Installing Gentoo Linux (x86)

by Ted Kosan

(See the bottom of the document for notes)

Table of Contents

1 Why bother to manually install an operating system?

1.1 This document will guide you through manually installing an operating system. You might ask yourself "why bother to manually install an operating system when it is much easier to install one automatically with a CD?" Manually installing an operating system does take a significant amount of time and effort. For people who just want to use a computer as a tool, automatically installing an operating system is usually the best choice. However, if you have a deeper interest in computers beyond just using them as tools, then the knowledge you will gain from manually installing an operating system will be very valuable.

2 Which operating system to install?

2.1 After you have made the decision to manually install an operating system, the next decision to make is which one to install? Since the most widely available kind of computer today is the IBM PC compatible computer, selecting an operating system that runs on this type of computer is a good choice. Another thing to consider is some operating systems are proprietary while others have an open source license. This means that the source code for the operating system can be viewed and modified by anyone who agrees to the license. If your goal is to learn as much about an operating system as possible, my opinion is that the best kind of operating system to work with is one that has a widely-used open source license.

2.2 Here is a list of the most popular operating systems that currently run on IBM PC compatible computers:

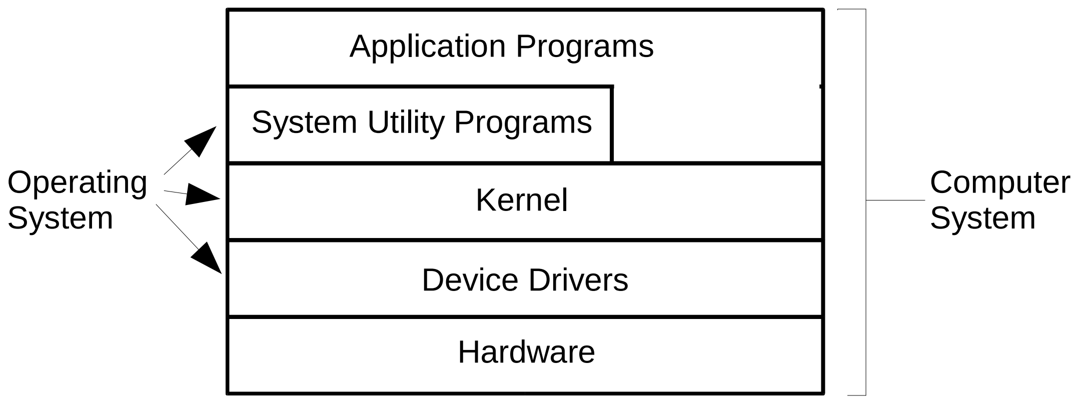
- Mac OS X.
- Microsoft Windows.
- FreeBSD (Berkley Standard Distribution).
- GNU/Linux.

2.3 Mac OS X and all of Microsoft's operating systems are proprietary and closed source which limits their effectiveness for learning about operating systems. FreeBSD and GNU/Linux are open source operating systems that are suitable for learning about operating systems. In this document we will focus on GNU/Linux.

3 The Parts Of An Operating System

3.1 Operating systems for PC and server class computers usually consist of the parts shown in Figure 1.

Figure 1



3.2 The **kernel** is the core part of an operating system, and it is actively running in the computer's memory map while the computer is on. It controls and coordinates almost all of the resources in the computer, and the computer would cease to function if the kernel was removed. The kernel accesses the computer's hardware through special programs called **device drivers**, and each piece of hardware needs to have a device driver written for it before the kernel can access it. Examples of hardware that need device drivers before the kernel can access them include the following:

- Keyboard.
- Hard Drive.
- CDROM Drive.
- Video Card.
- Sound Card.

3.3 The kernel of an operating system provides the core functionality of a computer. However, deep software and operating systems knowledge is needed to access this functionality. In order to make the operating system easier to use, **system utility programs** are included with it that access the kernel and then make its resources available to other utility programs and application programs.

3.4 A significant amount of the effort that is needed to manually install an operating system consists of installing its system utility programs, selecting which device drivers are needed for the computer's hardware, and configuring the kernel.

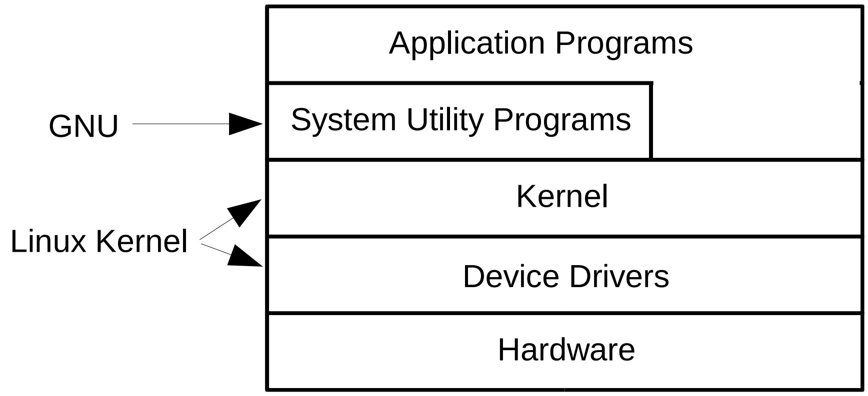
4 The GNU/Linux operating system

4.1 In order to understand what GNU/Linux is, one must first understand a little bit about UNIX because GNU/Linux is modeled after UNIX. UNIX is an operating system that was developed for mainframe and minicomputers in the 1960s and 1970s at AT&T Bell Laboratories. AT&T licensed UNIX to universities, governments, and companies throughout the 1970s and the license included access to the operating system's source code, which was mostly written in the C programming language. A number of UNIX versions were created by companies and universities throughout the 1970s and 1980s and it became widely adopted.

4.2 In 1983, Richard Stallman announced the GNU project. One of its goals was to create a UNIX-like operating system that consisted of 100% free (as in free speech) open source software. By the early 1990s, the GNU project had succeeded in creating most of the software needed for an operating system, except a kernel and device drivers.

4.3 In 1991 Linus Torvalds, a student at the University of Finland, wanted an open source version of UNIX that would run on standard personal computers. Since one did not exist at the time, he decided to start a project to create one, and he used the communications capabilities of the Internet to invite people from all over the world to help him. The project started by developing a kernel called "Freax" which stood for "free UNIX". The person who maintained the FTP server that the Freax software was placed on created a directory for it called "Linux" and the kernel became known as "Linux" soon afterwards. Instead of creating a kernel and all of the system utility software for it, the Linux developers decided to adapt their kernel to the already existing GNU systems software, thereby creating the **GNU/Linux** open source operating system. Figure 2 shows the relationship between the GNU software and the Linux kernel.

Figure 2



5 What is a GNU/Linux distribution?

5.1 The source code needed to build a GNU/Linux system exists on numerous servers on the Internet, and these servers are located throughout the world. Before a GNU/Linux system can be built, copies of this source code need to be brought into one place, compiled, configured, and then tested. While it is possible for an individual to do this, it takes a significant amount of skill and experience to accomplish. **GNU/Linux distributions** were created to help solve this problem. A GNU/Linux distribution is usually put together by a group of experienced developers who copy the source code needed to create a GNU/Linux distribution to one place, compile and configure it, and then make the result available to others. There are numerous GNU/Linux distributions available and new ones are being created all the time. Some distributions are cost-free while others are commercial.

5.2 The following is a list of the more popular GNU/Linux distributions:

- Debian.
- Fedora.
- Gentoo.
- Knoppix.
- Linspire.
- LFS (Linux From Scratch).
- Red Hat Enterprise.
- SUSE.
- Ubuntu.
- Mint.

6 Gentoo GNU/Linux, a good distribution for our purposes

6.1 Of the GNU/Linux distributions listed in the previous section, LFS is too difficult to install for a first-time GNU/Linux user, and the rest of the distributions except Gentoo are too easy to install. I have found that Gentoo GNU/Linux (or Gentoo Linux for short) is easier to install than LFS but difficult enough so that one learns a great deal during the installation process.

6.2 Here is the URL for the Gentoo Linux main website. Look through the website and then continue reading:

http://gentoo.org

6.3 The normal way that a beginner learns how to install Gentoo Linux is by reading the **Gentoo Installation Manual** and following the step-by-step instructions that are contained there. The Installation Manual, however, assumes that the user has a mid-level computer background, and it leaves out information that a beginner would find useful. This document covers much of the same material that the Gentoo Installation Manual does, but it moves slower through the installation process, and it contains more detailed explanations.

7 Obtain an IBM PC compatible computer to install Gentoo Linux on

7.1 Since IBM PC compatible computers are the most common type of personal computer, this document focuses on installing Gentoo Linux on these type of machines. The machine you select should have the following minimum requirements:

- x86_64 CPU.
- 348MB of RAM.
- 8GB hard drive.
- CDROM drive.

7.2 **As an alternative, you can use a virtual machine that emulates an x86_64-based PC (recommended).**

7.2.1 Follow the instructions that were covered in class for how to configure a new virtual machine.

8 Download the minimal install .iso image, check it, and then burn it on a CD

8.1 Many GNU/Linux distributions enable the user to download a .iso file that contains a bootable software image that can be burned onto a CDROM. After the image has been burned on the CDROM, the CDROM can be used to boot the machine into a GNU/Linux environment. This environment can then be used to install GNU/Linux on a hard drive.

8.2 A recent Gentoo Linux bootable image for x86-based PCs is called **install-amd64-minimal-20180311T214502Z.iso** .

8.3 Download the .iso file.

8.3.1 Download the file **install-amd64-minimal-20180311T214502Z.iso** from **http://patternmatics.org/ssu/etec1302/gentoo_2018**. This is the file that you will be burning onto a CD (**or loading into VirtualBox**). You have a problem, though. What if the file is corrupted? It would not be good to spend time burning the file onto a CD, booting a computer with it, getting half way through the installation process, and then discovering that a part of the file was corrupted. Most files that are downloaded from the Internet are checked to make sure they are not corrupted by using a **hash algorithm**.

8.4 Check the .iso file with a hash algorithm.

8.4.1 A program that implements a hash algorithm scans every byte in a file and generates a number that is sometimes called the **digital fingerprint** or **message digest** of the file. As long as the bytes in the file do not change, each time a given hash algorithm is run on a file the same message digest number is generated. A popular hash algorithm is called sha512 and here is an example sha512 message digest for the install-amd64-minimal-20180311T214502Z.iso file:

```
b326653e877b21fc6a84f3e428dcb3f33721aa76ac7da051b5cacd9b9b6a50c
8b3bb82e5f559db0370209a173124d4a771ed08d8c0d2d00ceeaf63e7b6a82d
54  install-amd64-minimal-20180311T214502Z.iso
```

8.4.2 This number is in hexadecimal (or base 16) format and this format is widely used with computers. Before a given file is made accessible on the Internet, a hash algorithm is run on it and a message digest number is generated. This digest number is put into a separate small file and then both the main file and the message digest file are placed on the Internet.

8.4.3 When a user wants to obtain a copy of the main file, both the main file and its digest file are downloaded to the user's computer. The user then runs a program that uses the same hash algorithm as the original one used on the main file. The message digest number that is generated is then compared to the number that is in the separate small file. If the numbers match, then the main file is not corrupted but if they do not match, it is corrupted and it must be downloaded again.

8.4.4 If you are using the Windows operating system, download **http://patternmatics.org/ssu/etec1302/gentoo_2018/QuickHash-Windows.exe** to check the checksum of the **install-amd64-minimal-20180311T214502Z.iso** file.

8.4.5 Download the **install-amd64-minimal-20180311T214502Z.iso.DIGESTS** file from the server, open it and if the md5 message digest numbers match, your file is not corrupted.

