access 391 x
   ```

   If you receive a message that says `X'391'DEVICE ERROR`, you must format the 391 disk by entering:

   ```
   format 391 x
   ```

   CMS responds as follows:

   ```
   FORMAT WILL ERASE ALL FILES ON DISK X (391).DO YOU WISH TO CONTINUE?
   (YES|NO).
   ```

   Answer `yes` and when CP responds with `ENTER DISK LABEL`, enter:

   ```
   acnt
   ```

   You can use any 1- to 6-character label name.

5. Copy the file named DVM PROFILE from MAINT's 193 disk (we have accessed the 193 disk as k) to the 391 disk by entering:

   ```
   copyfile dvm profile k profile exec x
   ```

Note: The DVM PROFILE is the PROFILE EXEC for the accounting, symptom record recording, and error recording virtual machines. The RETRIEVE utility, which does the IUCV connect to the *ACCOUNT system service, is invoked from this PROFILE EXEC.

6. Release and detach the 391 disk by entering:

   ```
   release x (det
   ```

7. If the accounting virtual machine is not logged on, use the XAUTOLOG command to log on the accounting virtual machine automatically. To do this for the DISKACNT user ID, enter:

   ```
   xautolog diskacnt
   ```

8. You can use the CP command QUERY RECORDING to ensure that accounting is active, for example, by entering the following:

   ```
   query recording
   ```

   Example 13-1 shows an example of the output.

Example 13-1 Query recording output

```
RECORDING    COUNT     LMT USERID   COMMUNICATION
EREP     ON  00000000  002 EREP     ACTIVE
ACCOUNT  ON  00001155  020 DISKACNT ACTIVE
SYMPTOM  ON  00000000  002 OPERSYMP ACTIVE
```

13.4.3 Virtual machine resource usage - record type 01

There are a number of VM accounting records available, but for Linux guest CPU consumption data, we're primarily interested in record type 01. This record is produced whenever a user logs off or whenever the ACNT command is entered. Among other things the record contains information on the following:

- User ID (Linux guest name)
- Number of seconds connected to CP
- Milliseconds of processor time used, including time for supervisor functions
- Milliseconds of virtual CPU time used
- Number of page reads
- Number of page writes
- Number of requested virtual I/O starts for non-spooled I/O

13.4.4 Processing accounting records

Over time, the accounting virtual machine's A disk fills with accounting records. CP sends a message to the primary system operator when the A disk is 75% full, when it is 90% full, and when it is completely full. You can also log on the accounting virtual machine and check the disk yourself. When the disk is full, you must process some of the old records and erase some files to make room for new ones.

The CMS Utility *ACCOUNT* can be used to process accounting records. Since z/VM 4.1, the CMS utilities have been bundled into the base z/VM installation and are no longer a separate, chargeable product.

Note: Refer to *CMS Command and Utility Reference* for complete information about the ACCOUNT utility.

Example 13-2 from the ACCOUNT command illustrates that we can use VM accounting to gather data on CPU consumption for all Linux guests running under VM.

Example 13-2 Output from the VM ACCOUNT command

```
VM SYSTEM USAGE OVER THE PERIOD         07/12/01 TO 07/12/01     ALL SHIFTS
USERID    SESS    CONNECT RATIO   REAL-CPU    VIRT-CPU   PG READ  PG WRITE       SIO
TUXOMSTR     1 000004:35 00974 0000:00:17 0000:00:10      6082     10874     33682
VMLINUXA     1 000844:50 00001 0596:31:23 0234:57:01     38644     95891    107830
VMLINUXB     1 000702:14 00663 0001:03:31 0000:43:40    237642    235045     49061
VMLINUXC     1 000702:14 00582 0001:12:23 0000:48:30    230075    256174     62389
VMLINUX2     1 000697:30 00309 0002:15:07 0001:54:22    621859    634902   2553131
VMLINUX3     1 000031:43 00638 0000:02:59 0000:01:50     10490     16233      9429
VMLINUX4     1 000702:16 00453 0001:32:53 0001:08:39     60941     80382    240205
VMLINUX5     1 000001:33 ***** 0000:00:00 0000:00:00         0         0       397
VMLINUX6     1 000863:18 00562 0001:32:07 0001:12:14    142996    230437   4847862
VMLINUX7     4 000078:35 00605 0000:07:47 0000:05:19        17      3395     70930
VMLINUX8     1 000080:20 ***** 0000:00:00 0000:00:00         0       876        26
VMLINUX9     1 000573:23 00548 0001:02:44 0000:45:39     39402    115060    704388
 TOTALS     41 020860:28 00033 0615:45:31 0242:59:18   1915393   2265598  10406917
```

13.4.5 Linux process accounting

Process accounting is the method of recording and summarizing processes executed on an individual Linux guest machine. Process accounting collects metrics such as the elapsed CPU time, average memory use, I/O information, and the name of the user who ran the process. The kernel will log process accounting information after a process terminates.

Note: Depending on the model of implementing Linux servers under VM, it might be sufficient to use VM accounting to record CPU consumption at a guest level rather than recording at a process level with individual Linux guests.

If you do require Linux process accounting, you should first install the acct rpm package. If you are running SuSE, this package resides in the ap1 package directory, with the filename acct.rpm.

After installing the rpm, you can edit /etc/rc.config to enable accounting at Linux boot time with the following command:

```
# rpm -ivh acct.rpm
```

The parameter to edit in /etc/rc.config is named `START_ACCT`. Make sure that it is set as follows:

```
START_ACCT=yes
```

Important: Remember to run the `SuSEconfig` command after you have edited /etc/rc.config.

Once the Linux system has been rebooted, process accounting will be started automatically.

The `lastcomm` command can be used to show the last commands that have been executed in a Linux system. The information displayed includes the command name, who ran the command, and the amount of CPU time consumed.

The `sa` command is a tool for summarizing and reporting on logged process accounting data stored in the acct file. Refer to the `sa` man page for a complete description of the syntax available.

13.5 Disk space utilization

In the context of this redbook, we are running many Linux guest systems under the VM operating system. As such, each Linux guest will have a number of minidisks. A simple approach to billing customers for the amount of disk space they consume would be to use existing VM utilities to report on DASD space utilization.

If the VM installation uses the User Directory (i.e., not DIRMAINT), then we can use the CP utility DISKMAP to provide us with space utilization information for Linux guests.

Example 13-3 Output from DISKMAP

```
VOLUME   USERID     CUU   DEVTYPE   START      END      SIZE
                                        0        0         1    GAP
LIS32A   LINMNT2    208     3390    00001    03338    03338
```

From this example we can see that the Linux guest system LINMNT2 has 3338 cylinders of DASD allocated to it.

If the VM installation is using the Directory Maintenance (DIRMAINT) utility, then the systems programmer can issue the command:

```
dirm dirmap
```

This command will generate a report detailing the current DASD utilization on the VM system. Example 13-4 illustrates the output that is generated:

Example 13-4 Output from DIRM DIRMAP

```
USER     DIRECT     Map of Minidisks    14:38:05   20010713
____     ______     ___ __ _________    ________   ________
Volser   Type Ownerid  Addr SysAffin   Start      End        Length   Flags
______   ____ _______  ____ ________   _____      ___        ______   _____
-----------------------------------------------------------------------------
LIUSR1   3390                              0          0          1 Gap
              LI2000   0191 *              1        100        100
              LI2000   0200 *            101       3100       3000
              MONWRITE 0203 *           3101       3338        238
-----------------------------------------------------------------------------
LIUSR2   3390                              0          0          1 Gap
              LI2000   0201 *              1       3000       3000
              MONWRITE 0200 *           3001       3338        338
-----------------------------------------------------------------------------
```

13.6 Network bandwidth usage

There are two options for attributing bandwidth consumption to individual Linux guests in a VM environment. You can either use the SNMP server that is provided as part of VM's TCP/IP stack, or you can use SNMP services provided within Linux. In our case, we've chosen to focus on SNMP services within a Linux environment. For detailed information on configuring an SNMP virtual machine under VM's TCP/IP stack, refer to *z/VM TCP/IP Planning and Customization*, SC24-5981.

13.6.1 An introduction to SNMP

Simple Network Management Protocol (SNMP) is an application-layer protocol that facilitates the exchange of management information between network devices. It is part of the TCP/IP protocol suite. There are two standard levels of SNMP: SNMPv1 and SNMPv2. There is a third version of SNMP SNMPv3, but acceptance of this as a standard is still pending.

The two primary components of an SNMP implementation are the SNMP agent and the Network Management Application. It is a client server architecture where the SNMP agent is the server and the SNMP manager is the client.

An *agent* is a software component that resides on a managed device and collects management information. A managed device could be a UPS, a router, a server, or one of a multitude of other device types. In our context, a managed device will be one or more Linux guest machines. The Network Management application can monitor and control devices on which an SNMP agent is running.

The three commands that are most commonly used in SNMP communications are read, write, and trap; they have the following characteristics:

Read	This command is used by the network management application to query SNMP agents for management information.
Write	This command is used by the network management application to modify variables maintained by the SNMP agent.
Trap	This command is used by SNMP agents to send alerts to network management applications when defined thresholds are met, or specific events occur.

The collection of management information that an agent is responsible for is called the Management Information Base (MIB). MIBs are organized hierarchically in a tree structure and are comprised of managed objects. Managed objects are the leaf nodes of the MIB tree.

SNMP is typically used to gauge network performance, find and resolve network problems, and plan for network growth. However, you can also use SNMP to monitor vendor-specific hardware such as the current load on a UPS, the CPU utilization on routers, hubs, and servers, and even disk I/O and free space.

13.6.2 SNMP installation

Most Linux distributions should include some version of SNMP. We chose to use the UCD-SNMP package for this redbook. We started on the Web at:

`http://net-snmp.sourceforge.net/`

Note: Although the RPM package is called ucdsnmp, the project has now been renamed to Net-SNMP.

UCD-SNMP includes various SNMP tools: an extensible agent, an SNMP library, tools for requesting or setting information from SNMP agents, tools for generating and handling SNMP traps, a version of the `netstat` command which uses SNMP, and a Tk/Perl MIB browser. You will probably also want to install the ucd-snmp-utils package, which contains UCD-SNMP utilities.

Note: The following example illustrates how we installed and configured SNMP using a SuSE system, with UCD-SNMP 4.2.1. The steps may differ if you are running a different Linux distribution, or if you are running a different level of UCD-SNMP.

To install the UCD-SNMP package, enter the following command:

```
# rpm -ivh /suse/cd1/n2/ucdsnmp.rpm
```

Once the package is installed, an example configuration file can be found in /usr/share/doc/packages/ucdsnmp/EXAMPLE.conf. Copy the EXAMPLE.conf file to the /etc directory as follows:

```
# cp /usr/share/doc/packages/ucdsnmp/EXAMPLE.conf /etc/ucdsnmpd.conf
```

13.6.3 SNMP configuration

We now want to configure SNMP for our local environment by editing the /etc/ucdsnmp.conf file. Example 13-5 shows the simple modifications we made to the file:

Example 13-5 Changes to /etc/ucdsnmp.conf

```
#       sec.name  source            community
com2sec local     localhost         localitso
com2sec mynetwork 9.0.0.0/8         itso
```

As shown, we set the community name (which is synonymous with a password) to `localitso`, in order to access the SNMP data from our local system. If we had wanted to access this machine's SNMP data from another machine in the network (limited to users with an IP address of 9.x.x.x), we would've used the password `itso`.

These modifications will be enough to get SNMP working in your environment; however, you should spend some time reviewing the configuration file to ensure you have the correct parameters set for your installation.

Now we edited the file /etc/rc.d/snmpd. We wanted to change the startup command so that the SNMP daemon uses our /etc/ucdsnmpd.conf as the configuration file.

Therefore, we changed the following line:

```
startproc /usr/sbin/snmpd -f || return=$rc_failed
```

to read:

```
startproc /usr/sbin/snmpd -f/etc/ucdsnmpd.conf  || return=$rc_failed
```

Finally, before starting the SNMP services, we edited /etc/rc.config by changing the following line:

```
START_SNMPD="no"
```

to read:

```
START_SNMPD="yes"
```

As a result, the SNMP daemon will start automatically from now on when Linux is booted. However, it is not actually running yet.

Note: Always remember to re-run SuSEconfig after making any changes to /etc/rc.config.

We were now ready to start SNMP services manually by using this command:

```
# rcsnmpd start
```

To test our SNMP implementation, we used the **snmpget** command. This command queries SNMP agents on specified hosts for one or more OID values. The syntax is as follows:

```
snmpget HOST COMMUNITY OID
```

Try the following command and you should get a similar response:

```
# snmpget localhost localitso .1.3.6.1.2.1.1.1.0
system.sysDescr.0 = Linux tux390 2.2.16 #1 SMP Wed Nov 8 10:57:03 GMT 2000 s390
```

The OID .1.3.6.1.2.1.1.1 maps to the system description. To see all of the available objects in our tree, we used the **snmpwalk** command. This command queries an entire tree, instead of individual OIDs.

The basic syntax is the same as **snmpget** (although the two commands have several different options):

```
# snmpwalk localhost public .1
```

With this command, you "walk" the entire tree of OIDs that are available to you. You can use the `snmpwalk` and `snmpget` commands from a remote Linux host on the network and get the same result.

This is a very basic implementation of SNMP. Included with the example ucdsnmpd.conf file are methods for monitoring CPU utilization, disk space, and several other useful examples. With these packages, you are also able to set traps to be sent to a specified host.

13.6.4 Network bandwidth monitoring

There are many tools available to monitor bandwidth consumption. In our case, we focus on just one of those tools, MRTG. This is not an endorsement of that product; rather, we picked one of the many available open source tools in this area merely to demonstrate how easy it is to install and configure.

13.6.5 MRTG

The Multi Router Traffic Grapher (MRTG) is an open source tool that utilizes SNMP to monitor the traffic load on servers, routers, or virtually anything that generates SNMP records. It can be found on the Internet at:

http://people.ee.ethz.ch/~oetiker/webtools/mrtg/

It is licensed for use under the terms of the GNU General Public License.

MRTG generates HTML pages containing PNG images which provide a snapshot visual representation of this traffic. MRTG is an excellent example of what you can do with SNMP. MRTG can be used to report on more than just network traffic; in our example, we also report on CPU consumption.

Note: The following example illustrates how we installed and configured MRTG using a SuSE system, using MRTG 2.9.10. The steps may differ if you are running a different Linux distribution or if you are running a different level of MRTG. Also note that we ran with UCD-SNMP 4.2.1.

13.6.6 MRTG installation and customization

If you are using SuSE, you can get the MRTG package from the n1 packages directory, filename mrtg.rpm

To install the MRTG package, enter the following command:

```
# rpm -ivh mrtg.rpm
```

In our example, we created a configuration file to monitor the network traffic on the localhost and provide us with CPU statistics. We first used the **`cfgmaker`** tool to create the configuration file:

```
# cfgmaker localitso@localhost > /etc/mrtg.conf
```

The **`cfgmaker`** program will discover the network interfaces that are defined to your Linux guest, and write this information (along with appropriate HTML tags) into the configuration file. In our example, the Linux guest has an OSA-Express Fast Ethernet interface.

Note: We encountered problems when MRTG tried to discover interface information for virtual CTC or IUCV devices. Refer to 13.6.8, "MRTG reporting for Virtual CTC or IUCV devices" on page 314 for a discussion of the extra steps needed to get bandwidth reporting to function using these devices.

Example 13-6 Ethernet interface as defined in /etc/mrtg.conf

```
Target[localhost_3]: 3:localitso@localhost:
SetEnv[localhost_3]: MRTG_INT_IP="9.12.6.73" MRTG_INT_DESCR="eth0"
MaxBytes[localhost_3]: 1250000
Title[localhost_3]: Traffic Analysis for 3 -- vmlinux7
PageTop[localhost_3]: <H1>Traffic Analysis for 3 -- vmlinux7</H1>
 <TABLE>
   <TR><TD>System:</TD>     <TD>vmlinux7 Guest Machine</TD></TR>
   <TR><TD>Maintainer:</TD> <TD>CCW <Caroline@javadog.org></TD></TR>
   <TR><TD>Description:</TD><TD>eth0  </TD></TR>
   <TR><TD>ifType:</TD>     <TD>ethernetCsmacd (6)</TD></TR>
   <TR><TD>ifName:</TD>     <TD></TD></TR>
   <TR><TD>Max Speed:</TD>  <TD>1250.0 kBytes/s</TD></TR>
   <TR><TD>Ip:</TD>         <TD>9.12.6.73 (vmlinux7.itso.ibm.com)</TD></TR>
 </TABLE>
```

We now needed to edit the newly created mrtg.conf file. At the top of the file, we added an entry for the working directory where MRTG will place the HTML and .PNG files. Because we were using Apache as the Web server in our example, we elected to use a subdirectory called mrtg off the default Apache DocumentRoot. We added the following WorkDir entry in the file /etc/mrtg.conf:

```
WorkDir: /usr/local/httpd/htdocs/mrtg
```

Before running MRTG, we also decided to add some additional CPU reporting definitions into the /etc/mrtg.conf file. This is an example of the extra reporting that can be achieved using SNMP—we are not limited to simply network statistics; instead CPU, memory, disk and many other resource measurements are available.

Example 13-7 CPU reporting definitions in /etc/mrtg.conf

```
LoadMIBs: /usr/share/snmp/mibs/UCD-SNMP-MIB.txt
Target[localhost.cpu]:ssCpuRawUser.0&ssCpuRawIdle.0:localitso@localhost
RouterUptime[localhost.cpu]: localitso@localhost
MaxBytes[localhost.cpu]: 100
Title[localhost.cpu]: CPU LOAD
PageTop[localhost.cpu]: <H1>User CPU Load %</H1>
Unscaled[localhost.cpu]: ymwd
ShortLegend[localhost.cpu]: %
YLegend[localhost.cpu]: CPU Utilization
Legend1[localhost.cpu]: User CPU in % (Load)
Legend2[localhost.cpu]: Idle CPU in % (Load)
Legend3[localhost.cpu]:
Legend4[localhost.cpu]:
LegendI[localhost.cpu]:  User
LegendO[localhost.cpu]:  Idle
Options[localhost.cpu]:  nopercent

LoadMIBs: /usr/share/snmp/mibs/UCD-SNMP-MIB.txt
Target[localhost.usrsys]:ssCpuRawUser.0&ssCpuRawSystem.0:localitso@localhost
RouterUptime[localhost.usrsys]: localitso@localhost
MaxBytes[localhost.usrsys]: 100
Title[localhost.usrsys]: CPU LOAD
PageTop[localhost.usrsys]: <H1>CPU (user and system) Load %</H1>
Unscaled[localhost.usrsys]: ymwd
ShortLegend[localhost.usrsys]: %
YLegend[localhost.usrsys]: CPU Utilization
Legend1[localhost.usrsys]: User CPU in % (Load)
Legend2[localhost.usrsys]: System CPU in % (Load)
Legend3[localhost.usrsys]:
Legend4[localhost.usrsys]:
LegendI[localhost.usrsys]:  User
LegendO[localhost.usrsys]:  System
Options[localhost.usrsys]:  nopercent

LoadMIBs: /usr/share/snmp/mibs/UCD-SNMP-MIB.txt
Target[localhost.cpusum]:ssCpuRawUser.0&ssCpuRawUser.0:localitso@localhost +
ssCpuRawSystem.0&ssCpuRawSystem.0:localitso@localhost /+
ssCpuRawNice.0&ssCpuRawNice.0:localitso@localhost
MaxBytes[localhost.cpusum]: 100
Title[localhost.cpusum]: CPU LOAD
PageTop[localhost.cpusum]: <H1>Active CPU Load %</H1>
Unscaled[localhost.cpusum]: ymwd
ShortLegend[localhost.cpusum]: %
YLegend[localhost.cpusum]: CPU Utilization
Legend1[localhost.cpusum]: Active CPU in % (Load)
Legend2[localhost.cpusum]:
Legend3[localhost.cpusum]:
```

```
Legend4[localhost.cpusum]:
LegendI[localhost.cpusum]:  Active
LegendO[localhost.cpusum]:
Options[localhost.cpusum]: nopercent
```

13.6.7 MRTG reporting

We were now ready to run MRTG. In your case, you should first run such a tool manually, and then, when you're happy with the reporting, you can use cron to automate the recording. The first couple of times MRTG is run, it gives warning messages such as those shown in Example 13-8. These messages are normal and can be safely ignored.

Example 13-8 Warning messages when first running MRTG

```
Rateup WARNING: /usr/bin//rateup could not read the primary log file for tux390.au.ibm.com
Rateup WARNING: /usr/bin//rateup The backup log file for tux390.au.ibm.com was invalid as well
Rateup WARNING: /usr/bin//rateup Can't remove tux390.au.ibm.com.old updating log file
Rateup WARNING: /usr/bin//rateup Can't rename tux390.au.ibm.com.log to tux390.au.ibm.com.old
updating log file
Rateup WARNING: /usr/bin//rateup could not read the primary log file for localhost.3
Rateup WARNING: /usr/bin//rateup The backup log file for localhost.3 was invalid as well
Rateup WARNING: /usr/bin//rateup Can't remove localhost.3.old updating log file
Rateup WARNING: /usr/bin//rateup Can't rename localhost.3.log to localhost.3.old updating log
file
```

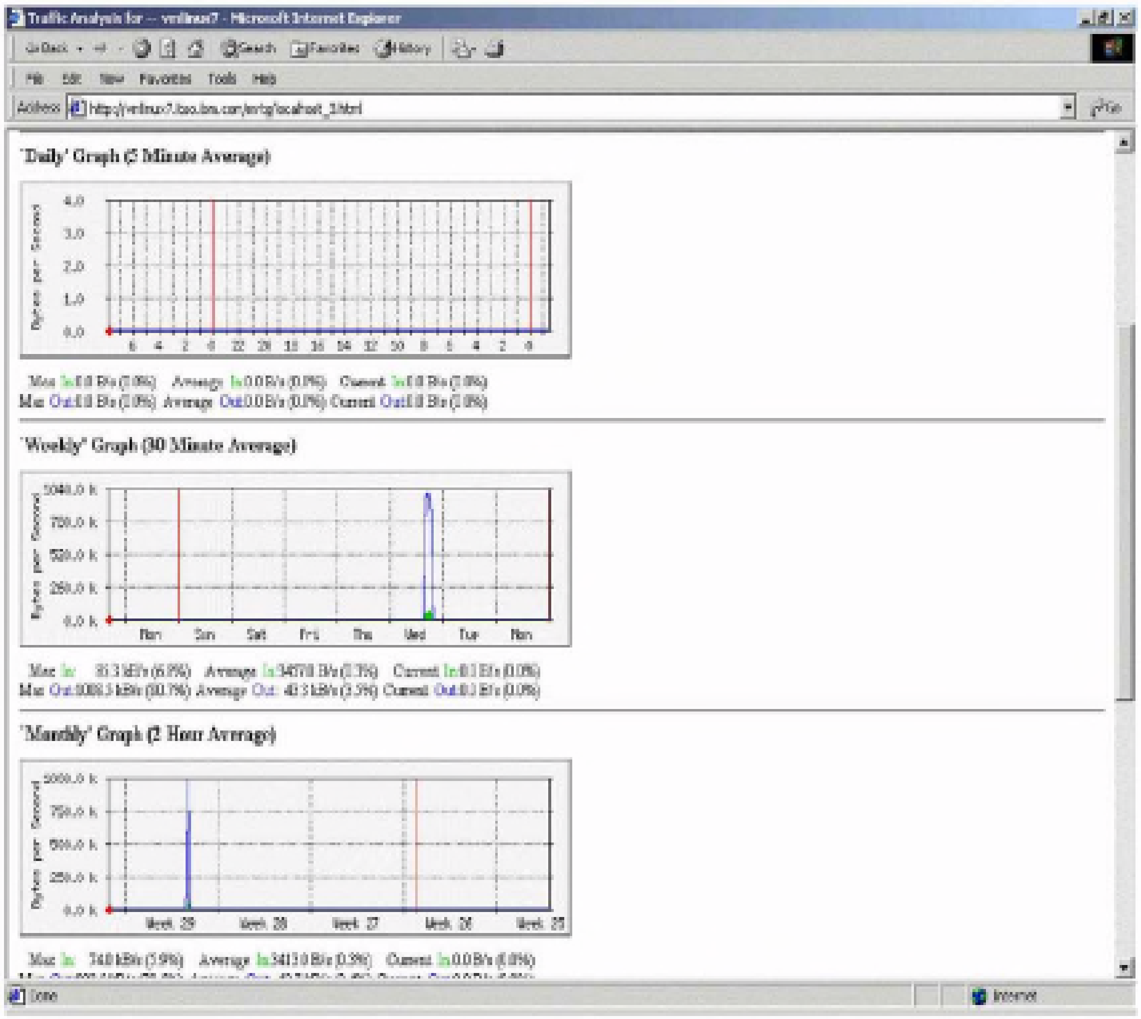
MRTG must be run regularly to capture SNMP statistics, with the recommended interval being every 5 minutes. You can update the /etc/crontab file to automate the running of MRTG, as shown in Example 13-9:

Example 13-9 An /etc/crontab definition to run MRTG automatically every 5 minutes

```
*/5     *       *       *       *       root    /usr/bin/mrtg /etc/mrtg.conf > /dev/null 2>&1
```

Using our example, we now saw a number of HTML pages populating the directory /usr/local/httpd/htdocs/mrtg.

An example of one of the pages MRTG generates is shown in Figure 13-1 on page 313:



'Daily' Graph (5 Minute Average)

Max In:0.0 B/s (0.0%) Average In:0.0 B/s (0.0%) Current In:0.0 B/s (0.0%)
Max Out:0.0 B/s (0.0%) Average Out:0.0 B/s (0.0%) Current Out:0.0 B/s (0.0%)

'Weekly' Graph (30 Minute Average)

Max In: 85.3 kB/s (6.8%) Average In:34570 B/s (2.7%) Current In:0.0 B/s (0.0%)
Max Out:1008.3 kB/s (80.7%) Average Out: 42.3 kB/s (3.3%) Current Out:0.0 B/s (0.0%)

'Monthly' Graph (2 Hour Average)

Max In: 74.0 kB/s (5.9%) Average In:3413.0 B/s (0.3%) Current In:0.0 B/s (0.0%)

Figure 13-1 MRTG Traffic Analysis page

It would be useful to see all the graphs for traffic analysis, CPU consumption, etc. on a single Web page. The MRTG package includes a utility called **indexmaker** which can be used to create an index.html page incorporating all the reports on to a single Web page.

```
# indexmaker /etc/mrtg.conf --output=/usr/local/httpd/htdocs/mrtg/index.html
```

In this example, we ran indexmaker against our MRTG configuration file, telling it to write the newly created index.html file to the MRTG HTML directory.

An example of the index.html page is shown in Figure 13-2 on page 314.

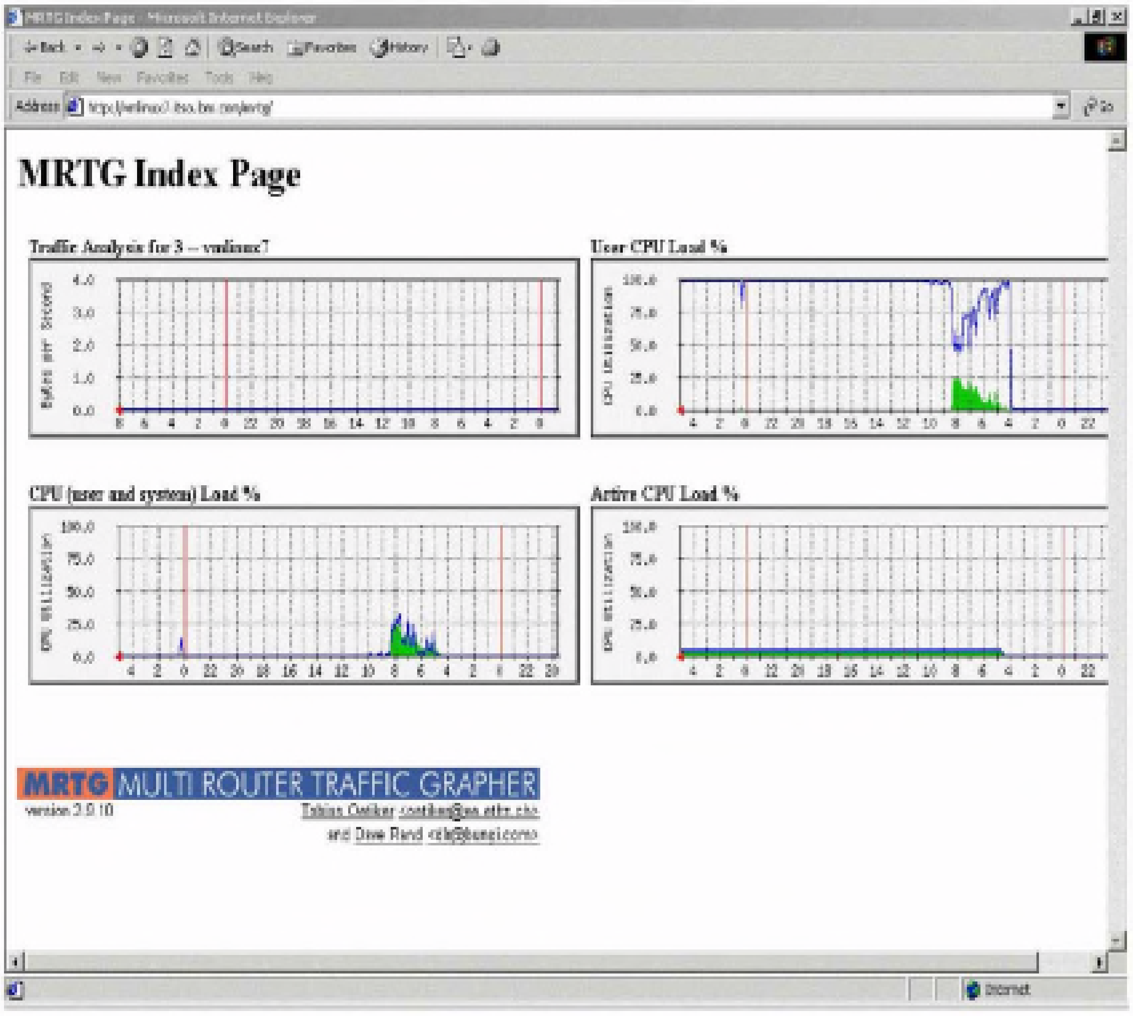

MRTG Index Page
Traffic Analysis for 3 -- vmlinux7
User CPU Load %
CPU (user and system) Load %
Active CPU Load %
MRTG MULTI ROUTER TRAFFIC GRAPHER

Figure 13-2 MRTG index page created by indexmaker

13.6.8 MRTG reporting for Virtual CTC or IUCV devices

When running many Linux guests under VM, it is highly probable that many of these guests will not have dedicated network interface cards. Instead, they will gain network connectivity via one or more virtual CTC or IUCV point-to-point connections with a VM TCP/IP stack.

We found that the MRTG program **cfgmaker** does not correctly recognize Virtual CTC or IUCV devices as their ifSpeed value in the MIB tree is set to 0. To get around this, we manually added a speed value into the MIB tree by adding a parameter into /etc/ucdsnmpd.conf as follows:

```
interface interface_name interface_type speed_in_bits/sec
```

Example 13-10 Changes to /etc/ucdsnmpd.conf for IUCV or CTC devices

```
interface ctc0 111 10000000
interface iucv0 111 10000000
```

In this example, we specified a maximum speed of 10Mbps. The `111` refers to the interface as a "Stack-to-Stack" device. For a complete listing of all available interface types, we referred to the file /usr/share/snmp/mibs/IANAifType-MIB.txt

Note: The speed value you specify will not affect the actual speed of the device; it will only alter the scaling of the MRTG graphs. You may need to alter this speed value to suit your environment to ensure the graph scaling is meaningful.

With these changes in place, we recycled the SNMP daemon as follows:

```
# rcsnmpd restart
```

At the level of SuSE we were running at time of writing (SuSE 7.2 beta, kernel 2.2.19), we found that adding a speed value in the MIB tree for the Virtual CTC or IUCV devices was not enough to get bandwidth information. The CTC and IUCV drivers at that level did not perform the necessary byte recording.

To resolve this issue, we made patches to both the CTC and IUCV drivers for the 2.2.19 level of the Linux kernel. These patches are in the file ctc-iucv-bytestat.patch and can be downloaded from:

```
ftp://www.redbooks.ibm.com/redbooks/SG246299
```

Ensure that you have the source code for the 2.2.19 Kernel in /usr/src/linux (this patch was only tested at 2.2.19). Copy the patch that you've downloaded from the Internet to the directory /usr/src/linux/drivers/s390/net/. You should see (among others) the files ctc.c and netiucv.c These are the source files for the CTC and IUCV drivers. To apply the patch from the shell, type:

```
# patch -b  -i ctc-iucv-bytestat.patch
```

You should see the output:

```
patching file ctc.c
patching file netiucv.c
```

If there are no errors listed, the module source code is now patched. Now compile the modules. To do this, go to the /usr/src/linux directory.

Note: This assumes that you have already selected that you want CTC and IUCV module support. If you have not already made this selection, then you need to first run the command **make menuconfig** from /usr/src/linux.

Then, under the selection:

```
S/390 Network device support
```

make sure the following selections are made:

```
<M> CTC device support
<M> IUCV device support (VM only)
```

Exit and save the configuration.

Compile the module support into binaries by typing:

```
# make modules
```

Once this has finished, type:

```
# make modules_install
```

This will install the modules into their runtime directories.

You can now use the modules; refer to *Linux for S/390 Device Drivers and Installation Commands* for information on module syntax. You should now have accurate byte recording for the Virtual CTC and IUCV devices.

Note: Because Linux kernel and device driver development are such fast-moving and dynamic fields, we will not be providing patches for any other level of the CTC/IUCV device drivers. From reviewing the source code for the CTC and IUCV drivers at the 2.4.5 level of the Linux kernel, it appears that the byte-recording limitation has been removed.

13.6.9 Monitoring multiple Linux guests

So far we've outlined how to set up SNMP and MRTG for a single Linux guest. However, as previously mentioned, in the context of this redbook we ran multiple Linux guest systems, so now we describe how to add more Linux guests into the MRTG reporting model.

To add more Linux guests into the MRTG reporting model, you first need to set up UCD-SNMP on any additional Linux guests that you wish to monitor. If you have established UCD-SNMP on a master Linux system that you use for cloning, then no further installation or configuration requirements may be necessary. However, if you do not use a cloning technique, then you have to set up UCD-SNMP according to the guidelines in this chapter.

Once the Linux systems that you wish to monitor have SNMP running, you can then run the MRTG **cfgmaker** utility to **snmpwalk** their MIB tree. This will determine what network interfaces they are using.

The syntax of *cfgmaker* is as follows:

```
cfgmaker community_name@host_name > config_file_name
```

Run it against another Linux guest, as follows:

```
# cfgmaker itso@vmlinux2.itso.ibm.com > mrtg.conf.vmlinux2
```

Now edit the mrtg.conf.vmlinux2 file, extracting the necessary interface information and adding it to your existing /etc/mrtg.conf file.

In our example, we added the definitions shown in Example 13-11 to the /etc/mrtg.conf file:

Example 13-11 Interface information gathered from walking vmlinux2's MIB tree