8.4.6 If you will be installing Gentoo on a physical PC, burn the .iso file onto a CD. If you are using the VirtualBox virtual machine, you do not need to burn the .iso file onto a CD. Instead, just remember where you placed it on your hard drive so you can tell VirtualBox where to find it.

9 VirtualBox Settings

9.1 Your instructor will demonstrate how to configure a VirtualBox virtual machine in class. The following are the settings that will be used for this virtual machine:

- Name: **<your_name>_etec1302_2018**
- Type: **Linux**.
- Version: **Gentoo (64 bit)**.
- Memory (RAM): **1024MB**.
- Virtual hard drive: **8.00 GB**. (After selecting “Create a virtual hard disk now”)
- Hard drive file type: **VDI**.
- Storage on physical hard drive: **Dynamically allocated**.
- Virtual hard drive file location and size: **Accept defaults**.

9.2 Installing the **install-amd64-minimal-20180311T214502Z.iso** file

- Settings → Storage → Controller: IDE → Empty → (Live CD/DVD: **Check)**.
- Settings → Storage → Controller: IDE → CD/DVD Drive: **Click on small blue CD icon, select “Choose Virtual Optical Disk File”, and then select the install-amd64-minimal-20180311T214502Z.iso file**.
- Click the "OK" button.

9.3 If you need to shut VirtualBox down before installation of Gentoo is complete, select the **Machine → Close... → Save the machine state** option in your virtual machine's window. When you launch VirtualBox again, your virtual machine will start in the same state it was in when it was closed. **Note: if you move your computer to another network, you may need**

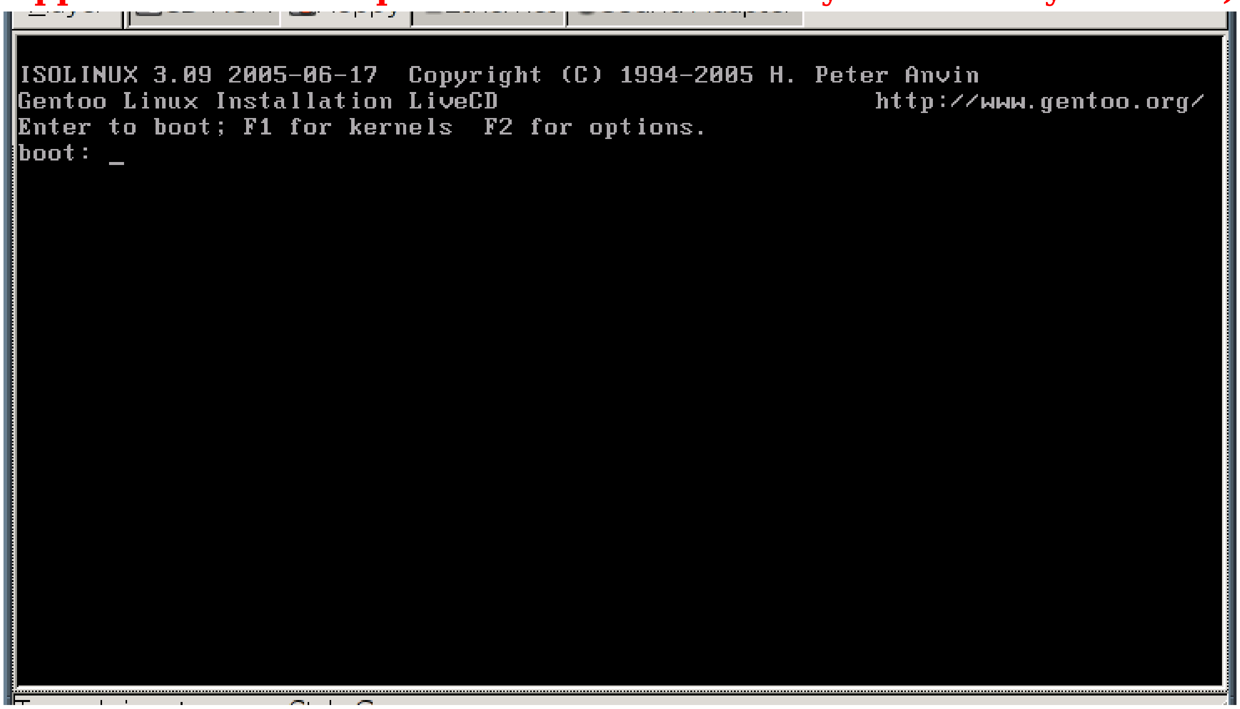
to renew your virtual machine's IP address.

10 Boot your computer with the CD

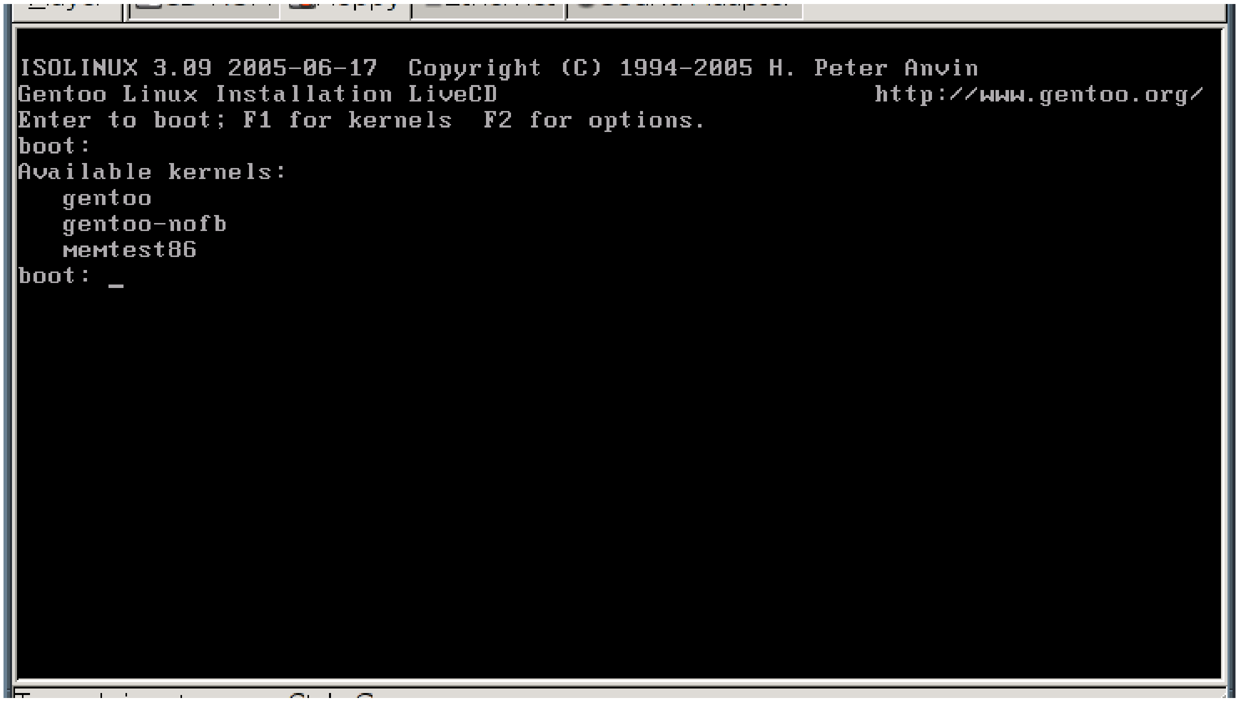
10.1 Your computer must be plugged into an Ethernet network that is attached to the Internet for these instructions to work.

10.2 Start your machine and enter the **setup utility** (usually by pressing <F2>, <F10>, or <Del>) to make sure it is configured to have the CDROM drive as the first boot device. Place the install CD into the CDROM drive and then boot the computer with it. **For people that are using VirtualBox, launch VirtualBox and then select the "Start" button to launch the virtual machine you created.**

10.3 When your machine boots from the install CD or image, the first screen it shows should be similar to the following **(Note: click inside the black window and press your <space> key once as soon as the boot: prompt appears or the boot process will start before you are ready for it to.)**:

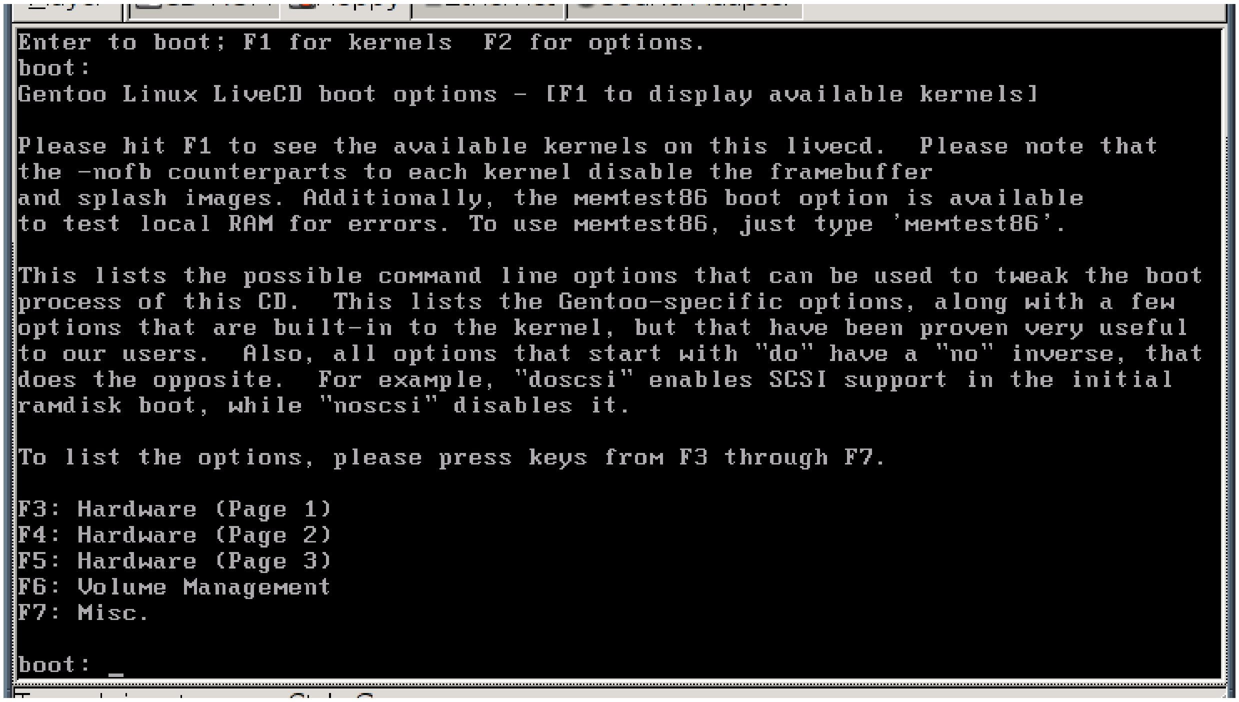


10.4 The CD will wait for input from the user for a short while and, if no input it given, it will boot the machine using the default configuration. If you press the F1 key, a list of kernels that are available on the CD is shown.