```
Target[vmlinux2.itso.ibm.com_4]: 4:itso@vmlinux2.itso.ibm.com:
SetEnv[vmlinux2.itso.ibm.com_4]: MRTG_INT_IP="9.12.6.99" MRTG_INT_DESCR="eth1"
MaxBytes[vmlinux2.itso.ibm.com_4]: 1250000
Title[vmlinux2.itso.ibm.com_4]: Traffic Analysis for -- vmlinux2
PageTop[vmlinux2.itso.ibm.com_4]: <H1>Traffic Analysis for -- vmlinux2</H1>
 <TABLE>
   <TR><TD>System:</TD>     <TD>vmlinux2 Guest machine.</TD></TR>
   <TR><TD>Maintainer:</TD> <TD>SEW <simon@42.org></TD></TR>
   <TR><TD>Description:</TD><TD>eth1  </TD></TR>
   <TR><TD>ifType:</TD>     <TD>ethernetCsmacd (6)</TD></TR>
   <TR><TD>ifName:</TD>     <TD></TD></TR>
   <TR><TD>Max Speed:</TD>  <TD>1250.0 kBytes/s</TD></TR>
   <TR><TD>Ip:</TD>         <TD>9.12.6.99 (vmlinux2.itso.ibm.com)</TD></TR>
 </TABLE>
```

Figure 13-3 on page 318 shows an example of an MRTG Web page reporting on network traffic for four Linux guest systems.

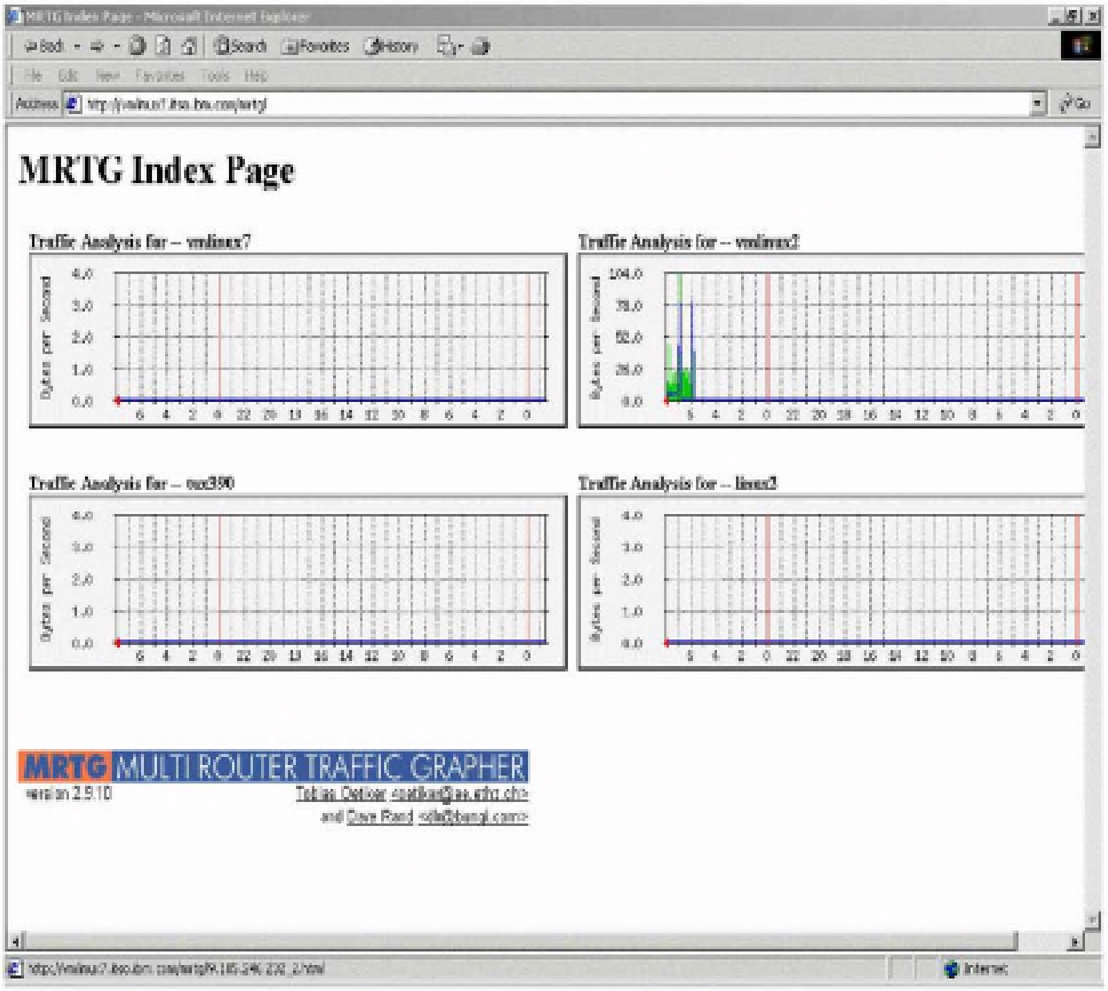

MRTG Index Page
Traffic Analysis for -- vmlinux7
Traffic Analysis for -- vmlinux2
Traffic Analysis for -- tux390
Traffic Analysis for -- linux2
Bytes per Second
MRTG MULTI ROUTER TRAFFIC GRAPHER

Figure 13-3 Multiple Linux guests on a single page

So far, we only looked at the traffic throughput reporting capabilities of MRTG. In the following section, we show how to determine how much total bandwidth over time is being used by the individual Linux guest machines.

13.6.10 Total bandwidth reporting

To report on the total network traffic profile for our Linux guests, we used the mrtg_total Perl script written by Josef Wendel. This script generates HTML reports for bandwidth usage on a per day and per month basis. The script is available on the Internet at:

http://www.geocities.com/josef_wendel/mrtg_total.html

There are a number of prerequisite Perl modules which you may not have installed on your system.

These modules are available from the Comprehensive Perl Archive Network (CPAN) Web site at the following URL:

```
http://cpan.valueclick.com/modules/by-module/GD
```

The modules we downloaded were:

- GD-1.33
- GDGraph-1.33
- GDTextUtil-0.80
- GDGraph3d-0.55

To use these modules, place the **mrtg_total** script and all the modules into a suitable directory on the Linux guest that is running MRTG. In our case, we placed the files in a directory called /home/gd.

Untar all of the files, then go into each subdirectory in the module order listed above and install that particular module. If you do not install them in the order listed, you may be trying to install a module that has a dependency on a module that hasn't been installed yet. Following is the first module installation, as an example:

```
$ cd /home/GD-1.33
$ perl Makefile.PL
$ make
$ make install
```

Now copy the **mrtg_total.pl** script to the /usr/local/bin directory.

Note: If you run **mrtg_total.pl** and get an error message stating `bad interpreter: No such file or directory`, then you need to change the location of the Perl program as referenced in the mrtg_total.pl script. The script expects to find the Perl executable in /usr/local/bin/perl. In a default SuSE system, that executable resides in /usr/bin/perl.

The **mrtg_total** script should be run once a day to generate its report. We ran it automatically by placing an entry into the /etc/crontab file.

Following is an /etc/crontab definition to run mrtg_total automatically once a day:

```
10 0  *  *  *     root  /usr/local/bin/mrtg_total.pl /etc/mrtg.conf
```

We were now ready to edit the MRTG configuration file to add extra parameters for each network interface so that the **mrtg_total** script will generate cumulative traffic information about each interface.

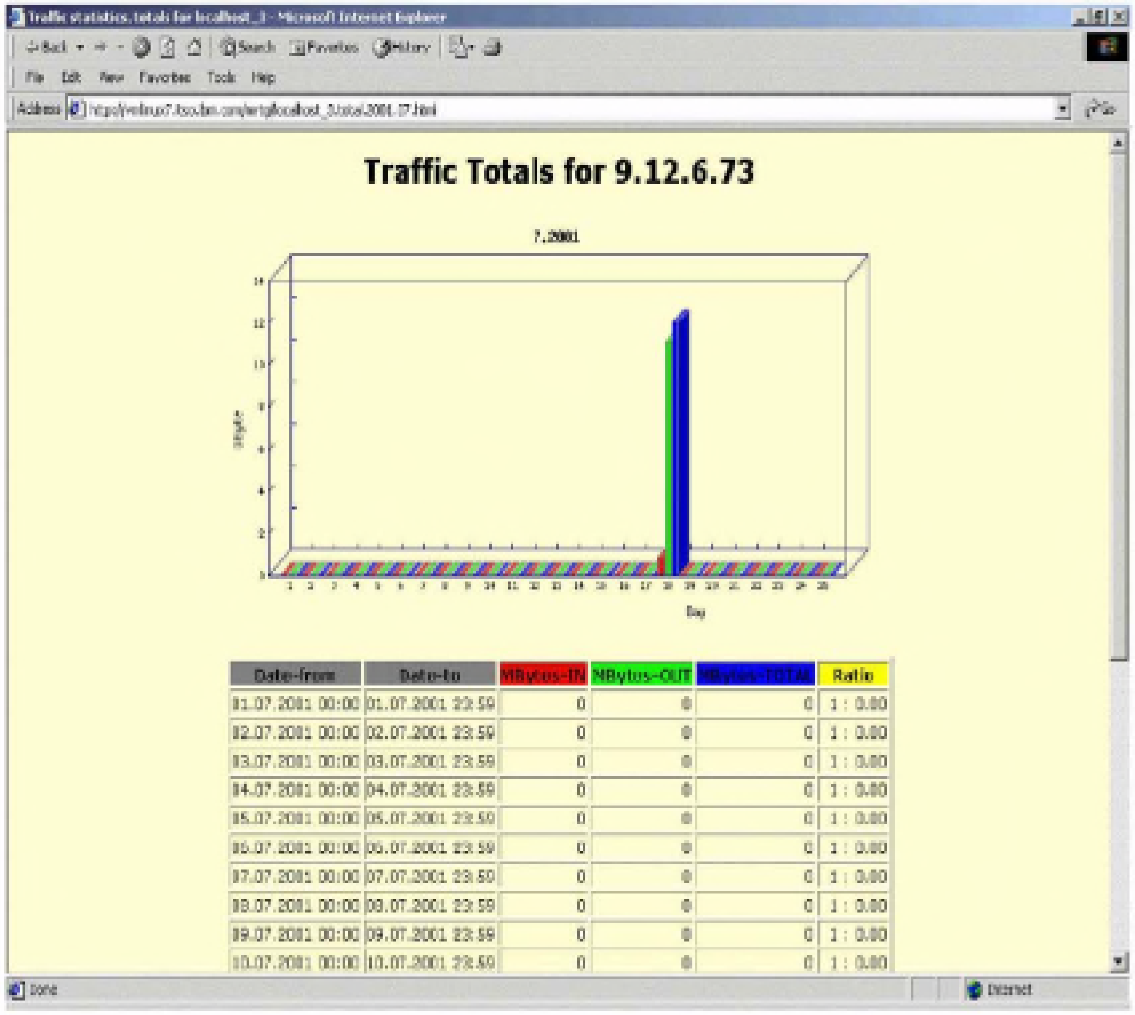
Using one of the interfaces from our previous example, we added the lines in italics to the /etc/mrtg.conf file; see Example 13-12 on page 320.

Example 13-12 Added parameters to include cumulative traffic information about each interface

```
### Interface 3 >> Descr: 'eth0' | Name: '' | Ip: '9.12.6.73' | Eth: '00-06-29-6c-cb-ce' ###
Target[localhost_3]: 3:localitso@localhost:
#-#Total[localhost_3]:      Traffic Totals for 9.12.6.73
#-#Total-Unit[localhost_3]: M
#-#Total-Ratio[localhost_3]:yes
SetEnv[localhost_3]: MRTG_INT_IP="9.12.6.73" MRTG_INT_DESCR="eth0"
MaxBytes[localhost_3]: 1250000
Title[localhost_3]: Traffic Analysis for 3 -- vmlinux7
PageTop[localhost_3]: <H1>Traffic Analysis for 3 -- vmlinux7</H1>
 <TABLE>
   <TR><TD>System:</TD>     <TD>vmlinux7 Guest machine.</TD></TR>
   <TR><TD>Maintainer:</TD> <TD>SEW <Simon@42.org></TD></TR>
   <TR><TD>Description:</TD><TD>eth0  </TD></TR>
   <TR><TD>ifType:</TD>     <TD>ethernetCsmacd (6)</TD></TR>
   <TR><TD>ifName:</TD>     <TD></TD></TR>
   <TR><TD>Max Speed:</TD>  <TD>1250.0 kBytes/s</TD></TR>
   <TR><TD>Ip:</TD>         <TD>9.12.6.73 (vmlinux7.itso.ibm.com)</TD></TR>
```

We would add additional lines, with relevant labels and comments, for each device we are reporting on in the MRTG configuration file.

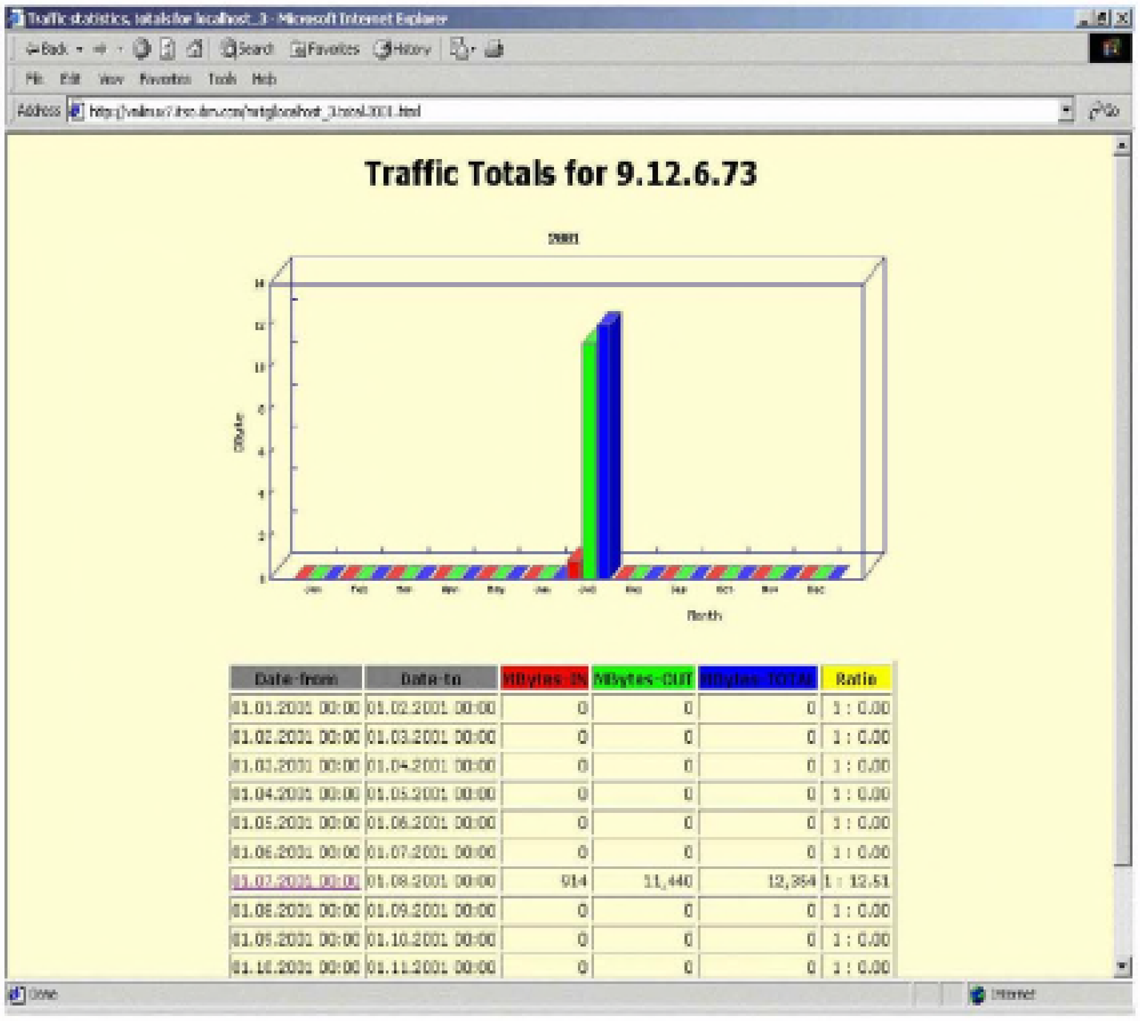
The graphs generated by the mrtg_total script reside in the same directory as all the other MRTG HTML files. Figure 13-4 on page 321 and Figure 13-5 on page 322 are two examples of traffic reporting by day and by month for one of our OSA-Express Fast Ethernet cards (as you can see, this was a test system and the traffic was only heavy for a short period).



Traffic Totals for 9.12.6.73

Date-from	Date-to	MBytes-IN	MBytes-OUT	MBytes-TOTAL	Ratio
01.07.2001 00:00	01.07.2001 23:59	0	0	0	1 : 0.00
02.07.2001 00:00	02.07.2001 23:59	0	0	0	1 : 0.00
03.07.2001 00:00	03.07.2001 23:59	0	0	0	1 : 0.00
04.07.2001 00:00	04.07.2001 23:59	0	0	0	1 : 0.00
05.07.2001 00:00	05.07.2001 23:59	0	0	0	1 : 0.00
06.07.2001 00:00	06.07.2001 23:59	0	0	0	1 : 0.00
07.07.2001 00:00	07.07.2001 23:59	0	0	0	1 : 0.00
08.07.2001 00:00	08.07.2001 23:59	0	0	0	1 : 0.00
09.07.2001 00:00	09.07.2001 23:59	0	0	0	1 : 0.00
10.07.2001 00:00	10.07.2001 23:59	0	0	0	1 : 0.00

Figure 13-4 Traffic totals by day



Date-from	Date-to	MBytes-IN	MBytes-OUT	MBytes-TOTAL	Ratio
01.01.2001 00:00	01.02.2001 00:00	0	0	0	1 : 0.00
01.02.2001 00:00	01.03.2001 00:00	0	0	0	1 : 0.00
01.03.2001 00:00	01.04.2001 00:00	0	0	0	1 : 0.00
01.04.2001 00:00	01.05.2001 00:00	0	0	0	1 : 0.00
01.05.2001 00:00	01.06.2001 00:00	0	0	0	1 : 0.00
01.06.2001 00:00	01.07.2001 00:00	0	0	0	1 : 0.00
01.07.2001 00:00	01.08.2001 00:00	914	11,440	12,354	1 : 12.51
01.08.2001 00:00	01.09.2001 00:00	0	0	0	1 : 0.00
01.09.2001 00:00	01.10.2001 00:00	0	0	0	1 : 0.00
01.10.2001 00:00	01.11.2001 00:00	0	0	0	1 : 0.00

Figure 13-5 Traffic totals by month

13.7 Availability monitoring

In any computing environment, a mechanism is needed to monitor the health of the systems. In an environment with hundreds of Linux guest machines running under VM, it would quickly prove impractical to monitor these systems manually.

As is often the case in the open source community, there are many availability monitoring packages to choose from. We focused on a tool called NetSaint, which is available on the Web at:

`http://www.netsaint.org`

The tool appeared to be straightforward to install and configure, and provided us with enough monitoring function to be useful in an environment with many Linux systems.

13.7.1 NetSaint

Note: The following example illustrates how we installed and configured NetSaint 0.0.6 using a SuSE system. The steps may differ if you are running a different Linux distribution or if you are running a different level of NetSaint.

NetSaint is an open source software package that carries the GPL license. It can be used to monitor not only Linux hosts, but also many other types of servers or routers. It can also monitor individual services that those machines are running.

For example, if a server is unavailable, or network traffic is running slow or disk drives are filling up, NetSaint can be used to e-mail or page support staff. NetSaint uses a modular, "plugin" architecture to perform the actual service checks, which allows you to choose from a large number of plugins and also allows you to develop your own plugins for the specific needs of your environment.

The NetSaint package can be downloaded from the following URL:

```
http://www.netsaint.org/download
```

The version we used in this redbook was NetSaint 0.0.6. As you may surmise from the version numbering, this was still very much a work in progress; nevertheless, from our testing we found it to be very useful.

We downloaded the package netsaint-0.0.6.tar.gz, which includes the core program, CGIs, and documentation from the URL:

```
http://www.netsaint.org/download/netsaint-0.0.6.tar.gz
```

We also downloaded the NetSaint plugins package (at level 1.2.9-4) from the Source Forge Web site at the following URL:

```
http://prdownloads.sourceforge.net/netsaintplug/netsaint-plugins-1.2.9-4.tar.gz
```

13.7.2 Installing NetSaint

Note: As long as you are running Linux and have a copy of the GNU C Compiler (gcc), the installation of NetSaint should prove to be simple. However, if you wish to see the graphs produced by the statusmap CGI, you will also need to install the GD library. If the GD library does not come as part of your distribution, you can download it from the following URL:

http://www.boutell.com/gd

To check whether or not you have the GD library installed, execute the following `find` command from the shell:

```
# find / -name libgd.a
```

If you get a match, it means that you have the GD library installed.

The installation manual that comes with the NetSaint package is the definitive guide to all aspects of installing and configuring the product. We include the following sections simply to give you a "jump-start" guide to getting the product and up and running quickly.

First decide which Linux guest machine will act as the NetSaint server. This machine will interrogate other machines, as defined in its configuration file, to determine their availability and service level. Once you have selected a suitable Linux guest, upload the two files mentioned above into a suitable directory. We used /home/netsaint/. Untar the files as follows:

```
# tar -zxvf netsaint-0.0.6.tar.gz
# tar -zxvf netsaint-plugins-1.2.9-4.tar.gz
```

You should now have two new subdirectories, netsaint-0.0.6 and netsaint-plugins-1.2.9-4. In our case, we first configured and compiled the core NetSaint program, and then configured and compiled the plugins.

Next, we created a base directory to be used as the runtime directory for NetSaint:

```
# mkdir /usr/local/netsaint
```

Before starting the compilation, we needed to create a NetSaint group and user ID to be used for file ownership. The following two commands illustrate how to do this:

```
# groupadd netsntg
# useradd netsaint -g netsntg
```

We were now ready to run the configuration script to initialize variables and create the makefile:

```
# ./configure --prefix=/usr/local/netsaint  --with-cgiurl=/cgi-bin/netsaint \
--with-htmurl=/netsaint/ --with-netsaint-user=netsaint \
--with-netsaint-grp=netsntg --with-gd-lib=/usr/local/lib/ \
--with-gd-inc=/usr/local/include/
```

When that completed successfully, a makefile had been built and we were ready to compile NetSaint core code and the CGIs:

```
# make all
```

After that completed successfully, we installed the binaries, documentation, and sample HTML files:

```
# make install
```

We created and installed sample configuration files using the following commands:

```
# make config
# make install-config
```

We needed to change directory to the plugins install directory, and run the configuration script for the NetSaint plugins, as follows:

```
# cd /home/netsaint/netsaint-plugins-1.2.9-4
# ./configure --prefix=/usr/local/netsaint --with-netsaint-user=netsaint \
--with-netsaint-group=netsntg --with-cgiurl=/cgi-bin/netsaint
```

After that completed successfully, we compiled the plugins with the following command:

```
# make all
```

Finally, we installed the binaries to the directory /usr/local/netsaint/libexec/ as follows:

```
# make install
```

13.7.3 Configuring the Web interface

The next step is to configure the Web interface so that you can access NetSaint status pages and run the NetSaint CGI programs from your Web browser.

Note: The following section makes the assumption that you are running Apache as your Web server.

We edited our Apache configuration file (by default, this usually resides in a file called /etc/httpd/httpd.conf).

We added the following line (in our example, we added it at the bottom of the file):

```
Alias /netsaint/ /usr/local/netsaint/share/
```

Next we needed to create an alias for the NetSaint CGIs:

```
ScriptAlias /cgi-bin/netsaint/ /usr/local/netsaint/sbin/
```

Important: The ScriptAlias definition must *precede* the default Apache cgi-bin ScriptAlias entry in the httpd.conf file.

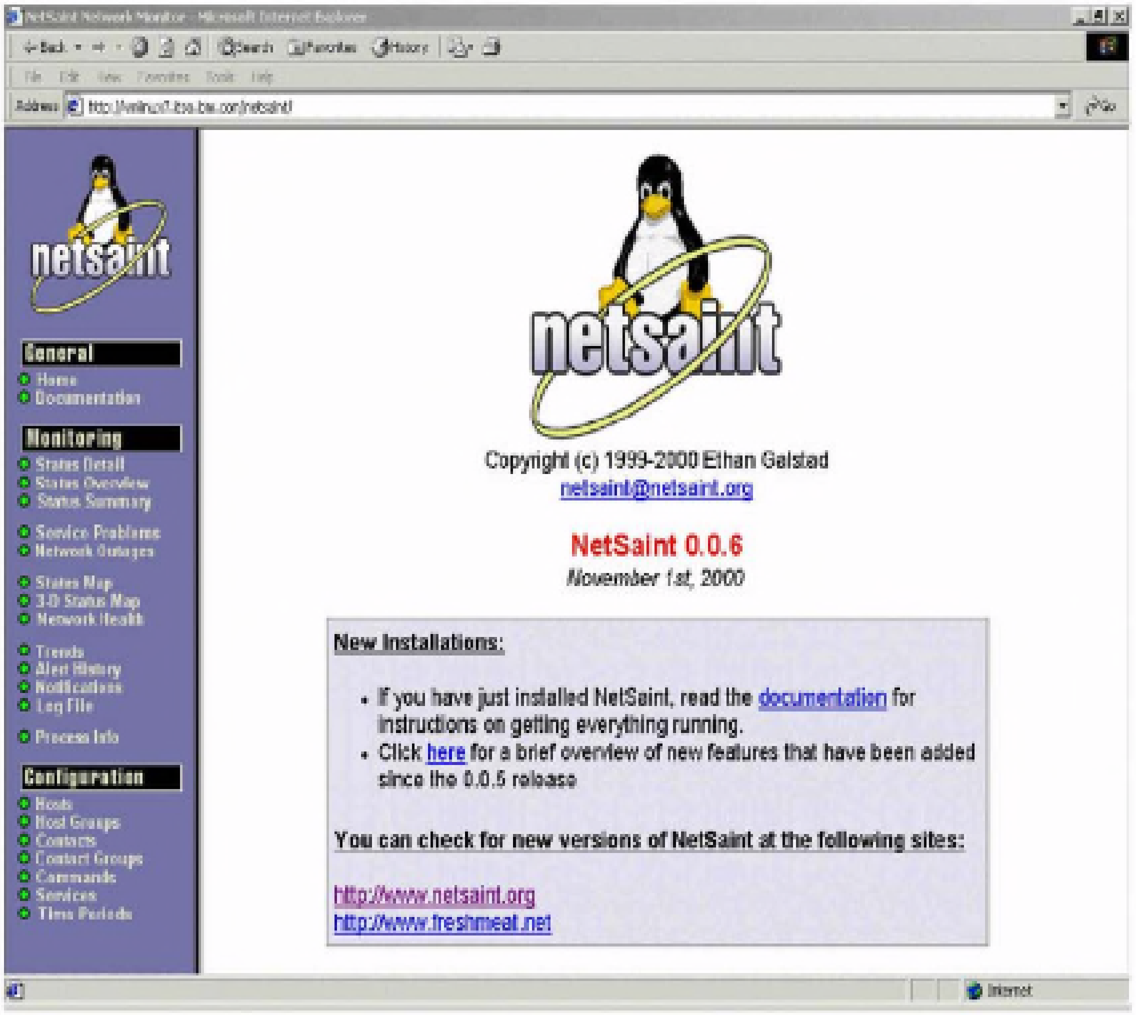
Now we restarted the Apache Web server; if running SuSE, this can be done as follows:

```
   # rcapache restart
```

You should now be able to go to the NetSaint Web interface by pointing your Web browser at the following URL:

```
   http://whatever_your_hostname_is/netsaint
```

You should be greeted with the screen shown in Figure 13-6 on page 327:



Copyright (c) 1999-2000 Ethan Galstad
netsaint@netsaint.org

NetSaint 0.0.6
November 1st, 2000

New Installations:

- If you have just installed NetSaint, read the documentation for instructions on getting everything running.
- Click here for a brief overview of new features that have been added since the 0.0.5 release

You can check for new versions of NetSaint at the following sites:

http://www.netsaint.org
http://www.freshmeat.net

Figure 13-6 NetSaint welcome screen

13.7.4 User authorization

We secured our NetSaint services, so that only authorized staff can use the tool, by adding the lines shown in Example 13-13 to the /etc/httpd/httpd.conf file:

Example 13-13 User Authentication statements in httpd.conf

```
<Directory /usr/local/netsaint/sbin>
AllowOverride AuthConfig
order allow,deny
allow from all
Options ExecCGI
</Directory>

<Directory /usr/local/netsaint/share>
AllowOverride AuthConfig
```

```
order allow,deny
allow from all
</Directory>
```

At this point we've only told Apache that access to the NetSaint CGI's and HTML files requires authorization. We now needed to define the users that are authorized to access those files using the Apache **htpasswd** program. The command syntax is as follows:

```
# htpasswd -c /usr/local/netsaint/etc/htpasswd.users netsaintadmin
```

This will create a file called htpasswd.users, with the first user ID being netsaintadmin. The **htpasswd** program will prompt you for a password for that user ID. To create additional users, use the following command:

```
# htpasswd /usr/local/netsaint/etc/htpasswd.users <username>
```

We now needed to create a file called .htaccess (yes, that is a period at the front of the file name). A copy of the file must reside in two locations: /usr/local/netsaint/sbin, and /usr/local/netsaint/share. The contents of the file should be as follows:

```
AuthName "NetSaint Access"
AuthType Basic
AuthUserFile /usr/local/netsaint/etc/htpasswd.users
require valid-user
```

Finally we needed to make some modifications to the CGI configuration file /usr/local/netsaint/etc/nscgi.conf.

The following changes were made for our configuration:

```
use_authentication=1
authorized_for_system_information=*
authorized_for_configuration_information=*
authorized_for_system_commands=*
authorized_for_all_services=*
authorized_for_all_hosts=*
authorized_for_all_service_commands=*
authorized_for_all_host_commands=*
```

The `use_authentication=1` parameter enables authorization checking. The `authorized_for` parameters all have a value of asterisk (*). This means that any user who has successfully been authenticated by Apache will have access to these CGIs.

13.7.5 Configuring NetSaint

Since we've configured the NetSaint infrastructure, we now turn to configuring NetSaint itself. NetSaint has three configuration files: a "main" configuration file, netsaint.cfg; a "host" configuration file, hosts.cfg; and a "CGI" configuration file, nscgi.cfg. The configuration files reside in the directory /usr/local/netsaint/etc/.

Unless you have made any changes to the default installation steps previously described above, there's no need to make changes to the main configuration file unless you specifically want to.

The bulk of the configuration occurs in the hosts.cfg file. It is here that you define the systems that you wish to monitor. This file is quite complex, so we'll use an example environment to describe how to configure it.

Figure 13-7 uses the NetSaint package to display our network and server configuration:

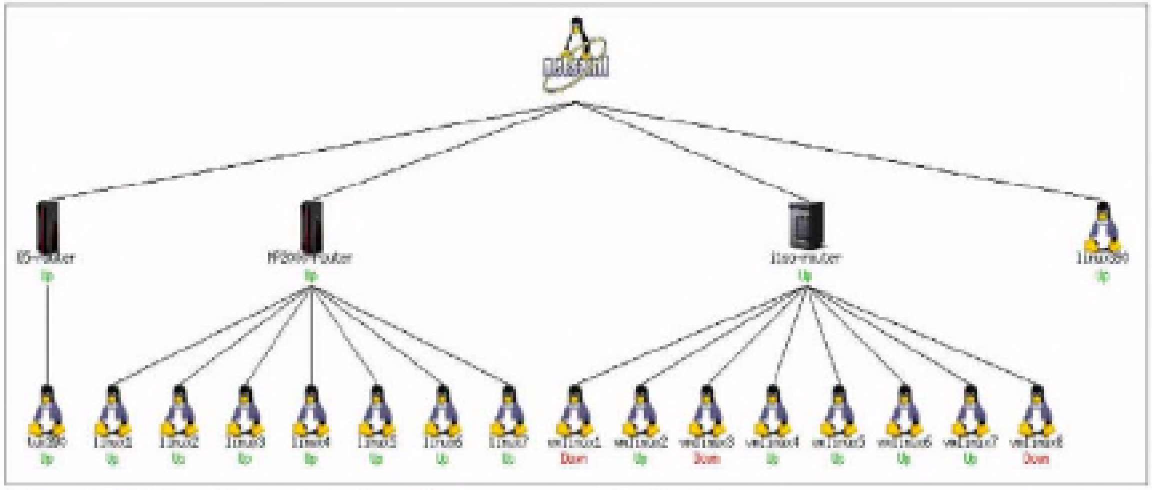


Figure 13-7 Example configuration - diagram generated by NetSaint

We focus on the two machines at the far left of the diagram, the G5-router and tux390. The definitions for all the other machines in the diagram will be very similar to those two.

We start by opening the file /usr/local/netsaint/etc/hosts.cfg in an editor.

Host definitions

We first need to discuss the host definition section. The syntax of that section is shown in Example 13-14:

Example 13-14 Host definitions syntax

```
host[<host_name>]=<host_alias>;<address>;<parent_hosts>;<host_check_command>;
```

```
<max_attempts>;<notification_interval>;<notification_period>;
<notify_recovery>;<notify_down>;<notify_unreachable>;
<event_handler>
```

Example 13-15 shows our coding:

Example 13-15 Host definitions

```
host[G5-router]=G5 VM 3.1 Router;9.185.122.219;;check-router-alive;20;60;24x7;1;1;1;
host[tux390]=tux390 VM Guest;9.185.122.217;G5-router;check-host-alive;10;120;24x7;1;1;1;
```

The G5-router is actually a VM TCP/IP stack on a z/VM LPAR running on a 9672 G5 processor. The tux390 host is a Linux VM guest connected to the VM TCP/IP stack via Virtual CTC links.

In the host definition for the G5-router, we say that we want to run the **check-router-alive** command against this machine. The **check-router-alive** command is defined in the file /usr/local/netsaint/etc/commands.cfg. It sends a single ICMP ping to the defined machine every 60 seconds.

To change the checking interval from 60 seconds to another value, you must change the interval_length parameter; this is set in the netsaint.cfg file. If there is 100% packet loss, or if the round trip average is 5000ms (5 seconds) or longer, an error is flagged. These settings are can be configured in the commands.cfg file.

Note: For a complete description of all the available configuration options, refer to NetSaint documentation.

Similar definitions were been made for the tux390 host. Note, however, that we defined the G5-router as the parent host of the tux390 machine. This is particularly useful when we use the status map and 3-D status map CGIs, because we can easily see the relationship between different machines in our network.

Host groups

Next, we need to review the host groups section. As the name implies, this allows us to group together one or more hosts for the purposes of notifying support staff when an outage is detected.

The syntax for this section is:

```
hostgroup[<group_name>]=<group_alias>;<contact_groups>;<hosts>
```

In our example, we coded the following:

```
hostgroup[G5Penguins]=All Linux Guests under VM on G5;linux-admins;tux390
hostgroup[routers]=All routers;linux-admins;itso-router,MP2000-router,G5-router
```

Note that we only defined the tux390 host under the G5Penguins group; if we had more Linux guests under this VM system, then we would add their names here.

Command configuration

The command configuration section can be used to define the functions you wish to run when an exception occurs. For example, we chose to use the default e-mail notification, which notifies the defined user whenever a problem is detected:

Example 13-16 Send an e-mail when an exception is detected

```
command[notify-by-email]=/bin/echo -e '***** NetSaint 0.0.6 *****\n\nNotification Type:
$NOTIFICATIONTYPE$\n\nService: $SERVICEDESC$\nHost: $HOSTALIAS$\nAddress: $HOSTADDRESS$\nState:
$SERVICESTATE$\n\nDate/Time: $DATETIME$\n\nAdditional Info:\n\n$OUTPUT$' | /bin/mail -s '**
$NOTIFICATIONTYPE$ alert - $HOSTALIAS$/$SERVICEDESC$ is $SERVICESTATE$ **' $CONTACTEMAIL$
```

Contact configuration

The contact section defines who to contact with the notification that an outage has occurred. We defined the *netsaintuser* as the recipient of all contact messages:

Example 13-17 Contact section

```
contact[netsaintuser]=NetsaintUser;24x7;24x7;1;1;1;1;1;1;notify-by-email,notify-by-epager;host-
notify-by-email,host-notify-by-epager;root
```

When there is a problem, the netsaintuser will be notified via e-mail, and also via a pager message sent through an e-mail pager gateway.

Service configuration

NetSaint not only monitors the availability of whole host machines, but it can also monitor individual services running within a machine. For example, we can define services to be monitored such as POP3, HTTP, and DNS. Netsaint will check that these services are indeed active on a machine.

The syntax of the service sections is shown in Example 13-18:

Example 13-18 Service section

```
service[<host>]=<description>;<volatile>;<check_period>;<max_attempts>;
              <check_interval>;<retry_interval>;<notification_group>;
              <notification_interval>;<notification_period>;<notify_recovery>;
```

```
<notify_critical>;<notify_warning>;<event_hander>;<check_command>
```

For the G5-router, we defined the following service definition:

Example 13-19 G5-router service definition

```
service[G5-router]=PING;0;24x7;3;5;1;linux-admins;240;24x7;1;1;0;;check_ping
```

Because this machine was simply acting as a router to our Linux guest, all we needed to do was ping the machine regularly to verify that it was still up. For our tux390 host, we added several more service definitions, as follows:

Example 13-20 tux390 host service definition

```
service[tux390]=PING;0;24x7;3;5;1;linux-admins;240;24x7;1;1;0;;check_ping
service[tux390]=HTTP;0;24x7;3;2;1;linux-admins;240;24x7;1;1;1;;check_http
service[tux390]=DNS;0;24x7;3;2;1;linux-admins;240;24x7;1;1;1;;check_dns
```

Along with regular pings to check machine availability, we also checked that the Web server and DNS were running.

The final step to complete before running NetSaint is to edit the nscgi.conf file. In our case, we wanted to add definitions for site-specifc graphics and enable NetSaint process checking by making the following additions:

Example 13-21 Changes to nscgi.conf file

```
hostextinfo[tux390]=/serverinfo/tux390.html;bluetux.gif;bluetux.jpg;bluetux.gd2;Linux 390;
hostextinfo[G5-router]=/serverinfo/g5.html;G5icon.gif;G5icon.jpg;G5icon.gd2;System 390 G5;

netsaint_check_command=/usr/local/netsaint/libexec/check_netsaint \
/usr/local/netsaint/var/status.log 5 '/usr/local/netsaint/bin/netsaint' #uncomment this line
```

We added graphics (.gif, .jpg and .gd2 files) so that the various screens available in the NetSaint Web interface will display our unique icons.

Note: You need to create these graphics yourself. To create .gd2 files, you can use the utility pngtogd2, which is part of the GD package.

13.7.6 Starting and stopping NetSaint

Note: Refer to the NetSaint documentation for a complete description of all options available for starting, stopping, and restarting NetSaint.

There are four methods of starting NetSaint: manually from a shell in the foreground; manually from a shell, but running NetSaint as a background task; manually as a Daemon; or automatically, as the Linux system boots.

While initial testing is carried out, we recommend that you manually run NetSaint as a foreground task. When you're comfortable that everything is configured correctly, you can automate NetSaint to start at boot time with the following command:

```
/usr/local/netsaint/bin/netsaint /usr/local/netsaint/etc/netsaint.cfg
```

To stop NetSaint when running in foreground mode, simply press <CTRL-C> to get out of the program. Refer to NetSaint documentation for other methods.

13.7.7 Using NetSaint

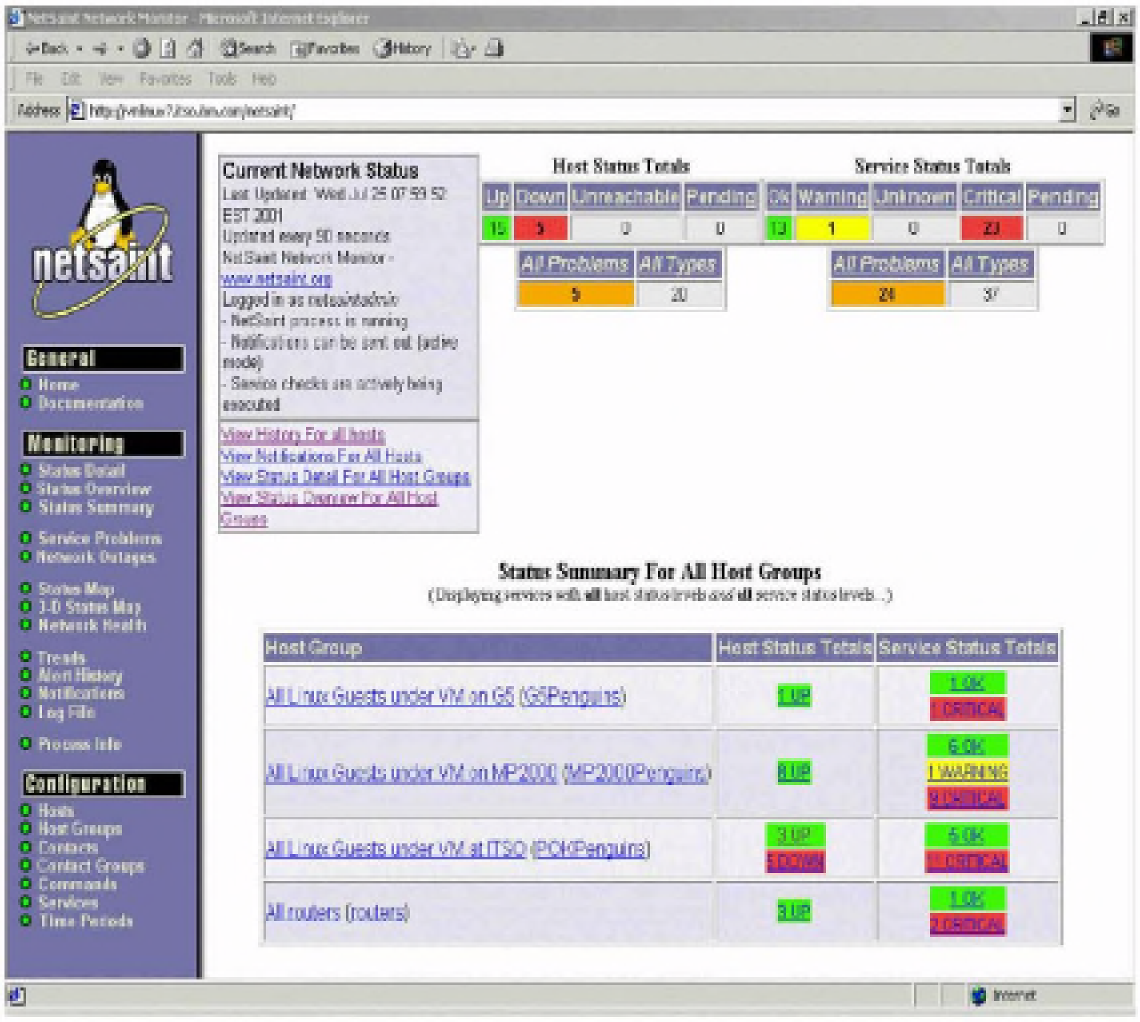
Once you have completed the configuration, you're ready to start NetSaint. Go to the main Web page to begin verifying that it is indeed working as you expect.

Type in the valid URL from your Web browser (it should be something like the following:

```
http://your_hostname/netsaint
```

You'll be prompted for a valid user ID and password; if you followed the example in this chapter, the user ID should be netsaintadmin. You will then be greeted by the Web page as displayed in 13.7.3, "Configuring the Web interface" on page 325.

To verify that your configuration definitions are successful, select the option **Status Summary**. Using that screen, as shown in Figure 13-8 on page 334, you can then drill down to individual servers.



Current Network Status
Last Updated: Wed Jul 25 07:59:52 EST 2001
Updated every 90 seconds
NetSaint Network Monitor - www.netsaint.org
Logged in as netsaintadmin
- NetSaint process is running
- Notifications can be sent out (active mode)
- Service checks are actively being executed

View History For all hosts
View Notifications For All Hosts
View Status Detail For All Host Groups
View Status Overview For All Host Groups

Host Status Totals

Up	Down	Unreachable	Pending
15	5	0	0

All Problems	All Types
5	20

Service Status Totals

Ok	Warning	Unknown	Critical	Pending
13	1	0	23	0

All Problems	All Types
24	37

Status Summary For All Host Groups
(Displaying services with **all** host status levels *and* **all** service status levels...)

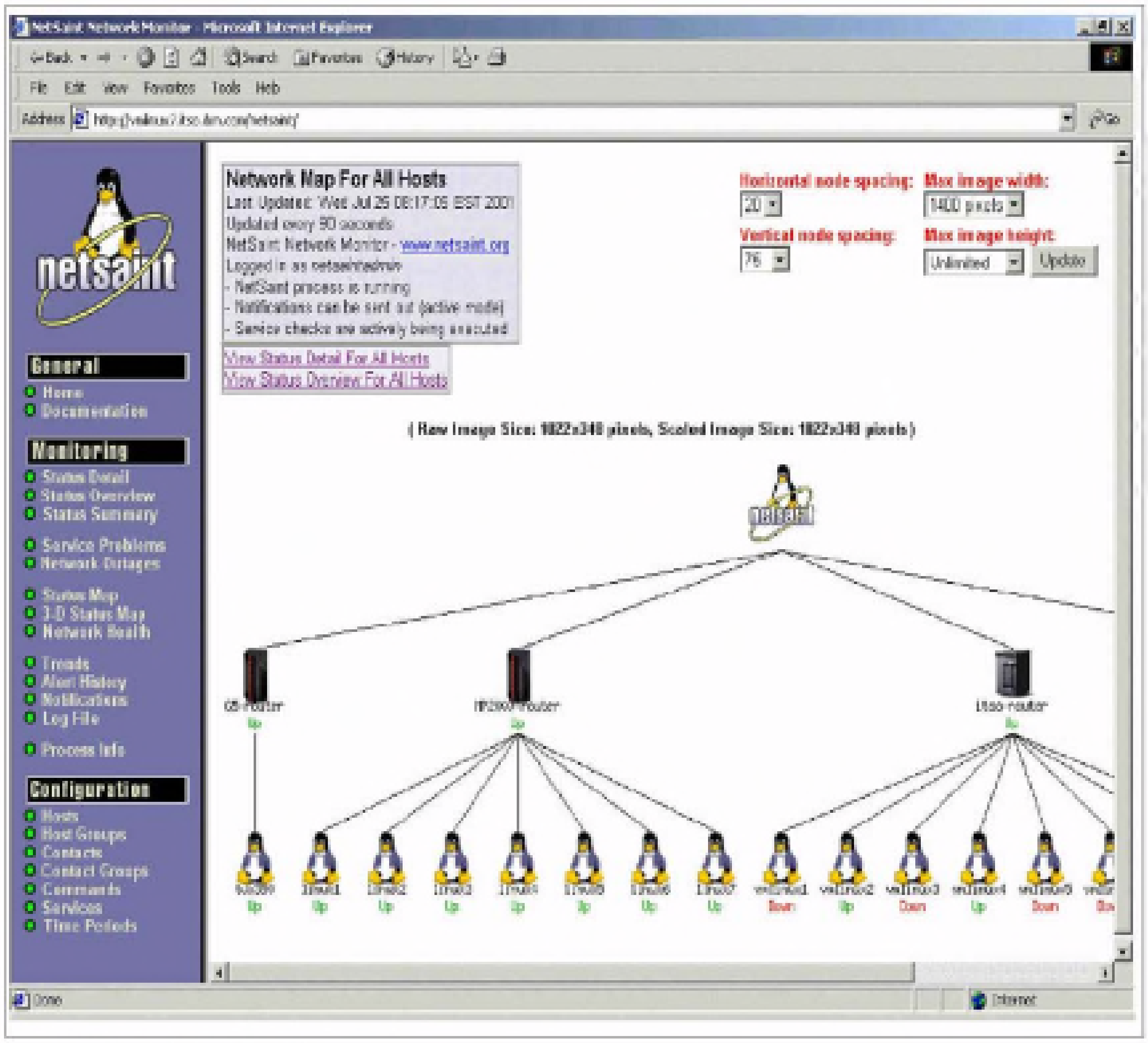
Host Group	Host Status Totals	Service Status Totals
All Linux Guests under VM on G5 (G5Penguins)	1 UP	1 OK 1 CRITICAL
All Linux Guests under VM on MP2000 (MP2000Penguins)	8 UP	6 OK 1 WARNING 9 CRITICAL
All Linux Guests under VM at ITSO (POKPenguins)	3 UP 5 DOWN	5 OK 11 CRITICAL
All routers (routers)	3 UP	1 OK 2 CRITICAL

Figure 13-8 NetSaint Status Summary screen

NetSaint has two very useful graphic-based CGIs: statusmap.cgi, and statuswrl.cgi. statuswrl.cgi produces VRML output.

Important: If you select the 3D-Status Map option from the Web page and it prompts you to save a file to disk, then it means you do not have a VRML plugin installed for your Web browser. The NetSaint documentation recommends using a VRML plugin such as Cortona, Cosmo Player or WorldView.

Figure 13-9 on page 335 and Figure 13-10 on page 336 show examples of these CGIs.



Network Map For All Hosts
Last Updated: Wed Jul 25 08:17:05 EST 2001
Updated every 90 seconds
NetSaint Network Monitor - www.netsaint.org
Logged in as netsaintadmin
- NetSaint process is running
- Notifications can be sent out (active mode)
- Service checks are actively being executed

Horizontal node spacing: 20
Vertical node spacing: 76
Max image width: 1400 pixels
Max image height: Unlimited

(Raw Image Size: 1822x348 pixels, Scaled Image Size: 1822x348 pixels)

Figure 13-9 Status map CGI

You can select individual machines and gather more information about their service status.

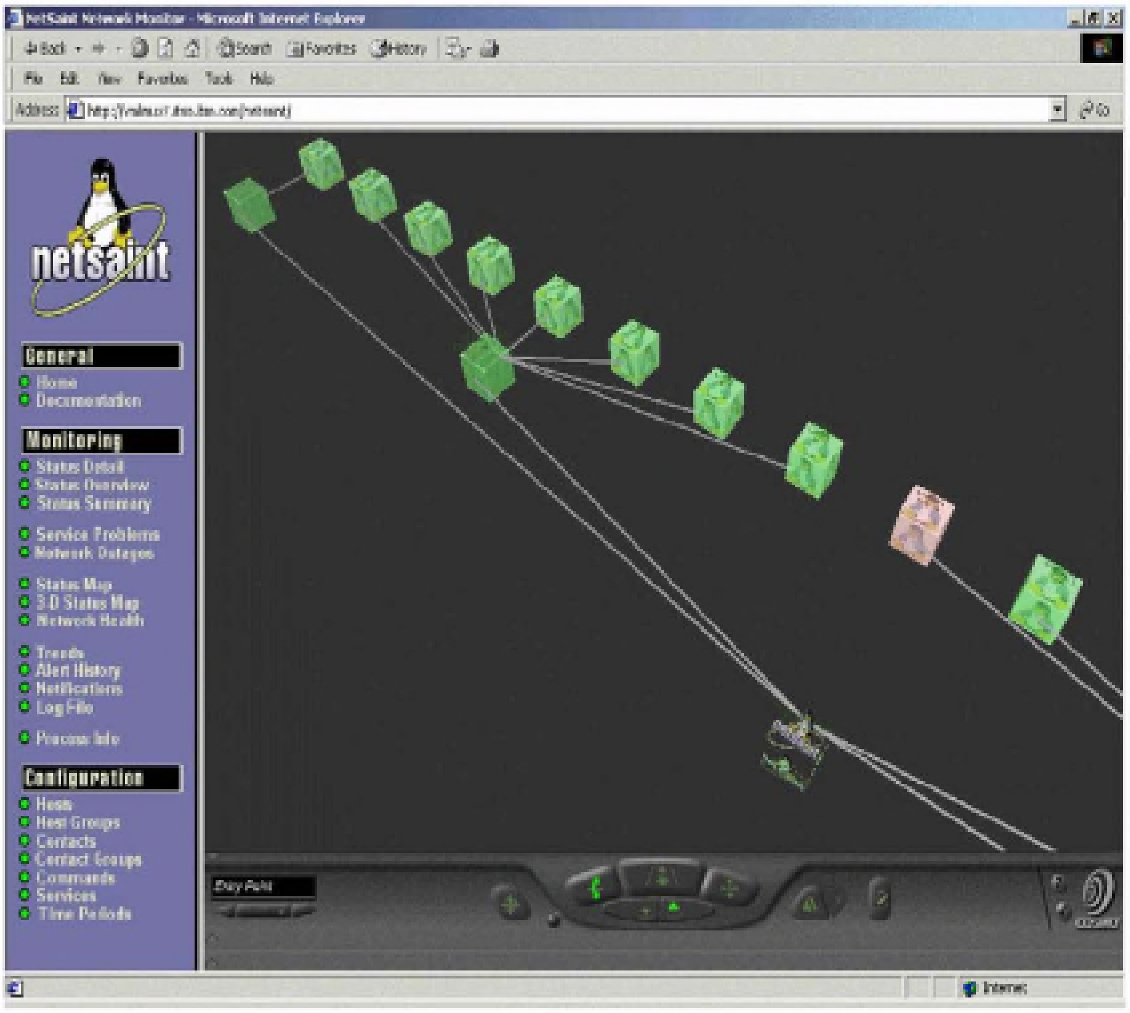
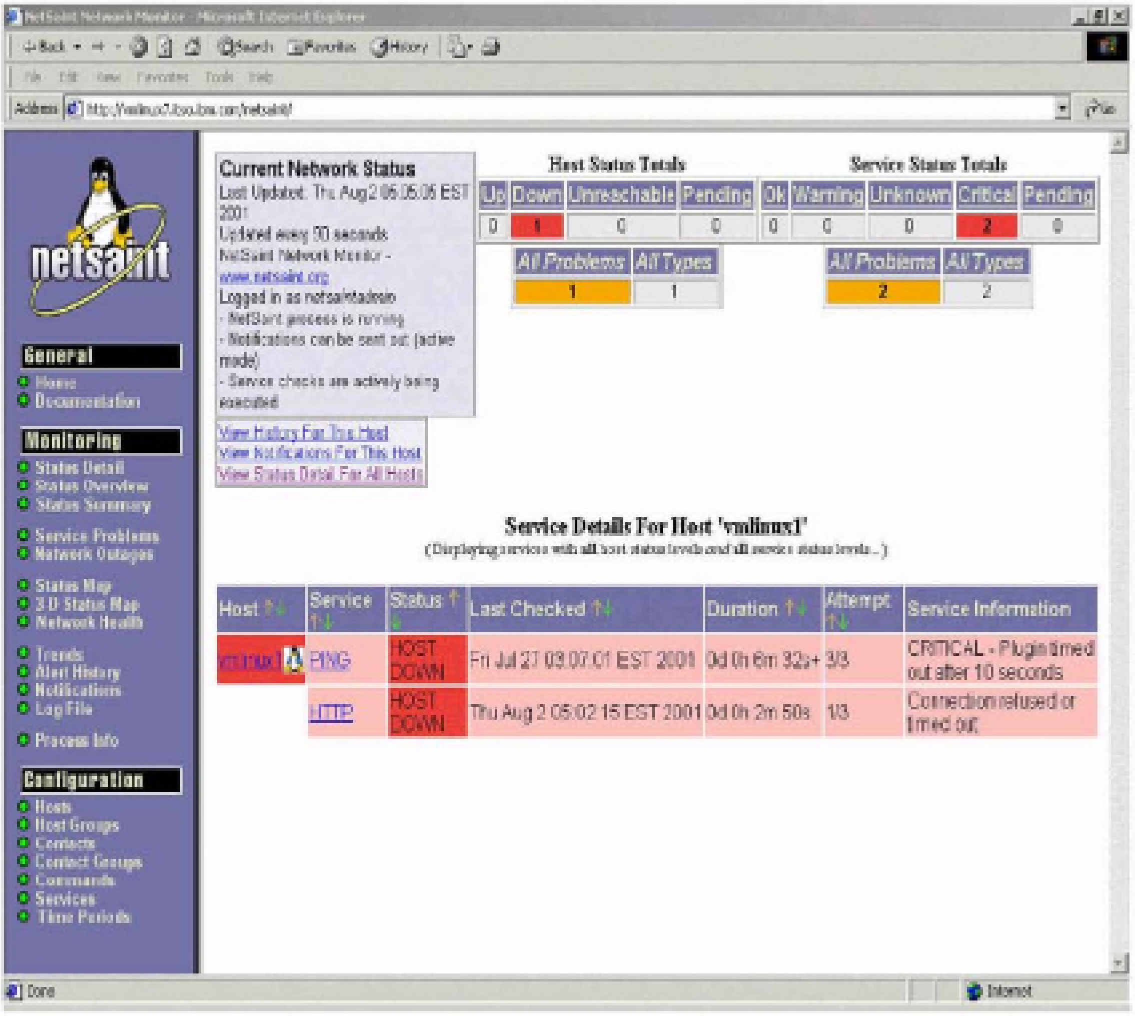


Figure 13-10 3D status map

With the 3D status map, you can literally fly around the Linux guests, identifying ones that may have problems (in our example, the problem guest displays as red).

If you find a machine that has a service problem, you can click that guest and be transferred to the status page for that specific machine. From there you can drill down further, thus pinpointing exactly what the problem is.

In Figure 13-11 on page 337, we can see that the Linux guest vmlinux1 is currently down:



Current Network Status
Last Updated: Thu Aug 2 06:06:06 EST 2001
Updated every 90 seconds
NetSaint Network Monitor - www.netsaint.org
Logged in as netsaintadmin
- NetSaint process is running
- Notifications can be sent out (active mode)
- Service checks are actively being executed

View History For This Host
View Notifications For This Host
View Status Detail For All Hosts

Host Status Totals

Up	Down	Unreachable	Pending
0	1	0	0

All Problems	All Types
1	1

Service Status Totals

Ok	Warning	Unknown	Critical	Pending
0	0	0	2	0

All Problems	All Types
2	2

Service Details For Host 'vmlinux1'
(Displaying services with all host status levels and all service status levels...)

Host	Service	Status	Last Checked	Duration	Attempt	Service Information
vmlinux1	PING	HOST DOWN	Fri Jul 27 09:07:01 EST 2001	0d 0h 6m 32s+	3/3	CRITICAL - Plugin timed out after 10 seconds
	HTTP	HOST DOWN	Thu Aug 2 05:02:15 EST 2001	0d 0h 2m 50s	1/3	Connection refused or timed out

Figure 13-11 Service details screen

13.8 Summary

In this chapter we have outlined some of the methods by which an organization can account for resources consumed by Linux/390 VM Guest systems. We have also examined just a few of the many ways in which organizations can monitor their Linux systems.

Comprehensive monitoring can be done very cheaply through many, very professional, Open Source packages. And as the popularity of Linux grows, more and more tools are becoming available every month. These tools are being written both by the open source community and by an ever-increasing number of software vendors, such as Tivoli.

Undoubtedly the methods and sophistication of accounting for Linux resource consumption will continue to mature, as we have seen on other platforms.

14

Web application servers

One of the most significant developments to appear recently in the software world is the Web application server. Although technically "application server" is a generic term, in common usage it refers to a server that implements the Java 2 Enterprise Edition (J2EE) standard. Some Java application servers include IBM WebSphere Application Server, BEA WebLogic, Lutris Enhydra, and Tomcat and Resin from the Apache Project.

Properly deploying WebSphere Application Server or any other application server is a very complex undertaking with many key decision points that are dependent on the particulars of each application. Fully exploring such a deployment is beyond the scope of this book.

There are issues with WebSphere Application Server (and other application servers) that require special consideration in the Linux for zSeries and S/390 environment. In this chapter, we discuss some of these issues and provide possible solutions.

14.1 WebSphere issues

In order for you to effectively deploy WebSphere on Linux for zSeries and S/390, we need to explain how WebSphere workloads differ from the other workloads we've discussed. These differences have a major effect on your deployment options and the relative merits of those options.

14.1.1 Java Virtual Machine (JVM)

As an implementation of the J2EE standard, WebSphere is based completely on Java technology. This means that all of WebSphere (both internal and application components) runs in a Java Virtual Machine (JVM). In some sense, the JVM can be viewed as its own small virtual server, which emulates a sort of system with a limited set of instructions and functions. Compiled Java byte-code is executed by the JVM, which runs as a process under the host operating system.

Some implementations of the JVM include Just-In-Time compilation (JIT). This technique involves compiling the Java byte-code into native instructions just before execution. This technique potentially provides significant improvements in execution time.

As a result of this interpreted implementation, Java applications tend to be somewhat more CPU-intensive than would otherwise be expected. For starters, the byte-code interpretation can take up to half of the execution time. There is also some overhead associated with Java's garbage collection routines. In the case of JIT compilation, some extra CPU time is still required to perform the translation to native code, though the total is usually still much less than for fully interpreted byte-code.

The JVM is also a pure stack-based machine; there are no "registers" to pass parameters, so data is always allocated on the stack (and thus uses memory).

14.1.2 Objects

Java is considered an object-oriented language. Object-oriented programming is a tool for improving software engineering methods and programmer effectiveness, not performance. Several artifacts of Java's object-orientation affect the resource demands of Java code:

- Loading of class hierarchies - loading one class requires loading all that class's ancestors, as well
- Indirection - most data accesses are through at least two levels of indirection, which makes memory access a more significant part of overall performance

- Dynamic binding - most methods are looked up by name (i.e. a string) rather than by an address, thus requiring lots of string parsing overhead at runtime[1]
- Cache-unfriendly behavior - objects tend to be less easily kept in cache, especially for architectures with a large number of distinct objects

In general, these behaviors mean that Java code tends to be a large consumer of memory, both in terms of usage and bandwidth.

It is important to also recognize that the architecture of WebSphere applications themselves can have a significant impact on the run-time behavior. Applications that are written as a few, relatively large objects can take advantage of a deep private cache (such as that provided by the pSeries) and thus are not as efficient on a shared-cache architecture such as the zSeries. Conversely, applications composed of many smaller objects will tend to be more efficient on the zSeries.

14.2 Options for running WebSphere

IBM WebSphere Application Server presents several different options for deployment in the Linux for zSeries and S/390 environment. Each deployment option has its own advantages and disadvantages, and is more appropriate for some environments than others. In fact, the particular behavior of a specific WebSphere Application Server application may make it more appropriately implemented in a particular way.

Note that because of the relatively high memory and CPU demands of WebSphere, we do not currently recommend running many (tens or hundreds) of separate WebSphere Application Server instances on Linux for zSeries and S/390.

14.2.1 WebSphere Application Server for zSeries and S/390 Linux

One option is to use WebSphere Application Server V3.5 for Linux for zSeries and S/390. This version is shipped with the Java 1.2.2 JVM rather than the Java 1.3, and thus will not have the performance enhancements offered by the new JVM. However, this option allows the user to leverage their existing expertise in Linux

We hope that eventually WebSphere Application Server V4 will be released for Linux for zSeries and S/390, and that some of the performance enhancements will carry over to this platform; however, at the time of writing no such release has been announced.

[1] This is not to say that this technique is necessarily a negative; it actually simplifies relocation, for one. But nothing comes for free, and there is a runtime cost associated with it.

14.2.2 WebSphere Application Server for z/OS

Another configuration option is to deploy WebSphere Application Server for z/OS on a separate LPAR running z/OS. At the time of writing, the current released version WebSphere Application Server for z/OS is V4, which includes Java 1.3 JVM. This JVM is expected to have significant performance gains over the Java 1.2.2 JVM included in WebSphere Application Server V3.5.

An additional cost will be the software and staffing costs associated with z/OS. This option may be most appropriate if z/OS is already installed in the organization and personnel are available to support it.

14.2.3 Separate servers

A third option, of course, is to deploy WebSphere Application Server on separate servers - pSeries AIX servers, for example. This method allows WebSphere Application Server to have exclusive access to all the memory and CPU resources on the machine. It also significantly increases the complexity of an installation, requiring additional hardware, power, network, and other resources. However, in situations where WebSphere Application Server is running extremely complex applications at very high utilizations, this may be the best way to handle the workload.

14.3 Planning for WebSphere Application Server

A key factor in planning for WebSphere Application Server is minimizing the impact it has on other z/VM guests while ensuring it has sufficient resources to meet its performance targets. Fortunately, the zSeries architecture provides powerful tools ideally suited to achieving these goals. By combining z/VM and LPARs, we can deploy WebSphere Application Server in ways that maximize performance and manageability.

We *strongly* recommend that WebSphere Application Server servers be set up in their own LPAR, separate from the other Linux for zSeries and S/390 guests (Web servers and such). This applies to WebSphere Application Server for Linux for zSeries and S/390, as well as to WebSphere Application Server for z/OS. In both cases, having a dedicated set of resources with the same general access patterns makes management simpler and minimizes unexpected interactions.

We also recommend that WebSphere Application Server servers be deployed as z/VM guests. The additional flexibility conferred by z/VM in administering the servers is significant, especially in a production ISP or ASP environment.

14.4 Test environments

One particular advantage of running WebSphere Application Server as a VM guest is that it allows the easy creation of development, test, and staging servers. This becomes especially important since it is not currently feasible to run individual WebSphere Application Servers for hundreds of clients. If customer applications need to be run on a shared WebSphere Application Server environment, then it is critical that there be cordoned-off areas where applications can be thoroughly tested before being deployed on the production server.

This is one area where the zSeries architecture has a distinct advantage. By creating test and staging servers as additional z/VM guests, it is possible to have a test environment that is essentially identical to the production environment, without additional equipment costs. These guests should be created within the WebSphere Application Server-dedicated LPAR to maintain segregation from other Linux for zSeries and S/390 guests.

An example deployment scenario is presented in Figure 14-1. Note that while the example shows the WebSphere Application Server LPAR as running on CPs rather than IFLs, this is only necessary in the case of WebSphere Application Server for z/OS; the processors for that LPAR can be IFLs if running WebSphere Application Server for Linux for zSeries and S/390.

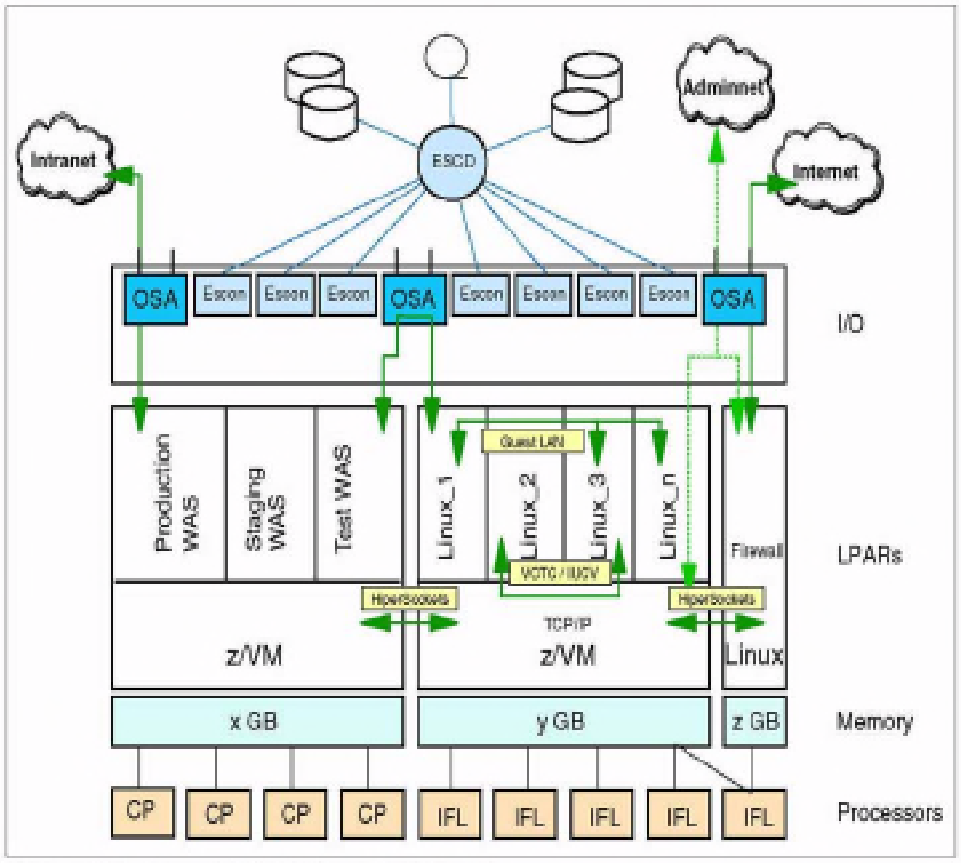


Figure 14-1 Sample WebSphere deployment

15

Integrating and replacing Microsoft servers

In this chapter we will discuss how to use you VM Linux images to replace existing Microsoft servers or integrate into the existing Microsoft Windows network.

15.1 Using Samba as a domain controller

Samba can also act as Windows NT 4.0 Primary Domain Controller (PDC) for Windows clients. When set up in this fashion, it has the following capabilities:

- Perform domain logons for Windows NT 4.0/2000 clients
- Place Windows 9x clients in user level security
- Retrieve a list of users and groups from a Samba PDC to Windows 9x/NT/2000 clients
- Supply roaming user profiles
- Allow Windows NT 4.0 style system policies

Note: Samba 2.2.1 is required for PDC functionality when using Windows 2000 SP2 clients

15.1.1 Setting up a Samba PDC

The following settings should be made in the /etc/smb.conf file for Samba to act as Windows PDC:

- In the [global] section, we add the following values:
  - `netbios name = VMLINUX8` // NETBIOS name of the server
  - `workgroup = SAMBAITSO` // domain name
  - `os level = 64`
  - `preferred master = yes` // we want to act as preferred master
  - `domain master = yes` // we want to act as domain master
  - `local master = yes` // we want to act local master
  - `security = user` // passwords are kept on the Samba server
  - `encrypt passwords = yes`
  - `domain logons = yes` // we allow domain logons
  - `logon path = \\%N\%U\profile` // where the user profiles are stored
  - `logon drive = H:` // where user home directories will be mounted
  - `logon home = \\%N\%U` // user home directory
  - `logon script = logon.cmd` // generic logon script for all users
- In the [netlogon] section we define the *netlogon* share. The `logon script` value in the global section is relative to this share.
  - `path = /sambashares/netlogon` // directory for *netlogon* share
  - `writeable = no`
  - `writelist = ntadmin` // only administrators can write here
- [profiles] section, here we define the share for storing user profiles:
  - `path = /sambashares/ntprofiles`
  - `writeable = yes`
  - `create mask = 0600`
  - `directory mask = 0700`

The /etc/smb.conf from our test system is as follows:

```
# Samba config file created using SWAT
# from tot67.itso.ibm.com (9.12.6.133)
# Date: 2001/08/07 14:05:20

# Global parameters
[global]
        workgroup = SAMBAITSO
        netbios name = VMLINUX8
        server string = Samba PDC 2.2.1a on VMLINUX8
        encrypt passwords = Yes
        map to guest = Bad User
        keepalive = 30
        domain logons = Yes
        os level = 64
        preferred master = True
        domain master = True
        wins support = Yes
        kernel oplocks = No
        winbind uid = 10000-20000
        winbind gid = 10000-20000
        template shell = /bin/bash
        winbind separator = +

[homes]
        comment = home-directory
        read only = No
        create mask = 0750
        browseable = No

[printers]
        comment = All Printers
        path = /tmp
        create mask = 0700
        printable = Yes
        browseable = No

[windowsshare]
        comment = Share for W2KSAMBA Domain Users group
        path = /sambashares/windowsshare

[netlogon]
        path = /sambashares/netlogon
        write list = ntadmin

[profiles]
        path = /sambashares/ntprofiles
        read only = No
        create mask = 0600
        directory mask = 0700
```

15.1.2 Creating a machine trust account

For each machine participating in a domain, a machine trust account has to be created on the Domain controller. The password for the machine account acts as a secret for secure communication with the Domain controller. This means that the PDC cannot be spoofed by another computer with the same NETBIOS name trying to access the data. Windows 9x computers are never true members of a domain because they do not have a machine trust account and thus there is no secure communication with the Domain controller.

In the current release, Samba requires a UNIX user ID to exist because Samba computes the Windows NT SIDs from the UNIX UIDs. This means that all machine accounts must have an entry in /etc/passwd and /etc/smbpasswd. You can create a machine trust account in two ways:

1. Manually

   Using the **useradd** command, we added a machine (in our example) with the NETBIOS name TOT11:

   ```
   # useradd -g 100 -d /dev/null -c Peters_W2K -m -s /bin/false TOT11$
   ```

   The following entry is now added to the /etc/passwd file:

   ```
   TOT11$:x:501:100:Peters_W2K:/dev/null:/bin/false
   ```

   Now you need to create the /etc/smbpasswd entry with the following command:

   ```
   # smbpasswd -a -m TOT11
   Added user TOT11$
   ```

   Now you should join the domain from the client computer.

   Important: Manually creating is the same as creating an account with Server Manager in Windows NT. From the time you create a machine trust account manually to the time you join to the domain, every client with the NETBIOS name can join the domain. This means that it can get a lot of data from the domain, because PDC inherently trusts the members of a domain.

2. Automatically

   The recommended way of creating an account is to add it on the fly. For this you need to add the following parameter to your /etc/smb.conf file:

   ```
   add user script = /usr/bin/useradd -d /dev/null  -g 100 -s /bin/false -M %u
   ```

Important: In Samba 2.2.1 only the *root* can be used to add the machine accounts. Therefore, it is necessary to create an entry in /etc/smbpasswd for root. For security reasons we recommend that you use a different password from the one used for the UNIX root account.

15.2 Using Samba in Windows domains

Integration of the Linux and Windows worlds is a big challenge of today's IT industry. One of the aspects of this challenge is how to use the same account created in a Windows Active Directory database on the Linux servers. For example, if we incorporate a Samba server into the Windows 2000 Active Directory and want to share the disks from this Samba server, we would normally need to reproduce all user IDs, group IDs, and passwords on the Linux server. To avoid this work, the package Winbind was written.

Winbind combines the worlds of UNIX and Windows NT/2000 by allowing a UNIX server to become a full member of a Windows domain. After joining the domain, the UNIX server sees all users and groups as if they were native UNIX users and groups. This means that whenever the UNIX programs query for the user ID or group ID, they ask the Windows Domain controller for the specified domain to validate the user or group ID.

Winbind hooks into the operating system at a low level, via the NSS name resolution modules in the C library. This redirection to the Windows Domain controller is transparent. Users on the UNIX machine can use Windows user and group names as they would use "native" UNIX names. For example, they can own the files with their user ID, log in to the system and even run an X-window session as Windows Domain users. The only difference in user names is that they have the Domain name incorporated into the user name, for example DOMAIN\username or DOMAIN\groupname. This is necessary for Winbind so it can determine which Domain controller is responsible for authentication.

Winbind also provides the Pluggable Authentication Module (PAM) to provide authentication via a Windows 2000 domain to any PAM-enabled application. This solves the problem of synchronizing passwords between the systems, because all the passwords are stored in a single location on the Windows Domain controller or Active Directory.

15.2.1 Recompiling the latest Samba package

In our environment we were using the SuSE 7.2 31 bit version of Linux. Before recompiling the newest version you should install the version which comes with the distribution so that the /etc/rc.config file gets updated with the correct configuration for Samba. You also need to ensure that the Samba daemon is started automatically by specifying "yes" in START_SMB in /etc/rc.config as shown in the following:

```
# start samba? ("yes" or "no")
# Windows 95 / NT  -  File- and Printservices
#
START_SMB="yes"
```

After changing the START_SMB setting, run the following command:

```
# SuSEconfig
```

To recompile the latest Samba package with the additional packages needed for integration into Windows domains you need to get the files samba-latest.tar.gz and samba-appliance-0.5-src.tar.gz. Start at the Samba home page:

```
http://www.samba.org
```

Then find a download mirror. The latest Samba package is in the top directory and the Samba appliance package is in the subdirectory named appliance. Copy those two files into the /usr/src directory. Before recompiling the two packages, check if you have installed the following packages:

- pam_devel

To successfully recompile the latest Samba (in our example we used version 2.2.1a), follow these steps:

1. Unpack the latest source:

```
# cd /usr/src
# tar xzf samba-latest.tar.gz
```

2. Start the configuration script for Makefile:

```
# cd samba-2.2.1a/source
# ./configure --prefix=/usr --libdir=/etc --with-privatedir=/etc \
--localstatedir=/var/log --with-codepagedir=/usr/lib/samba/codepages \
--sbindir=/usr/sbin --with-smbmount --with-automount --with-vfs \
--with-quotas --with-profile --with-msdfs --mandir=%{_mandir} \
--with-swatdir=/usr/lib/samba/swat \
--with-sambabook=/usr/lib/samba/swat/using_samba --with-pam \
--with-pam_smbpass
```

3. After the Makefile is created, compile the package:

```
# make LOCKDIR=/var/lock/samba
```

4. Stop the Samba daemon:

```
# /etc/init.d/smb stop
```

5. Install the newly compiled Samba:

```
# make install LOCKDIR=/var/log/samba
```

6. Start the newly installed Samba:

```
# /etc/init.d/smb start
```

To check if you are really running the new version, execute the following command; the output should be similar to the following:

```
# smbd -V
Version 2.2.1a
```

If you managed to do all the tasks just described, you are now running the latest version of Samba. Congratulations!

Now we need to compile utilities from the Samba appliance source code. To do this, follow these steps:

1. Unpack the latest source:

```
# cd /usr/src
# tar xzf samba-appliance-0.5-src.tar.gz
```

2. Start the configuration script for Makefile:

```
# cd samba-appliance-0.5/source/tng
# cp /usr/share/automake/config.sub .
# ./configure --prefix=/usr --libdir=/etc --with-privatedir=/etc \
--localstatedir=/var/log --with-codepagedir=/usr/lib/samba/codepages \
--sbindir=/usr/sbin --with-smbmount --with-automount --with-vfs \
--with-quotas --with-profile --with-msdfs --mandir=%{_mandir} \
--with-swatdir=/usr/lib/samba/swat -enable-static=yes -enable-shared=no\
--with-sambabook=/usr/lib/samba/swat/using_samba --with-pam \
--with-pam_smbpass
```

3. After the Makefile is created, compile the following packages: samedit, nsswitch and winbind:

```
# make nsswitch LOCKDIR=/var/lock/samba
# make bin/samedit LOCKDIR=/var/lock/samba
```

4. Copy the following files into your Samba bin directory (in SuSE 7.2, this is /usr/sbin):

```
# cp bin/windindd /usr/sbin
# cp bin/wbinfo /usr/sbin
# cp bin/sanmedit /usr/sbin
# cp nsswitch/libnss_winbind.so /lib/libnss_winbind.so.2
# cp nsswitch/pam_winbind.so /lib/security/pam_winbind.so
```

15.2.2 Joining the Active Directory

By using the tools we compiled, we now join our Linux Samba server to the Windows 2000 Active Directory. In our example, we installed Windows 2000 Server with the following setup:

- Windows 2000 server with SP2 installed
- Domain name: itso.ibm.com
- NETBIOS domain name: ITSO
- Host name: itsont1.itso.ibm.com
- IP Address: 9.12.0.60/255.255.255.0, gateway 9.12.0.1

The VM Linux Samba server has the following attributes:

- Samba 2.2.1a with Winbind extensions from Samba-appliance-0.5 version
- Host name: vmlinux8.itso.ibm.com
- IP Address: 9.12.6.72/255.255.255.0, gateway 9.12.6.75

To join the Windows 2000 Active Directory we use the **samedit** command. Follow these steps to join the Linux Samba server to the Windows 2000 Active Directory:

1. Modify the /etc/smb.conf file to include the following parameters:

   ```
   workgroup = ITSO
   security = DOMAIN
   password server = 9.12.0.60
   encrypt passwords = yes
   ```

2. Connect to the Windows 2000 server:

   ```
   # samedit -S ITSONT1 -W ITSO -U Administrator
   ```

   Type in the Administrator password; the output of the command should be similar to the following:

   ```
   # samedit -S ITSONT1 -W ITSO -U Administrator
   added interface ip=9.12.6.72 bcast=9.12.6.255 nmask=255.255.255.0
   Enter Password:
   Server: \\ITSONT1:    User:   Administrator   Domain: ITSO
   Connection:     1st session setup ok
   2nd session setup ok
   OK
   ```

   If the NETBIOS server name (in our example ITSONT1) could not be resolved into the IP address by your DNS server, you can add the NETBIOS server name into the /etc/lmhists file, which provides the NETBIOS-to-IP address resolution. You can see the example of the /etc/lmhosts file in Example 15-1.

Example 15-1 The /etc/lmhosts file

```
# This file provides the same function that the
# lmhosts file does for Windows.
```

```
# It provides another way to map netbios names to ip addresses.
# See the section on 'name resolve order' in the manual page to
# smb.conf for more information.