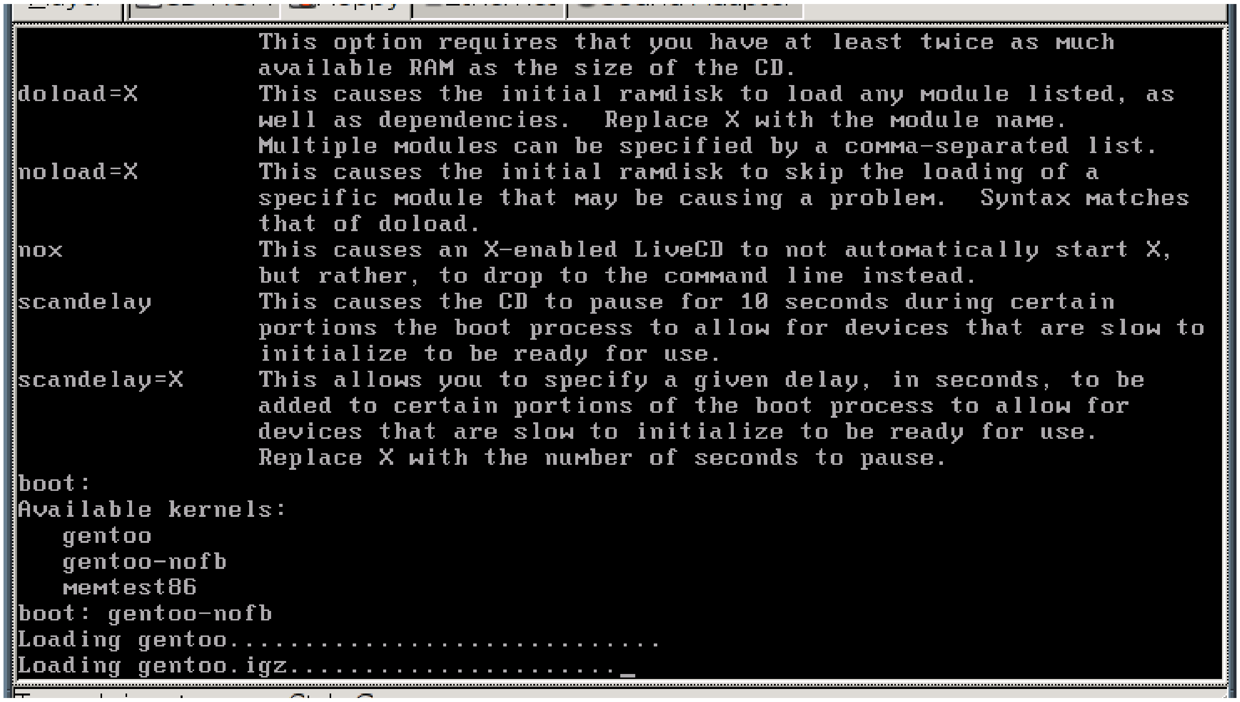


10.5 The kernel named **gentoo** is the default kernel and the one named **gentoo-nofb** is a kernel that does not use the frame buffer to switch into graphics mode when it boots. In a moment we will boot the computer using the **gentoo-nofb** kernel to make it easier to read the information that is displayed when the system is booting.

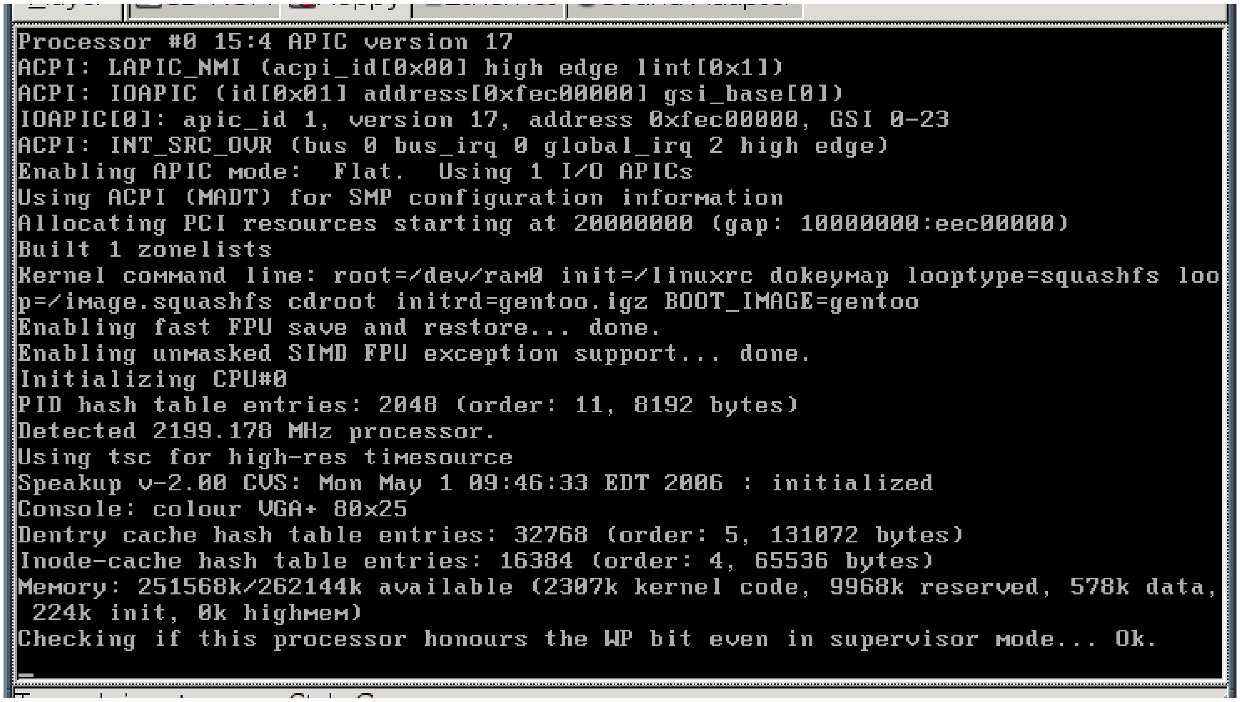
10.6 If you press the **F2** key, the following options screen is shown:



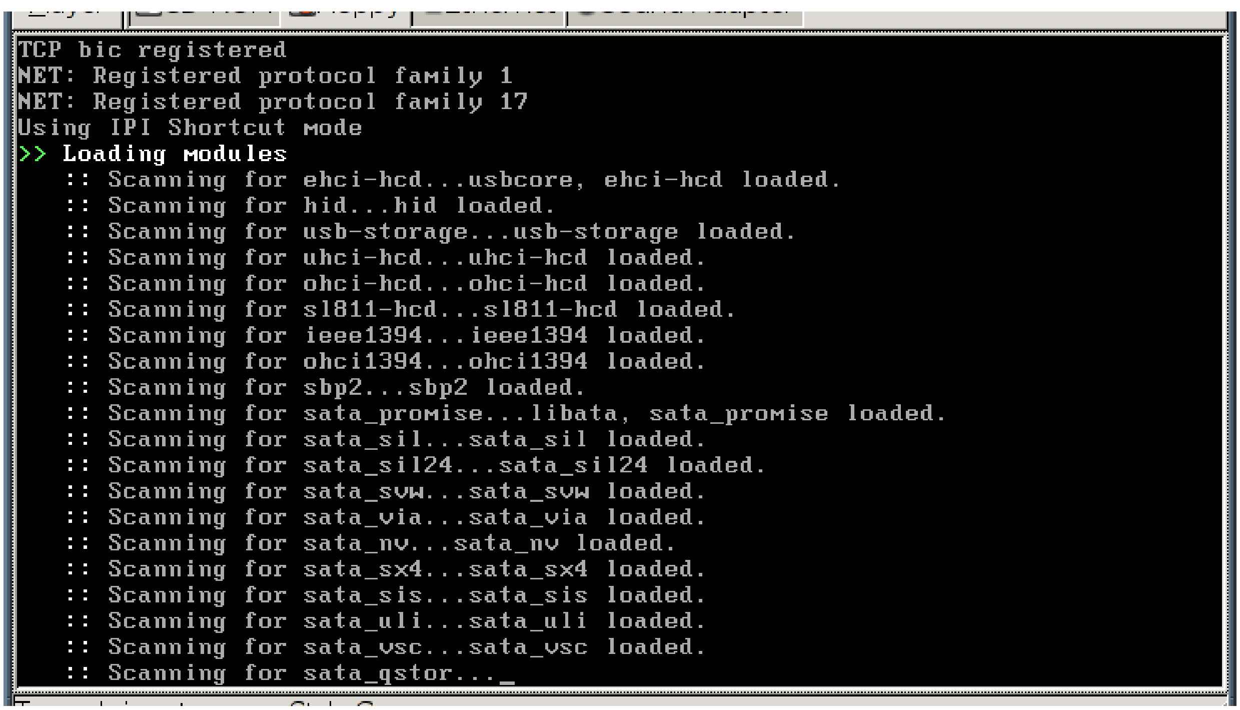
10.7 You can press the **F3 -F7** keys to read about the options that are available during the boot process. After you are done reading about the options, type **gentoo-nofb** at the **boot:** prompt (which is shown next) and the system will begin to boot the copy of Gentoo Linux which is on the CD.



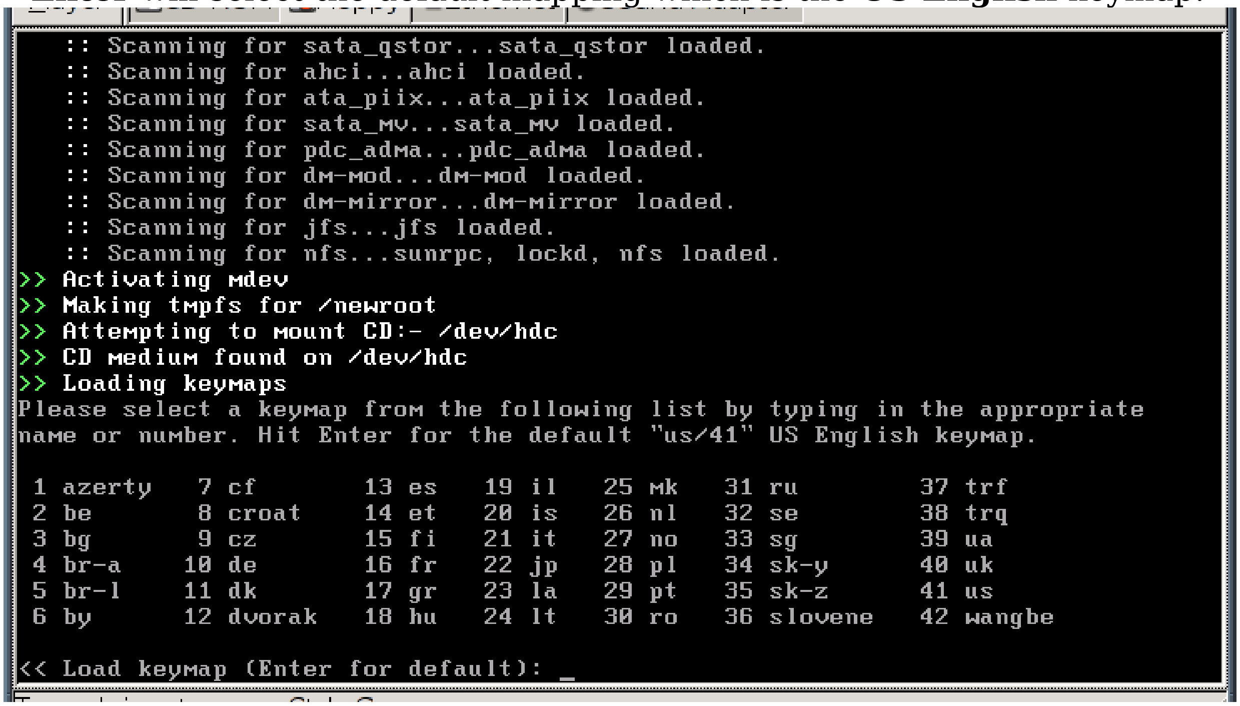
10.8 As Gentoo Linux boots, it is going to show the details of the boot process on the screen. As each part of the operating system is loaded into memory and initialized, it prints a message on the screen which indicates whether it was successfully loaded or not. It may also print information about the hardware it is responsible for, such as the speed of the system's processor or the amount of RAM that it has. The following screen shot shows both of these pieces of information. Can you locate them on the following screen capture?