# Sample entry:
# 192.168.1.1 samba
9.12.0.60 ITSONT1
```

3. After successfully logging into the Windows 2000 server, use the following commands (Example 15-2) to add your Linux Samba server to the Active Directory:

Example 15-2 Adding a user to a domain

```
# samedit -S ITSONT1 -W ITSO -U Administrator
added interface ip=9.12.6.72 bcast=9.12.6.255 nmask=255.255.255.0
Enter Password:
Server: \\ITSONT1:     User:   Administrator   Domain: ITSO
Connection:     1st session setup ok
2nd session setup ok
OK
[ITSONT1\Administrator@ITSO]$ createuser VMLINUX8$ -L
createuser VMLINUX8$ -L

SAM Create Domain User
Domain: ITSO Name: vmlinux8$ ACB: [W          ]
Resetting Trust Account to insecure, initial, well-known value: "vmlinux8"
vmlinux8 can now be joined to the domain, which should
be done on a private, secure network as soon as possible
Create Domain User: OK
[ITSONT1\Administrator@ITSO]$
```

Note: In the case that you get message "Create Domain User: FAILED", this means that account was created, but it is disabled. To start using the account you need to reset and enable it in "Active Directory Users and Groups" tool on the Windows 2000 server. You will find this account under Computer accounts.

There is an alternative way to join the Windows 2000 Active Directory by using `smbpasswd`. To use this approach, follow these steps:

1. Create a computer account with the "Active Directory Users and Groups" tool on the Windows 2000 server, with the name vmlinux8.
2. Reset the account by right-clicking it and select "Reset Account."
3. On VM Linux server, join the domain by executing the command:

   ```
   # smbpasswd -j ITSO -r ITSONT1
   ```

15.2.3 Setting up Winbind

Before starting Winbind, add the following lines to the /etc/smb.conf file:

- winbind separator = +

  This is the separator for separating the Windows Domain name from the user name; for example, ITSO+Administrator.

- winbind uid = 10000-20000

  This means that Windows Domain users will be mapped to this range of UNIX user IDs.

- winbind gid = 10000-20000

  This means that Windows Domain groups will be mapped to this range of UNIX group IDs.

- winbind cache time = 15
- winbind enum users = yes
- winbind enum groups = yes
- template homedir = /home/%D%U

  This is the definition for the home directory of Windows Domain users.

- template shell = /bin/bash

  This is the login shell for Windows Domain users logging into the VM Linux Samba server. If you do not want to allow remote logins for Windows Domain users, use /bin/false.

Now you can start Winbind with the command:

```
# windind
```

You can test the Winbind functionality by listing users and groups from the Windows 2000 Active Directory.

Users:

```
# wbinfo -u
```

You will see output similar to that shown in Example 15-3.

Example 15-3 Windows Domain users

```
# wbinfo -u
ITSO+Administrator
ITSO+Guest
ITSO+IUSR_NF5500W2K
ITSO+ivo
ITSO+IWAM_NF5500W2K
```

```
ITSO+krbtgt
ITSO+mikem
ITSO+openldap
ITSO+tot67
ITSO+TsInternetUser
#
```

Users:

```
# wbinfo -g
```

You will see output similar to that shown in Example 15-4.

Example 15-4 Windows Domain groups

```
# wbinfo -g
ITSO+Domain Admins
ITSO+Domain Users
ITSO+Domain Guests
ITSO+Domain Computers
ITSO+Domain Controllers
ITSO+Cert Publishers
ITSO+Schema Admins
ITSO+Enterprise Admins
ITSO+Group Policy Creator Owners
ITSO+DnsUpdateProxy
#
```

15.2.4 Setting up /etc/nsswitch.conf

To fully incorporate Winbind functionality into the Linux security layout, you should also modify /etc/nsswitch.conf with the following entries:

- passwd: files winbind
- groups: files winbind

You can check the setup of the nsswitch configuration with the `id` command as shown in Example 15-5.

Example 15-5 Checking the id data for Windows user name

```
# id ITSO+ivo
uid=10000(ITSO+ivo) gid=10000(ITSO+Domain Users) groups=10000(ITSO+Domain
Users)
#
```

With this setup you can now share a directory from your Samba server and assign the Windows Domain group permission to it. In this case, Windows Domain users can use Samba shared directories. In our example, we created the directory /sambashares/windowsshare with the commands:

```
# mkdir /sambashares
# mkdir /sambashares/windowsshare
```

Then we assigned full permissions for the Domain Users group from ITSO Domain as is shown in Example 15-6.

Example 15-6 Setting the permissions for the Windows Domain Users

```
# chgrp "ITSO+Domain Users" windowsshare
# ls -l
total 12
drwxr-xr-x    3 root     root         4096 Aug  3 12:46 .
drwxr-xr-x   19 root     root         4096 Aug  3 12:46 ..
drwxr-xr-x    2 root     ITSO+Dom     4096 Aug  3 12:47 windowsshare
# chmod 770 windowsshare
# ls -l
total 12
drwxr-xr-x    3 root     root         4096 Aug  3 12:46 .
drwxr-xr-x   19 root     root         4096 Aug  3 12:46 ..
drwxrwx---    2 root     ITSO+Dom     4096 Aug  3 12:47 windowsshare
#
```

To share /sambashares/windowsshare, add the following lines to the /etc/smb.conf file as shown in Example 15-7.

Example 15-7 Defining a windows share

```
[windowsshare]
        comment = Share for ITSO Domain Users group
        path = /sambashares/windowsshare
        read only = No
```

15.2.5 Setting up the PAM authentication

With the Winbind package you also get the Pluggable Authentication Modules (PAM) for Winbind. This module allows the setup of your Linux server to authenticate the users against the Windows Domain controller. This means that you can log on to the Linux server running Winbind with the username from the Windows Active Directory. In Figure 15-1 on page 357 we show how this process is done.

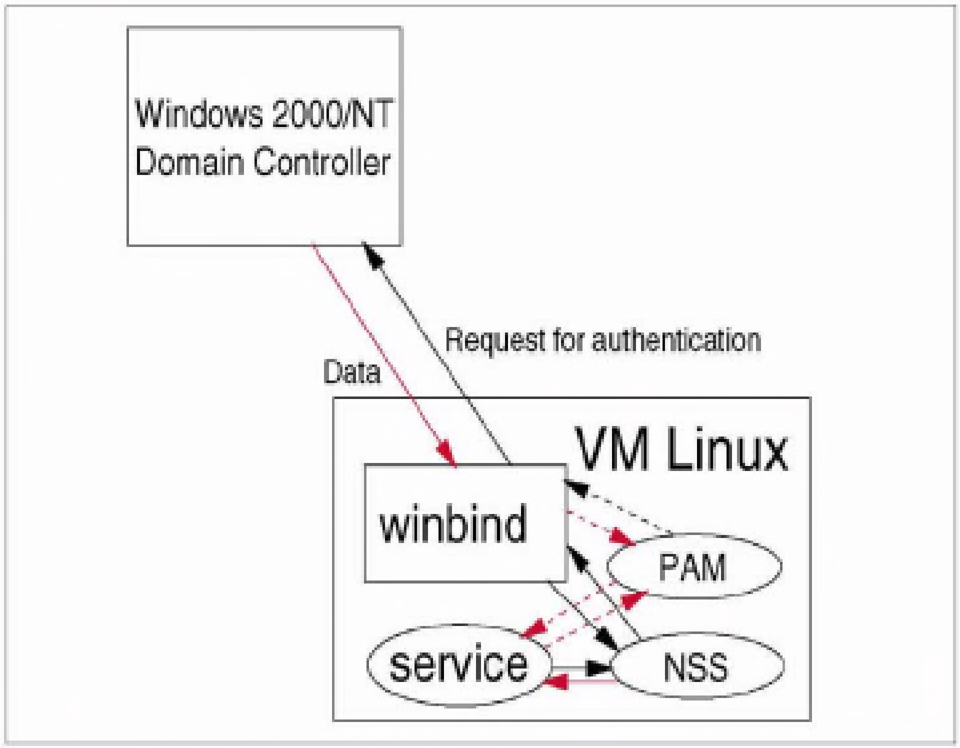


Figure 15-1 Using Winbind for authentication

As you can see in Figure 15-1, the process of authentication is as follows:

1. Service gets the request.
2. Because service is PAM-aware, it passes the request handling to the underlying PAM modules.
3. The PAM module for Winbind passes the request to the Winbind daemon.
4. The Winbind daemon passes the request to the Windows Domain controller.
5. If the user exists in the Windows Active Directory database, the information is passed back to the Winbind daemon.
6. The Winbind daemon then passes this information back to the PAM module.
7. PAM then informs service that this user is allowed to use it.

If the user does not exist, this information comes back to the service and service denies the access.

To achieve this, modify the file for the service which you want to access with this user ID. In SuSE 7.2, these files are located in the /etc/pam.d directory as shown in the following:

```
# ls -l /etc/pam.d/
total 92
drwxr-xr-x    2 root     root         4096 Aug  2 17:50 .
```

```
drwxr-xr-x   38 root     root         4096 Aug  7 09:52 ..
-rw-r--r--    1 root     root          305 Jun 17 22:23 chfn
-rw-r--r--    1 root     root          305 Jun 17 22:23 chsh
-rw-r--r--    1 root     root          631 Jun 18 01:03 ftp
-rw-r--r--    1 root     root           95 Jun 18 06:30 imap
-rw-r--r--    1 root     root          749 Aug  3 11:37 login
-rw-r--r--    1 root     root          623 Aug  2 17:50 login.org
-rw-r--r--    1 root     root          259 Jun 18 01:04 netatalk.pamd
-rw-r--r--    1 root     root          517 Jun 17 20:33 other
-rw-r--r--    1 root     root          305 Jun 17 22:23 passwd
-rw-r--r--    1 root     root           95 Jun 18 06:30 pop
-rw-r--r--    1 root     root          311 Jun 18 01:23 ppp
-rw-r--r--    1 root     root          322 Jun 18 12:34 proftpd
-rw-r--r--    1 root     root          263 Jun 17 22:11 rexec
-rw-r--r--    1 root     root          455 Jul 18 09:38 rlogin
-rw-r--r--    1 root     root          292 Jun 17 22:11 rsh
-rw-r--r--    1 root     root          216 Aug  3 13:59 samba
-rw-r--r--    1 root     root          439 Aug  2 17:50 sshd
-rw-r--r--    1 root     root          352 Jun 17 22:36 su
-rw-r--r--    1 root     root          108 Jun 17 23:05 sul
-rw-r--r--    1 root     root           60 Jun 17 23:16 sudo
-rw-r--r--    1 root     root          318 Jun 17 23:39 xdm
```

1. To allow use of the Windows Active Directory user names to log on to your Linux system, modify the /etc/pam.d/logon file as shown in the following:

```
# cat /etc/pam.d/login
#%PAM-1.0
auth     required       /lib/security/pam_nologin.so
auth     required       /lib/security/pam_env.so
auth     required       /lib/security/pam_mail.so
auth     sufficient     /lib/security/pam_winbind.so
account  required       /lib/security/pam_unix.so       audit
account  required       /lib/security/pam_winbind.so
password required       /lib/security/pam_pwcheck.so    nullok
password required       /lib/security/pam_unix.so       nullok \
                         use_first_pass use_authtok
password required       /lib/security/pam_winbind.so
session  required       /lib/security/pam_unix.so
session  required       /lib/security/pam_limits.so
```

Now you can try to log on to the Linux server running the Winbind daemon using the Windows Active Directory account. The following was our successful logon:

```
# telnet vmlinux8
Trying 9.12.6.72...
Connected to vmlinux8.
Escape character is '^]'.
Welcome to SuSE Linux 7.2 (S390) - Kernel 2.4.5 (2).
```

```
vmlinux8 login: ITSO+ivo
Password:
Last login: Fri Aug  3 17:15:32 from localhost
Have a lot of fun...
No directory /home/ITSO/ivo!
Logging in with home = "/".
```

15.3 Replacing Microsoft Exchange Server

An Internet Services Provider has to offer at least a basic mail serving functionality to its customers. Once messages have been delivered into the mailbox of the recipient's message store, the recipient needs message access methods to retrieve and work with the messages. These mail servers usually are based on the simple Post Office Protocol, version 3 (POP3). POP3 treats the message store as a single in-box. The user agent can retrieve and delete messages from this in-box. Once messages are retrieved and deleted from the POP3 server, it is the user agent's responsibility, if necessary, to retain messages in some local message store. While a POP3 client can leave mail on the server (by not deleting it), the POP3 protocol lacks mechanisms to categorize, file, or search the mail, so the POP3 server message store can quickly become unmanageable. Also, most large-scale POP3 servers enforce a storage limit, refusing to accept new mail for a user whose limit has been exceeded.

Thus, the POP3 model strongly encourages the complete transfer of mail to the client, where a well-designed client can provide many more capabilities to the user. This has the advantage that the communication with the server is simple, but it has the disadvantage that the user cannot conveniently use more than one computer to read mail: the mail remains on whichever computer the user reads it.

Enterprise customers, which are thinking about consolidating their servers on Linux on zSeries, usually are using more complex mail serving applications, based on or at least supporting the Internet Mail Access Protocol, version 4 (IMAP4). This newer access protocol defines a much richer message store, allowing mail to be stored in multiple mailboxes. A rich set of message and mailbox manipulation functions exists. While a POP3 message can be handled only as a single block, IMAP4 allows access to individual parts of the message. Provisions exist to allow message stores to be replicated to a local store (and resynchronized later) for the mobile user. The IMAP4 model, in contrast to the POP3 model, involves storing mail on the server, where it may be accessed by any client, and using the client's storage only for caching messages for efficiency or for traveling.

The standards described so far allow messages to be transmitted through the Internet, but only "in the clear". Features such as authentication and encryption are needed to make message transmission secure. Authentication allows messages to be signed, so the recipient can confirm that the sender is the person claimed. Encryption allows data to be sent in such a fashion that only a recipient with a key can decrypt the data. For e-mail, directory services are needed to access user information, such as a given user's e-mail address. Lightweight Directory Access Protocol (LDAP), is the standard that describes how to access directory data. Directory services will play an even greater role for storing and accessing public keys to enable secure messaging.

Earlier e-mail systems were developed for a homogeneous group of users on a single network. They typically have a large set of features allowing the creation and manipulation of compound documents. Their delivery systems often support guaranteed deliveries and receipt notifications. Additional integrated functions for calendars and schedules are not uncommon. On the other hand, they often do not scale well to large user communities, because they were developed for a small, homogeneous domain. They cannot exchange mail with other systems except through specially designed gateways, which lose information in the process of converting between mail formats.

For these reasons, widely accepted groupware and e-mail systems (such as Microsoft Exchange or Lotus Notes) are now designed to also support the common Internet standards like POP3 and IMAP4. Of course, the rich document layout offered, for example, by Lotus Notes is lost when sending an e-mail to a server that is not of the same kind, and advanced functions like calendars can only be used between two Lotus Notes servers. But the pure mailing functionality is still sustained, even when communicating with simple POP3 servers.

So a mail serving application on a Linux server running on zSeries is expected to at least support the IMAP4 protocol, and to communicate with Lotus Notes or Exchange clients. Additionally it is preferred that a functionality comparable to those of the major groupware applications is offered. Various applications are available that meet these requirements, some commercial, some open source. We will have a closer look at one example of each group.

Naturally, from an IBM point of view, Lotus Domino is the recommended application for mail serving and collaboration. However, although there is an Intel-based Linux version, until now no Linux version for zSeries is available, so we will not discuss this software now.

15.3.1 The Bynari Insight Server

In April 2001, Bynari Inc. announced that its messaging and collaboration product, Insight Server, runs on the IBM zSeries and S/390 under the Linux operating system ports for those platforms. See also:

`http://www.bynari.net`

Insight is a commercial UNIX/Linux collaboration tool for enterprises. It was developed due to the computer industry's need of a UNIX/Linux client for MS Exchange with Outlook functionality. Insight bridges the gap between Outlook users and UNIX/Linux workstations, allowing them to collaborate in a seamless fashion. It enables a number of messaging protocols to communicate so that various platforms can work together—the way systems designers intended.

Bynari began offering Insight Server as an entry level platform for small to medium size businesses wanting functionality found in Microsoft Exchange. Soon, demand for larger user populations began, asking for more robust hardware platforms. Having already scaled Insight Server for Compaq's non-stop clusters, Bynari's developers began looking at IBM mainframes as the next step on its product roadmap.

Originally designed as a platform for the large populations of Microsoft Outlook users not connected to Exchange, Insight Server has also proven its value as a cost effective replacement for Exchange. Insight Server supports IMAP, POP3, SMTP, and LDAP protocols for numerous e-mail clients including Outlook, Outlook Express, Eudora and Netscape Messenger. Additionally, Insight Server offers meeting management, shared calendars and folders, and other collaboration functions for workgroup oriented clients including Microsoft Outlook and Bynari's own Insight client.

15.3.2 The Cyrus IMAP server

The Cyrus IMAP server, a scalable enterprise mail system designed for use in small to large enterprise environments using standards-based technologies, is developed by the Cyrus Project at Carnegie Mellon University. It provides access to personal mail and system-wide bulletin boards through the IMAP protocol. Documentation is provided at :

`http://asg2.web.cmu.edu/cyrus`

The software can be downloaded from:

`ftp://ftp.andrew.cmu.edu/pub/cyrus-mail`

A full Cyrus IMAP implementation allows a seamless mail and bulletin board environment to be set up across multiple servers. It differs from other IMAP server implementations in that it is run on "sealed" servers, where users are not normally permitted to log in. The mailbox database is stored in parts of the file system that are private to the Cyrus IMAP system. All user access to mail is through software using the IMAP, POP3, or KPOP protocols. The private mailbox database design gives the server large advantages in efficiency, scalability, and in the ease of administration. Multiple concurrent read/write connections to the same mailbox are permitted. The server supports access control lists on mailboxes and storage quotas on mailbox hierarchies.

A brief description of how to install the Cyrus IMAP server in a Linux environment, together with Sendmail and OpenLDAP, is provided in "The Exchange Server Replacement HOWTO" by Curt Johnson, and in the "Cyrus IMAP HOWTO" by Aurora Skarra-Gallagher, both of which can be found at:

http://www.linuxdoc.org

15.4 Using AFS in an enterprise environment

In this section we talk about the Andrew File System (AFS) and how it can participate and be used in an IT VM Linux environment. We will also outline the installation instructions for setting up an AFS server and client in a VM Linux environment. In our test environment we used OpenAFS, which was donated by IBM from its version of commercial AFS implementation. IBM branched the source of the AFS product, and made a copy of the source available for community development and maintenance.

15.4.1 What is AFS

AFS makes it easy for people to work together on the same files, no matter where the files are located. AFS users do not have to know which machine is storing a file, and administrators can move files from machine to machine without interrupting user access. Users always identify a file by the same path name and AFS finds the correct file automatically, just as happens in the local file system on a single machine. While AFS makes file sharing easy, it does not compromise the security of the shared files. It provides a sophisticated protection scheme.

AFS uses a client/server computing model. In client/server computing, there are two types of machines: Server machines store data and perform services for client machines; client machines perform computations for users and access data and services provided by server machines. Some machines act as both client and server. In most cases, you work on a client machine, accessing files stored on a file server machine.

15.4.2 Building OpenAFS

You can get the latest OpenAFS from:

http://www.openafs.org

In our test environment we used OpenAFS version 1.1.1. After downloading the package, you can unwind it with the command:

```
# tar xzf openafs-1.1.1-src.tar.bz2
```

Then move to the source directory and run the configuration program with the commands:

```
# cd openafs-1.1.1
# mv config.sub config.sub.org
# cp /usr/share/automake/config.sub .
# ./configure --with-afs-sysname=s390_linux24 \
    --with-linux-kernel-headers=/usr/src/linux
```

Important: You have to install the kernel source before compiling. In our example we installed the source in /usr/src/linux.

Compile the package with the command:

```
# make
```

After compilation, all the binary and configuration files reside in the openafs-1.1.1/s390_linux24 directory. For a minimal configuration of your AFS system, you need to install at least one server. This server will then act as:

- File server machine
- Database server machine
- Binary distribution machine
- System control machine

15.4.3 Installing OpenAFS

To install and configure OpenAFS on the VMLinux server, follow the following steps.

Creating AFS directories

Create the AFS directories with the commands:

```
# mkdir /usr/afs
# mkdir /usr/vice
# mkdir /usr/vice/etc
```

Loading AFS modules into the kernel

1. Change to the directory as indicated (assuming that /usr/src/openafs-1.1.1 is the source code directory):

   ```
   # cd /usr/src/openafs-1.1.1/s390_linux24/dest/root.client/usr/vice/etc
   ```

2. Copy the AFS kernel library files to the local /usr/vice/etc/modload directory with the command:

   ```
   # cp -rp modload /usr/vice/etc
   ```

3. Copy the initialization scripts to the local directory for initialization files (in our example, /etc/init.d):

   ```
   # cp -p afs.rc /etc/init.d/afs
   ```

4. Run the AFS initialization script to load the AFS extensions into the kernel:

   ```
   # /etc/init.d/afs start
   ```

Configuring server partitions

Each AFS file server must have at least one partition or logical volume dedicated to storing AFS volumes. Each partition is mounted on /vicepxx, where xx is one or two lowercase letters. The /vicepxx directories must reside in the machine's root directory. In our example, we selected the /dev/dasd/0209/part1 as the partition for AFS file serving. Follow these steps to configure the partitions:

1. Create the mount directory:

   ```
   # mkdir /vicepa
   ```

2. Create the file system on the partition, add an entry to the /etc/fstab file, and mount the partition with the command:

   ```
   # mke2fs /dev/dasd/0209/part1 -b 4096
   ```

   An example of the /etc/fstab file after adding the mounting entry is as follows:

   ```
   # cat /etc/fstab
   /dev/dasd/0204/part1    swap                      swap     defaults   0   0
   /dev/dasd/0201/part1    /                         ext2     defaults   1   1
   /dev/dasd/0209/part1    /vicepa                   ext2     defaults   0   2
   proc            /proc                     proc            defaults   0   0
   # mount /vicepa
   ```

Enabling AFS Login

AFS also provides the PAM authentication for PAM-capable clients. The following steps show you how to set up the configuration for each service for which you wish to use AFS authentication. You can skip this section if you do not want to use client functionality on this server.

1. Copy the PAM libraries into the /lib/security directory:

If you plan to use the AFS Authentication Server (**kaserver** process):

```
# cd /lib/security
# cp /usr/src/openafs-1.1.1/s390_linux24/dest/lib/pam_afs.so.1 .
# ln -s pam_afs.so.1 pam_afs.so
```

If you plan to use Kerberos implementation of AFS authentication:

```
# cd /lib/security
# cp /usr/src/openafs-1.1.1/s390_linux24/dest/lib/pam_afs.krb.so.1 .
# ln -s pam_afs.krb.so.1 pam_afs.krb.so
```

2. For each service from /etc/pam.d, put the following line into the auth section:

```
auth sufficient /lib/security/pam_afs.so try_first_pass ignore_root
```

Insert this line just after the entries that impose conditions under which you want the service to fail. The ignore_root parameter means that the AFS PAM module will ignore the local superuser root and also any user with UID 0. Following is our /etc/pam.d/login file for login service:

```
# cat /etc/pam.d/login
#%PAM-1.0
auth     required   /lib/security/pam_nologin.so
auth     required   /lib/security/pam_env.so
auth     required   /lib/security/pam_mail.so
auth     sufficient /lib/security/pam_afs.so    try_first_pass ignore_root
auth     requisite  /lib/security/pam_unix.so   nullok #set_secrpc
account  required   /lib/security/pam_unix.so
password required   /lib/security/pam_pwcheck.so nullok
password required   /lib/security/pam_unix.so nullok use_first_pass \
                      use_authtok
session  required   /lib/security/pam_unix.so     none # debug or trace
session  required   /lib/security/pam_limits.so
```

Starting the BOS Server

The BOS (Basic OverSeer) Server is used to monitor and control other AFS server processes on its server. Follow these steps to install and start the BOS Server:

1. Copy the files:

```
# cd /usr/src/openafs-1.1.1/s390_linux24/dest/root.server/usr/afs
# cp -rp * /usr/afs
```

2. Start the BOS Server, include the *-noauth* flag to disable authorization checking (authentication is not running yet):

```
# /usr/afs/bin/bosserver -noauth &
```

3. Verify that the BOS server created /usr/vice/etc/ThisCell and /usr/vice/etc/CellServDB as symbolic links to the corresponding files in the /usr/afs/etc directory:

```
# ls -l /usr/vice/etc
```

If the links do not exist, create them with the commands:

```
# cd /usr/vice/etc
# ln -s /usr/afs/etc/ThisCell ThisCell
# ln -s /usr/afs/etc/CellServDB CellServDB
```

Defining cell name and membership for the server process

Here we assign the cell name. You should know that changing the name is very difficult, so you should plan the name carefully. Usually, the cell name is the same as the name of the Internet domain you are using. There are two important restrictions: the name cannot include uppercase letters or more than 64 characters.

Use the following steps to set the cell name:

1. Change to the directory with the AFS programs with the command:

   ```
   # cd /usr/afs/bin
   ```

2. With the **`bos setcellname`** command, set the cell name:

   ```
   # ./bos setcellname vmlinux8.itso.ibm.com itso.ibm.com -noauth
   ```

   As you can see, we issued **`bos setcellname`** with two parameters:

   – machine name = vmlinux8.itso.ibm.com
   – cell name = itso.ibm.com

3. Verify that the server you are installing is now registered as the cell's first database server:

   ```
   # ./bos listhosts vmlinux8.itso.ibm.com -noauth
   Cell name is itso.ibm.com
   Host 1 is vmlinux8
   ```

Starting the database server process

Now we create four database server processes in the /usr/afs/local/BosConfig file and start them running. They run on the database server machine only.

- The Authentication Server (the **`kaserver`** process) maintains the Authentication Database.
- The Backup Server (the **`busserver`** process) maintains the Backup Database.
- The Protection Server (the **`ptserver`** process) maintains the Protection Database.
- The Volume Location (VL) Server (the **`vlserver`** process) maintains the Volume Location Database (VLDB).

Note: AFS's authentication and authorization software is based on algorithms and other procedures known as *Kerberos*, as originally developed by Project Athena at the Massachusetts Institute of Technology. Some cells choose to replace the AFS Authentication Server and other security-related protocols with Kerberos as obtained directly from Project Athena or other sources. If you wish to do this, contact the AFS Product Support group now to learn about necessary modifications to the installation.

Follow these steps to create these server processes (we assume that you are in the /usr/afs/directory):

1. Start the Authentication Server:

```
# ./bos create vmlinux8.itso.ibm.com kaserver simple \
    /usr/afs/bin/kaserver -cell itso.ibm.com -noauth
```

2. Start the Backup Server:

```
# ./bos create vmlinux8.itso.ibm.com buserver simple \
    /usr/afs/bin/buserver -cell itso.ibm.com -noauth
```

3. Start the Protection Server:

```
# ./bos create vmlinux8.itso.ibm.com ptserver simple \
    /usr/afs/bin/ptserver -cell itso.ibm.com -noauth
```

4. Start the VL Server:

```
# ./bos create vmlinux8.itso.ibm.com vlserver simple \
    /usr/afs/bin/vlserver -cell itso.ibm.com -noauth
```

Initializing cell security

Now we initialize the cell's security mechanisms. We begin by creating two initial entries in the Authentication Database:

- A generic administrative (in our example we call it admin)

  After installation, all administrators can use this account or you can create a separate account for each of them.

- The entry for the AFS server process, called afs

  There are no logons under this user ID, but Authentication Server's Ticket Granting (TGS) module uses the associated key to encrypt the server tickets that it grants to AFS clients.

In the following steps we show how to create these two entries. Keep in mind that this process does not configure all of the security mechanisms related to the AFS Backup System. To do this you need to refer to the *IBM AFS Administration Guide* on the Web at:

http://oss.software.ibm.com/developerworks/opensource/afs/docs.html

1. Change to the directory with the AFS programs with the command:

```
# cd /usr/afs/bin
```

2. Enter the kas interactive mode with the -noauth option, because the server is in no-authorization checking mode. Then create admin and afs entries:

```
# ./kas -cell itso.ibm.com -noauth
ka> create afs
initial_password:
Verifying, please re-enter initial_password:
ka> create admin
initial_password:
Verifying, please re-enter initial_password:
```

3. Examine the afs entry checksum:

```
ka> examine afs

User data for afs
key (0) cksum is 824768179, last cpw: Wed Aug  8 09:42:29 2001
password will never expire.
An unlimited number of unsuccessful authentications is permitted.
entry never expires.  Max ticket lifetime 100.00 hours.
last mod on Wed Aug  8 09:42:29 2001 by <none>
permit password reuse
```

4. Turn on the ADMIN flag for the admin entry and then examine the entry to verify that the admin flag appears:

```
ka> setfields admin -flags admin
ka> examine admin
User data for admin (ADMIN) - you can see the ADMIN flag is present
key (0) cksum is 824768179, last cpw: Wed Aug  8 09:42:39 2001
password will never expire.
An unlimited number of unsuccessful authentications is permitted.
entry never expires.  Max ticket lifetime 25.00 hours.
last mod on Wed Aug  8 09:47:32 2001 by <none>
permit password reuse
```

5. Quit the kas server:

```
ka> quit
```

6. Now we need to add admin to the /usr/afs/etc/UserList file, to enable admin to issue privileged **bos** and **vos** commands:

```
# ./bos adduser vmlinux8.itso.ibm.com admin -cell itso.ibm.com -noauth
```

7. Next define the AFS server encryption key in /usr/afs/etc/KeyFile:

   ```
   # ./bos addkey vmlinux8.itso.ibm.com -kvno 0 -cell itso.ibm.com -noauth
   input key:
   Retype input key:
   ```

   For the input key, type in the password you used for creating the `afs` entry in step 1.

8. Verify that the checksum for the new key in the Keyfile is the same as the checksum defined for the key Authentication Database's `afs` entry that we displayed in step 2.:

   ```
   # ./bos listkey vmlinux8.itso.ibm.com -cell itso.ibm.com -noauth
   key 0 has cksum 824768179
   Keys last changed on Wed Aug  8 12:03:59 2001.
   All done.
   ```

   As you can see in our example, the keys are the same.

9. Now we need to create the Protection Database Entry for the admin user. By default, the Protection Server assigns AFS UID 1 to the admin user, because it is the first entry you are creating. If the local password file (/etc/passwd or equivalent) already has an entry for admin that assigns it a UNIX UID other than 1, it is best to use the `-id` argument on the **`pts createuser`** command to make the new AFS UID match the existing UNIX UID. Otherwise, it is best to accept the default. In our example we already have the admin user on the system with a UID of 501:

   ```
   # ./pts createuser -name admin -cell itso.ibm.com -id 501 -noauth
   User admin has id 501
   ```

10. Next add the admin user to the system:administrators group and then check if this was successful:

    ```
    # ./pts adduser admin system:administrators -cell itso.ibm.com -noauth
    # ./pts membership admin -cell itso.ibm.com -noauth
    Groups admin (id: 1) is a member of:
    system:administrators
    ```

11. Now we need to restart the bos server with the `-all` flag to restart the database server processes, so that they start using the new server encryption key:

    ```
    # ./bos restart vmlinux8.itso.ibm.com -all -cell itso.ibm.com -noauth
    ```

    You can check whether the AFS server processes are running with the command:

    ```
    # ps ax | grep afs
     9050 ?        S      0:00 /usr/afs/bin/bosserver -noauth
    11419 ?        S      0:00 /usr/afs/bin/kaserver
    11420 ?        S      0:00 /usr/afs/bin/buserver
    11421 ?        S      0:00 /usr/afs/bin/ptserver
    ```

```
11422 ?        S      0:00 /usr/afs/bin/vlserver
```

Starting the file server, volume server, and salvager

To start the **fs** process, which consists of the File Server, Volume Server, and the Salvager (**fileserver, volserver** and **salvager** processes), follow these steps:

1. Change to the directory with the AFS programs with the command:

   ```
   # cd /usr/afs/bin
   ```

2. Create the **fs** process:

   ```
   # ./bos create vmlinux8.itso.ibm.com fs fs /usr/afs/bin/fileserver \
   /usr/afs/bin/volserver /usr/afs/bin/salvager -cell itso.ibm.com -noauth
   ```

   You can verify that the **fs** process has started successfully with the command:

   ```
   # ./bos status vmlinux8.itso.ibm.com fs -long -noauth
   Instance fs, (type is fs) currently running normally.
   Auxiliary status is: file server running.
   Process last started at Wed Aug  8 12:39:05 2001 (2 proc starts)
   Command 1 is '/usr/afs/bin/fileserver'
   Command 2 is '/usr/afs/bin/volserver'
   Command 3 is '/usr/afs/bin/salvager'
   ```

   You can see that in our example the servers are running with no problems.

3. Because this is the first AFS file server in our cell, we need to create the first AFS volume, root.afs:

   ```
   # ./vos create vmlinux8.itso.ibm.com /vicepa root.afs -cell itso.ibm.com \
      -noauth
   Volume 536870912 created on partition /vicepa of vmlinux8.itso.ibm.com
   ```

   As you can see, we used our /vicepa partition for the root.afs AFS volume.

Starting the server portion of the update process

Start the server portion of the Update Server (the **upserver** process) to distribute the contents of directories on this machine to other server machines in the cell. It becomes active when you configure the client portion of the Update Server on additional server machines.

Distributing the contents of its /usr/afs/etc directory makes this server the cell's system control machine. The other servers in the cell run the **upclientetc** process (an instance of the client portion of the Update Server) to retrieve the configuration files. Use the `-crypt` argument to the upserver initialization command to specify that the Update Server distributes the contents of the /usr/afs/etc directory only in encrypted form. Several of the files in the directory, particularly the KeyFile file, are crucial to cell security and so must never cross the network unencrypted.

(You can choose not to configure a system control server, in which case you must update the configuration files in each server's /usr/afs/etc directory individually. The **bos** commands used for this purpose also encrypt data before sending it across the network.)

Distributing the contents of its /usr/afs/bin directory to other servers of its system type makes this server a binary distribution machine. The other servers of its system type run the **upclientbin** process (an instance of the client portion of the Update Server) to retrieve the binaries.

The binaries in the /usr/afs/bin directory are not sensitive, so it is not necessary to encrypt them before transfer across the network. Include the `-clear` argument to the upserver initialization command to specify that the Update Server distributes the contents of the /usr/afs/bin directory in unencrypted form unless an upclientbin process requests encrypted transfer.

Note that the server and client portions of the Update Server always mutually authenticate with one another, regardless of whether you use the `-clear` or `-crypt` arguments. This protects their communications from eavesdropping to some degree.

Start the Update Server process with the commands:

```
# cd /usr/afs/bin
# ./bos create vmlinux8.itso.ibm.com upserver simple \
    "/usr/afs/bin/upserver -crypt /usr/afs/etc -clear /usr/afs/bin" \
    -cell itso.ibm.com -noauth
```

Starting the Controller for NTPD

Keeping the clocks on all server and client machines in your cell synchronized is crucial to several functions, and in particular to the correct operation of AFS's distributed database technology, Ubik. The time skew can disturb Ubik's performance and cause service outages in your cell.

The AFS distribution includes a version of the Network Time Protocol Daemon (NTPD) for synchronizing the clocks on server machines. If a time synchronization program is not already running on the machine, then in this section you start the **runntp** process to configure NTPD for use with AFS.

Note: Do not run **runntp** process on top of another NTPD or another time synchronization protocol is already running on the machine.

In our example we did not have another time synchronization protocol running, so we decided to use **runntp** from the AFS server:

1. Create the **runntp** process:

If you have a reliable network connection to an outside time source:

```
# ./bos create vmlinux8.itso.ibm.com runntp \
simple "/usr/afs/bin/runntp hostname+" -cell itso.ibm.com -noauth
```

If you plan to use the local clock as the time source (as we did in our example):

```
# ./bos create vmlinux8.itso.ibm.com runntp \
simple "/usr/afs/bin/runntp -localclock" -cell itso.ibm.com -noauth
```

If you have a connection to an outside time source, but it is not reliable:

```
# ./bos create vmlinux8.itso.ibm.com runntp \
    simple "/usr/afs/bin/runntp -localclock hostname+" \
    -cell itso.ibm.com -noauth
```

Note: In the OpenAFS version we used, the NTP package was not compiled, because it is obsolete. The clients are getting time from the AFS servers anyway. You should install the NTP server on the AFS server. It is available on the Web at:

http://www.eecis.udel.edu/~ntp

Or use any other NTP server. In SuSE 7.2 they have included the XNTP package, which can be used for this purpose.

In our example we deleted the definition for the **runntp** process with the command:

```
# ./bos delete vmlinux8.itso.ibm.com runntp
```

15.4.4 Installing client functionality

The server which we just installed is the AFS file server, database server, system control server, and binary distribution server. Now we need to make this server also the client machine.

Copying client files to the local disk

Before installing and configuring the AFS client, we need to copy the necessary files from the build directory:

```
# cd /usr/src/openafs-1.1.1/s390_linux24/dest/root.client/usr/vice/etc
# cp  -p * /usr/vice/etc
cp: omitting directory `C'
cp: omitting directory `modload'
# cp -rp C /usr/vice/etc
```

Defining cell membership for client processes

Every AFS client machine has a copy of the /usr/vice/etc/ThisCell file on its local disk to define the machine's cell membership for the AFS client programs that run on it. The ThisCell file you created in the /usr/afs/etc directory (in "Defining cell name and membership for the server process" on page 366) is used only by server processes.

Among other functions, the ThisCell file on a client machine determines the following:

- The cell in which users authenticate when they log onto the machine, assuming it is using an AFS-modified login utility
- The cell in which users authenticate by default when they issue the `klog` command
- The cell membership of the AFS server processes that the AFS command interpreters on this machine contact by default

//TODO: Should "To define this" be "To define cell membership"?

To define this, remove the symbolic link created in "Starting the BOS Server" on page 365 and create the new ThisCell file by copying the server copy of this file from /usr/afs/etc/ThisCell. With this you define the same cell for both server and client processes, which gives you the most consistent AFS performance:

```
# cd /usr/vice/etc
# rm ThisCell
# cp /usr/afs/etc/ThisCell ThisCell
```

Creating the client CellServDB file

The /usr/vice/etc/CellServDB file on a client machine's local disk lists the database server machines for each cell that the local Cache Manager can contact. If there is no entry in the file for a cell, or if the list of database server machines is wrong, then users working on this machine cannot access the cell.

Because the **afsd** program initializes the Cache Manager, it copies the contents of the CellServDB file into kernel memory. The Cache Manager always consults the list in kernel memory rather than the CellServDB file itself. Between reboots of the machine, you can use the `fs newcell` command to update the list in kernel memory directly.

Follow these steps to create the CellServDB file:

1. Remove the symbolic link created in "Starting the BOS Server" on page 365:

```
# cd /usr/vice/etc/
# rm CellServDB
```

2. Create CellServDB with the local cell entry and display it to verify the file:

```
# cat /usr/afs/etc/CellServDB > CellServDB
# cat CellServDB
>itso.ibm.com   #Cell name
9.12.6.72    #vmlinux8
```

Configuring the cache

The Cache Manager uses a cache on the local disk or in machine memory to store local copies of files fetched from file server machines. As the `afsd` program initializes the Cache Manager, it sets basic cache configuration parameters according to definitions in the local /usr/vice/etc/cacheinfo file.

The file has three fields:

1. The first field names the local directory on which to mount the AFS file space. The conventional location is the /afs directory.
2. The second field defines the local disk directory to use for the disk cache. The conventional location is the /usr/vice/cache directory, but you can specify an alternate directory if another partition has more space available. There must always be a value in this field, but the Cache Manager ignores it if the machine uses a memory cache.
3. The third field specifies the number of kilobyte (1024 byte) blocks to allocate for the cache.

The values you define must meet the following requirements:

- On a machine using a disk cache, the Cache Manager expects always to be able to use the amount of space specified in the third field. Failure to meet this requirement can cause serious problems, some of which can be repaired only by rebooting. You must prevent non-AFS processes from filling up the cache partition. The simplest way is to devote a partition to the cache exclusively.
- The amount of space available in memory or on the partition housing the disk cache directory imposes an absolute limit on cache size.
- The maximum supported cache size can vary in each AFS release; see the Release Notes for the current version.
- For a disk cache, you cannot specify a value in the third field that exceeds 95% of the space available on the partition mounted at the directory named in the second field. If you violate this restriction, the `afsd` program exits without starting the Cache Manager and prints an appropriate message on the standard output stream. A value of 90% is more appropriate on most machines. Some operating systems (such as AIX) do not automatically reserve some space to prevent the partition from filling completely; for them, a smaller value (say, 80% to 85% of the space available) is more appropriate.

- For a memory cache, you must leave enough memory for other processes and applications to run. If you try to allocate more memory than is actually available, the **afsd** program exits without initializing the Cache Manager and produces the following message on the standard output stream:

  ```
  afsd: memCache allocation failure at number KB
  ```

  The number value is how many kilobytes were allocated just before the failure, and so indicates the approximate amount of memory available.

Tip: Disk caches smaller than 10 MB do not perform well, and also memory caches smaller than 5 MB do not perform well. The cache size depends on the number of users using the client machine.

Configuring a disk cache

To configure the disk cache, perform the following steps:

1. Create a local directory for caching with the command:

   ```
   # mkdir /usr/vice/cache
   ```

2. Create the cacheinfo file; in our example, we define a 50 MB disk cache:

   ```
   # echo "/afs:/usr/vice/cache:50000" > /usr/vice/etc/cacheinfo
   ```

Configuring a memory cache

To configure the memory cache, do the following:

Create the cacheinfo file; in our example, we create a 25 MB memory cache:

```
# echo "/afs:/usr/vice/cache:25000" > /usr/vice/etc/cacheinfo
```

Configuring the Cache Manager

By convention, the Cache Manager mounts the AFS file space on the local /afs directory. The **afsd** program sets several cache configuration parameters as it initializes the Cache Manager, and starts daemons that improve performance. These options are stored in the afsd options file. In the afs configuration file there are three predefined cache sizes:

- `SMALL` is suitable for a small machine that serves one or two users and has approximately 8 MB of RAM and a 20 MB cache.
- `MEDIUM` is suitable for a medium-sized machine that serves two to six users and has 16 MB of RAM and a 40 MB cache.
- `LARGE` is suitable for a large machine that serves five to ten users and has 32 MB of RAM and a 100 MB cache.

By default the distributed afs.conf file options are set to `MEDIUM`.

Follow these steps to configure the Cache Manager:

1. Create the local directory on which to mount the AFS file space:

   ```
   # mkdir /afs
   ```

2. Copy the AFS configuration option file to the /etc/sysconfig directory (in the case of the Suse 7.2 distribution we used, you need also to create this directory):

   ```
   # mkdir /etc/sysconfig
   # cp /usr/vice/etc/afs.conf /etc/sysconfig/afs
   ```

3. Edit the /etc/sysconfig/afs file if you want to incorporate any changes:

   Change `AFS_SERVER=off` to `AFS_SERVER=on`.

   In our example we added `-nosettime`, because this is a file server that is also a client. This flag prevents the machine from picking up the file server in the cell as its source for the correct time.

   There are also two more parameters you can use:

   – `-memcache` specifies that the machine will use a memory cache.
   – `-verbose` specifies that the trace of Cache Manager's initialization will be displayed on the standard output stream.

   Example 15-8 shows an example of the AFS configuration file.

Example 15-8 Our /etc/sysconfig/afs file

```
#! /bin/sh
# Copyright 2000, International Business Machines Corporation and others.
# All Rights Reserved.
#
# This software has been released under the terms of the IBM Public
# License.  For details, see the LICENSE file in the top-level source
# directory or online at http://www.openafs.org/dl/license10.html

# Configuration information for AFS client

# AFS_CLIENT and AFS_SERVER determine if we should start the client and or
# the bosserver. Possible values are on and off.
AFS_CLIENT=on
AFS_SERVER=on

# AFS client configuration options:
LARGE="-stat 2800 -dcache 2400 -daemons 5 -volumes 128"
MEDIUM="-stat 2000 -dcache 800 -daemons 3 -volumes 70 -nosettime"
SMALL="-stat 300 -dcache 100 -daemons 2 -volumes 50"
OPTIONS=$MEDIUM

# Set to "-verbose" for a lot of debugging information from afsd. Only
# useful for debugging as it prints _a lot_ of information.
```

```
VERBOSE=

# OPTIONS are the options passed to afsd.
OPTIONS="$OPTIONS $VERBOSE"


# Sample server preferences function. Set server preferences using this.
# afs_serverprefs() {
#    /usr/afsws/etc/fs setserverprefs <host> <rank>
#}

# Either the name of an executable script or a set of commands go here.
# AFS_POST_INIT=afs_serverprefs
AFS_POST_INIT=
```

15.4.5 Completing the installation of the first AFS server

The machine is now configured as an AFS file server and client machine. In this final phase of the installation, we initialize the Cache Manager and then create the upper levels of AFS file space, among other procedures.

Verifying the AFS initialization script

Follow these step to complete this task:

1. Shut down the **bos** server:

   ```
   # /usr/afs/bin/bos shutdown vmlinux8.itso.ibm.com -wait
   ```

2. Issue the **ps** command to learn the **bosserver** process's ID and then kill that process:

   ```
   # ps ax | grep bosserver
   9050 ?        S      0:00 /usr/afs/bin/bosserver -noauth
   # kill -9 9050
   ```

3. Reboot the VM Linux server, log on as root, and then start the AFS initialization script and wait for the message that all daemons are started:

   ```
   # cd /
   # shutdown -h now

   login: root
   password: root_password

   # /etc/init.d/afs start
   Starting AFS services.....
   afsd: All AFS daemons started.
   ```

4. As a basic test of correct AFS functioning, try to authenticate as `admin`:

```
# /usr/afs/bin/klog admin
Password: admin_passwd
```

5. Issue the **tokens** command to verify that the **klog** command was successful:

```
# /usr/afs/bin/tokens

Tokens held by the Cache Manager:

User's (AFS ID 501) tokens for afs@itso.ibm.com [Expires Aug  9 18:09]
--End of list--
```

6. Issue the **bos status** command to verify that the output of each process reads "`currently running normally`":

```
# /usr/afs/bin/bos status vmlinux8.itso.ibm.com
Instance kaserver, currently running normally.
Instance buserver, currently running normally.
Instance ptserver, currently running normally.
Instance vlserver, currently running normally.
Instance fs, currently running normally.
Auxiliary status is: file server running.
Instance upserver, currently running normally.
```

7. Check the volumes with the command:

```
# cd /
# /usr/afs/bin/fs checkvolumes
All volumeID/name mappings checked.
```

Activating the AFS initialization script

After confirming that the AFS initialization script works correctly, we take the action necessary to have it run automatically at each reboot.

On the SuSE 7.2 distribution you can do this by creating two symbolic links into run level 3, which is the default run level used:

```
# cd /etc/init.d/rc3.d/
# ln -s ../afs S99afs
# ln -s ../afs K01afs
```

Configuring the top levels of the AFS file space

If you have not previously run AFS in your cell, you now configure the top levels of your cell's AFS file space. We created the root.afs volume in "Starting the file server, volume server, and salvager" on page 370. Now we set the Access Control List (ACL) on the /afs directory. Creating, mounting, and setting the ACL are the three steps required when creating any volume.

After setting the ACL on the root.afs volume, create your cell's root.cell volume, mount it as a subdirectory of the /afs directory, and set the ACL. Create both a read/write and a regular mount point for the root.cell volume. The read/write mount point enables you to access the read/write version of replicated volumes when necessary. Creating both mount points essentially creates separate read-only and read-write copies of your file space, and enables the Cache Manager to traverse the file space on a read-only path or read/write path as appropriate.

Then replicate both the root.afs and root.cell volumes. This is required if you want to replicate any other volumes in your cell, because all volumes mounted above a replicated volume must themselves be replicated in order for the Cache Manager to access the replica.

When the root.afs volume is replicated, the Cache Manager is programmed to access its read-only version (root.afs.readonly) whenever possible. To make changes to the contents of the root.afs volume (when, for example, you mount another cell's root.cell volume at the second level in your file space), you must mount the root.afs volume temporarily, make the changes, release the volume, and remove the temporary mount point.

To set up the ACL for the /afs directory, follow these steps:

1. Edit the ACL on the /afs directory with **`fs setacl`**. We add the entry that grants the `l` (lookup) and `r` (read) permissions to the system:anyuser group. With this we enable all AFS users who can reach your cell to traverse trough the directory. If you prefer to enable access only to locally authenticated users, substitute the system:authuser group.

   Note: By default the **`system:administrators`** have all seven rights. This is the default entry that AFS places on every new volume's root directory.

   ```
   # /usr/afs/bin/fs setacl /afs system:anyuser rl
   ```

2. Create the root.cell volume and then mount it in the subdirectory of /afs, where it serves as the root of our cell's local AFS file space. At the end we create an ACL entry for the system:anyuser group:

   ```
   # /usr/afs/bin/vos create vmlinux8.itso.ibm.com /vicepa root.cell
   Volume 536870915 created on partition /vicepa of vmlinux8.itso.ibm.com
   /usr/afs/bin/fs mkmount /afs/itso.ibm.com root.cell
   /usr/afs/bin/fs setacl /afs/itso.ibm.com system:anyuser rl
   ```

3. To shorten the path names for users in the local cell we create a symbolic link to a shortened cell name:

   ```
   # cd /afs
   # ln -s itso.ibm.com itso
   ```

```
# ls -l
total 8
drwxrwxrwx    2 root     root         2048 Aug 8 20:44 .
drwxr-xr-x   22 root     root         4096 Aug 8 20:29 ..
lrwxr-xr-x    1 admin    root           12 Aug 8 20:44 itso -> itso.ibm.com
drwxrwxrwx    2 root     root         2048 Aug 8 20:41 itso.ibm.com
```

4. Create a read/write mount point for the root.cell volume (we created the regular mount point in step 2). By convention, the read/write mount point begins with a period:

```
# cd /usr/afs/bin
# ./fs mkmount /afs/.itso.ibm.com root.cell -rw
```

5. Define the replication site for the root.afs and root.cell volumes:

```
# ./vos addsite vmlinux8.itso.ibm.com /vicepa root.afs
Added replication site vmlinux8.itso.ibm.com /vicepa for volume root.afs
# ./vos addsite vmlinux8.itso.ibm.com /vicepa root.cell
Added replication site vmlinux8.itso.ibm.com /vicepa for volume root.cell
```

6. Verify that the Cache Manager can access both the root.afs and root.cell volumes before you attempt to replicate them:

```
# ./fs examine /afs
Volume status for vid = 536870912 named root.afs
Current disk quota is 5000
Current blocks used are 5
The partition has 6737196 blocks available out of 6737440

# ./fs examine /afs/itso.ibm.com
Volume status for vid = 536870915 named root.cell
Current disk quota is 5000
Current blocks used are 2
The partition has 6737196 blocks available out of 6737440
```

7. Release the replica of root.afs and root.cell you created in the previous steps:

```
# ./vos release root.afs
Released volume root.afs successfully
# ./vos release root.cell
Released volume root.cell successfully
```

8. Check the volumes to force the Cache Manager to notice that you have released read-only versions of the volumes, then examine the volumes again:

```
# ./fs checkvolumes
All volumeID/name mappings checked.
# ./fs examine /afs
Volume status for vid = 536870912 named root.afs
Current disk quota is 5000
Current blocks used are 5
```

```
The partition has 6737248 blocks available out of 6737440

# ./fs examine /afs/itso.ibm.com
Volume status for vid = 536870915 named root.cell
Current disk quota is 5000
Current blocks used are 2
The partition has 6737248 blocks available out of 6737440
```

Storing AFS binaries in AFS

In the conventional configuration, you make AFS client binaries and configuration files available in the subdirectories of the /usr/afsws directory on client machines (afsws is an acronym for AFS workstation). You can conserve local disk space by creating /usr/afsws as a link to an AFS volume that houses the AFS client binaries and configuration files for this system type.

In this section we create the necessary volumes. The conventional location to which to link /usr/afsws is /afs/cellname/sysname/usr/afsws.

Follow these steps to complete this task:

1. Create volumes for storing the AFS client binaries for this system type. In our example we create the volumes *s390_linux24*, *s390_linux24.usr* and *s390_linux.usr.afsws:*

```
# ./vos create vmlinux8.itso.ibm.com /vicepa s390_linux24
Volume 536870918 created on partition /vicepa of vmlinux8.itso.ibm.com
# ./vos create vmlinux8.itso.ibm.com /vicepa s390_linux24.usr
Volume 536870921 created on partition /vicepa of vmlinux8.itso.ibm.com
# ./vos create vmlinux8.itso.ibm.com /vicepa s390_linux24.usr.afsws
Volume 536870924 created on partition /vicepa of vmlinux8.itso.ibm.com
```

2. Now mount those volumes:

```
# ./fs mkmount -dir /afs/.itso.ibm.com/s390_linux24 -vol s390_linux24
# ./fs mkmount -dir /afs/.itso.ibm.com/s390_linux24/usr \
    -vol s390_linux24.usr
# ./fs mkmount -dir /afs/.itso.ibm.com/s390_linux24/usr/afsws \
    -vol s390_linux24.usr.afsws
```

3. Release the new root.cell replica and check the volumes so the local Cache Manager can access them:

```
# ./vos release root.cell
Released volume root.cell successfully
# ./fs checkvolumes
All volumeID/name mappings checked.
```

4. Grant the lookup and read access to the system:anyuser group on each new directory's ACL:

```
# cd /afs/.itso.ibm.com/s390_linux24
# /usr/afs/bin/fs setacl -dir . usr usr/afsws -acl system:anyuser rl
```

5. We set an unlimited quota an the s390_linux.usr.afsws volume so we do not have any problems copying appropriate files for distribution, without exceeding the quota:

```
# /usr/afs/bin/fs setquota /afs/.itso.ibm.com/s390_linux24/usr/afsws 0
```

6. Copy the neccessary files:

```
# cd /afs/.itso.ibm.com/s390_linux24/usr/afsws/
# cp -rp /usr/src/openafs-1.1.1/s390_linux24/dest/bin .
# cp -rp /usr/src/openafs-1.1.1/s390_linux24/dest/etc .
# cp -rp /usr/src/openafs-1.1.1/s390_linux24/dest/include .
# cp -rp /usr/src/openafs-1.1.1/s390_linux24/dest/lib .
```

7. Set the permissions to allow system:authuser to look up and read on directories /etc, /include, and /lib and deny the access for the group system:anyuser to those directories. The group system:anyuser still needs the lookup and read permissions to the /bin directory to enable unauthenticated users to access the **klog** binary:

```
# cd /afs/.itso.ibm.com/s390_linux24/usr/afsws
# /usr/afs/bin/fs setacl -dir etc include lib -acl system:authuser rl \
    system:anyuser none
```

8. Create the symbolic link /usr/afsws on the local disk to the directory /afs/itso.ibm.com/@sys/usr/afsws:

```
# ln -s /afs/itso.ibm.com/@sys/usr/afsws /usr/afsws
```

Tip: If you do not want to type the whole path to AFS suite commands, such as **fs**, you should include the following paths to the PATH environment variable: /usr/afsws/bin and /usr/afsws/etc.

Congratulations! You have just completed the installation of the AFS server on your VM Linux.

15.4.6 Installing clients on other servers

In this section we describe how to install the AFS client on the server from which you want to access the AFS files.

Transfer the installation files to the client server

Follow these steps to transfer the installation files from the server where you compiled the OpenAFS package to the client server. In our example, the server with compiled packages was vmlinux8.itso.ibm.com.

1. Create the gz package:

```
# cd usr/src/openafs-1.1.1/s390_linux24/
# tar -c -z dest > s390_linux24afs.gz
```

2. Transfer the file to the client computer with ftp; for example, create the directory and unpack into this directory:

```
# mkdir /afsinstall
# cd /afsinstall/
# ftp vmlinux8.itso.ibm.com
Connected to vmlinux8.itso.ibm.com.
...
ftp> cd /usr/src/openafs-1.1.1/s390_linux24
ftp> bin
ftp> get s390_linux24afs.gz
...
11253760 bytes received in 00:00 (24.35 MB/s)
ftp> bye
221 Goodbye.
# tar zxf s390_linux24afs.gz
```

Creating AFS directories on the local disk

Create the directories for holding binary and configuration files with the commands:

```
# mkdir /usr/vice
# mkdir /usr/vice/etc
```

Loading AFS into the kernel

Follow these steps to load the AFS modules into the kernel:

1. Copy the AFS kernel files into the /usr/vice/etc/modload directory:

```
# cd /afsinstall/dest/root.client/usr/vice/etc/
# cp -rp modload /usr/vice/etc
```

2. Copy the initialization script and start it:

```
# cp -p afs.rc /etc/init.d/afs
# /etc/init.d/afs start
Starting AFS services.....
```

Enabling AFS login

Follow the instructions in “Enabling AFS Login” on page 364 to enable AFS login on the client server. Keep in mind that the installation files are now in the /afsinstall directory, not in /usr/src/openafs-1.1.1/s390_linux24.

Loading and creating client files

Follow these steps to complete this task:

1. Copy the client files:

```
# cd /afsinstall/dest/root.client/usr/vice/etc/
# cp -p * /usr/vice/etc
cp: omitting directory `C'
cp: omitting directory `modload'
# cp -rp C /usr/vice/etc
```

2. Create the /usr/vice/etc/ThisCell file. With this cell you define the membership of this client server:

```
# echo "itso.ibm.com" > /usr/vice/etc/ThisCell
```

3. From your AFS server, copy CellServDB to /usr/vice/etc/CellServDB.

Configuring the cache

We already explained, in "Configuring the cache" on page 374, how the cache works and what the needed parameters are. Here we just outline the procedure to implement this on the client machine. In our example, we use the disk cache:

1. Create the directory for the cache:

```
# mkdir /usr/vice/cache
```

2. Create the cacheinfo file with a cachesize of 25000 KB:

```
# echo "/afs:/usr/vice/cache:25000" > /usr/vice/etc/cacheinfo
```

Configuring the Cache Manager

We explained the function of the Cache Manager in "Configuring the Cache Manager" on page 375. Follow these steps to set up the Cache Manager on the client server:

1. Create the directory for the cache:

```
# mkdir /afs
```

2. Copy the configuration file:

```
# mkdir /etc/sysconfig
# cp /usr/vice/etc/afs.
# cp /usr/vice/etc/afs.conf /etc/sysconfig/afs
```

3. Edit the configuration file to suit your needs. Following is our example file:

```
#! /bin/sh
# Copyright 2000, International Business Machines Corporation and others.
# All Rights Reserved.
#
# This software has been released under the terms of the IBM Public
# License.  For details, see the LICENSE file in the top-level source
# directory or online at http://www.openafs.org/dl/license10.html

# Configuration information for AFS client
# AFS_CLIENT and AFS_SERVER determine if we should start the client and or
# the bosserver. Possible values are on and off.
AFS_CLIENT=on
AFS_SERVER=off

# AFS client configuration options:
LARGE="-stat 2800 -dcache 2400 -daemons 5 -volumes 128"
MEDIUM="-stat 2000 -dcache 800 -daemons 3 -volumes 70 -memcache"
SMALL="-stat 300 -dcache 100 -daemons 2 -volumes 50"
OPTIONS=$MEDIUM

# Set to "-verbose" for a lot of debugging information from afsd. Only
# useful for debugging as it prints _a lot_ of information.
VERBOSE=

# OPTIONS are the options passed to afsd.
OPTIONS="$OPTIONS $VERBOSE"


# Sample server preferences function. Set server preferences using this.
# afs_serverprefs() {
#    /usr/afsws/etc/fs setserverprefs <host> <rank>
#}