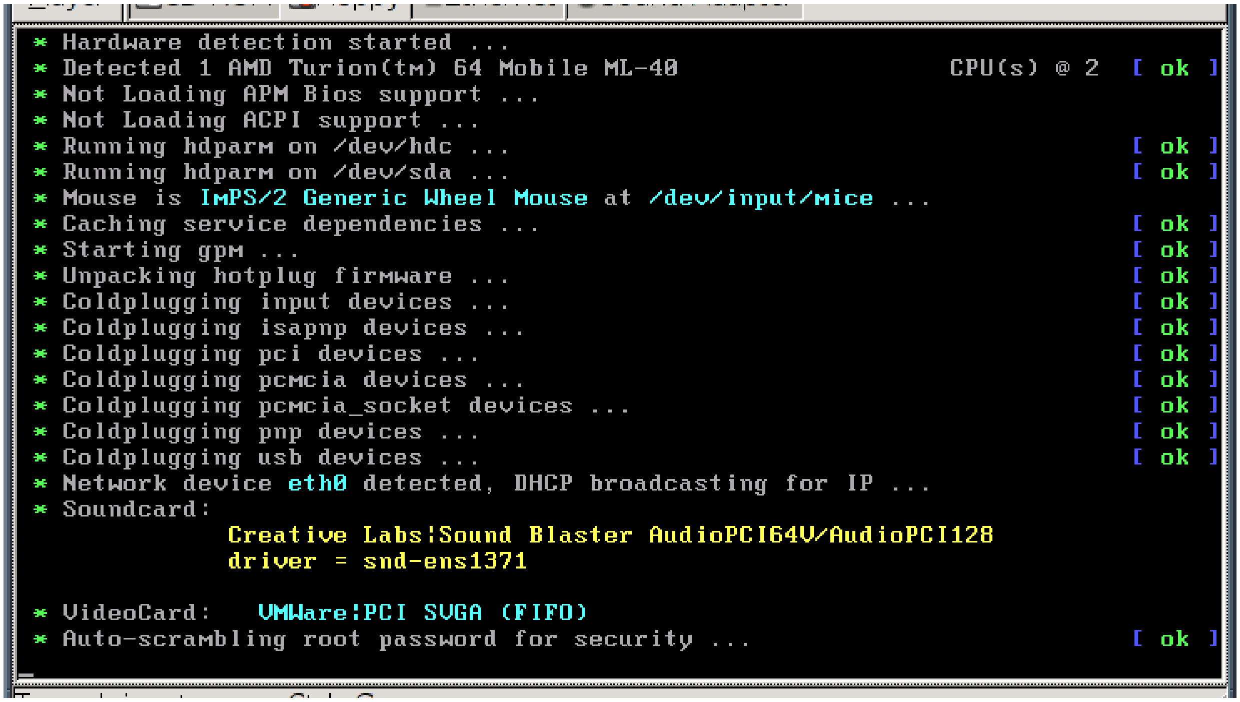
10.9 A little bit later in the boot process, the CD scans the system to see what the make and model are for the various pieces of hardware in the system and then it loads the device drivers that match this hardware. Device drivers in Linux are often loaded as kernel modules and the following screen shows kernel modules being loaded into RAM:



10.10 When you are asked for which keyboard **keymap** to select, hitting **Enter** will select the default mapping which is the **US English** keymap:

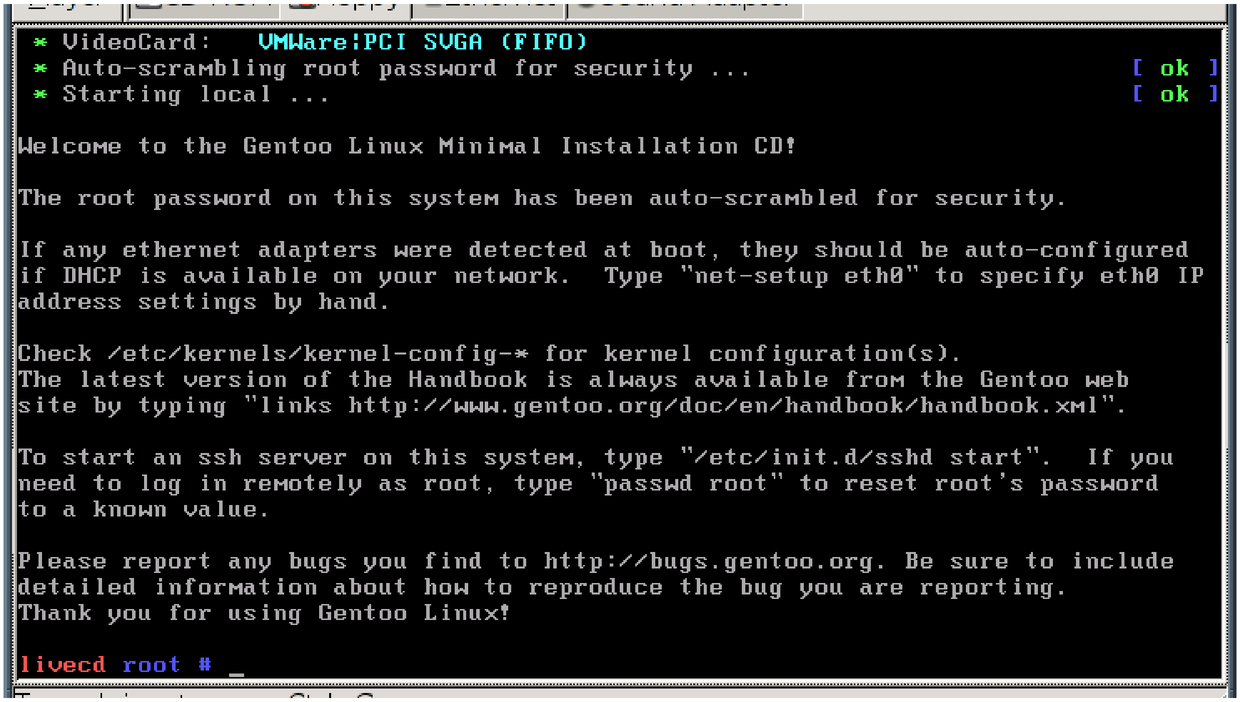


10.11 Towards the end of the boot process, the CPU is detected, the mouse driver is attached to **/dev/input/mice** (**make a note of this**), the main network device is found (usually **eth0**), and a **DHCP** (Dynamic Host Configuration Protocol) request is sent to the local network asking for local configuration information. The sound card and video card are then located and finally, a **login password** is randomly generated so that someone else on the network cannot log into your machine without you knowing about it. These steps can be seen in the following screen shot:



10.12 The last part of the boot process provides information that describes options that the user may want to explore. One of these options is to launch the **ssh server** which will permit the user to remotely log into the current machine from another machine on the network. This is useful if the user wants to leave the current machine running at one location (like school) and work through the installation process from another location (like home). For now, though, we will proceed by working right at the machine.

10.13 This last screen shot shows the end of the boot process:



11 Exploring the copy of Gentoo Linux that is on the CD itself

11.1 The way that the minimal install CD works is that it **boots the machine into the copy of Gentoo Linux that came on the CD itself** and then the user enters this environment and uses it to **install Gentoo Linux on the system's hard drive**. After Gentoo Linux has been successfully installed on the hard drive, the CD is removed, the system is rebooted, and the copy of Gentoo that was placed on the hard drive is used to boot the system from then on.

11.2 Before starting the installation of Gentoo Linux on the system's hard drive, we are going to explore the CD's copy of Gentoo Linux in order to gain a better understanding of the Gentoo Linux environment.

12 Testing the network connection

12.1.1 The first thing I want you to do is to enter the command **ifconfig** at the command prompt. The ifconfig command is used to configure the system's network interfaces, but it will also show the current configuration of each network interface if it is entered without any additional options.

```
livecd / # ifconfig

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.1.8  netmask 255.255.255.0  broadcast 206.21.94.255
        inet6 fe80::a00:27ff:fe28:853a  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:28:85:3a  txqueuelen 1000  (Ethernet)
        RX packets 13271  bytes 2275608 (2.1 MiB)
        RX errors 0  dropped 4  overruns 0  frame 0
        TX packets 127  bytes 13087 (12.7 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 16436
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 0  (Local Loopback)
        RX packets 45  bytes 810 (810.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 45  bytes 810 (810.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

12.1.1.1 **If your machine does not have a valid IP address**, make sure the machine is properly connected to the network. Now, execute the following command: **net-setup eth0** and configure your machine to use a **wired** network and **DHCP**.

12.1.1.2 After this command is finished executing, check to make sure you have a valid IP address again using the ifconfig command.

12.1.2 The output of the **ifconfig** command executed above shows that this machine has two network interfaces attached to it which are called **eth0** and **lo**. The **eth0** interface is bound to an **Ethernet** network device, and it is also the system's main network connection. Ethernet is the most popular networking technology for connecting PCs to networks, and it is likely that your PC is attached to a network using Ethernet too. The DHCP request that was sent to the network while the system was booting received a response that configured **this** machine with IP address **10.0.1.8**. Your machine should also have been configured with an address from your network (**Note: your address will most likely be different than 10.0.1.8**).

12.1.3 The **lo** interface is attached to what is called a **loopback** network, which is a simulated network local to the machine itself. The **lo** interface is used for testing purposes and for allowing different applications on the machine to communicate with each other even when the machine is not attached to an actual network. Most **lo** interfaces are given IP address **127.0.0.1**.

13 Exploring the CD's filesystem

13.1.1 The next aspect of the CD's version of Gentoo Linux we are going to explore is its **filesystem**. A **file** is a sequence of numbers that are associated with each other. A **filename** is the name that is given to a file so that the file can be referred to. A **filesystem** is a system for organizing a storage device (like a hard drive, flash drive, or CDROM drive) so that files can be stored on it. Most computers use an **hierarchal filesystem** which means that the filesystem can contain directories. **Directories**, which are also called **folders** in GUIs, are containers that can hold both **files** and other **directories**. A directory that is inside of another directory is called a **subdirectory**, and the top-level directory in an hierarchal filesystem is called the **root directory** **(most UNIX-like systems also have a subdirectory in the top-level root directory which is named "root". Even though they have the same name, they are different directories and the purpose of the root subdirectory will be explained later)**.

13.1.2 Issue the following commands:

```
livecd ~ # cd /

livecd / # pwd
/
```

13.1.3 The top-level directory in a Linux system is always given the forward slash symbol (**/**) as its name. The **cd** command stands for **Change Directory**, and it allows the user to change from the current **working directory** to another directory. The new directory is now the working directory, which means it is the directory we are currently working in. The **cd /** command issued above changed the terminal into the top-level root directory. The **pwd** command stands for **Print Working Directory**, and it shows which directory we are currently in. When the above pwd command was issued, it indicated that we were now in the / directory. Notice also that the **command prompt** changed from "**livecd ~ #"** to "**livecd / #"**.

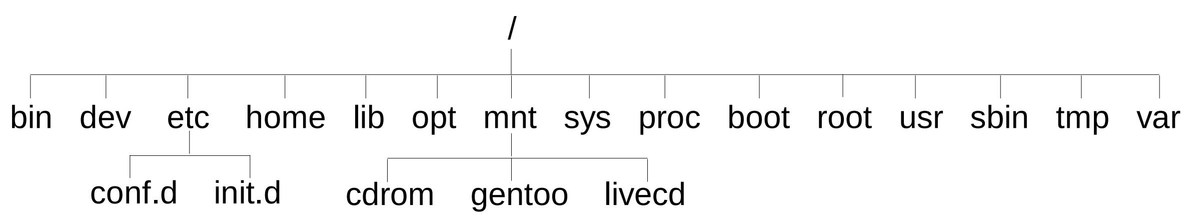
13.1.4 A command prompt is used in a **command line interface** (CLI) to inform the user that it is ready to accept typed input. Most command prompts can be configured to provide useful information to the user, and the one we are working with has been configured to show the name of the working directory. You might be wondering why the prompt had the working directory name change from **~** to **/** and this will be explained shortly.

13.1.5 Now that we are in the top-level root directory, issue the **ls** command (the 'l' is a lower case L, not a number one):