# Either the name of an executable script or a set of commands go here.
# AFS_POST_INIT=afs_serverprefs
AFS_POST_INIT=
```

Starting the Cache Manager

1. Reboot the VM Linux server, log on as root, and then start the AFS initialization script and wait for the message that all daemons are started:

   ```
   # cd /
   # shutdown -h now
   ...
   login: root
   password: root_password
   ...
   # /etc/init.d/afs start
   Starting AFS services.....
   afsd: All AFS daemons started.
   ```

2. Follow the instructions in “Activating the AFS initialization script” on page 378 to enable the script to load automatically.

Setting up volumes and loading binaries into AFS

Here we create /usr/afsws on the local disk to the directory in AFS that houses AFS binaries for this system type. We prepared those binaries in the "Storing AFS binaries in AFS" on page 381.

1. Create /usr/afsws on the local disk as a symbolic link to the directory /afs/itso.ibm.com/@sys/usr/afsws with the command:

   ```
   # ln -s /afs/itso.ibm.com/@sys/usr/afsws /usr/afsws
   ```

Congratulations! Now you are ready to use the AFS file system on your client. Next time you log on to your system, you will already be authenticated to the AFS server.

> **Important**: On any Linux system on which you want to use the AFS logon with authentication to the AFS server, you still have to define a /etc/passwd entry with the same username and user ID as defined on the AFS server. This is required by the AFS PAM module. It authenticates you to the AFS server, but it still requires a local entry in the /etc/passwd file.

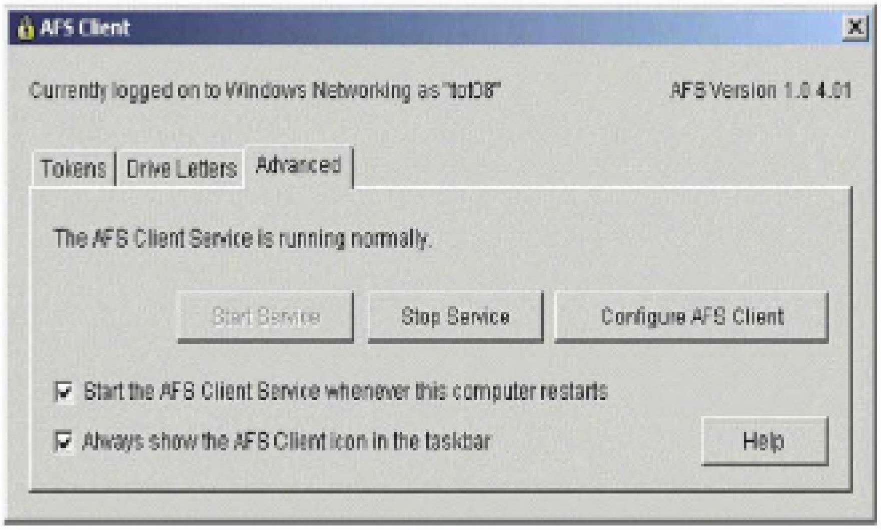
15.4.7 \Installing Windows 2000 OpenAFS Client

In this section we explain how to install and configure Windows 2000 OpenAFS Client. You can obtain the compiled version of AFS client from:

http://www.openafs.org/release/latest.html

In our example we used version 1.0.4.a.

After installing the package, start the AFS client. You will see a window similar to Figure 15-2.



Currently logged on to Windows Networking as "tot08"

AFS Version 1.0 4.01

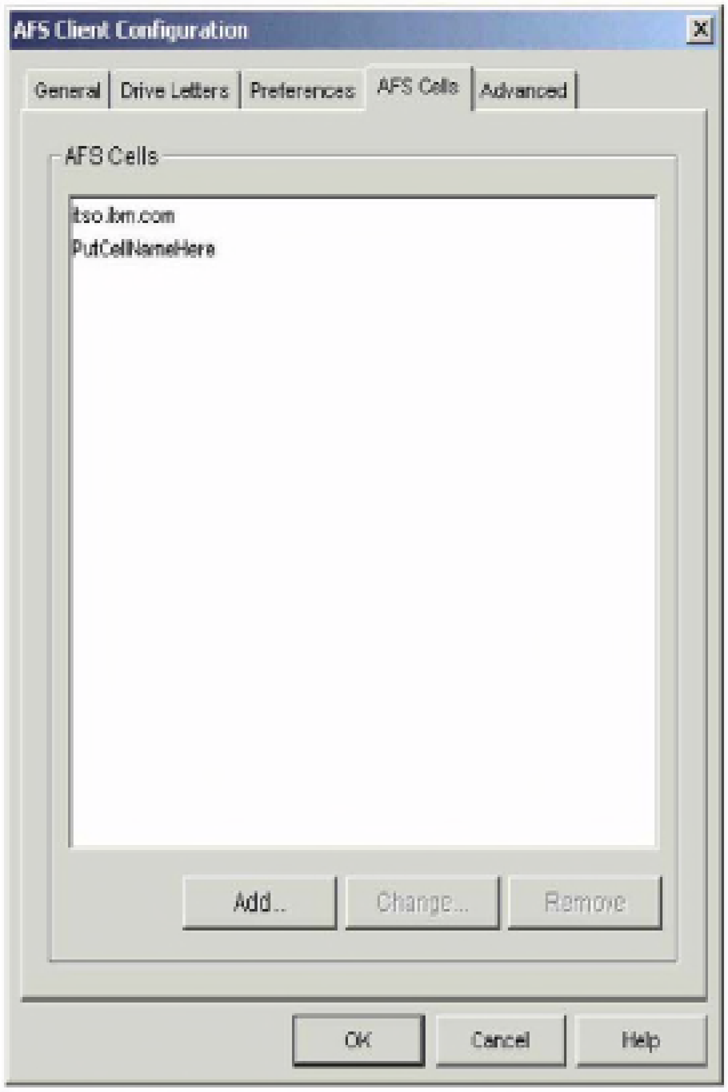
The AFS Client Service is running normally.

[✓] Start the AFS Client Service whenever this computer restarts

[✓] Always show the AFS Client icon in the taskbar

Figure 15-2 Windows 2000 AFS Client

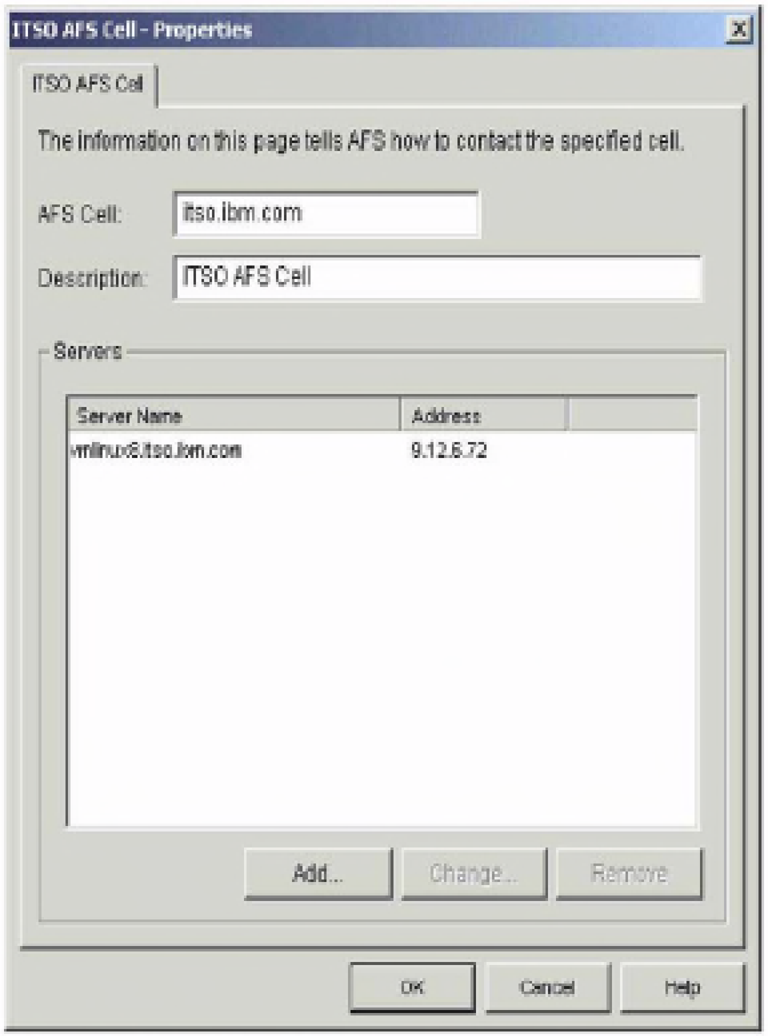
On the Advanced tab, click `Configure AFS Client` and you will see a window similar to Figure 15-3 on page 388.



itso.ibm.com
PutCellNameHere

Figure 15-3 Configuring Windows 2000 AFS Client

Select the **AFS Cells** tab and click **Add...** , and you will see a window similar to Figure 15-4 on page 389.


ITSO AFS Cell - Properties

The information on this page tells AFS how to contact the specified cell.

AFS Cell: itso.ibm.com

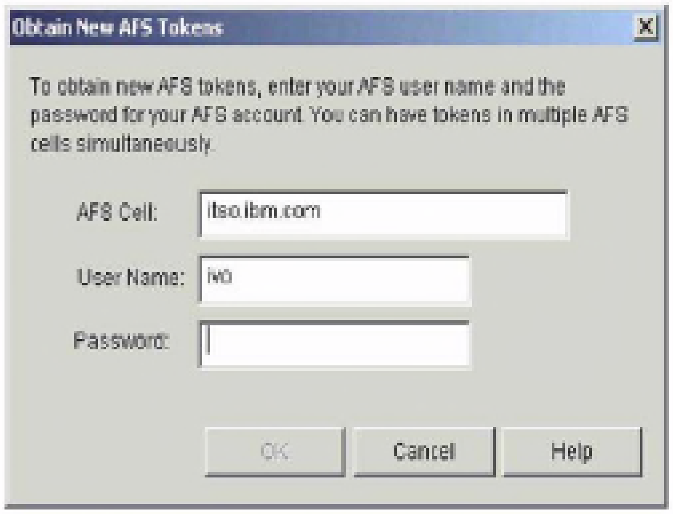
Description: ITSO AFS Cell

Server Name	Address
vmlinux8.itso.ibm.com	9.12.6.72

Figure 15-4 Defining the AFS Cell

Define your AFS Cell name here, and the server that is holding this cell. In our example you can see that the cell name is itso.ibm.com and that the server is vmlinux8.itso.ibm.com. Now start the AFS client service.

After defining the AFS Cell and starting the service, you can log on to get the tokens. In the main AFS client setup shown in Figure 15-2 on page 387, select the **Tokens** tab and click **Obtain New Tokens...**. You will see a window similar to Figure 15-5 on page 390.


Obtain New AFS Tokens

To obtain new AFS tokens, enter your AFS user name and the password for your AFS account. You can have tokens in multiple AFS cells simultaneously.

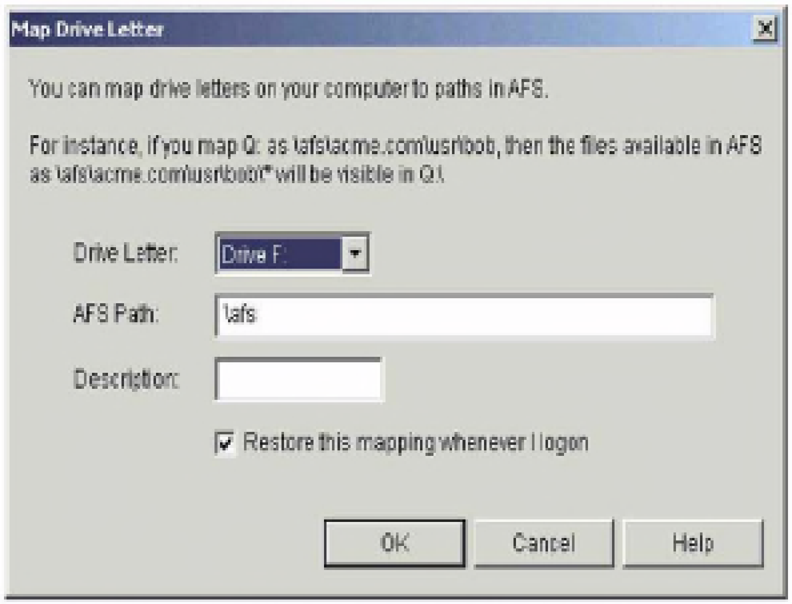
AFS Cell: itso.ibm.com

User Name: ivo

Password:

Figure 15-5 Logging on to the AFS server

Now you can map the AFS directory /afs to the drive letter on your Windows 2000 workstation. In the main AFS client setup shown in Figure 15-2 on page 387, select the **Drive Letters**, and you will see a window similar to Figure 15-6.


Map Drive Letter

You can map drive letters on your computer to paths in AFS.

For instance, if you map Q: as \afs\acme.com\usr\bob, then the files available in AFS as \afs\acme.com\usr\bob\* will be visible in Q:\

Drive Letter: Drive F:

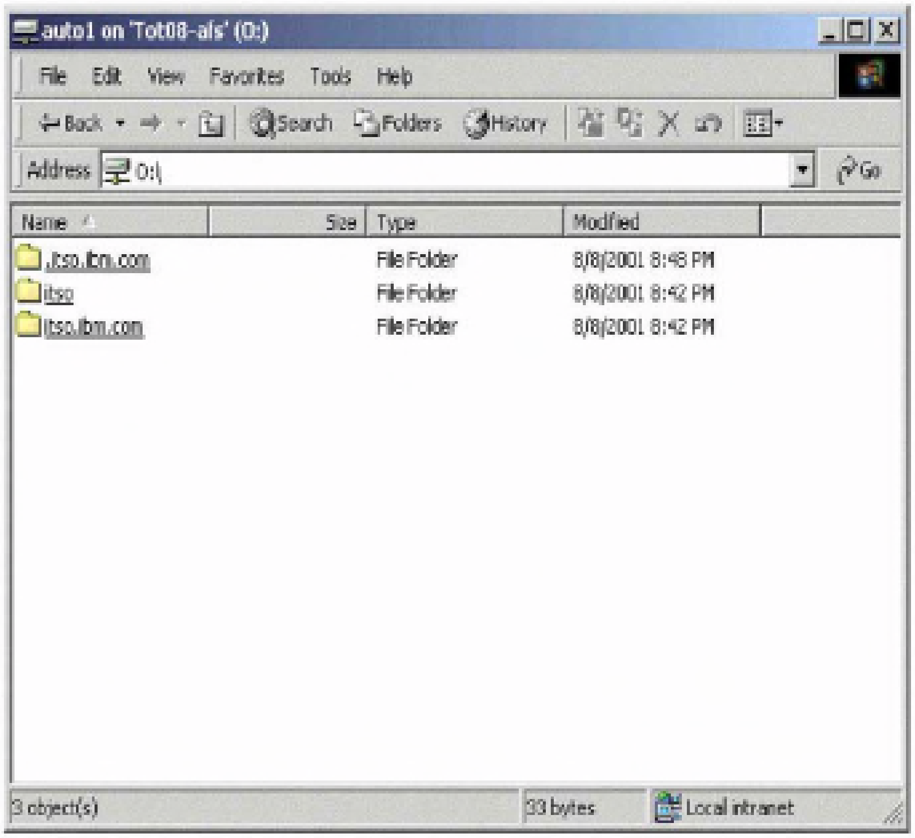
AFS Path: \afs

Description:

[✓] Restore this mapping whenever I logon

Figure 15-6 Mapping the /afs directory to the local drive letter

After you mapped the /afs directory to the drive letter, you can see the /afs content by exploring the drive you assigned. You will see a window similar to Figure 15-7 on page 391.


auto1 on 'Tot08-afs' (O:)
File Edit View Favorites Tools Help
Back Search Folders History
Address O:\
Go
Name Size Type Modified

Name	Size	Type	Modified
.itso.ibm.com		File Folder	8/8/2001 8:43 PM
itso		File Folder	8/8/2001 8:42 PM
itso.ibm.com		File Folder	8/8/2001 8:42 PM

3 object(s) 33 bytes Local intranet

Figure 15-7 /afs directory mapped to the drive

Tip: If you define the same username and password on the AFS server as you use on you Windows 2000 workstation you can integrate the logon to AFS server with you workstation logon. With this you can use single logon to the workstation and AFS server.

16

z/VM 4.2 Linux features

In this chapter, we describe the enhancements shipped with IBM's z/VM 4.2 Operating System that can benefit customers running Linux guest virtual machines.

We review the System Administration Facility, which can help in the creation and management of multiple Linux guests. This facility can also be used to migrate from an existing Virtual Image Facility (VIF) environment.

We also review the VM LAN Support introduced in z/VM 4.2. This facility allows z/VM to support multiple internal, virtualized LAN segments. Individual guest machines can define a virtual network interface card (NIC) and then connect to the virtual or guest LAN segment using existing communications software, such as the Linux TCP/IP stack (we refer to these as "guest LANs" to avoid confusion with virtual LAN, since that term has been adopted by an IEEE standard, 802.1q).

16.1 System Administration Facility

The System Administration Facility is in part based on a number of ease-of-use functions developed for the S/390 Virtual Image Facility (VIF) product. These functions include the creation of Linux guest machines, the assignment of disk space for those guests, and the ability to start and stop Linux servers. The facility is comprised of a client component which runs in either a Linux or CMS guest. This client component communicates with a server component known as the VMADMIN server.

16.1.1 Who should use the System Administration Facility

The System Administration Facility is intended for use in a *new, non-tailored* z/VM system. It will not function if you want to use a pre-existing system. VMADMIN manages the user directory (which means that you can't do it—either manually, or through a directory management product such as DIRMAINT).

Before deciding on whether or not this facility is going to be useful in your environment, we recommend that you review the following flowchart, which comes from the *System Administration Facility Guide*, SC24-6034.

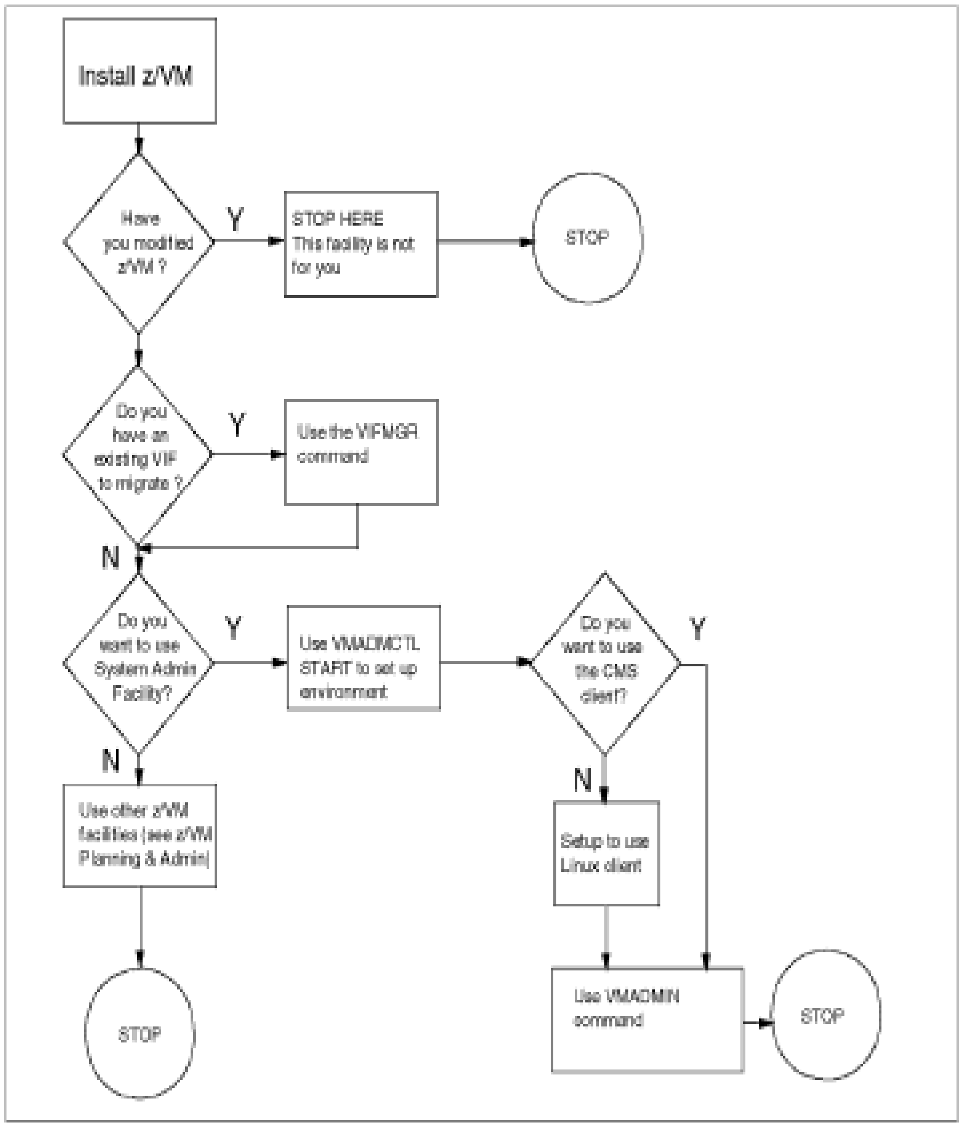

Install z/VM
Have you modified z/VM ?
Y
STOP HERE This facility is not for you
STOP
Do you have an existing VIF to migrate ?
Y
Use the VIFMGR command
N
Do you want to use System Admin Facility?
Y
Use VMADMCTL START to set up environment
Do you want to use the CMS client?
Y
N
Use other z/VM facilities (see z/VM Planning & Admin)
N
Setup to use Linux client
STOP
Use VMADMIN command
STOP

Figure 16-1 Figure 5-1 How to proceed after installing z/VM

16.1.2 Initializing the System Administration Facility

If you decide to use the System Administration Facility in your environment, execute the following steps to initialize the facility (before proceeding, however, we recommend you first read the instructions given in *System Administration Facility Guide*, SC24-6034).

1. Logon to CMS on the newly installed z/VM system, using the VMADMIN user ID.
2. Enter the command `VMADMCTL START`. (Note that this command can only be entered once; you cannot stop and then restart VMADMCTL.)
3. Reply to the prompts with your environment-specific responses.

You'll need the following information about your environment in order to answer the VMADMCTL initialization questions:

- Network device address for the VMADMIN server

  Network devices such as OSA cards have two (and sometimes, three) device addresses assigned to them. There is always an even number and an odd device number, which represent read and write subchannels.

  On an OSA Express card running in QDIO mode, there's a third device for the control subchannel. To answer this prompt, specify the even device address number of the network card you'll be using for VMADMIN. For example, if using an OSA card with device addresses 0x0500, 0x0501, and 0x0502, you'd reply with `0500`.
- The network device's port number or port name

  This is the number or name of the starting even port, or the name associated with the network device that is assigned to the VMADMIN server.
- VMADMIN server network type
- The type of local area network (LAN) to which the server is connected. For a QDIO device, specify either FastEthernet, FE, GigabitEthernet, or GB. For a non-QDIO device, specify either Ethernet, 802.3, TokenRing, TR, or FDDI.
- VMADMIN server network MTU size (576, 1492, 1500, 2000, 4096, 4352 or 8902)
- VMADMIN server IP address
- VMADMIN server subnet mask
- IP address of gateway to be used by the System Administration Facility

Once configuration has completed, the VMADMIN server will be initialized and a message will be generated, informing you that VMADMIN is now operational.

Following is an example of the initialization process that includes the prompts and example responses:

```
VMADMCTL START
HLEVMA0050I Reply RESTART at any time to start over or QUIT to terminate
HLEVMA0052R Enter Server network device address:
9.12.6.73
HLEVMA0036E Device address must be hexadecimal
HLEVMA0052R Enter Server network device address:
292C
HLEVMA0053R Enter Server network port number:
0
```

```
HLEVMA0054R Enter Server network type (Ethernet, 802.3, TokenRing, TR,
FDDI):
Ethernet
HLEVMA0055R Enter Server network MTU size (576, 1492, 1500, 2000, 4096,
4352, 8902):
1500
HLEVMA0056R Enter Server IP address:
9.12.6.73
HLEVMA0057R Enter Server IP mask:
255.255.255.0
HLEVMA0058R Enter Server gateway IP address:
9.12.6.75
HLEVMA0079I
HLEVMA0079I Here is the configuration (please make a note of it):
HLEVMA0079I
HLEVMA0079I Server:
HLEVMA0079I         Network device address: 292C
HLEVMA0079I          Network port: 0
HLEVMA0079I          Network type: ETHERNET
HLEVMA0079I          Network MTU size: 1500
HLEVMA0079I          IP address: 9.12.6.73
HLEVMA0079I          IP mask: 255.255.255.0
HLEVMA0079I          Gateway IP address: 9.12.6.75
HLEVMA0079I          Client IP address:
HLEVMA0079I
HLEVMA0080R Is this correct (Yes(1),No(0)):
1
14:51:28 AUTO LOGON  ***       VMADMIN  USERS = 8     BY MAINT
HLEVMA0498I VMADMCTL complete - VMADMIN is now operational
```

Verify that all the settings are correct by using the **VMADMIN Q ALL** command:

```
VMADMIN Q ALL
HLE$QU0079I
HLE$QU0079I Here is the configuration (please make a note of it):
HLE$QU0079I
HLE$QU0079I Server:
HLE$QU0079I         Network device address: 292C
HLE$QU0079I                  Network port: 0
HLE$QU0079I                  Network type: ETHERNET
HLE$QU0079I              Network MTU size: 1500
HLE$QU0079I                    IP address: 9.12.6.73
HLE$QU0079I                       IP mask: 255.255.255.0
HLE$QU0079I            Gateway IP address: 9.12.6.75
HLE$QU0079I             Client IP address: NOT DEFINED
HLE$QU0079I
HLE$QU1300I 0 of 0 MB of server paging space in use
HLE$QU1301I 0 MB of 0 MB of Linux image partition space in use
```

```
HLE$QU1302I VMADMIN performance: CPU is Green, Paging is Green, I/O is
Green
HLE$QU1324I No paging volumes are defined
HLE$QU1324I No image volumes are defined
HLE$QU1303I Server uses IP address 9.12.6.73 with device 292C
HLE$QU1306I Server level: 18, Service 000
HLE$QU1307I Last boot on 2001-08-02 at 15:15:06
------------------------------------------------------------------------
```

16.1.3 Using VMADMIN

VMADMIN functions can be run from a CMS session or from a Linux guest machine. Before you can run VMADMIN from Linux, however, you must first create a Linux image, using the VMADMIN CMS client. Once you have created the first Linux system, all other VMADMIN work can be done from Linux.

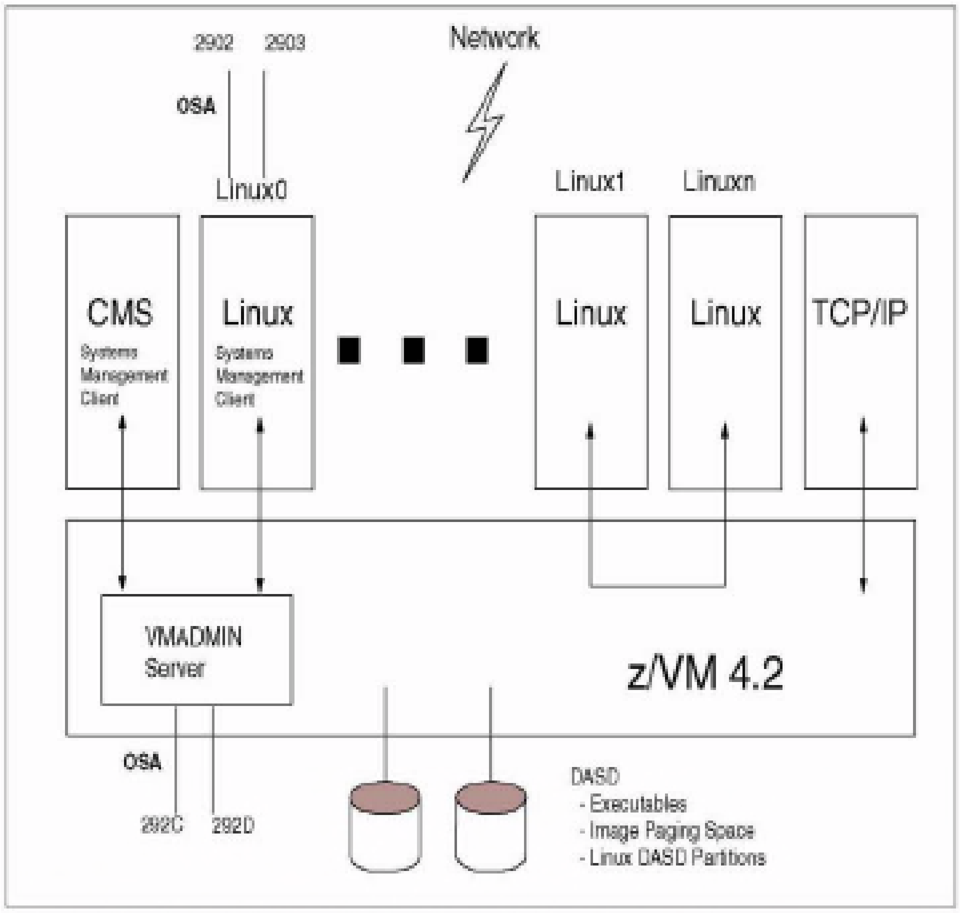


Figure 16-2 Example System Administration Facility environment

16.1.4 Creating the initial Linux guest

The following steps take you through the creation of the initial Linux virtual machine. These steps must be performed from a CMS session that has access to the VMADMIN facility. (Typically, in a new installation you should use the MAINT user ID for this task.)

1. Using the VMADMIN SERVER VOLUME command, create the paging and image (Linux filesystem) space that will be used by the Linux guest.

   Note: The following examples show user input in bold.

   Create a paging device, using device address 3E44, and give it a volume label of VM3E44.

   ```
   vmadmin server volume add paging 3e44 vm3e44
   16:19:38 DASD 3E44 ATTACHED TO VMADMIN 1000 BY VMADMIN WITH DEVCTL
   HLE$VO2004I Command may take up to 16 minutes; please wait
   ```

   Once the format has completed, map the volume to ensure that everything worked.

   ```
   vmadmin server volume map used vm3e44
   HLE$VO2216E VM3E44 (3E44) is a PAGING volume
   Command Complete
   ```

   Add a volume for "image" space (i.e., space that will be used by the Linux filesystems). Once this completes, also map this volume to ensure that the format completed successfully.

   ```
   vmadmin server volume add image 3ca3 vm3ca3
   HLE$VO2200I IMAGE volume VM3CA3 added
   Command Complete
   ```

2. Create the first Linux guest by using the command **VMADMIN IMAGE CREATE**. (In our case we named it LNXMSTR since it is the first Linux guest, but you can name it whatever you prefer.)

   Note: This command merely defines the guest to VM and does not perform the actual Linux install.

   ```
   vmadmin image create LNXMSTR
   Enter password for Image:
   PASSWORD
   Re-enter password:
   PASSWORD
   HLE$IM1500I Image LNXMSTR created successfully
   Command Complete
   ```

3. Use the **VMADMIN SERVER INSTALL** command to copy the kernel image, parmline, and initial ramdisk files from a nominated FTP server to the guest machine. Once this completes, you'll be ready to boot the Linux system starter system.

Select the FTP server that contains the kernel image, parmline, and initial RAMdisk that you want to install. In our case we used SuSE, so the location of these files is stored in the suse.ins file.

Note: We had a problem in getting this step to work, at first. We resolved the problem by editing the suse.ins file, entering fully qualified path names (i.e. paths from the root directory) for the image, parmline, and initrd files. This problem may now be resolved, however, as we were using an early build of z/VM 4.2.

```
VMADMIN SERVER INSTALL 9.12.6.134 ftpuser ftppwd suse.ins
HLE$IN2305I Transferring Linux from 9.12.6.134 suse.ins
HLE$IN2003I Command may take up to 20 seconds; please wait
HLE$IN2304I Linux installed from 9.12.6.134 suse.ins
Command Complete
```

4. Define the network device that'll be used by the first Linux guest, by using the **VMADMIN IMAGE NETWORK** command. In our example, the Linux guest will use an OSA card with device addresses 2902 and 2903 (see Figure 16-2 on page 398 for details).

```
VMADMIN IMAGE NETWORK lnxmstr add 2902
HLE$IM1506I NETWORK ADD completed successfully
Command Complete
```

5. Authorize the Linux master guest to be able to be able to run VMADMIN commands.

```
vmadmin server clientipaddress 9.12.6.65
HLE$CL2504I Ping to Client IP address 9.12.6.65 failed
HLE$CL2501I Client IP address is set to 9.12.6.65 successfully
Command Complete
```

6. Provide read-only access to VMADMIN's 203 minidisk, which holds the Linux vmadmin command.

```
vmadmin partition share vmadmin 203 with lnxmstr 203
HLE$PA1506I PARTITION SHARE completed successfully
Command Complete
```

You're now ready to boot the initial Linux guest machine.

7. Logon to a CMS session using (in our example) user ID LNXMSTR. The Linux system will automatically boot.

```
LOGON LNXMSTR PASSWORD
There is no logmsg data
FILES:  NO RDR,  NO PRT,  NO PUN
LOGON AT 21:55:18 EDT THURSDAY 08/16/01
Linux version 2.2.19 (root@s390l6) (gcc version 2.95.2
(SuSE+gcc-2.95.2.4-diffs+gcc-bugfixes)) #1 SMP Mon Jun 18 05:19:40 2001
Commandlineis:ramdisk_size=32768root=/dev/ram0ro
We are running under VM
```

```
This machine has an IEEE fpu
Initial ramdisk at: 0x02000000 (16777216 bytes)
Detected device 001F on subchannel 0000 - PIM = 80, PAM = 80, POM = FF
Detected device 2902 on subchannel 0001 - PIM = 80, PAM = 80, POM = FF
Detected device 2903 on subchannel 0002 - PIM = 80, PAM = 80, POM = FF
Detected device 0203 on subchannel 0003 - PIM = F0, PAM = F0, POM = FF
...
```

8. Because this is the first time this Linux image has been booted, you'll be prompted for networking definitions. Configure the network, and then telnet to the image. You can now install the full system by using an appropriate installation script (we used YaST in our example).

   To issue VMADMIN commands from the Linux guest, once you log onto the image, you must make the DASD at device number 203 known to Linux and it must be mounted read-only. (For more information about the procedure you'll need to follow to accomplish this step, refer to the documentation for the Linux distribution you are using.)

   Note: For a complete description of the functions available with the System Administration Facility, refer to the z/VM 4.2 publication *System Administration Facility Guide*, SC24-6034.

16.2 VM LAN support

Prior to z/VM 4.2, virtual connectivity options for connecting one or more virtual machines were restricted to virtual channel-to-channel (CTC) links, and the Inter User Communications Vehicle (IUCV) facility. These are point-to-point connections, which means that in the case of CTC links, when you want two virtual machines to communicate with each other, you must define CTC device pairs in *each* machine and couple those devices between guest machines.You also have to define static routing statements in each guest that needs to communicate with another guest in the system.

Another problem with point-to-point links is that, if one side of the connection went down, it was often difficult to subsequently reconnect the two machines. Frequently, one of the Linux guest machines would have to reboot in order to pick up the connection.

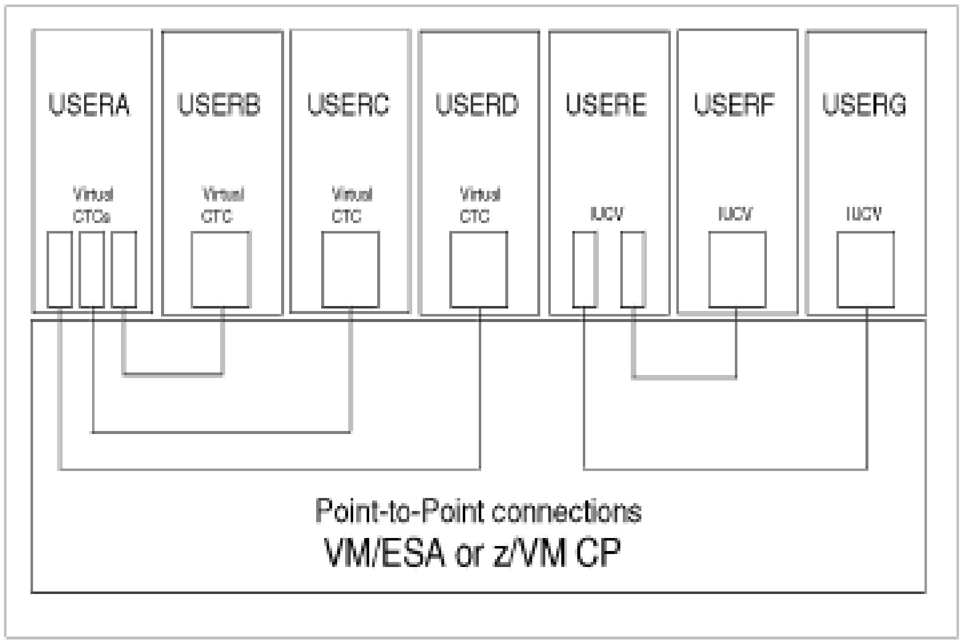


Figure 16-3 Guest virtual connectivity options prior to z/VM 4.2"

From z/VM 4.2, CP has been enhanced to provide a feature known as "VM LAN"; see Figure 16-4 on page 403. This feature allows you to create multiple virtual LAN segments within a z/VM environment, and there is no limit on the number of LAN segments that you can create. Individual guest machines can create a virtual Network Interface Card (NIC) to allow them to connect to the virtual LAN and communicate with other guests using standard TCP/IP protocols.

The virtual NIC emulates a HiperSockets device, as introduced by the zSeries z900 GA-2 machines in late 2001. As the VM LAN is a virtualization technique, it is not limited to the use of zSeries hardware; support for VM LAN goes back to 9672 Generation 5 machines and the Multiprise 3000.

Unlike with the complexity of point-to-point connections, when using the VM LAN facility, you only have to define the virtual network adapter in each guest and connect that adapter to the LAN.

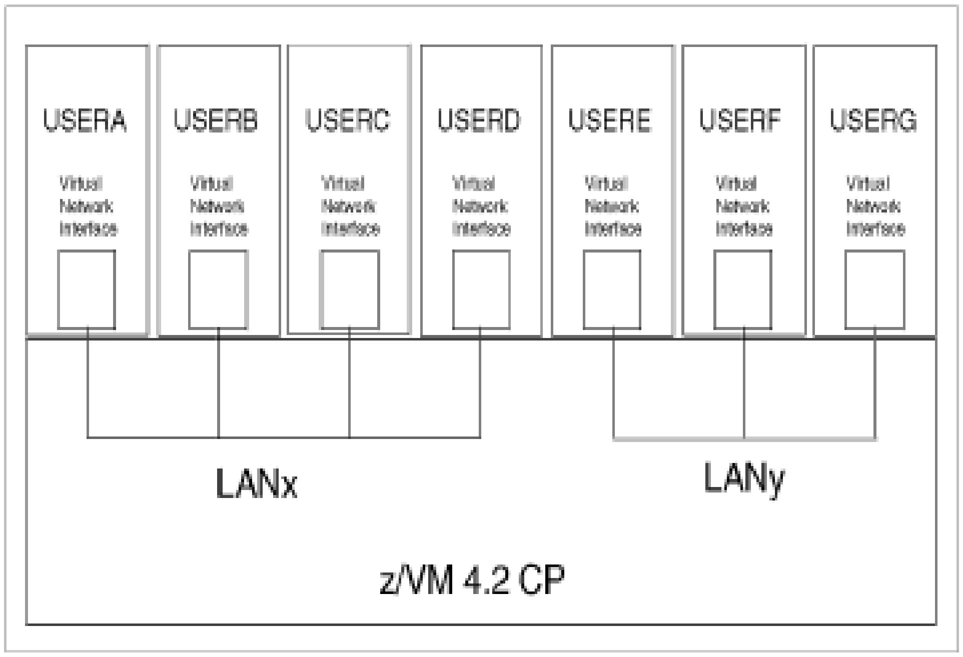


Figure 16-4 VM LAN support in z/VM 4.2

To use the VM LAN, the guest O/S (for example, Linux) must be able to support HiperSockets. In our case, we used an IBM internal test Linux system which had Hipersockets support built into the QETH driver.

VM LANs can be created or destroyed dynamically. It is possible to have both unrestricted and restricted LANs. Access to a restricted LAN is controlled via an access control list.

16.2.1 Creating a VM LAN

To manually create a virtual LAN for a group of VM guests, follow these steps:

1. Create a VM LAN segment in the VM host system.

   ```
   CP define lan SEWLAN MAXCONN 100 ownerid system
   23:00:55 LAN SYSTEM SEWLAN is created
   ```

   We created a LAN called SEWLAN, and the maximum number of guests that can connect to this LAN is 100. The MAXCONN setting can be changed to any number between 1 and 1024. Alternatively, if you do not use a MAXCONN value, then there is no limit on the number of guests that can connect to this LAN.

   The LAN definition given in our example is not permanent across IPLs of VM, so you should add a DEFINE LAN statement to VM's SYSTEM CONFIG file. Refer to the DEFINE LAN section in the z/VM *CP Command and Utility Reference* for additional details on this subject.

2. On each individual guest machine, you must create a virtual network interface card (NIC).

   ```
   CP define nic 500 hiper devices 3
   23:08:01 NIC 0500 is created; devices 0500-0502 defined
   ```

   This creates a set of devices that will look like a HiperSockets interface to Linux. Again, this is not a permanent definition; it will only exist for the life of the guest session. To make this definition permanent, add a SPECIAL statement in the CP directory for that guest, either by editing the USER DIRECT file or by running DIRMAINT.

3. On each guest, connect the virtual NIC to the LAN.

   ```
   CP couple 500 to system sewlan
   23:10:12 NIC 0500 is connected to LAN SYSTEM SEWLAN
   ```

   To ensure this happens whenever a Linux guest starts up, put this **COUPLE** command into each guest's PROFILE EXEC file.

16.2.2 Using the VM LAN with Linux guests

Note: The information in this section is based on tests using a pre-GA QDIO driver on an internal IBM system; your experiences may be different. Both the VM LAN facility and the supporting Linux device drivers will be available by late 2001 as GA code.

To use the VM LAN with Linux guests, we followed these steps:

1. We defined the VM LAN, created a virtual NIC for each our Linux guests, and then connected those NICs to the LAN as discussed in 16.2.1, "Creating a VM LAN" on page 403.
2. We booted Linux, using an initial RAMdisk installation.
3. When we got to the networking prompts, we entered the following details:

```
Welcome to Linux for S/390
Is your machine connected to a network (Yes/No) ? yes

Select the type of your network device
1) for lcs osa token ring
2) for lcs osa ethernet
3) for qdio osa ethernet
4) for channel to channel and escon channel connections
5) for IUCV
6) for CLAW
Enter your choice (1-6): 3

Please type in the channel device configuration options, e.g
qeth0,0xfd00,0xfd01,0xfd02,0,1
```

```
qeth parameter:
qeth0,0x0500,0x0501,0x0502,0,0,0

Please enter your IP address: 192.168.0.10
Please enter the net mask: 255.255.255.0
Please enter the net address:  [192.168.0.0] 192.168.0.0
Please enter the gateway address:  [192.168.0.1] 192.168.0.1
Please enter the IP address of the DNS server:
Please enter your host name: zvmlnx3.itso.ibm.com
Please enter the DNS search domain: itso.ibm.com

Configuration will be:
Channel device   : qeth0,0x0500,0x0501,0x0502,0,0,0
Host name        : zvmlnx3.itso.ibm.com
IP address       : 192.168.0.10
Net mask         : 255.255.255.0
Broadcast address: 192.168.0.255
Gateway address  : 192.168.0.1
Net address      : 192.168.0.0
DNS IP address   :
DNS search domain: itso.ibm.com
Is this correct (Yes/No) ?
Yes
```

4. When we got to the Linux shell, we entered the command **ifconfig -a** in order to determine if we had a hipersockets device defined.

```
# ifconfig -a
hsi0      Link encap:Ethernet  HWaddr 00:00:00:00:00:00
         NOARP  MTU:8192  Metric:1
         RX packets:0 errors:0 dropped:0 overruns:0 frame:0
         TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
         collisions:0 txqueuelen:100
         Interrupt:14

lo         Link encap:Local Loopback
           inet addr:127.0.0.1  Mask:255.0.0.0
           UP LOOPBACK RUNNING  MTU:16436  Metric:1
           RX packets:14 errors:0 dropped:0 overruns:0 frame:0
           TX packets:14 errors:0 dropped:0 overruns:0 carrier:0
           collisions:0 txqueuelen:0
```

We also entered a **CP QUERY NIC** command to determine the status of our Linux guest machine's virtual Network Interface Card. As seen from the following example, we had an established session, but did not have an IP address bound to the NIC.

```
CP Q NIC 500 DETAILS
Adapter 0500  Type: HIPER     Name: UNASSIGNED  Devices: 3
Port 0 MAC: 00-04-AC-00-00-05  LAN: SYSTEM ITSOLAN     MFS: 16384
```

```
Connection Name: HALLOLE   State: Session Established
Device: 0500  Unit: 000   Role: CTL-READ
Device: 0501  Unit: 001   Role: CTL-WRITE
Device: 0502  Unit: 002   Role: DATA
```

5. We displayed the contents of the /proc/chandev file in order to verify that the devices 0x500,0x501,and 0x0502 have been detected.

```
# cat /proc/chandev

channels detected
       chan   cu     cu     dev    dev                                    in chan
Irq    devno  type   type   model   type  model pim  chpids               use reg.
===============================================================================
0x0000 0x2946 0x04  0x3088 0x60  0x0000 0x00 0x80 0x1900000000000000  no    no
0x0001 0x2947 0x04  0x3088 0x60  0x0000 0x00 0x80 0x1900000000000000  no    no
0x000e 0x0500 0x10  0x1731 0x05  0x1732 0x05 0x80 0x0500000000000000  yes   yes
0x000f 0x0501 0x10  0x1731 0x05  0x1732 0x05 0x80 0x0500000000000000  yes   yes
0x0010 0x0502 0x10  0x1731 0x05  0x1732 0x05 0x80 0x0500000000000000  yes   yes
```

6. We were now ready to activate the hipersockets device via an **`ifconfig`** command, as follows:

```
# ifconfig hsi0 192.168.0.1 netmask 255.255.255.0 multicast up

hsi0   Link encap:Ethernet  HWaddr 00:00:00:00:00:00
       inet addr:192.168.0.10  Mask:255.255.255.0
       UP RUNNING NOARP MULTICAST  MTU:8192  Metric:1
       RX packets:0 errors:0 dropped:0 overruns:0 frame:0
       TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
       collisions:0 txqueuelen:100
       Interrupt:14
```

7. Finally, we performed another **`QUERY NIC`** command and noted that this time, we had an IP address bound to the virtual NIC.

```
#CP Q NIC 500 DETAILS
Adapter 0500  Type: HIPER   Name: UNASSIGNED  Devices: 3
Port 0 MAC: 00-04-AC-00-00-05  LAN: SYSTEM ITSOLAN    MFS: 16384
Connection Name: HALLOLE  State: Session Established
Device: 0500  Unit: 000   Role: CTL-READ
Device: 0501  Unit: 001   Role: CTL-WRITE
Device: 0502  Unit: 002   Role: DATA
Unicast IP Addresses: 192.168.0.10
```

This Linux guest can now communicate with any other member of the virtual LAN without requiring static routes to individual machines or specific COUPLE statements to link it to other guests.

17 Roadmap

In this chapter we discuss some areas we would have liked to have investigated further. Some research was done to test "good ideas." In some cases experiments were done and prototypes were coded to verify the ideas. We believe these areas could allow significant improvements to running Linux images on VM, but time did not allow us to complete the work.

17.1 Ability to reconfigure CTC and IUCV

In "Linux as a virtual router" on page 68 we drew attention to the lack of ability to reconfigure the CTC and IUCV drivers.

The way that these drivers work is that connections are defined when the device driver is loaded. The restrictions are different for each driver:

CTC The virtual CTCs must be defined at Linux IPL time because the current Linux kernel does not properly handle dynamically defined CTC devices. The devices do not need to be coupled to the peer user ID until the moment you want to activate the connection, so you can postpone that decision and use the `hcp` command to do the couple. Thus, the restriction is only in the number of devices you want to use.

IUCV The IUCV driver does not use S/390 devices, so it is not subject to restrictions in the device layer of the kernel. The current driver, however, requires all peer user IDs to be specified when the driver is loaded. You can unload the driver and specify additional peer user IDs, but that means you must bring down the other IUCV connections in this image.

We believe it would not be very difficult to change the IUCV driver such that you specify only the number of connections when it is loaded, and specify the peer user ID just before the `ifconfig` command by writing into a /proc entry. This would give the IUCV driver at least the flexibility of the CTC driver.

There are also some scaling issues with the CTC driver that prevent this from being effective. When a number of connections are down and the virtual router is trying to establish the connection, this appears to keep the CTC driver from sending packets on the other connections that are up.

Currently a maximum of 8 or 10 connections is defined for the CTC and IUCV drivers. People have suggested that this could be changed (based on the fact that it is defined as a constant), but that might be less trivial than it looks. The CTC driver allocates 2 times a 64 KB buffer for each connection, so 100 connections would require 12.5 MB of storage. This would make it hard to create a small compact Linux router that can be kept resident by z/VM. The IUCV driver requires the user IDs to be listed when the driver is loaded. That causes some practical problems as well if you want to specify hundreds of user IDs.

It would be very attractive if z/VM would provide a kind of virtual network based on IUCV or some other interface to CP (as opposed to virtual point-to-point connections). Such a virtual network should offer broadcast capability as well so that a DHCP server (in a Linux image on VM, connected to that same virtual LAN) would provide the information to let a DHCP client in each Linux image configure the IP stack automatically (as suggested in 10.4.4, “Reducing the number of changes required” on page 220). Given the penetration of Ethernet in the Linux arena, it would be attractive if the Linux device driver would make it appear as a network interface on a virtual Ethernet LAN.

17.2 SOURCEVIPA equivalence for Linux

As discussed in 4.2.3, “Virtual IP addressing” on page 79, we can use a dummy interface in our Linux instance to provide a persistent traffic path for TCP/IP connections. This is used to provide a single addressing point for applications, even when multiple interfaces are used. It also allows for private IP addressing ranges to be used in the virtual routing setup, conserving the Internet address range.

While the dummy interface provides resilient connectivity support for incoming connections, it does not assist when the Linux instance establishes an outbound connection. This is because of the way the TCP/IP `connect()` function works. Part of `connect()` processing uses the TCP/IP routing table to determine which interface the connection request (TCP SYN packet) will be sent over. The IP address of this interface is used as the source address for the connection.

In z/OS and z/VM, this default behavior has been modified, adding extra processing to support SOURCEVIPA. z/OS and z/VM check the configuration for the interface chosen in `connect()`, and if SOURCEVIPA has been specified for that interface, the address of the appropriate VIPA will be used as the source address.

Note: This assessment of the SOURCEVIPA function in z/OS and z/VM has been done simply by looking at how the function works, not by actual inspection of the code. None of the authors of this redbook have access to z/OS or z/VM code, so the way the function is implemented may be different from this.

It would be possible to make the same changes to the Linux INET code to support the same feature. There are many considerations, however, that would have to be considered:

- The change would have to be duplicated in TCP and UDP (and may operate differently for both).
- IPV4 and IPV6 would need to be looked at.
- Other parts of the kernel might be affected. For example, the code that sends the SYN packet may be dependent on the interface address determined in `connect()`, and would have to be changed as well.

- Applications that initiate connections might not perform as expected.

A fairly large amount of work would be involved in making this kind of modification to the INET code. However, at the end of the task Linux would be able to benefit from the same high-availability connectivity as z/OS and z/VM for incoming and outgoing connections.

Even without a function like SOURCEVIPA in your penguin colony, though, you obtain high availability for incoming connections using the dummy interface ad previously described.

17.3 DCSS-mapped block devices

Reduction of the storage requirements ("footprint") of the Linux images is one of the prerequisites for effectively running a large number of Linux images on a VM system. One of the options for reducing the footprint is to share storage among the images. Sharing the read-only portions of the kernel as suggested in 10.7.1, "Using an NSS with just the kernel" on page 228 will help somewhat, but the kernel is only a small portion of the Linux image. It would be much more attractive to share the application code (the shared libraries and binaries).

17.3.1 Sharing between processes

When a process needs access to an executable or shared library, an `mmap()` function call is used to map this file into the virtual storage of the process. The file is not completely read into memory before the process starts, but portions are brought into storage when needed (like demand paging). When the process accesses a part of the file that is not in Linux storage, a page fault exception passes control to the kernel to start the I/O to read in a portion of that file and resume the process when the I/O has completed. In the case of shared libraries, another process might need the same portion of the mapped file and it would find the portion already in storage and not encounter a page fault. This means that portion of the file will be loaded into Linux storage only once. Popular portions of files will continue to reside in storage and will happen to be available for processes when needed.

This is something Linux does on each platform. It is not unique to S/390 and it does not exploit S/390 facilities other than the Dynamic Address Translation hardware (referred to as Memory Management Unit (MMU) on other platforms).

17.3.2 Sharing between images

This sharing process, described in 17.3.1, "Sharing between processes" on page 410, happens within a single Linux image. When Linux images are sharing disks with code, many of the Linux images will have read in the same portions of the same files. From a VM point of view, this means that duplicates of the same data is in the virtual storage of each of these Linux images. If the pages are referenced frequently enough by the Linux images, they will be part of the resident storage of the virtual machine. It would be very attractive if VM could play similar tricks as Linux does and map these virtual machine pages on the same real page frames. Since Linux is using normal I/O operations to read the data, it is not trivial for VM to recognize the pattern and perform a form of mapping.

Some benefit may be gained from VM Minidisk Cache (MDC) that will cache portions of the data in real storage page frames.

Note: One could argue whether MDC is effective in this case. Because Linux images already buffer these popular pages themselves, MDC will not notice that the page is more popular than other pages read once by a Linux image. The Linux I/O tends to be rather "MDC unfriendly" (except when using the diagnose interface), so this may be a moot point. Rather than argue, we probably should measure.

When the Linux image issues the I/O to get the data because it page faults, it is probably too late for VM to intercept and try to do smart things. This is further complicated by the fact that each Linux image will read that block of disk into a different page of the virtual machine storage.

One option would be to use the z/VM facilities to do something similar to the Linux `mmap()` function. The z/VM facilities, however, build on the XC architecture, and Linux for S/390 currently cannot run in that mode.

17.3.3 Using shared segments

Another z/VM facility to share storage is a discontiguous saved segment (DCSS). A virtual machine can "attach" a DCSS, which means that a defined part of its address space is mapped on a DCSS. The pages of the DCSS reside in a special spool file and are brought in when necessary through the VM paging subsystem, and will remain in storage when referenced frequently enough. Multiple virtual machines share the real page frames, much like the `mmap()`

approach in Linux. The DCSS is normally located outside the virtual storage of the virtual machine (i.e., at a virtual address beyond the size of the virtual machine). At boot time Linux will set up tables to map what it sees as real page frames, not segments attached to it after the boot process.

The way we can talk Linux into using a DCSS could be to have a block device driver. The block device driver will issue a Diagnose 64 call to attach the segment when the device is opened. The way to access the segment would be through the **ioremap()** function in Linux (this is used on PC platforms to access memory on PCI cards). The kernel currently does not export the symbol for device driver modules to use it, but this is trivial to do in arch/s390/kernel/s390_ksyms.c. We made that change and the code appears to work as expected, in that a device driver module can access the data in the segment.

The first attempt to implement this DCSS block device was to build a device driver on top of the device file system (devfs) so that segment names would show up in the /dev tree when attached. While this might be an elegant approach for a production driver, it turned out to be "a lot of work."

The second attempt involved taking the XPRAM device driver that is part of the Linux for S/390 source tree, and change it to use a DCSS. The module was changed to take the names of the DCSS as a parameter when loading.

```
Aug  9 01:05:34 tux60000 kernel: dcssinfo:trying to load module
Aug  9 01:05:34 tux60000 kernel: dcssinfo:initializing:
Aug  9 01:05:34 tux60000 kernel: dcssdebug:dcss: this is 0 TUXTEST
Aug  9 01:05:34 tux60000 kernel: dcssdebug:  major 34
Aug  9 01:05:34 tux60000 kernel: dcssinfo:  hardsector size: 4096B
Aug  9 01:05:34 tux60000 kernel: dcssdebug:diag64 TUXTEST  is   0 20000000 200fffff
Aug  9 01:05:34 tux60000 kernel: dcssinfo:  1024 kB expanded memory found.
Aug  9 01:05:34 tux60000 kernel: dcssdebug: device(0) offset = 0 kB, size = 1024 kB
Aug  9 01:05:34 tux60000 kernel: dcssinfo:Module loaded successfully
```

While writing the redbook the device driver was already doing the **ioremap**() call and it kept the pointer to the mapped memory for the segment. The **request**() function was changed to copy from and to mapped memory. In fact, we do not want to copy the page into Linux storage. We need to convince the system to use the page sitting outside virtual storage. Some extra stuff is needed to load the segment in non-shared mode to have it writable and to issue the SAVESYS when the segment is detached.

Another interesting application for the DCSS driver would be to attach a segment that has been prepared with the **mkswap** command. This would be like a swap disk in virtual disk (VDISK), but without the expensive channel programs to drive it.

17.4 Shadowed disk support

In the case where many hundreds to thousands of Linux images are essentially similar installations, with the exception of a few configuration files, it would be advantageous to have some means of minimizing the number of disk devices needed. To this end we began investigating the feasibility of a "shadowed" disk driver. This driver would use a common master drive as read-only data, but writes would be directed to a guest-specific shadow disk. Subsequent reads would retrieve changed blocks from the shadowed disk and unchanged blocks from the master disk.

One significant possible benefit to using a "shadowed" approach is that it becomes a much simpler matter to upgrade software for all the systems simultaneously. Several issues still remain; for example, configuration file format changes will still require some careful consideration and thought. In the main, though, it will be highly desirable to be able to "insta-patch" all the Web servers at once. It also greatly simplifies the management of hundreds of images by ensuring that they are all running the same version of software.

Another use of a disk shadow is to allow extremely simple recovery to a "known good" state. If one of the guest images manages to damage its configuration to the point that it cannot be easily repaired, or even that it will no longer boot, all that must be done is to delete the shadow and replace it with a blank, new shadow. In a matter of seconds, the system is up and running again with a known baseline configuration. With some of the other configuration automation techniques we have discussed, the newly "rebuilt" machine could even automatically make the first changes (e.g. IP address) such that when it comes up it is already alive and well on the network. Carrying this idea even further, for well-defined servers all carrying out a similar function (i.e., a cluster of Web servers) an automated process could automatically "resurrect" a failed server by bringing online a new, freshly configured image while retaining the old shadow copy for the system administrator to look at to determine the cause of the failure.

Combined with the DCSS driver in "DCSS-mapped block devices" on page 410, this would not only save disk space, but would also allow portions of the common master driver to reside in storage that is shared among Linux images.

One aspect of the driver that is somewhat more complex is maintaining shadow consistency with the master disk. If the master disk changes at all, then the block map on the shadow is invalid. Some tools could be developed that will:

- Generate a new shadow based on the old shadow and master
- Rebuild shadow/master to reclaim unused space (this will depend on how the file system behaves)

- Add new shadows to an existing shadow (multiple shadow disks to expand shadow capacity)
- Factor common changes out of multiple shadows to generate a new, more efficient master
- Simplify shadow management for the system administrator

A prototype implementation using the Linux md driver was developed during this residency, but was not completed in time for significant results to be documented in this book. Development is continuing, and we hope to be able to publish results at some future date.

17.5 File system access tool for systems management

Linux currently does not properly handle the dynamic LINK and DETACH of minidisks as CMS users are used to having.

There is a need to allow one Linux image to access the file system of another (not running) Linux image. When authorized to do so, users can link to each other's minidisks. Unfortunately, the current implementation of Linux for S/390 does not support dynamic linking to a minidisk and use of this disk in the DASD driver. Even when Linux would correctly handle the machine check interrupts involved with configuration changes of the virtual machine, the DASD driver would still need to be unloaded and reloaded with new parameters (which is not possible when the root file system is on disk).

The possibility to do this builds on VM facilities and is therefore not present on other Linux platforms.

17.5.1 Possible use for the utility

In general, systems management can be simplified when a running Linux image can dynamically access the file system on a minidisk other than the minidisks linked to the virtual machine at IPL. The restrictions of the kernel and DASD driver currently do not offer this option.

Ad-hoc file system repair activities

For CMS-based applications, a VM systems programmer or application programmer would want to link and access minidisks of service machines to investigate issues and fix problems. A similar facility would be useful for systems management of Linux images on VM.

Alternative data sharing

Instead of sharing data via permanent network connections, applications can exploit VM facilities to link to the file system of another Linux image. This should be done while the other Linux image is not running (in which case a network connection would not be possible, anyway).

Automated configuration changes of cloned images

When a new Linux image is created by copying the disks of an existing Linux image, several files must be customized for the new image (IP address, host name, root password). This process can be simplified if normal Linux utilities can be used to apply these changes.

DirMaint-controlled file system copy

For minidisks in CMS format with CMS files, DirMaint can automatically copy the files when the size of a minidisk needs to be increased. If a Linux system can dynamically link to the old and new minidisk, the same function could be implemented for minidisks containing a Linux file system. This is very important for storage management.

17.5.2 Design outline

A program on the user's Linux image (the master) will take the arguments needed to access the minidisk (user ID, virtual address, optionally read password, mount point). Because of the limitations of the DASD driver to dynamically add devices, a new Linux image (the worker) must be started when a minidisk needs to be accessed. This new image links the proper minidisk and IPL Linux from an NSS that also contains a RAMdisk image. The DASD driver can now be loaded and the file system on the minidisk can be mounted in the root file system. The new Linux image connects to the master Linux image via an IUCV connection and exports the mounted file system via NFS. The program that initiated this can now mount that exported file system in its own file system.

With an enhancement to pick up the IPL parameters, as shown in 10.7.3, "Picking up IPL parameters" on page 230, we do not even need to boot with initrd. The additional disks can be specified in the IPL command: an extra option could be passed to the boot scripts to indicate what action is required from the worker.

17.5.3 Detailed design

The IPL of the worker should be reasonably fast to make this work. We have seen that booting a kernel with an uncompressed 30 MB RAMdisk image from NSS can be done within 3 seconds when the pages needed are already in storage. The RAMdisk image can probably be made smaller, which would further speed up the process. If necessary, the system can be IPLed from disk if that turns out to be faster.

To get IUCV connections between worker and master, the proper IUCV statements must be in the CP directory. The netiucv driver requires all peer user IDs to be defined when the driver is loaded. These restrictions suggest that we create a few of these workers for each virtual machine that must use this facility. These workers should be dedicated to this master. This simplifies security issues, because the workers can have the same authorization as the master. The workers do not need disk space, so the cost of half a dozen workers for each system administrator is not much.

The program to initiate this function can find the first free IUCV connection and XAUTOLOG the corresponding worker. The XAUTOLOG command can be issued using the **hcp** command in the cpint package. An **ifconfig** command can be issued to define the IUCV connection. Root authorization in Linux is normally needed for the **hcp** command, but that applies to the final **mount** command as well.

17.5.4 Additional points for the implementation

Booting from RAMdisk may be too restrictive for a full implementation. The alternative is to keep a separate root device for each worker. Because of the restrictions of the DASD driver, the kernel must be IPLed from NSS and a small modification must be made to the bootstrap such that it takes parameters from the IPL command to insert them in the command line for the kernel. This is not rocket science and has been done before. It would be very useful for other purposes as well and greatly simplify the IPL from NSS.

To make the facility more generic, it must be possible to pass the command to be executed on the worker. By default, this would be the script to set up the network connection with the master, but it will also be possible to have a command executed without connecting the network. This results in a kind of background processing outside the virtual machine.

To copy a file system to a new disk, two minidisks must be linked and the tar and untar for the copy must be issued.

An elegant solution would be if the worker could mount part of the master's file system so that local scripts and commands in the master could be executed. Different levels of commands and libraries could make this very complicated.

Some complications with respect to UID and GID may occur when files in the target file system are modified or created.

The private IUCV network between each worker and the master is a secure solution and there are no security risks for the NFS export in the worker.

The DASD driver currently has a delay of 10 seconds in the startup. The reason for this is unclear. It should be possible to remove this delay or improve the process.

Most parameters will be fixed for each worker and could be taken from a file on the 191 disk using the cmsfs driver (or computed from the user ID of the virtual machine). The actual parameters for the command could be passed to the Linux boot scripts via CP Globalv variables (for example, the TAG of a virtual device).

There are no specific requirements for the level of Linux running in the worker. The only requirement is that it should be reasonably current, but it probably can be an off-the-shelf kernel with RAMdisk. With recent levels of Linux, the incompatibilities between different versions of the DASD driver seem to be fixed.

17.6 Synergy with CMS Pipelines

CMS Pipelines is the z/VM built-in productivity tool for CMS. Many CMS applications are written to use *CMS Pipelines* facilities. In the hands of an experienced plumber, *CMS Pipelines* is also very useful for ad-hoc analysis of the output of experiments, as we did while writing the redbook.

While *CMS Pipelines* design was originally mildly inspired by the concept of pipes in UNIX, it has been enhanced significantly beyond that. It features multistream pipes as well as pipes that dynamically modify the topology of the pipeline by adding segments to the running pipeline. *CMS Pipelines* has drivers to interface with various z/VM facilities.

The *CMS Pipelines* home page is hosted by Princeton University at

`http://pucc.princeton.edu/~pipeline`

The home page has pointers to several papers on *CMS Pipelines*. It also offers the latest version of the *CMS Pipelines* Runtime Library free for download to allow customers with slightly older levels of CMS to run the latest version of *CMS Pipelines* on their system. The z/VM documentation comes with two publications for *CMS Pipelines* users:

- *CMS Pipelines Reference*, SC24-5971
- *CMS Pipelines User's Guide*, SC24-5970

Experienced plumbers tend to prefer the documentation in the "*CMS Pipelines* Author's Edition," which is available on the VM Collection CDs as well as on the home page:

```
http://pucc.princeton.edu/~pipeline/pipeline.book
```

In an earlier residency, John Hartmann, the author of *CMS Pipelines*, created the "Plumber's Workbench" (PWB). It is a workstation application with CMS in its back room. It allows access to all of *CMS Pipelines* from the workstation. PWB is available for OS/2 and MS Windows workstations. John Hartmann also worked on a PWB client for Linux (including Linux for S/390). This gives Linux applications access both to *CMS Pipelines* and to CMS applications.

We did some experiments with the code and it certainly works, though the syntax of the `vmpipe` command is error prone due to the overloading of the "|" character. To manipulate Linux data with *CMS Pipelines*, the PWB client sends the data over a TCP/IP connection to the PWB agent running in a CMS user ID. Since Linux for S/390 can have a fast connection to the CMS user ID, this is less likely to be a showstopper.

For people with the appropriate skills, it could be very attractive to use those skills for their Linux work. The ideal would be to have a Linux implementation of *CMS Pipelines*. This is currently not available.

17.7 Make DirMaint the registration vehicle

Several things need to be arranged in VM to create a new Linux image, as shown in 9.2, "Things to do for new Linux images" on page 190. It could be attractive to enhance DirMaint so that it takes the role of central registration vehicle in VM.

To do this, there would be a need for new options in the prototype files that invoke exit routines to do the "things" that are needed for the new Linux images. For example, a new statement in the prototype file could look like this:

```
ADDIP subnet-17
```

This could invoke an exit that allocates a new IP address for this image in a specific subnet, and creates the definitions in the TCP/IP configuration files, DNS, DHCP, etc. If real network interfaces are used, the exit should probably pass the correct DEVICE statements to DirMaint to have these included in the directory entry for the new image.

Because we do not know yet what is needed, a flexible implementation should make the statements, as well as the exit routines, user-defined.

The same processes obviously would take care of removing the definitions when the Linux image is deleted with DirMaint, or when the creation process is rolled back for some reason.

Discussion with people associated with DirMaint development showed they are aware of the need to make DirMaint assist in cloning Linux images. One of the possible enhancements could be to have the DATAMOVE virtual machine create new minidisks as a copy of an original disk rather than format them.

A

Linux Community Development System

The Linux Community Development System (LCDS) was created by a team of IBMers in the spring of 2001. Its purpose was to provide the open source community with free access to Linux on a mainframe. In this chapter, we describe the experiences and lessons learned.

Components of the system

The following sections discuss the system components.

Linux on a mainframe for free

The first component of the Linux Community Development System is the Linux part—in other words, making Linux systems on S/390 available to the open source community. Here is the invitation as it appears on the LCDS home page:

http://www-1.ibm.com/servers/eserver/zseries/os/linux/lcds/index.html

```
 Welcome to the Linux Community Development System (the 'Service'), a Service
provided by IBM. The Service provides you with access to a Linux on S/390
environment for the purpose of providing the Open Source community with a
platform to develop, port and/or test drive your products or applications on
this platform. We anticipate the majority of users to include entrepreneur
developers/vendors that otherwise might not have the opportunity to test/port
their code to the S/390 platform. However, we invite all interested parties
that meet the established terms and conditions to register and experience
'Linux for S/390'.
```

Community: the global response

The LCDS home page opened for business on May 22, 2001. In three days the page had received 27,000 hits. Not all of those hits led to a request for a Linux system, but as of late July 2001, there were a little over 600 images running on the system. The users are a truly global community, representing these countries:

- Angola
- Argentina
- Australia
- Austria
- Belgium
- Brazil
- Bulgaria
- Canada
- Chile
- China
- Croatia
- Czech Republic
- Denmark
- Dominican Republic
- Egypt

- Estonia
- Finland
- France
- Germany
- Great Britain
- Greece
- Hungary
- Iceland
- India
- Indonesia
- Ireland
- Israel
- Italy
- Japan
- Malaysia
- Mexico
- Netherlands
- New Zealand
- Norway
- Pakistan
- Peru
- Poland
- Romania
- Russia
- Singapore
- South Korea
- Spain
- Sri Lanka
- Sweden
- Switzerland
- Taiwan
- Thailand
- Turkey
- Ukraine
- United Arab Emirates
- United Kingdom
- United States
- Venezuela
- Vietnam
- Yugoslavia

Development: what is being tried

The users represent a global range of applications as well as geographies. Film production, aerospace, pharmaceutical, insurance and banking companies are participating, as well as many universities from around the world, and gnu.org. The following list shows the variety of reasons users gave for wanting a Linux system on S/390:

- Rotund prime sequencing
- Samba, Apache, Sendmail
- Digital document system
- C++ compiles, C code front ends, general tests
- Cryptography, security, intrusion detection
- Java, XML, XMK
- Wireless, voice, embedded devices
- Working on, experimenting with, testing...
- Want to see:
  - If this works
  - How easy it is
  - If I can port
- Pong
- Oriental herbology

System: what it is being run on

The operating system software that runs the LCDS is z/VM. This allows hundreds of unique Linux images to exist on one physical machine. The hardware is S/390 technology, not zSeries. The techniques learned and refined on this system will deliver even better results as they are deployed on 64-bit, zSeries hardware. The full details of the system are described in the following section.

Technical implementation of the LCDS

Hardware specifications

CPU

The LCDS is hosted on a 9672 G6 Model ZX7 machine. This hardware has IEEE floating point and is 31-bit technology. It is a 10-way processor, with 32 GB of memory. It is part of the S/390 family.

DASD/disk capacity

The Linux images and z/VM operating system have access to disk storage on a Shark (Enterprise Storage Server) Model 2105-F20 configured with 2.1 terabytes of capacity.

Network

The network design of the LCDS had to accommodate an interesting mix of legal and physical characteristics. The z/VM operating system had to be accessible to IBM employees who were setting it up and administering it on the internal IBM network. The Linux guests had to be on a direct connection to the Internet, and there could be *no* connection between the two (Internet and internal IBM). The physical constraints included the use of an existing T1 connection to the Internet over a 3172 LAN Channel Station. Although there was only one physical connection to the Internet, the design had to accommodate hundreds of unique IP addresses—one for each Linux guest.

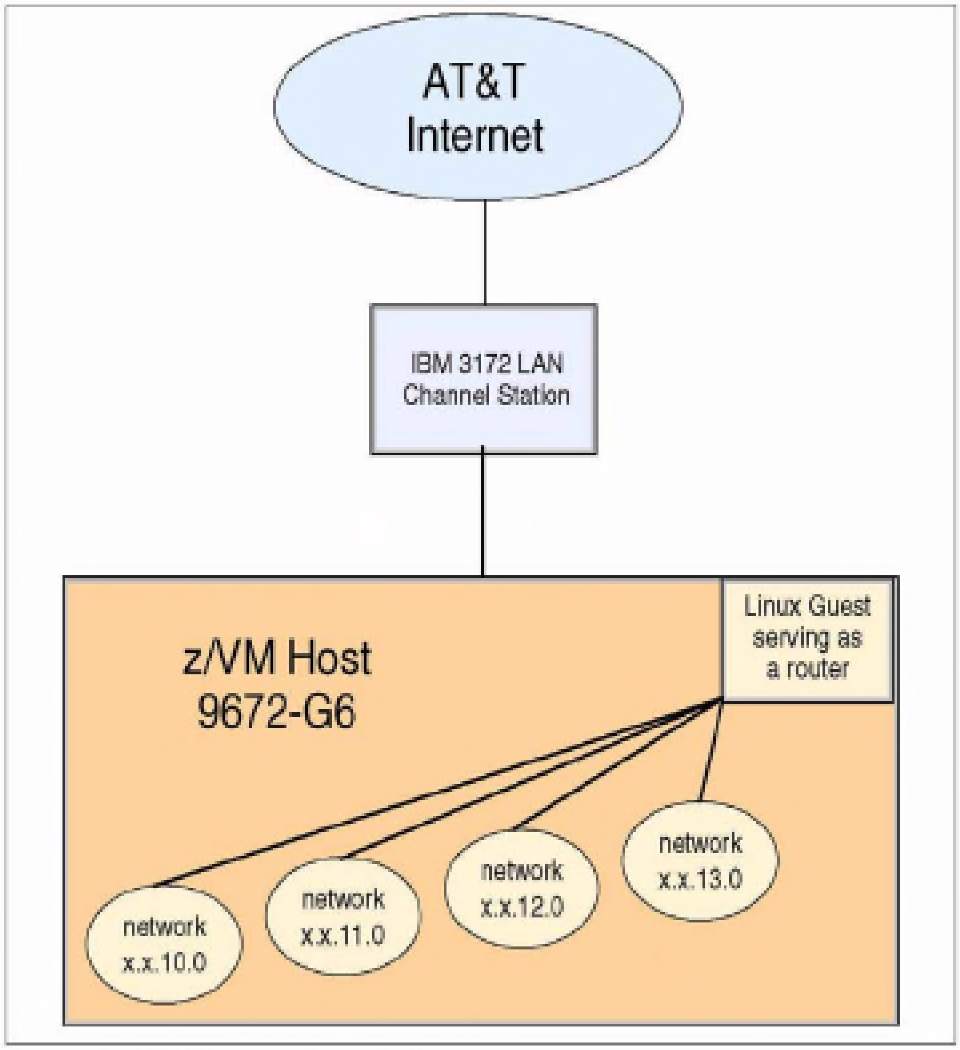


Figure 17-1 LCDS connection to the Internet

A total of 2000 IP addresses were obtained. These addresses are spread across 20 routers in groups of 100. All the routers to the Linux guests are themselves Linux guests. This allows the exploitation of z/VM architecture by networking all the Linux guests through virtual, or software-defined, connections. The LCDS uses IUCV-type connections, as opposed to Virtual Channel To Channel (VCTC). The IUCV connections were found to initialize quicker and recover more automatically, for example after a reboot.

One task of the Linux routers is to do Network Address Translation (NAT). Within the LCDS virtual network, the Internet addresses are translated to class A 10.x.x.x addresses. This means the 2000 purchased Internet addresses do not have to be consumed by internal routers, name servers, and gateways.

The other advantage of the 10.x.x.x addresses is that they cannot be routed. The network traffic isn't going to "leak". Within the LCDS network, ICMP is turned off. This means ping cannot be used to discover the network topology. Once users of the LCDS are logged on to their Linux image, they can access any address on the Internet using any protocol that is authorized on the target server.

Within the LCDS network, the Linux guests are architecturally isolated from each other. This isolation is achieved under the control of z/VM. Each Linux guest can only have access to a resource that is defined to it. There are no pathways available for a Linux guest to communicate with, access, or modify any resource that is not defined to it.

The domain name server is within the scope of the LCDS network. Since this is a very dynamic setup, with Linux images being defined by the dozen, it seemed better to control the configuration of the name server within the LCDS staff.

Back-routers are used to contain and shorten the network traffic among the Linux guests. They allow an asset to be shared without exposing the traffic (and asset) to the Internet.

As mentioned before, ping (ICMP) is turned off. The only access to log in as root is through SSH access. Telnet and ftp are enabled, and can be used once you have successfully logged in.

Staff and processes

Team

The Linux Community Development System was designed, implemented, and is administered by a small team based mostly in Endicott, NY. Everyone involved made their contributions while still keeping their day job, in keeping with the open source community tradition. They are officially part of the Advanced Technical Support (ATS) organization. The team members are:

John Sutera	Manager
Bill Bitner	Performance, VM Development
Pamela Bryant	PROP
Steve Gracin	Networking, RedHat, WebSphere Application Server
Stanley Jones Jr	Registration, Lotus Notes work flow
Bob Leicht	Enrollment, SSH, System
Richard Lewis	VM, Networking, System
John Schnitzler Jr	Hardware, IOCP, SSH
Jon vonWolfersdorf	Networking, LCDS Home Page

Pam Bryant and Richard Lewis are based in Gaithersburg. The rest of the team is in Endicott, as is the hardware.

Register

Access to the LCDS is open to anyone (except internal IBMers), anywhere in the world. A form is provided on the Internet, asking for a minimal amount of information, including the user's purpose in testing Linux on a mainframe. Once the form is filled out, a Lotus Note is sent to an administrative ID. This ID is monitored by two team members. When a request for a Linux system is received, it is converted to an entry in a Lotus database. The request is reviewed, then accepted or rejected. An accepted request triggers a Lotus Notes agent to assign an ID and password, which are sent to the requestor.

Generate a system

One requirement for access to the LCDS is SSH (Secure SHell) encryption. The requestor is responsible for getting a terminal emulator that is SSH capable, and for generating public and private keys.

Note: The freeware program PuTTY does both, with the PuTTYgen.exe and putty.exe programs. They can be found on the Web at:

    http://www.chiark.greenend.org.uk/~sgtatham/putty

When requestors of a Linux system receive the note with the ID and password, they use these to sign on to a secure Web page. This Web page is on the LCDS z/VM system where the Linux guests are defined and run. The ID and password are validated, then a new Linux guest is generated. The automated generation process creates a VM guest Linux user, with associated disk space, virtual memory, and a network address.

The CLONEM exec takes advantage of CMS techniques such as PIPES. It also uses drivers from the open source community. Rick Troth of BMC Software has written a driver that allows CMS files to be read from Linux, and Neale Ferguson of Software AG has written a device driver that allows CP commands to be issued from Linux (see Appendix B, "Using the hcp command" on page 437 for more details.) This permits an architecture where customization information is managed from z/VM, which allows one person to administer hundreds of systems from one central focal point.

There are two other key pieces of interface technology. The first is the Web page mentioned earlier. It is CA's VM:Webgateway. It allows the requestor information to be collected in the z/VM environment and propagated to each new Linux guest. The second interface is specialized customization to the boot process of Linux. Richard Lewis of the IBM Washington System Center created a shell script that runs very early in the boot process, before the network connection is started.

The script reads the customization information on the Linux guest's A-disk (gathered from the Web page). It assigns the correct network address and essentially answers the questions a user answers when installing a Linux distribution.

The first access that requestors of the Linux guest have to their system is when they SSH into it as root. They do not have to go through SuSE or Turbo panels to configure the distribution. This automated process provides a high level of security, as it shields the user from the underlying z/VM system, prevents network configuration errors (accidental or deliberate), and reserves control of the configuration process to the system owner.

Help, Support

When someone downloads Linux and installs it on their home PC, they understand they are on their own as far as technical support is concerned. It is their responsibility to find (or contribute!) answers through the use of news groups and mailing lists. The same is true on the Linux Community Development System. Free access to a somewhat hard-to-acquire and expensive hardware platform has been provided by IBM. The goal is to prove that Linux on the mainframe is the same as any other Linux. "The same" includes the same style of support. There is a forum on the LCDS Web site, where community members share their experiences. They can describe problems they have encountered, and may receive technical help. However, no one is restricted to using only that forum, and there is no guarantee they will get an answer there. Technical support comes from the open source community at large.

It often happens that a Linux system crashes after some user tests or modifications. Since the requestor of an LCDS Linux guest machine does not have access to the "big red switch" (the power switch) to do a reset of the hardware, it was necessary to provide a way to reboot a seriously incapacitated Linux guest. The REBOOT service machine is accessed using SSH and accepts the name of your Linux machine as the login ID. There are four options to choose from. You can:

- Exit without doing anything.
- IPL your Linux with a rescue system. This reboots a Linux rescue system from a RAM disk.
- IPL your Linux from a specified device. This performs a normal reboot.
- Force your Linux offline; do not restart it. This forces a shutdown -h now, which will then require a reboot with either option 2 or 3.

As of the time of writing, there were over 600 Linux guests running on the LCDS. At no time has a reboot of one Linux guest impacted any of the others. The architecture of z/VM allows complete freedom for individual users to try any high-risk change they like, while completely isolating the other Linux machines from any impact of that change.

Monitoring

The LCDS usage has been monitored both interactively and using accumulated statistics. The historical data has turned up interesting facts, such as z/VM setting a new record for paging of 259,000 pages per second. The previous record was 45,000 pages per second. This is a testament to the robustness of the z/VM architecture.

Network monitoring showed no particular bottlenecks. Once network traffic is within the virtual network of z/VM and the Linux routers, communication is at very high speed and bandwidth. The physical limitation is the capacity of the T1 line. A sample graph of the daily usage is shown in Figure A-1.

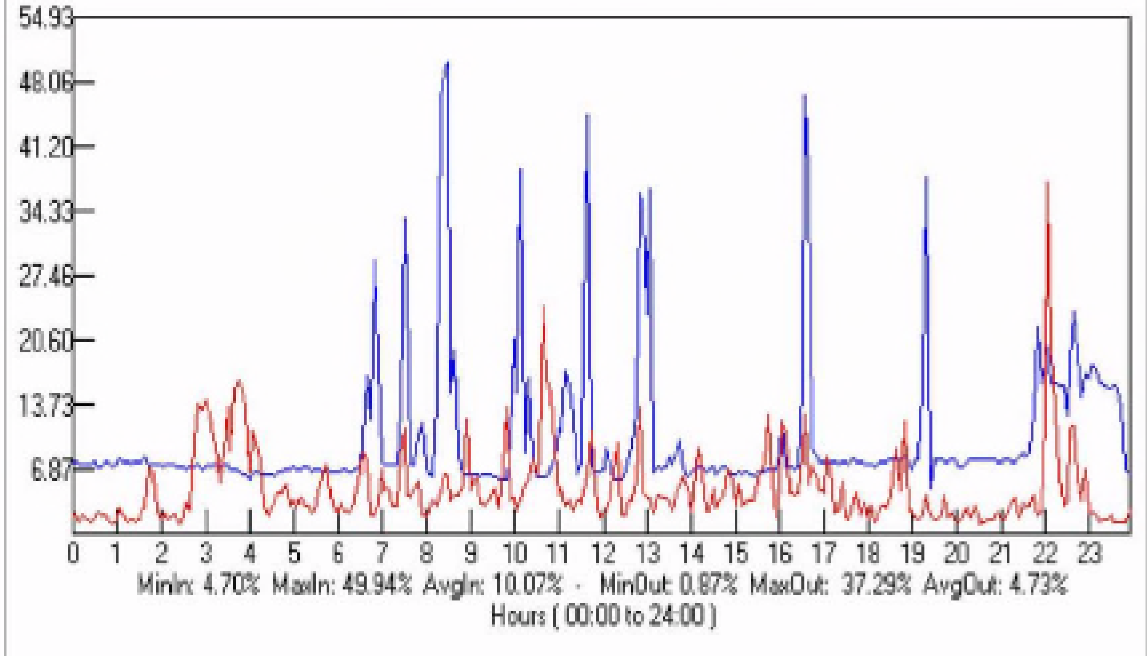


Figure A-1 percent utilization of the T1 line

The higher line (blue) is inbound traffic, and the lower line (red) is outbound traffic. The usage pattern is a pretty typical workday series of peaks and valleys as people come in to work, come back from lunch, and hurry to finish something at the end of the day. The graph of weekly activity showed that Saturday was one of the busier days—an interesting finding.

Interactive observation of the Linux guests was done by the LCDS staff. On occasion, a spike in CPU activity would be noted and investigated. In a production ISP/ASP environment, this process could be automated. The CP ACCOUNT facility also collected the CPU activity for each guest, and could have been used to track high CPU usage.

One area that several people would like to explore is the possibility of using the Monitor facility. This system interface is used in CMS to report more detailed usage information from within the virtual machine. CMS reports statistics on its Shared File System, as an example. Neale Ferguson has started work on a Linux driver to talk to the Monitor Application Data interface. Some members of the LCDS team hope to cooperate in refining this driver.

Termination

Users are given access to a Linux guest for 30, 60, or 90 days. At the end of the time they requested, the image is deleted from the system. It is the users' responsibility to retrieve any data they wish to keep. The automated process to return resources to the system is fairly basic, since there is no requirement for any information to be preserved.

Evolution and lessons learned

The LCDS has been a very dynamic experiment, and a fast-changing environment. A lot has been learned, both about z/VM running Linux guests, and about the nature of the Linux kernel. Several refinements to z/VM tuning were made, and there was at least one contribution to the Linux kernel.

z/VM

The IBM labs have been doing validation of the early Linux code drops, even before they go to the various distributors. The LCDS staff has been actively engaged in that validation. Two areas of interest have been the DASD drivers and how Linux behaves with mini-disk caching. There was an iteration of the kernel that did not respond well to mini-disk caching, but that was quickly resolved. The code drops that are being tested as of this writing will be out in the fall of 2001.

z/VM Release 4.10 includes an enhancement to CCW translation. Code was written for VSE guests that improved I/O to DASD devices. (VSE is another operating system.) This fast path code was only available for DASD, since VSE systems typically do a lot of data processing, and very little network activity. IBM developers working with Linux under z/VM realized that although VSE did very little network activity, Linux does a lot of network activity. They thought of making the I/O commands to network devices eligible for the fast path code. This was done for LAN Channel Station (LCS) and CTC connections and a 40% improvement in processor efficiency for network I/O was achieved for Linux guests.

Linux

The Linux kernel has a bit of logic that wakes up to check for work. It is referred to as *jiffies* or the *jiffies timer pop*. This results in wasteful overhead on a mainframe processor that is optimized to respond to interrupts. David Boyes of Sine Nomine Associates has experimented with altering the Hz value in the Linux kernel. The default value is 100, but it has been set to a value such as 16. This means more useful work is done, and there is less dispatching of a Linux machine that has no work to do, simply to check for work. Setting the Hz value too low can be a problem. Responsiveness goes down, and some things stop working. At this time the LCDS timer is set to its normal default value of 100, in order to maintain the consistency of Linux on other platforms.

A patch has been submitted to the Linux organization that implements a much different scheduling technique. It is not in the platform-dependent code. It would affect all platforms, and is designed to help all platforms that have multiple processors, but it has not been accepted into the kernel. Users or distributors do have the option of including the patch, which applies to Linux 2.4. There is a great deal of interest in this area of the code, so a lot of innovation can be expected. The Linux news groups and mailing lists will have the most current information.

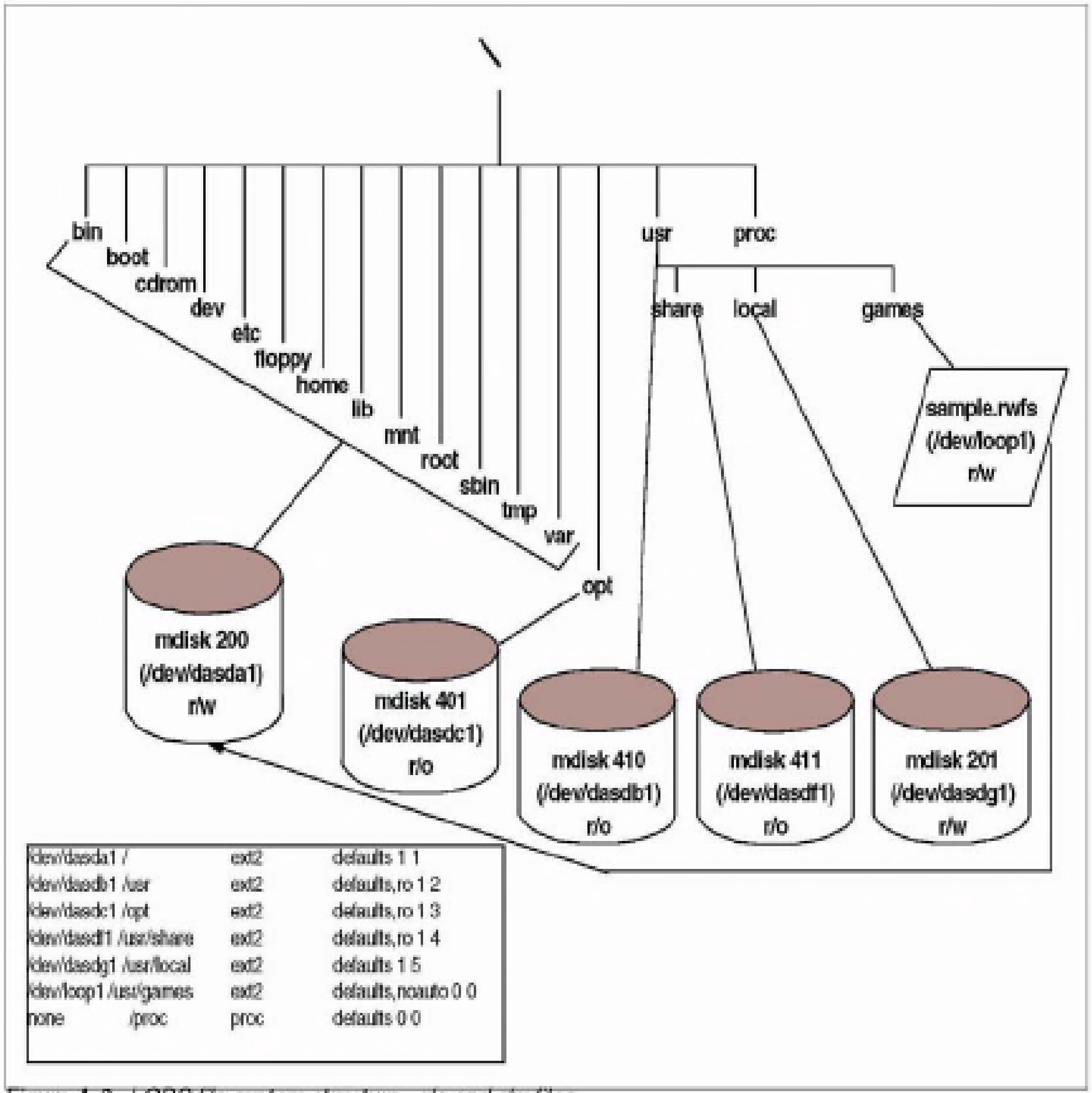
Tuning both

One problem that was encountered very quickly was default settings for `cron`. There is a security package that scans for trivial passwords, which by default was started at midnight. On a single Linux system, this is a very conscientious thing to do. When hundreds of Linux guests on the same hardware all do the same thing, at the same time, it is a very bad thing to do! The system spiked to 1000% busy (which means all 10 CPUs were at 100% utilization) and paging went to 259,000/s. The system did not crash. Eventually all the Linux guests completed the scan for trivial passwords, and CPU busy returned to normal. It took a bit of investigation to discover the cause of the activity, but it was simple to fix. As upgrades are made and Linux is reinstalled, the defaults in `cron` are checked for tasks that should not be scheduled to run.

A characteristic of Linux is that the more memory it has allocated, the more it will use that memory for file caching. In some architectures, this is very desirable. However, when running under z/VM, it is more efficient to use the mini-disk caching capability. In fact, when a change was made to the LCDS system to reduce the amount of mini-disk caching storage and give it to paging, performance actually got worse. Paging and file I/O are covered extensively in Chapter 3, "Virtual server architecture" on page 45.

Structuring the file system to save space

One concern the LCDS staff had was to not use up the available disk space any faster than could be helped. After some experimentation, the structure shown in Figure A-2 on page 434 was chosen. The greatest amount of file space is used in the part of the directory tree under /usr. Therefore, much of that part of the file system is mounted read only (r/o).


\
bin
boot
cdrom
dev
etc
floppy
home
lib
mnt
root
sbin
tmp
var
opt
usr
proc
share
local
games
sample.rwfs (/dev/loop1) r/w
mdisk 200 (/dev/dasda1) r/w
mdisk 401 (/dev/dasdc1) r/o
mdisk 410 (/dev/dasdb1) r/o
mdisk 411 (/dev/dasdf1) r/o
mdisk 201 (/dev/dasdg1) r/w