```
livecd / # ls
bin   dev  home       lib  opt   root  sbin  tmp  var
boot  etc  initramfs  mnt  proc  run   sys   usr
```

13.1.6 The **ls** command stands for **List directory** and it shows the contents of the working directory. A directory can contain files or **subdirectories**, and a subdirectory is simply a directory that is inside of another directory. Most of the names that the above **ls** command listed are the names of the standard subdirectories that are present in the root directory of most Gentoo Linux systems. Figure 3 shows most of the upper levels of this filesystem along with some of the subdirectories inside of the **etc** and **mnt** directories that we will be working with soon.

Figure 3



13.1.7 You may be wondering how to tell the difference between the names of files and the names of directories when the **ls** command is executed. If the **-l** option (which is a lower case L and stands for **long listing**) is passed to the ls command, it will add extra information to the listing that will indicate this:

```
livecd / # ls -l
total 0
lrwxrwxrwx  1 root root   15 Mar 25 03:32 bin -> /mnt/livecd/bin
lrwxrwxrwx  1 root root   16 Mar 25 03:32 boot -> /mnt/livecd/boot
drwxr-xr-x 14 root root 3840 Mar 25 03:32 dev
drwxr-xr-x 49 root root 2320 Mar 25 03:47 etc
drwxr-xr-x  3 root root   80 Mar 25 03:47 home
drwxr-xr-x  2 root root   40 Mar 25 03:32 initramfs
lrwxrwxrwx  1 root root   15 Mar 25 03:32 lib -> /mnt/livecd/lib
drwxr-xr-x  5 root root  100 Mar 25 03:32 mnt
lrwxrwxrwx  1 root root   15 Mar 25 03:32 opt -> /mnt/livecd/opt
dr-xr-xr-x 78 root root    0 Mar 25 03:31 proc
drwx------  2 root root  100 Dec 13  2012 root
drwxr-xr-x  6 root root  380 Mar 25 03:45 run
```

```
lrwxrwxrwx  1 root root   16 Mar 25 03:32 sbin -> /mnt/livecd/sbin
dr-xr-xr-x 11 root root    0 Mar 25 03:31 sys
drwxrwxrwt  4 root root   80 Mar 25 03:32 tmp
lrwxrwxrwx  1 root root   15 Mar 25 03:32 usr -> /mnt/livecd/usr
drwxr-xr-x  9 root root  240 Dec 13  2012 var
```

13.1.8 This version of **ls** places each name in a separate row. The names that represent directories have a row that begins with a letter '**d**' and rows that begin with a letter '**l**' represent a **symbolic link** or reference to a separate directory or file. For example, the name **etc** above refers to a directory that is in the **/** directory but the name **bin** is a link that refers to another directory called **bin** which is inside a directory called **livecd**. The **livecd** directory, in turn, is inside the **mnt** directory.

13.1.9 **/mnt/livecd/bin** is called a **path**, and a **path is a concise method for indicating which files and directories are contained within other directories.** Paths are read from left to right. In this path, notice that the leftmost character is the '**/**' character which represents the top-level directory in the filesystem. The other '/' characters in the path are called **path separators**, and they are used to separate one path name from another.

13.1.10 A **file** has a '**-**' character at the beginning of its row. Since none of the above rows begin with a '-', the / directory does not contain any files.

13.1.11 Let's change into the **bin** directory and see what it contains:

```
livecd / # cd bin

livecd bin # pwd
/bin

livecd bin # ls
attr            chroot          fgrep     lsmod          rbash      tail
awk             cp              findmnt   mkdir          rc-status  tar
basename        cut             fuser     mkfifo         readlink   touch
bash            date            gawk      mknod          red        tr
bashlogin       dd              getfacl   mktemp         rm         true
bb              df              getfattr  more           rmdir      tty
brltty          dir             grep      mount          rnano      umount
brltty-config   dirname         groups    mountpoint     route      uname
brltty-install  dmesg           gunzip    mv             sed        uncompress
bunzip2         dnsdomainname   gzip      nano           seq        vdir
busybox         domainname      head      netstat        setfacl    vi
bzcat           du              hostname  nisdomainname  setfattr   vstp
bzip2           echo            ifconfig  passwd         sh         wc
cat             ed              kill      pidof          sleep      yes
chacl           egrep           ln        ping           sort       ypdomainname
chgrp           env             login     ping6          stty       zcat
```

```
chmod           expr           ls         ps              su
chown           false          lsblk      pwd             sync
```

13.1.12 The bin directory contains a significant number of files. These files are special because they contain numbers which represent machine language instructions that the CPU can execute directly. The name of this directory stands for **binary** because programmers sometimes refer to files that contain machine language instructions as **binaries**.

13.1.13 Generate a long listing for this directory so that we can confirm that the **bin** directory contains at least some files:

```
livecd bin # ls -l
total 6825
-rwxr-xr-x 1 root root    9576 Dec 13  2012 attr
lrwxrwxrwx 1 root root       4 Dec 11  2012 awk -> gawk
-rwxr-xr-x 1 root root   22056 Dec 11  2012 basename
-rwxr-xr-x 1 root root  652220 Dec 11  2012 bash
-rwxr-xr-x 1 root root     134 Dec 13  2012 bashlogin
lrwxrwxrwx 1 root root       7 Dec 13  2012 bb -> busybox
-rwxr-xr-x 1 root root  302204 Dec 13  2012 brltty
-rw-r--r-- 1 root root    1509 Dec 13  2012 brltty-config
-rwxr-xr-x 1 root root    3192 Dec 13  2012 brltty-install
lrwxrwxrwx 1 root root       5 Dec 13  2012 bunzip2 -> bzip2
-rwxr-xr-x 1 root root 1668168 Dec 13  2012 busybox
lrwxrwxrwx 1 root root       5 Dec 13  2012 bzcat -> bzip2
-rwxr-xr-x 1 root root   34304 Dec 13  2012 bzip2
-rwxr-xr-x 1 root root   42664 Dec 11  2012 cat
-rwxr-xr-x 1 root root    9616 Dec 13  2012 chacl
-rwxr-xr-x 1 root root   50824 Dec 11  2012 chgrp
-rwxr-xr-x 1 root root   46696 Dec 11  2012 chmod
-rwxr-xr-x 1 root root   50856 Dec 11  2012 chown
-rwxr-xr-x 1 root root   26216 Dec 11  2012 chroot
-rwxr-xr-x 1 root root   96104 Dec 11  2012 cp
-rwxr-xr-x 1 root root   34408 Dec 11  2012 cut
-rwxr-xr-x 1 root root   54920 Dec 11  2012 date
-rwxr-xr-x 1 root root   54988 Dec 11  2012 dd
-rwxr-xr-x 1 root root   83728 Dec 11  2012 df
-rwxr-xr-x 1 root root  100392 Dec 11  2012 dir
-rwxr-xr-x 1 root root   22056 Dec 11  2012 dirname
-rwxr-xr-x 1 root root   21964 Dec 13  2012 dmesg
lrwxrwxrwx 1 root root       8 Dec 11  2012 dnsdomainname -> hostname
lrwxrwxrwx 1 root root       8 Dec 11  2012 domainname -> hostname
-rwxr-xr-x 1 root root   95976 Dec 11  2012 du
-rwxr-xr-x 1 root root   22056 Dec 11  2012 echo
-rwxr-xr-x 1 root root   43728 Dec 13  2012 ed
-rwxr-xr-x 1 root root  124808 Dec 11  2012 egrep
--More--
```

13.1.14 The **ls -l | more** command does not send its output to the screen. Instead, the '**|**' symbol (which is typed by holding down the <shift> key and pressing the **backslash** '\' key) is used to **pipe** the output from the ls -l command into the **more** command. The **more** command is used to show long output one page at a time so that it can be seen. Press the <space> key to view the next page of output. You can keep pressing the <space> key until all of the output has been viewed, or you can press the 'q' key in order to exit the **more** command early.

13.1.15 If you look closely at the files that were listed, you will notice the names of three commands that we recently executed (**ls**, **pwd** and **more**). Most commands that are executed at a command line are simply executable files that are present somewhere in the filesystem. You may have noticed that the **cd** command is not present in the bin directory. We will cover the reason for this later.

13.1.16 We will now go back to the **/** directory, change to the **etc** directory, and then enter the **conf.d** directory which is inside the **etc** directory:

```
livecd bin # cd /

livecd / # pwd
/
livecd / # cd etc

livecd etc # pwd
/etc

livecd etc # cd conf.d

livecd etc # pwd
/etc/conf.d

livecd conf.d # ls
acpid         device-mapper  hostname     lvm         ntpd        sshd
alsasound     dmcrypt        hwclock      mdadm       partimaged  syslog-ng
apmd          dmesg          ip6tables    mdraid      pciparm     tmpfiles
autoconfig    espeakup       iptables     modules     pydoc-2.7   udev
bootmisc      fsck           keymaps      net         pydoc-3.2   urandom
consolefont   gpm            killprocs    netmount    rdate       wpa_supplicant
crypto-loop   hddtemp        local.start  nfs         rdnssd
dante-sockd   hdparm         localmount   ntp-client  rpcbind

livecd conf.d # ls -a
.             crypto-loop    hddtemp    local.start  nfs         rdnssd
..            dante-sockd    hdparm     localmount   ntp-client  rpcbind
acpid         device-mapper  hostname   lvm          ntpd        sshd
alsasound     dmcrypt        hwclock    mdadm        partimaged  syslog-ng
apmd          dmesg          ip6tables  mdraid       pciparm     tmpfiles
autoconfig    espeakup       iptables   modules      pydoc-2.7   udev
bootmisc      fsck           keymaps    net          pydoc-3.2   urandom
consolefont   gpm            killprocs  netmount     rdate       wpa_supplicant
```

13.1.17 After changing into the conf.d directory, I executed an **ls** command followed by an **ls -a** command. Can you see what the difference is? The **ls -a** command stands for **list all**, and it will show any hidden names that are in a directory. In this case, two hidden names are in the directory (which are **.** and **..**). If you execute an **ls -la** command, you will see that the . and .. names both refer to directories. These directories are special, however, because every directory in the filesystem contains these two hidden directories.

13.1.18 The . directory refers to the working directory and the .. directory refers to the directory that the working directory is inside of. We are currently working in the **/etc/conf.d** directory. If we issue a **cd ..** command, notice what happens:

```
livecd conf.d # pwd
/etc/conf.d

livecd conf.d # cd ..

livecd etc # pwd
/etc
```

13.1.19 The **cd ..** command placed us into the **etc** directory which is one above the **conf.d** directory that we were inside of. If we now execute a **cd .** command, notice that we remain in the **etc** directory because a single . refers to the working directory:

```
livecd etc # pwd
/etc

livecd etc # cd .

livecd etc # pwd
/etc
```

13.1.20 Change back to the top-level directory by executing a **cd ..** command or a **cd /** command. No matter where you are in the directory hierarchy, executing the **cd /** command will take you back to the top-level directory. You can also pass a path to the **cd** command and it will move you to the last directory on the path. Let's try this. In the following example, I will change to the top-level directory, change to the **/etc/conf.d** directory, and then change back to the top-level directory:

```
livecd / # pwd
/

livecd / # cd /etc/conf.d
```

```
livecd conf.d # pwd
/etc/conf.d

livecd conf.d # cd /

livecd / # pwd
/
```

13.1.21 Typing commands and paths can become tedious so most command line interfaces provide a feature to help with this. From the / directory, try typing **cd e** <tab>. You should see the rest of the **etc** directory's name automatically filled out. The <tab> key provides automatic command and path completion, and it saves a significant amount of typing. You should currently have **cd etc/** on your command line and if you now type **con** <tab>, the rest of the **conf.d**'s name will be automatically typed, and you can then change into this directory.

13.1.22 Here is a list of the standard subdirectories that are typically in a Gentoo Linux's root directory, along with a short explanation of what the purpose of each one is:

```
bin  - Contains utility programs that can be run from a command line.
boot - Holds a bootloader program and the kernel image that will be loaded.
dev  - Device drivers are bound to the names in this directory.
etc  - Contains most of the system's configuration files.
home - Each user on the system gets their own directory in the home directory.
lib  - Contains the system's library files and kernel modules.
mnt  - Removable storages devices are bound to the names in this directory.
opt  - Applications that are added to the system are often stored here.
proc - Special directory that shows live information about the kernel.
root - The superuser's home directory.
sbin - Contains utility programs that only the superuser can execute.
sys  - Similar to proc but allows parameters to also be changed.
tmp  - Space for temporary files.
usr  - Houses much of the software that is loaded on the system.
var  - Contains data that varies as the system runs, such as system logs.
```

14 Managing storage device complexity

14.1 There are many kinds of storage devices available today including CDROM drives, flash drives, and hard drives. Personal computers and servers usually use a hard drive as their main storage device, and this is the kind of storage device we will be installing Gentoo Linux onto. However, Gentoo Linux can be installed onto almost any of the wide range of storage device types that are currently available.

14.2 Storage device types are usually implemented in very different ways from each other. For example, a hard drive consists of one or more spinning metal disks which use a magnetic head to write information onto them and read information from them, while flash drives use solid state electronics for these tasks. Something that should be bothering you at this point is how all of these different types of storage devices, along with ones that are yet to be invented, are treated in a uniform way by the operating system.

15 Abstraction

15.1.1 Our world is an extremely complex place, and it is becoming more complex all the time. If we did not have ways to deal with all of this complexity, our civilization would collapse and we would be thrown back into stone age conditions. Fortunately, techniques do exist for managing complexity, and one of the most powerful of these techniques is called **abstraction**. **Abstraction** is the process of hiding certain details of a concept or object so that only those details that are important for a given purpose remain exposed.

15.1.2 As an example, consider the device that city planners use to measure the amount of traffic that passes a given point on a street during a certain time frame. The device usually consists of a rubber hose that is run across the road and attached to a box with a counter in it. Each time a vehicle's tires roll over the hose, the air pressure in the hose is momentarily increased, a sensor in the box senses this increase, and it advances a counter.

15.1.3 From the point of view of the counting system, it does not care whether the vehicle that has just run over it was a small red Chevy car, a green Volkswagen luxury sedan, or a white Mac truck. Details like this are not needed for the purpose of counting vehicles, and they would only serve as unwanted distractions. Therefore, the designers of the vehicle counter used abstraction to hide these unwanted details from the device so that only the properties they wanted to measure remained visible.

16 Interfaces

16.1.1 Abstraction is used heavily in all areas of computing in order to manage the enormous amounts of complexity contained within this field. One area of computing that takes great advantage of the process of abstraction involves the mechanisms that are used when one computing entity needs to communicate with another computing entity. These communications mechanisms are usually called **interfaces**.

16.1.2 Using the terminology of abstraction, an **interface** is an abstraction of an entity that can be used by other entities to communicate with it. Keyboards, hard drives, CDROM drives, flash drives, video cards, and sound cards all use interfaces to communicate with the computer's motherboard. Software also makes use of interfaces, and we will cover some examples of this later. Before we do, though, let's discuss an example of an interface that is in common use in the world today in order to gain a better understanding of how abstraction and interfaces work.

16.1.3 This example consists of the interface that humans use to drive a car. If you think about it, almost all of the details about how a car works have been hidden from the driver and the few details they absolutely have to deal with have been abstracted into an **automatic-car-operator interface**. This automatic-car-operator interface consists of a steering wheel, an accelerator pedal, a brake pedal, and a transmission selection lever which has Park, Drive and Neutral positions on it (most transmissions also have the ability to force the selection of lower gear ranges like L1 and L2 but we will ignore these for this discussion). For the most part, this automatic-car-operator interface is all a person needs to know in order to operate an automatic car, and a knowledge of this interface will give this person the ability to operate any car in the world.

16.1.4 The interesting thing is that interfaces are more durable (and almost more real) than their implementations. This can be illustrated by imagining a person who was put into hibernation in the 1950s and awakened today. The automatic-car-operator interface in use today is identical to the one used to operate 1950's automatic transmission cars. Therefore, this person would have no problem operating a modern car even though the details of how this automatic-car-operator interface are implemented have changed significantly since the 1950s.

16.1.5 The typical automatic car in the 1950s had a carbureted engine, rear wheel drive, drum brakes, and a hydraulically shifted transmission. The typical modern car has a fuel-injected engine, front wheel drive, disc brakes, and an electronically shifted transmission. A 1950s mechanic that had been placed in hibernation and awakened today would have to be completely retrained before being capable of working on a modern automobile. The reason for this is that a mechanic works with the implementation details behind the automatic-car-operator interface and these details are constantly changing.

17 Different types of devices can provide the same abstract interface

17.1.1 As discussed in the automatic-car-operator interface example, abstraction and interfaces can be used to manage the changes in complexity that occur when devices are improved over time. Abstraction and interfaces can also be used, however, to allow very different types of devices to act in a uniform way so that these differences are hidden from the users of these devices. The following example will use the automatic-car-operator interface in another way to illustrate this.

17.1.2 In the physical world, there are many things that move around in a primarily two dimensional plane. A partial list of these things include cars, trucks, buses, boats, motorcycles, hovercraft, snowmobiles, and horses. In theory, if a person knew how to use the automatic-car-operator interface, and if they needed to use a motorcycle but had never ridden one before, abstraction and interfaces could assist them.

17.1.3 If the motorcycle implemented the automatic-car-operator interface then, when the person looked at the motorcycle, all they would see was the automatic-car-operator interface. They would get 'into' the car, put it in drive, press the accelerator pedal, and drive away. The person would have no idea that they were actually riding a motorcycle. This concept can even be extended to something like a horse. If a given horse implemented the automatic-car-operator interface then, when a person who knew this interface (but did not know how to ride a horse) looked at the horse, all they would see is the automatic-car-operator interface and they could use this interface to 'drive' the horse. Any of the things in the above list could be made to implement the automatic-car-operator interface, and then it could be used by a person who only knew the automatic-car-operator interface to 'drive' it around.

17.1.4 Unix-like operating systems also use abstraction and interfaces to deal with the **complexity of change** and the **complexity of diversity** and this is covered in the next section.

18 Block devices and character devices are abstract interfaces

18.1.1 There are numerous kinds of devices that can be attached to a computer including mice, keyboards, sound cards, hard drives, CDROM drives, flash drives, and RAM drives. This creates a significant amount of complexity that needs to be managed, and the way that UNIX-like systems manage this complexity is by having all devices implement either the **character device interface** or the **block device interface**.

18.1.2 A **character device** communicates with a computer one byte at a time. Examples of devices that implement the character device interface include:

Mouse - Sends a series of bytes to the computer as it is moved and clicked.
Keyboard - Sends bytes to the computer as its keys are pressed.
Sound card - Receives a sequence of bytes from the computer and turns these into sounds.

18.1.3 **Block devices** communicate with the computer using groups or blocks of bytes. Examples of block devices include:

Hard drive - Uses spinning metal disks to hold information using magnetics.
CDROM drive - Uses spinning plastic disk to hold information using optics.
Flash drive - Uses computer chips to hold information.
RAM drive - A program that pretends that it is a physical storage devices but it stores its information in RAM.

19 All devices are bound to names in the /dev directory

19.1.1 Change into the /dev directory and execute an **ls -l** command (I have edited this listing to make it shorter.):

```
livecd dev # ls -l
total 4
crw-rw---- 1 root root  254,   0 Mar 25 03:31 0:0:0:0
crw-rw---- 1 root root  189, 129 Mar 25 03:32 2-1
crw-rw---- 1 root root  254,   1 Mar 25 03:32 2:0:0:0
drwxr-xr-x 2 root root       560 Mar 25 03:32 block
drwxr-xr-x 2 root root        80 Mar 25 03:32 bsg
crw------- 1 root root   10, 234 Mar 25 03:32 btrfs-control
drwxr-xr-x 3 root root        60 Mar 25 03:32 bus
drwxr-xr-x 2 root root      3000 Mar 25 03:32 char
crw------- 1 root root    5,   1 Mar 25 03:32 console
crw------- 1 root root   10,  62 Mar 25 03:32 cpu_dma_latency
crw------- 1 root root   10, 252 Mar 25 03:32 dac960_gam
crw-rw---- 1 root root   10, 236 Mar 25 03:32 device-mapper
crw-rw---- 1 root root  152,   3 Mar 25 03:32 discover
drwxr-xr-x 3 root root        60 Mar 25 03:32 disk
crw-rw---- 1 root root  152,   2 Mar 25 03:32 err
drwxr-xr-x 2 root root       140 Mar 25 03:31 etherd
crw-rw---- 1 root root   13,  64 Mar 25 03:32 event0
crw-rw---- 1 root root   13,  65 Mar 25 03:32 event1
crw-rw---- 1 root root   13,  66 Mar 25 03:32 event2
crw-rw---- 1 root video  29,   0 Mar 25 03:32 fb0
lrwxrwxrwx 1 root root        13 Mar 25 03:32 fd -> /proc/self/fd
crw-rw---- 1 root root  152,   6 Mar 25 03:32 flush
crw-rw-rw- 1 root root    1,   7 Mar 25 03:32 full
crw-rw-rw- 1 root root   10, 229 Mar 25 03:32 fuse
srwxrwxrwx 1 root root         0 Mar 25 03:32 gpmctl
crw------- 1 root root  253,   0 Mar 25 03:32 hidraw0
prw------- 1 root root         0 Mar 25 03:32 initctl
```

```
drwxr-xr-x 4 root root       240 Mar 25 03:32 input
crw-rw---- 1 root root  152,   4 Mar 25 03:32 interfaces
crw-r----- 1 root kmem    1,   2 Mar 25 03:32 kmem
crw-r--r-- 1 root root    1,  11 Mar 25 03:32 kmsg
srw-rw-rw- 1 root root         0 Mar 25 03:32 log
crw------- 1 root root   10, 237 Mar 25 03:32 loop-control
brw-rw---- 1 root disk    7,   0 Mar 25 03:32 loop0
brw-rw---- 1 root disk    7,   1 Mar 25 03:32 loop1
brw-rw---- 1 root disk    7,   2 Mar 25 03:32 loop2
brw-rw---- 1 root disk    7,   3 Mar 25 03:32 loop3
brw-rw---- 1 root disk    7,   4 Mar 25 03:32 loop4
brw-rw---- 1 root disk    7,   5 Mar 25 03:32 loop5
brw-rw---- 1 root disk    7,   6 Mar 25 03:32 loop6
brw-rw---- 1 root disk    7,   7 Mar 25 03:32 loop7
drwxr-xr-x 2 root root        60 Mar 25 03:32 mapper
crw------- 1 root root   10, 227 Mar 25 03:32 mcelog
-rw-r--r-- 1 root root         3 Mar 25 03:32 mdev.seq
crw------- 1 root root   10,  58 Mar 25 03:32 megadev0
crw-r----- 1 root kmem    1,   1 Mar 25 03:32 mem
crw-rw---- 1 root root   13,  63 Mar 25 03:32 mice
lrwxrwxrwx 1 root root        15 Mar 25 03:32 mouse -> /dev/input/mice
crw-rw---- 1 root root   13,  32 Mar 25 03:32 mouse0
crw-rw---- 1 root root   13,  33 Mar 25 03:32 mouse1
crw------- 1 root root   10, 221 Mar 25 03:32 mpt2ctl
drwxr-xr-x 2 root root        60 Mar 25 03:32 net
crw------- 1 root root   10,  61 Mar 25 03:32 network_latency
crw------- 1 root root   10,  60 Mar 25 03:32 network_throughput
crw-rw-rw- 1 root root    1,   3 Mar 25 03:32 null
crw-r----- 1 root kmem    1,   4 Mar 25 03:32 port
crw------- 1 root root  108,   0 Mar 25 03:32 ppp
crw------- 1 root root   10,   1 Mar 25 03:32 psaux
crw-rw-rw- 1 root tty     5,   2 Mar 25 04:23 ptmx
drwxr-xr-x 2 root root         0 Mar 25 03:31 pts
brw-rw---- 1 root disk    1,   0 Mar 25 03:32 ram0
brw-rw---- 1 root disk    1,   1 Mar 25 03:32 ram1
brw-rw---- 1 root disk    1,  10 Mar 25 03:32 ram10
<snip>
```

19.2 In the **ls -l** long listing, **character devices** have a '**c**' in the left column and **block devices** have a '**b**'. On the computer I generated this list on, the hard drive is attached to **/dev/sda**. We will be using **/dev/sda** shortly to access the main hard drive so that we can prepare it for holding our Gentoo Linux installation.

19.3 Before we do that, however, I want you to experiment with the mouse device so that you can get a better feel for how devices work. Change into the **/dev/input** directory and execute the following commands:

```
livecd dev # cd /dev/input
```

```
livecd input # pwd
```

```
/dev/input

livecd input # ls
event0  event1  mice  mouse0 <snip>

livecd input # hexdump mice
0000000 0008 2802 ff00 fe38 28ff ff00 0028 28ff
0000010 ff00 0028 28ff ff00 0028 28fd fe00 0028
0000020 28fe fe00 0028 28fe fe00 0028 28fe fe00
<snip>
```

19.4 After you have entered the **hexdump** command, move your mouse around on the screen, and notice what happens. The mouse is generating numbers as it is being moved, and it is sending these numbers one at a time to the **mouse driver** (which is part of the kernel). The mouse driver, in turn, is attached to the file named **/dev/input/mice** so that it is easily accessible to other programs in the system.

19.5 The **hexdump** command is designed to open a character device (or a file) and then display each number that is sent by the device to the screen. By default, hexdump displays numbers in hexadecimal format, although it can be configured to display the numbers in other formats too. When you are finished sending numbers to the hexdump command with the mouse, hold down the **<ctrl>** key on your keyboard and then press the **'c'** key. **<ctrl> c** sends a signal to a program that tells it to exit. If you run a program from the command line and you can not get it to stop running, entering **<ctrl> c** will usually force it to exit.

19.6 All devices that are attached to the computer are bound to a name somewhere in the **/dev** directory. Now that you have a better understanding of how devices are accessed in a UNIX-like system, we are going to prepare the main storage device so that Gentoo Linux can be loaded onto it.

20 Partitioning the main storage device

20.1 Most PCs use a hard drive as their main storage device, so we are going to assume that you are going to install Gentoo Linux on a hard drive. Hard drives implement the **block device interface**, which means they communicate with the computer using blocks of numbers instead of one number at a time like character devices do. Hard drive block devices have such large capacities, however, that they are often made to appear as a set of smaller block devices (called **logical drives)** in order to increase their manageability. The process of making a hard drive look like a group of smaller block devices is called **partitioning**. Before you partition your hard drive, let's look at where it is attached inside of the **/dev directory**.

20.2 Change into the **/dev** directory and issue a **ls -l sda** command. If you pass a name to the **ls** command, it will just list that one name instead of all the names in a directory.

```
livecd dev # cd /dev
```

```
livecd dev # ls -l sda
brw-rw---- 1 root disk 8, 0 Mar 23 04:33 sda
```

20.3 If your main storage device is an IDE (Integrated Drive Electronics) hard drive, then it will usually be attached to the name **sda** in the **/dev** directory Notice that this is a block device because a '**b**' is listed in the leftmost column. What we are going to do is use the **fdisk** command to **partition** the **sda** drive into three smaller block devices called **sda1**, **sda2**, and **sda3**.

20.4 First, let's have **fdisk** show us information about all of the storage devices that are currently attached to the computer by passing it the **-l** option, which stands for 'List partitions' ('l' is a lower case L):

```
livecd dev # fdisk -l

Disk /dev/sda: 8 GiB, 8589934592 bytes, 16777216 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
```

The above information is what **fdisk** listed on the VirtualBox virtual computer that I used to prepare the materials for this class. It indicates that the virtual computer has one IDE hard drive attached to it at **/dev/sda** and the size of this drive is 8 **Gigabytes**. The drive has not been partitioned yet, and a valid partition table does not yet exist on the drive.

When you issue the **fdisk -l** command on a machine that already has an operating system on it, partitions probably exist on the drive and these will be listed. As an example, here is the information that was listed when I ran **fdisk -l** on my portable computer:

```
the_count tkosan # fdisk -l

Disk /dev/sda: 80.0 GB, 80026361856 bytes
240 heads, 63 sectors/track, 10337 cylinders
Units = cylinders of 15120 * 512 = 7741440 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1               1           5       37768+  83  Linux
/dev/sda2               6          72      506520   82  Linux swap / Solaris
/dev/sda3              73        8469    63481320   83  Linux
```

20.5 This hard drive has an 80 Gigabyte capacity and it has been partitioned into three smaller block devices called **sda1**, **sda2** and **sda3**. If your computer has already been partitioned, the first thing you will need to do when you execute the **fdisk** command is to delete any existing partitions on the drive.

20.6 If your machine has more than one **IDE** drive, the second drive will be named **hdb**, the third one **hdc**, and so on. If your machine is using **SCSI** hard drives instead of IDE hard drives, the SCSI drives will be named **sda**, **sdb**, **sdc**, etc.

20.7 Let us now use **fdisk** to **partition** your main hard drive. Assuming your hard drive is named **sda**, execute the following command, and then type in the commands highlighted in green:

```
livecd dev # fdisk /dev/sda

Welcome to fdisk (util-linux 2.21.2).

Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.

Device does not contain a recognized partition table
Building a new DOS disklabel with disk identifier 0x92ad8a6d.

Command (m for help): n
Partition type:
   p   primary (0 primary, 0 extended, 4 free)
   e   extended
Select (default p): p
Partition number (1-4, default 1): 1
First sector (2048-16777215, default 2048): <enter>
Using default value 2048
Last sector, +sectors or +size{K,M,G} (2048-16777215, default 16777215): +32M

Created a new partition 1 of type 'Linux' and of size 32 MiB.

Command (m for help): p

Disk /dev/sda: 8 GiB, 8589934592 bytes, 16777216 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0xa5dc726b

Device     Boot Start   End Sectors Size Id Type
/dev/sda1        2048 67583   65536  32M 83 Linux

Command (m for help): n
Partition type:
```

```
   p   primary (1 primary, 0 extended, 3 free)
   e   extended
Select (default p): p
Partition number (1,2, default 2): 2
First sector (67584-16777215, default 67584): <enter>
Using default value 67584
Last sector, +sectors or +size{K,M,G} (67584-16777215, default 16777215): +512M

Created a new partition 2 of type 'Linux' and of size 512 MiB.

Command (m for help): p

Disk /dev/sda: 8 GiB, 8589934592 bytes, 16777216 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0xa5dc726b

Device     Boot Start     End Sectors  Size Id Type
/dev/sda1        2048   67583   65536   32M 83 Linux
/dev/sda2       67584 1116159 1048576  512M 83 Linux

Command (m for help): t
Partition number (1,2, default 2): 2
Hex code (type L to list codes): L

 0  Empty           24  NEC DOS         81  Minix / old Lin bf  Solaris
 1  FAT12           27  Hidden NTFS Win 82  Linux swap / So c1  DRDOS/sec (FAT-
 2  XENIX root      39  Plan 9          83  Linux           c4  DRDOS/sec (FAT-
 3  XENIX usr       3c  PartitionMagic  84  OS/2 hidden C:  c6  DRDOS/sec (FAT-
 4  FAT16 <32M      40  Venix 80286     85  Linux extended  c7  Syrinx
 5  Extended        41  PPC PReP Boot   86  NTFS volume set da  Non-FS data
 6  FAT16           42  SFS             87  NTFS volume set db  CP/M / CTOS / .
 7  HPFS/NTFS/exFAT 4d  QNX4.x          88  Linux plaintext de  Dell Utility
 8  AIX             4e  QNX4.x 2nd part 8e  Linux LVM       df  BootIt
 9  AIX bootable    4f  QNX4.x 3rd part 93  Amoeba          e1  DOS access
 a  OS/2 Boot Manag 50  OnTrack DM      94  Amoeba BBT      e3  DOS R/O
 b  W95 FAT32       51  OnTrack DM6 Aux 9f  BSD/OS          e4  SpeedStor
 c  W95 FAT32 (LBA) 52  CP/M            a0  IBM Thinkpad hi eb  BeOS fs
 e  W95 FAT16 (LBA) 53  OnTrack DM6 Aux a5  FreeBSD         ee  GPT
 f  W95 Ext'd (LBA) 54  OnTrackDM6      a6  OpenBSD         ef  EFI (FAT-12/16/
10  OPUS            55  EZ-Drive        a7  NeXTSTEP        f0  Linux/PA-RISC b
11  Hidden FAT12    56  Golden Bow      a8  Darwin UFS      f1  SpeedStor
12  Compaq diagnost 5c  Priam Edisk     a9  NetBSD          f4  SpeedStor
14  Hidden FAT16 <3 61  SpeedStor       ab  Darwin boot     f2  DOS secondary
16  Hidden FAT16    63  GNU HURD or Sys af  HFS / HFS+      fb  VMware VMFS
17  Hidden HPFS/NTF 64  Novell Netware  b7  BSDI fs         fc  VMware VMKCORE
18  AST SmartSleep  65  Novell Netware  b8  BSDI swap       fd  Linux raid auto
1b  Hidden W95 FAT3 70  DiskSecure Mult bb  Boot Wizard hid fe  LANstep
1c  Hidden W95 FAT3 75  PC/IX           be  Solaris boot    ff  BBT
1e  Hidden W95 FAT1 80  Old Minix
Hex code (type L to list codes): 82
Changed system type of partition 2 to 82 (Linux swap / Solaris)
```

```
Command (m for help): p

Disk /dev/sda: 8 GiB, 8589934592 bytes, 16777216 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0xa5dc726b

Device     Boot Start     End Sectors  Size Id Type
/dev/sda1        2048   67583   65536   32M 83 Linux
/dev/sda2       67584 1116159 1048576  512M 82 Linux swap / Solaris

Command (m for help): n
Partition type:
   p   primary (2 primary, 0 extended, 2 free)
   e   extended
Select (default p): p
Partition number (3,4, default 3): 3
First sector (1116160-16777215, default 1116160): <enter>
Last sector, +sectors or +size{K,M,G} (1116160-16777215, default 16777215): <enter>

Created a new partition 3 of type 'Linux' and of size 7.5 GiB.

Command (m for help): p

Disk /dev/sda: 8 GiB, 8589934592 bytes, 16777216 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0xa5dc726b

Device     Boot   Start      End  Sectors  Size Id Type
/dev/sda1          2048    67583    65536   32M 83 Linux
/dev/sda2         67584  1116159  1048576  512M 82 Linux swap / Solaris
/dev/sda3       1116160 16777215 15661056  7.5G 83 Linux

Command (m for help): a
Partition number (1-3, default 3): 1

The bootable flag on partition 1 is enabled now.