```
/dev/dasda1 /             ext2     defaults 1 1
/dev/dasdb1 /usr          ext2     defaults,ro 1 2
/dev/dasdc1 /opt          ext2     defaults,ro 1 3
/dev/dasdf1 /usr/share    ext2     defaults,ro 1 4
/dev/dasdg1 /usr/local    ext2     defaults 1 5
/dev/loop1 /usr/games     ext2     defaults,noauto 0 0
none         /proc        proc     defaults 0 0
```

Figure A-2 LCDS file system structure - r/o and r/w files

The attribute of being read/only is enforced at the z/VM level. Editing the /etc/fstab file and changing /usr to be read/write does not affect access to the underlying physical device. To provide users with their own, private read/write files, a separate device is mounted at /usr/local. This allows about 80% of the file system to exist in one copy, shared read/only, by all 600 Linux guests.

Perhaps the most outstanding result of running the LCDS has been to witness the integrity of the architecture. The inadvertent stress test that drove the system to 1000% busy did not result in an outage. We have seen how z/VM ensures the isolation of each server. This allows for the most effective economy of scale in consolidating servers in one hardware footprint, while at the same time permitting owners of the Linux guest the same freedom and autonomy they could have on a workstation-based server.

Summary

There are many options to the S/390 Linux architecture and system design that may be feasible for accommodating multiple Linux guests. The team utilized existing hardware, software, Linux distributions and network topology that was available. The LCDS demonstrates S/390 and z/VM versatility, strength and security within an existing I/T environment that is worldwide in scope and responsiveness.

B

Using the hcp command

Neale Ferguson of Software AG wrote the package cpint for Linux for S/390 to provide an interface to some CP functions. One of the components is the **hcp** command, which uses the Diagnose 8 interface to issue CP commands on behalf of the guest virtual machine that runs the Linux image. The cpint package for the 2.2 series kernel can be downloaded from the following Web site:

`http://penguinvm.princeton.edu/programs/cpint.tar.gz`

To use the package you need to compile the programs and install them on each of the systems where you want to use the package. This can be very impractical if you have many different systems. It is possible to package the compiled binaries and installation script in an rpm package that can be installed along with the other packages on each Linux system.

Creating an rpm package for cpint

An rpm package consists of the source files, patches, and the spec file that describes how to apply the patches, how to build the binaries and how to install the binaries on the system.

For a Red Hat installation, the rpm files reside in the /usr/src/redhat/ directory. In a SuSE installation, these files are in the /usr/src/packages/ directory.

To build an rpm package from the source files, we need to create a spec file:

Example: B-1 The cpint.spec file

```
Vendor:       Rob van der Heij
Distribution: SuSE Linux 7.0 (s390)
Name:         cpint
Packager:     rob@rvdheij.com

Copyright:    GPL
Group:        Applications/System
Provides:     cpint
Autoreqprov:  on
Version:      2.2.16
Release:      3
Summary:      Device driver for CP diagnose functions
Source:       http://penguinvm.princeton.edu/programs/cpint.tar.gz[1]
Buildroot:    /var/tmp/cpint-root/

%define cpintmajor 254

%description
Author:Neale Ferguson <Neale.Ferguson@SoftwareAG-USA.com>

The cpint package provides the cpint device driver that communicates with
VM/ESA through the CP Diagnose interface. Most useful right now is the
diagnose 8 which allows the virtual machine to issue CP commands to query
or modify the status of the virtual machine.

%prep
%setup  -n cpint

%build
make

%install
linuxvers=`uname -r`
rm -rf $RPM_BUILD_ROOT
mkdir -p $RPM_BUILD_ROOT/dev
mknod $RPM_BUILD_ROOT/dev/cpcmd c %{cpintmajor} 8
chown root:wheel $RPM_BUILD_ROOT/dev/cpcmd
install -p -D hcp $RPM_BUILD_ROOT/sbin/
install -p -D cpint.o $RPM_BUILD_ROOT/lib/modules/$linuxvers/misc/

%post
if grep " cpint" /etc/modules.conf > /dev/null
then
   true
else
   echo "alias char-major-%{cpintmajor} cpint" >> /etc/modules.conf
```

[1] Unfortunately Neale packaged also binaries in his tar file so we untarred it, did a **make clean** and then tarred it again.

```
   /sbin/depmod -a
fi

%postun
if grep -w "cpint" /etc/modules.conf > /dev/null
then
   grep -v -w "cpint" /etc/modules.conf > /etc/modules.conf.new \
   && install -b -p /etc/modules.conf.new /etc/modules.conf
   rmmod cpint 2> /dev/null
   /sbin/depmod -a
fi

%files
%attr(600,root,root)/dev/cpcmd
/sbin/hcp
/lib/modules/%{version}/misc/cpint.o

%changelog
* Mon Jun 11 2001 rob@rvdheij.com
- Fixed incorrect mode for /dev/cpcmd node
* Fri Jun 08 2001 rob@rvdheij.com
- Use modules.conf rather than conf.modules
```

The next step is to copy the cpint.tar.gz (and optionally any patch files for the package) into the SOURCES/ directory, and then build the binaries rpm.

```
   # rpm -bb cpint.spec
```

This creates the binaries rpm in the SPECS/s390 directory. To package the source and patches with the spec file, you create the source rpm package. The **rpm -ba** command creates both binaries and source rpm.

Installing the cpint rpm package

When the binaries rpm is created, the install is very simple:

```
   # rpm -Uvh cpint-2.2.16-3.s390.rpm
```

If you do this on another system, you don't even have to copy the rpm file over by hand. A single **rpm** command can take care of the FTP and install (provided you put the rpm package in a place that can be reached via FTP):

```
   # rpm -Uvh ftp://hostname/path/cpint-2.2.16-3.s390.rpm
```

Using the hcp command

Because the install scripts in the rpm package also register the major number for cpint in /etc/modules.conf, the kernel module will be loaded automatically when you issue the **hcp** command:

```
# hcp q t
TIME IS 20:00:55 EDT FRIDAY 07/13/01
CONNECT= 99:59:59 VIRTCPU= 077:40.06 TOTCPU= 098:48.55
```

Seeing this will make the average VM user probably feel at home immediately, but there are a few gotchas to watch out for. Make sure you specify the command in double quotes when necessary:

```
# hcp m * test
HCPMSG020E Userid missing or invalid
```

What happened in this case is that the shell substituted the "*" with the list of files in the current directory, as it does with each shell command. Depending on the number of files in the directory, you may also get another indication of the problem:

```
# hcp m * test
Write: Cannot allocate memory
```

Another one to watch out for is typing the **hcp** command without any parameters. This causes the virtual machine to drop in CP READ and get logged off after some time.

C

Using the Linux 2.4 kernel

In this appendix we describe how we upgraded to the Linux 2.4.5 kernel. There are different ways to get a Linux-2.4.5 kernel running on your system. We experienced, howeve, that not all obvious routes to that goal were easy to travel for someone with minimal Linux skills.

On June 29, just before we started our work on this Redbook, another "code drop" was done by the IBM team in Boeblingen. Part of the patches published on the IBM DeveloperWorks Web site were the S/390 patches for the Linux-2.4.5 kernel. This latest patch is called linux-2.4.5-s390, just like the previous one published on June 13. The size of the patch, however, has grown from 25 KB to 1.5 MB. The "readme" with the patch explains the new function introduced with this patch. One of the biggest chunks in this patch appears to be a rework of the DASD driver.

Note: We were unable to get a system running with this new DASD driver on existing disks. Because of this we decided to stick with the June 13 version of the patch (which is not available on the Web site any more). Later experiments suggest that this may have been caused by devfs unintendedly being enabled. Unfortunately, we did not have time to retrace our steps and repeat the process.

Can of worms: Somewhere along the line, the ELF architecture number for s390 has changed from 0xa390 to the now official 0x0016. The binaries contain this number to prevent binaries from another architecture to be executed on your system (which is good). This new number is defined in binutils and will cause the loader to produce binaries marked with the new number. At the same time the kernel was changed such that it can support both types of binaries, so this is upwards compatible.

This same architecture code is also present in the kernel modules, including the object code only (OCO) ones for the OSA interface. You need to get the correct version of the modules to make sure they can be loaded or you will be without your network connection. The **depmod** command will warn you about the architecture difference.

Silo and the DASD driver

Changes to the ioctl() functions in the DASD driver make the silo from your 2.2.16 kernel fail with the 2.4.5 kernel. From the /usr/src/linux directory you need to **make silo** again. Unfortunately, the changes in silo and the DASD driver are incompatible, so you should keep both versions of silo around. We renamed the old silo to /sbin/silo-2.2.16 and copied the new one as /sbin/silo-2.4.5. This way you are constantly reminded to use the correct version.

There is a bug in **silo** in that it fails to clean up the temporary device node. This happens when the version of **silo** does not match the DASD driver in the kernel. The error message to recognize is shown in Example C-1. You need to **rm /tmp/silodev** to run **silo** again.

Example: C-1 Error message indicating the left-over device node

```
silo.c (line:428) 'mknod ("/tmp/silodev", S_IFBLK | S_IRUSR | S_IWUSR,
fst.st_dev)' returned 17='File exists'
```

If you want you can even write your own shell script /sbin/silo to pick the correct version based on the level of the kernel. The shell script is shown in Example C-2.

Later versions of **silo** support a configuration file. We did not use this because it mainly caused us problems and did not play well with alternate boot volumes.

Example: C-2 Shell script to pick the correct version of silo

```
#! /bin/sh
$0-`uname -r` $*
```

Another bug in `silo` is that it leaves temporary files like parm.6Djosg in the /boot directory. You may want to enhance your `silo` script such that it removes these temporary files before running `silo` again.

> **Attention:** When you copy a new kernel to your /boot directory, you should avoid overwriting the active kernel (for example, by avoiding the default name "image"). The way `silo` works is that it records the block numbers of the kernel in a special bootmap file. When Linux is booted, these blocks are read into memory from disk without checking the directories on the disk. If you overwrite your active kernel, these blocks become unused and may be overwritten later by other files. If you do not run `silo` at this point (for example, because it does not work on your new kernel), you are close to being unable to boot your old kernel again. For peace of mind you should implement the process outlined in 9.5, "Using an alternate boot volume" on page 206 when you play with kernel upgrades.

In the latest 2.4.5 patches the `silo` command has disappeared. We have not yet worked with `zipl`, which should perform the same function.

Steps to upgrade SuSE 7.0 to Linux-2.4.5 kernel

Using the intermediate May 7 version of SuSE 7.0, you should be able to get 2.4.5 running following this recipe.

Untar the linux-2.4.5 sources

Get the linux-2.4.5.tar.gz file from the Web at:

ftp://ftp.kernel.org

or your other favorite FTP site and put that in /usr/src for now. You probably should save the current 2.2.16 sources as well, as follows:

```
# cd /usr/src
# rm linux
# tar xzf linux-2.4.5.tar.gz
# mv linux linux-2.4.5
# ln -s linux-2.4.5 linux
```

Apply the IBM patches

You need to download linux-2.4.5-s390.tar.gz from the DeveloperWorks Web site and `untar` that. The following assumes you do this in /root/patch245/:

```
cd /usr/src/linux
patch -p1 -i /root/patch245/linux-2.4.5-s390.diff
```

The CTC driver at this level of code has a small problem that, at the very least, floods the console with error messages. There were also reports about sudden hangs of the driver, which may be caused by the same bug. The patch for this bug is in Example C-3. It must be applied in the same way as the linux-2.4.5-s390.diff patch.

Example: C-3 Patch for the ctc driver in 2.4.5

```
--- boelinux-2.4.5/drivers/s390/net/ctcmain.c   Wed Apr 18 23:40:07 2001
+++ linux-2.4.5/drivers/s390/net/ctcmain.c      Wed Jun 20 23:48:45 2001
@@ -988,7 +988,7 @@
                        first = 0;
                }
                atomic_dec(&skb->users);
-               dev_kfree_skb(skb);
+               dev_kfree_skb_any(skb);
        }
        spin_lock(&ch->collect_lock);
        if (ch->dccw) {
```

Restriction: The IUCV driver in this 2.4.5 level appears to be broken as well in that it caused a kernel oops with the first IP packet transmitted. A new version is being tested, but for the time being you will need to use CTC instead (and apply this patch).

Copy the config file from your old source tree

You probably want the new kernel configured similar to what you had on the previous one. If you have been building the 2.2.16 kernel yourself, the config file for the kernel will be the .config file in /usr/src/linux-2.2.16SuSE. If you did not do this already, then you find a copy of it as image.config in your /boot directory. Copy the config file to /usr/src/linux/.oldconfig and run `make oldconfig` now. You will be prompted for the new configuration options that were not in your old kernel.

Build the kernel and modules

You can now build the kernel and modules, starting in the /usr/src/linux/. This process may take a while, depending on the processing resources you have available:

```
#cd /usr/src/linux
# make image modules modules_install
```

Because of the incompatibility with `silo` and the new DASD driver as explained in “Silo and the DASD driver” on page 442, you should also build `silo` and make yourself the shell script for it:

```
# make silo
# cp arch/s390/tools/silo/silo /sbin/silo-2.4.5
```

Prepare the boot disk

Unless you are very certain of what you are doing, you should prepare yourself an alternate boot disk as explained in 9.5, "Using an alternate boot volume" on page 206 and mount this at /boot. Copy the new kernel to the /boot directory and run **silo** with your normal parameter file:

```
# cp System.map /boot/System.map-2.4.5
# cp arch/s390/boot/image /boot/image-2.4.5
# cd /boot
# silo -d /dev/dasd? -f image-2.4.5 -p parmfile
```

Install your network driver

Depending on the type of network interface used in the system, you may need to get the version of one of the Object Code Only network drivers from the DeveloperWorks Web site. You can create a /lib/modules/2.4.5/oco directory for these (if you put them in the kernel directory they may disappear when you run a **make modules_install** again). If you add the network drivers after running the **make modules_install**, then you need to run **depmod -a** again.

Attention: Do not pick the 2.4.5-2 version of the network drivers. These versions use the new 0x0016 ELF architecture code which does not work with the modutils and kernel we have here.

If you use a CTC or IUCV connection to the system, the driver will have been built already by the **make modules** step.

Shut down and reboot

You can now shut down your Linux system and boot from the new boot disk. Though the messages during boot will be slightly different, things should look fairly normal with this boot. If it fails to get your network driver going you should log on from the virtual console as root and run **depmod -a** again.

About the timer patch

During the weeks we were writing this redbook, there was a lively discussion on the Linux-390 mailing list about the possible benefit of the so-called "timer patch." A few months before that Martin Schwidefsky from IBM Boeblingen posted a possible way in which Linux for S/390 could do without the 10 mS timer tick. The description of the proposed patch can be found on the Web at:

http://lwn.net/2001/0412/kernel.php3

The patch removes the global variable “jiffies” that is used by the kernel and device drivers for time measurement. Instead, it provides a macro called “jiffies” to compute the current value using the STCK instruction. Though Martin posted portions of the patch to the kernel mailing list, there still is work to be done if you want to implement this yourself.

For the work on the redbook we had access to a preliminary version of the patch that Martin proposed. This patch is not part of the mainstream kernel sources. It is unclear whether this is going to happen at all.

The patch must be applied just like the linux-2.4.5-s390.diff, on top of what you have there now:

```
# cd /usr/src/linux
# patch -p1 -i /root/patches245/timer-2.4.5-s390.diff
```

Unfortunately, the lcs.o driver for 2.4.5 has a dependency on the “jiffies” symbol that is removed by the timer patch. The qdio driver does not have this dependency, so we expect this should continue to work (but we were not able to try it). Since the source for the lcs driver is not part of the kernel sources, we cannot rebuild it ourselves. Apart from the qdio driver the only alternative left now is the CTC driver.

The version of the timer patch that we used failed to export a symbol, thus causing unresolved references if you want to build the CTC driver as a module. The patch in Example C-4 fixes this problem. Apply it just like the timer patch before rebuilding the kernel.

Example: C-4   Exporting the “init_timer_cc” symbol

```
--- boeblinux-2.4.5/arch/s390/kernel/s390_ksyms.c Wed Apr 11 21:02:28 2001
+++ linux-2.4.5/arch/s390/kernel/s390_ksyms.c   Sun Jul 29 17:07:02 2001
@@ -8,6 +8,7 @@
 #include <asm/checksum.h>
 #include <asm/delay.h>
 #include <asm/setup.h>
+#include <asm/io.h>
 #if CONFIG_IP_MULTICAST
 #include <net/arp.h>
 #endif
@@ -20,7 +21,7 @@
 EXPORT_SYMBOL(_zb_findmap);
 EXPORT_SYMBOL(__copy_from_user_fixup);
 EXPORT_SYMBOL(__copy_to_user_fixup);
-
+EXPORT_SYMBOL(__ioremap);
 /*
 * semaphore ops
 */
```

After applying the timer patch you should run **make dep** again, followed by a **make image modules modules_install** as before. Then write the new kernel and map to your /boot directory and run **silo**.

Using the device file system

The /dev directory as we know it contains the entries that describe the hardware devices for your Linux image. To be more precise, /dev contains an entry for each hardware device you could possibly have in your Linux image. You see lots of /dev/dasd* entries because some Linux image could have 20 disks and need an inode to access the device. To have all /dev inodes in the private disk space of each Linux image is difficult to maintain. It also is a waste of disk storage, even though the entries themselves are rather small.

It is not practical to put /dev on an R/O shared disk because some applications need to change the owner of the device entry. This is not possible on an R/O shared disk. For specific cases (like tty devices) there is a virtual file system, but a more attractive solution is the device file system (Devfs). The patches for Devfs were done by Richard Gooch and have been included in the 2.3.46 kernel.

Devfs is a virtual file system, similar to the proc file system mounted in Linux as /proc. The entries shown in such a file system with the **ls** command do not really exist. Specific entry points of the device driver for the file system (like procfs or devfs) are called when a directory listing is required. Other entry points are called when you read or write a "file" in the virtual file system. If you type the command **cat /proc/uptime**, that value is computed because you want to display it. Just as /proc provides the peepholes to expose kernel variables to the applications, devfs shows the device configuration of the Linux image to the applications. The device file system is mounted on the /dev mount point, so with devfs running /dev contains only the devices present in your configuration at that moment.

The device drivers must be aware of devfs. They must call the correct devfs functions to register and they must provide the proper data structures to hold the device information. Whenever devfs needs information about a directory in the device file system, it invokes the appropriate functions of the device driver. For compatibility, a devfsd package is available that provides access to the old inodes through the new devfs interfaces.

The DASD driver with Linux for S/390 has been enhanced to support devfs as well. The difference is obvious when you use the **df** command to show the mounted file systems.

Example: C-5 Device names shown with devfs

```
tux60002:/proc # df
Filesystem          1k-blocks      Used Available Use% Mounted on
/dev/dasd/01A0/part1    174116     63256    101872  38% /
/dev/dasd/01A1/part1   1415848   1169352    174576  87% /usr
```

Because the DASD driver keeps the active devices in a specific subtree of the /dev file system, it is easy to see what disks are accessible to the Linux image.

Example: C-6 Full list of disk devices in the system

```
tux60002:/proc # find /dev/dasd
/dev/dasd
/dev/dasd/0200
/dev/dasd/0200/disc
/dev/dasd/0200/part1
/dev/dasd/01A0
/dev/dasd/01A0/disc
/dev/dasd/01A0/part1
/dev/dasd/01A1
/dev/dasd/01A1/disc
/dev/dasd/01A1/part1
/dev/dasd/01C0
/dev/dasd/01C0/disc
/dev/dasd/01C0/part1
```

The listing in Example C-6 shows the entries in /dev/dasd for this particular system. The DASD driver creates a subdirectory named after the device address. In this subdirectory is one entry for the raw device (the disc entry) and one for each partition on the disk. In addition to the subdirectories named after the device address, there is also a /dev/discs directory, shown in Example C-7, that lists the disks in sequential order. These entries are links to the corresponding entries in the /dev/dasd directory.

The /dev/discs directory is the standard Linux devfs structure. This is not specific to Linux for S/390. On another platform these entries would be links to SCSI or IDE devices.

Example: C-7 Available disk devices in /dev/discs

```
tux60002:/dev/discs # ls -l
total 0
drwxr-xr-x   1 root     root            0 Dec 31  1969 .
drwxr-xr-x   1 root     root            0 Dec 31  1969 ..
lr-xr-xr-x   1 root     root           12 Dec 31  1969 disc0 -> ../dasd/0200
lr-xr-xr-x   1 root     root           12 Dec 31  1969 disc1 -> ../dasd/01A0
lr-xr-xr-x   1 root     root           12 Dec 31  1969 disc2 -> ../dasd/01A1
```

```
lr-xr-xr-x   1 root     root           12 Dec 31  1969 disc3 -> ../dasd/01C0
```

The FAQ at the following Web site is recommended reading material if you want to get started with the device file system:

http://www.atnf.csiro.au/~rgooch/linux/docs/devfs.html

Installing the device file system on SuSE

We installed the device file system on a SuSE 7.0 system upgraded with a Linux-2.4.5 kernel as described in Appendix C, "Using the Linux 2.4 kernel" on page 441. The first step is to configure the kernel to support a device file system (but not enable the option to mount/dev automatically).

With this kernel active you can install devfsd (from the SuSE distribution). Because the device file system mounts on top of the /dev directory, anything in there becomes unaccessible. The devfsd package contains the devfsd daemon that provides the compatibility interface. This way the old programs will continue to work, even when you mount devfs over/dev.

Attention: The startup script /sbin/init.d/boot.devfs use by SuSE refers to a file /sbin/init.d/mygrep (because grep lives in /usr/bin, which is not yet available at that point during startup). The installation of devfsd did not put the file there, so we copied it over from /usr/share/doc/packages/devfsd/ to get it working

When you have verified that devfsd is working properly at the next reboot, you can enable the option devfs=mount in the kernel parameter file (and run `silo` again).

Attention: We found that `mingetty` did not work with devfs for us, despite the comments about devfs in the code (it looks like support was added for some devices needed as virtual console, but not the /dev/console we have with Linux for S/390). This resulted in error messages about /dev/console permissions. The easiest way out for the moment was to remove `mingetty` and use `sulogin` instead in /etc/initab.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information on ordering these publications, see "How to get IBM Redbooks" on page 453.

- *Linux for S/390*, SG24-4987

  `http://www.ibm.com/redbooks/abstracts/sg244987.html`
- *Linux for zSeries and S/390: Distributions*, SG24-6264

  `http://www.ibm.com/redbooks/abstracts/sg246264.html`
- *IBM @server zSeries 900 Technical Guide*, SG24-5975

  `http://www.ibm.com/redbooks/abstracts/sg245975.html`
- *OSA-Express Implementation Guide*, SG24-5948

  `http://www.ibm.com/redbooks/abstracts/sg245948.html`

Other resources

These publications are also relevant as further information sources:

- *z/VM V4R1.0 General Information*, GC24-5991
- *z/VM TCP/IP Planning and Customization*, SC24-5981
- *z/VM TCP/IP Programmer's Reference*, SC24-5983 (available softcopy only)
- *z/VM TCP/IP User's Guide*, SC24-5982 (available softcopy only)

Referenced Web sites

These Web sites are also relevant as further information sources:

- The Linux for S/390 developerWorks page:

  `http://www10.software.ibm.com/developerworks/opensource/linux390/index.shtml`
- The Coda Web site:

  `http://www.coda.cs.cmu.edu`

- The paper "Linux IP Networking - A Guide to the Implementation and Modification of the Linux Protocol Stack" by Glenn Herrin, May 2000:

  http://kernelnewbies.org/documents/ipnetworking/linuxipnetworking.html

- The paper "Addressing Security Issues in Linux" by Mark Chapman, December 2000, and the paper "Linux Security State of the Union" by Robb Romans and Emily Ratliff, May 2001, both available at:

  http://oss.software.ibm.com/developer/opensource/linux/papers.php

- The Linux kernel FTP site:

  ftp://ftp.kernel.org

- The OpenLDAP organization:

  http://www.openldap.org

- The NET-SNMP home page:

  http://net-snmp.sourceforge.net

- Linux Devfs (Device File System) FAQ:

  http://www.atnf.csiro.au/~rgooch/linux/docs/devfs.html

- Linux Documentation Project home page:

  http://www.linuxdoc.org

- Advanced Maryland Automatic Network Disk Archiver (Amanda) home page:

  http://www.amanda.org

- Gnuplot Central home page:

  http://www.gnuplot.org

- Multi Router Traffic Grapher (MRTG) home page:

  http://people.ee.ethz.ch/~oetiker/webtools/mrtg/

- NetSaint home page:

  http://www.netsaint.org

- OpenAFS home page:

  http://www.openafs.org

- Samba home page:

  http://www.samba.org

- Bynari home page:

  http://www.bynari.net

- A resource for learning how the kernel works with regard to TCP/IP:

  http://kernelnewbies.org/documents/ipnetworking/linuxipnetworking.html

- CMS Pipelines home page:

  `http://pucc.princeton.edu/~pipeline`

- Dante home page:

  `http://www.inet.no/dante`

How to get IBM Redbooks

Search for additional Redbooks or Redpieces, view, download, or order hardcopy from the Redbooks Web site:

ibm.com/redbooks

Also download additional materials (code samples or diskette/CD-ROM images) from this Redbooks site.

Redpieces are Redbooks in progress; not all Redbooks become Redpieces and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

IBM Redbooks collections

Redbooks are also available on CD-ROMs. Click the CD-ROMs button on the Redbooks Web site for information about all the CD-ROMs offered, as well as updates and formats.

Special notices

Index

IBM

Redbooks

Linux for IBM @server zSeries and S/390: ISP/ASP Solutions

(1.0" spine)
0.875"<->1.498"
460 <-> 788 pages

Linux for IBM @server zSeries and S/390: ISP/ASP Solutions

Create and maintain hundreds of virtual Linux images

Running on the mainframe

Managed by z/VM

This IBM Redbook describes how Linux can be combined with z/VM on zSeries and S/390 hardware - the mainframe. This combination of hardware and operating systems enables Internet Service Providers (ISP) and Application Service providers (ASP) to more efficiently provide services. We assume a broad definition of ASP to include production enterprise solutions as simple as file serving.

In a world of discrete servers, when a new resource is required, workload can either be added to an existing server or a new one can be purchased. Often a new server is installed and the *server farm* grows in the enterprise.

S/390 and zSeries hardware, microcode and software allow physical resources to be made virtual among Linux systems. This allows many hundreds of Linux systems to exist on a single server. Running multiple Linux *images* as guests of VM/ESA or z/VM is a smart choice.

SG24-6299-00 ISBN 0738423521