Command (m for help): p

Disk /dev/sda: 8 GiB, 8589934592 bytes, 16777216 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0xa5dc726b

Device     Boot   Start      End  Sectors  Size Id Type
/dev/sda1  *       2048    67583    65536   32M 83 Linux
/dev/sda2         67584  1116159  1048576  512M 82 Linux swap / Solaris
```

```
/dev/sda3       1116160 16777215 15661056  7.5G 83 Linux

Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.
livecd / # fdisk -l

Disk /dev/sda: 8 GiB, 8589934592 bytes, 16777216 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0xa5dc726b

Device     Boot   Start      End  Sectors  Size Id Type
/dev/sda1  *       2048    67583    65536   32M 83 Linux
/dev/sda2         67584  1116159  1048576  512M 82 Linux swap / Solaris
/dev/sda3       1116160 16777215 15661056  7.5G 83 Linux
```

20.8 The last thing I want you to do before we move on to the next step is to look in the **/dev** directory to see that the new partitions/block devices (**sda1**, **sda2** and **sda3**) have been added there so that they can be accessed by other parts of the system (the asterisk at the end of sda is called a **wildcard** character, and it will match any group of characters that start in the position it is put in.):

```
livecd dev # cd /dev

livecd dev # ls -l sda*
brw-rw---- 1 root disk 8, 0 Mar 25 04:37 sda
brw-rw---- 1 root disk 8, 1 Mar 25 04:37 sda1
brw-rw---- 1 root disk 8, 2 Mar 25 04:37 sda2
brw-rw---- 1 root disk 8, 3 Mar 25 04:37 sda3
```

20.9 We will be using the names **/dev/sda1**, **/dev/sda2** and **/dev/sda3** throughout the rest of the installation process to access the partitions we have created.

20.10 THIS IS A GOOD BREAKING POINT IF YOU DO NOT HAVE TIME TO WORK THROUGH SECTION 20.

20.11 You can close VirtualBox by selecting Machine -> Close.

20.12 When the "Close Virtual Machine" dialog is shown, select "Save the machine state" option then select "Ok".

20.13 You can close the VirtualBox application and when you open it again, you simply have to select the "Start" button to resume where you left off.

21 Placing filesystems on the partitions

21.1 When a new partition has been created, it is unable to have files and directories placed on it until it has been formatted with a specific **filesystem type**. The Linux kernel is able to work with a significant number of filesystems and here is a partial list of the ones it supports:

```
Ext2
Ext3
Reiserfs
JFS
XFS
OCFS2
Minix
ISO 9660
MSDOS
VFAT
NTFS
Amiga FFS
Apple Macintosh
BeOS
SquashFS
OS/2 HPFS
```

21.2 Any of these filesystems could be placed on the partitions we have created, but we are only going to use the two most common ones used with GNU/Linux systems, which are **Ext2** and **Ext3**.

21.3 The **Ext2 filesystem**, which stands for **second extended filesystem**, was one of the earliest filesystems that was supported by the Linux kernel. It is still very popular, but one of its drawbacks is that it does not support **journaling**. If the power is suddenly removed on a non-journaling filesystem like **Ext2**, much of the information that was about to written to the disk was still in **RAM** and it becomes lost. This sudden loss of disk information often results in damaged files that will need to be repaired during the next system boot.

21.4 A **journaling filesystem** solves this problem by recording the additions and changes that are about to be made to the disk in a log. The log is usually updated every few seconds. If the computer loses power, the log can be used to automatically restore the filesystem during the next system boot. The **Ext3** filesystem is an extension to the Ext2 filesystem that supports journaling along with some other advanced features. These extended capabilities, however, mean that an Ext3 filesystem requires more resources than an Ext2 filesystem does, so one must decide which filesystem is appropriate for a given use.

21.5 We are going to apply the **Ext2** filesystem to the **/dev/sda1 boot** partition and the **Ext3** filesystem to the **/dev/sda3** top-level **root** filesystem. The **boot** filesystem is only used during the boot process, and it is mostly read from during this time. Therefore, it does not need the extended capabilities that the Ext3 filesystem offers. You can apply the **Ext2** filesystem to **partition 1** using the **mke2fs** command as follows:

```
livecd / # mke2fs /dev/sda1
mke2fs 1.42.13 (17-May-2015)
Creating filesystem with 32768 1k blocks and 8192 inodes
Filesystem UUID: 9e9b9654-4c6c-4b90-8235-85ae8449c594
Superblock backups stored on blocks:
        8193, 24577

Allocating group tables: done
Writing inode tables: done
Writing superblocks and filesystem accounting information: done
```

21.6 Since the **root partition** is going to hold the main directory hierarchy for our Gentoo Linux installation, it is a good idea to use the **Ext3** filesystem with this partition. The **mke2fs** command is also used to apply the **Ext3** filesystem to a partition, but a **-j** (journaling) option needs to be passed to this command to tell it to create an Ext3 filesystem instead of an Ext2 filesystem:

```
livecd / # mke2fs -j /dev/sda3
mke2fs 1.42.13 (17-May-2015)
Creating filesystem with 1957632 4k blocks and 489600 inodes
Filesystem UUID: 8c17601d-5bf1-409f-b38b-14f0edb50d6f
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632

Allocating group tables: done
Writing inode tables: done
Creating journal (32768 blocks): done
Writing superblocks and filesystem accounting information: done
```

21.7 The last partition that needs to be initialized is the **/dev/sda2 swap partition**. The swap partition does not use a normal filesystem, and instead it has a special swap format applied to it. The **mkswap** command is used to prepare the swap partition for use, and the **swapon** command is used to enable it:

```
livecd / # mkswap /dev/sda2
Setting up swapspace version 1, size = 512 MiB (536866816 bytes)
no label, UUID=013245c7-5fab-4901-b9de-5a74e61de341
```

```
livecd / # swapon /dev/sda2
```

21.8 The **boot** and **root** filesystems are now ready to have **files** and **directories** added to them and we will do this in the next section.

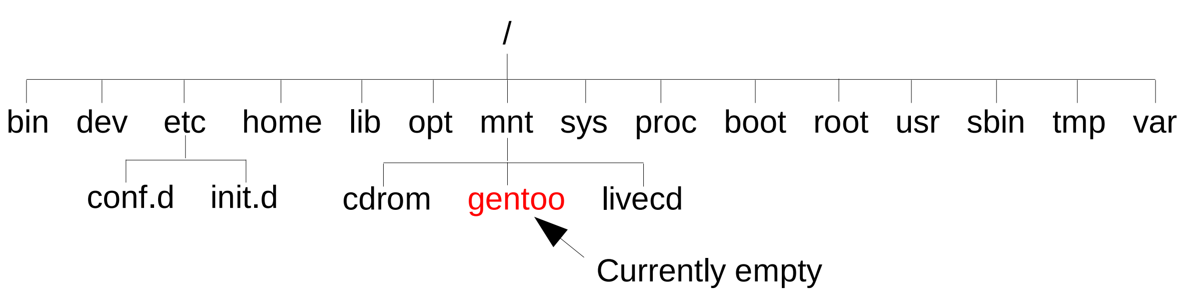
22 Mounting the boot and root partitions

22.1 Now that the **boot** partition (**/dev/sda1**) and the **root** partition (**/dev/sda3**) have had filesystems applied to them, the next step is to make these partitions accessible to the rest of the system. In UNIX-like systems, **the way that devices with filesystems on them are made accessible is by attaching them to a directory in the main directory hierarchy.** The process of attaching a device to a directory is called **mounting** and it is done using the **mount** command. The place where a device is mounted to a directory hierarchy is called a **mount point**.

22.2 Figure 5 shows the upper levels of the CD's directory hierarchy. In a moment we are going to **mount** the **root partition** to the **/mnt/gentoo directory** but before we do, let's change into this directory and see what is there:

```
livecd gentoo # cd /mnt/gentoo

livecd gentoo # pwd
/mnt/gentoo

livecd gentoo # ls -l
total 0
```

Figure 5



22.3 As you can see, the **/mnt/gentoo** directory is empty. This directory's only purpose for being on the CD is to provide a place to mount the **/dev/sda3 root partition** so that it can be accessed. Let's do this now. First, you must change out of the **/mnt/gentoo** directory and a safe directory to change into is the top-level root directory:

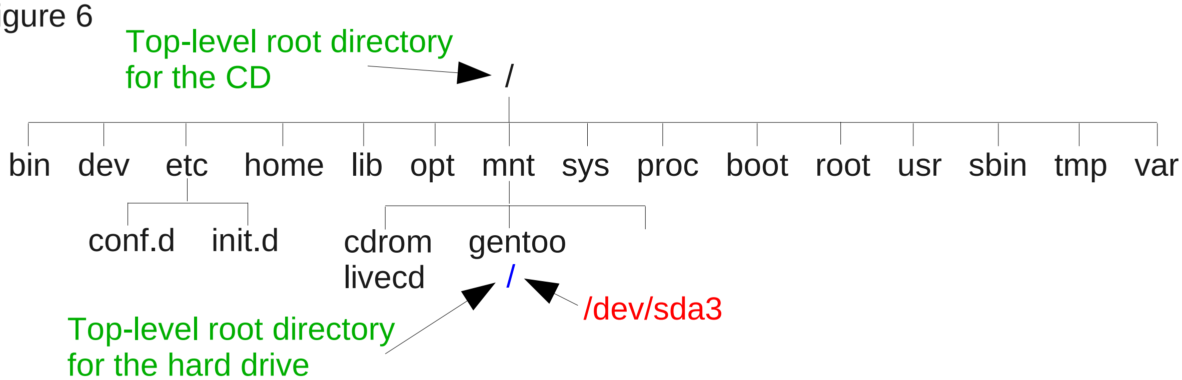
```
livecd gentoo # cd /

livecd / # pwd
/
```

22.4 Now, mount the **/dev/sda3** partition to the **/mnt/gentoo** directory, change back into the **/mnt/gentoo** directory, and then execute an **ls -l** command in order to see if anything appeared there:

```
livecd / # mount /dev/sda3 /mnt/gentoo

livecd / # cd /mnt/gentoo

livecd gentoo # ls -l
total 16
drwx------ 2 root root 16384 Mar 21 01:46 lost+found
```

22.5 Notice that there is now a subdirectory inside of the /mnt/gentoo directory called **lost+found**. **mke2fs** always places a directory called **lost+found** in a partition after it is done preparing it. The fact that this directory is present inside the **/mnt/gentoo** directory means that we have successfully mounted the **/dev/sda3** partition to **/mnt/gentoo**. (See Figure 6).

Figure 6



22.6 Figure 6 shows that the **/dev/sda3** partition has been mounted to the **gentoo** directory, and the **gentoo** directory is inside of the **/mnt** directory. The **/mnt** directory's name is short for **mount** and **normally its purpose is to provide a place to mount removable storage devices**, such as flash drives and CDROMs. In this case, however, we are using the **/mnt/gentoo** directory as a place to **temporarily mount the partitions** we have created so that we can place information on them. After we are done placing information on the /dev/sda1 and /dev/sda3 partitions, they will be capable of booting the PC without the help of the CD.

22.7 As you study Figure 6, another thing you should notice is that a label has been added to the top of the figure which reads "**Top-level root directory for the CD**" and a label near the bottom of the figure has been added which reads "**Top-level root directory for the hard drive**". Both labels point to a '**/**' top-level root directory symbol, but the CD's top-level root directory is currently the **active** one. This means that if you type cd /, it will be the **CD's** root directory that you will be placed into.

22.8 What we are in the process of doing is creating a copy of the standard Gentoo Linux directory hierarchy on the **/dev/sda3** partition. The next step is to create a **directory** called **boot** inside the **/dev/sda3** '**/**' partition and then mount the **/dev/sda1 boot** partition to this directory. The command that creates directories is called **mkdir**, and you must make sure you are in the **/mnt/gentoo** directory before using it:

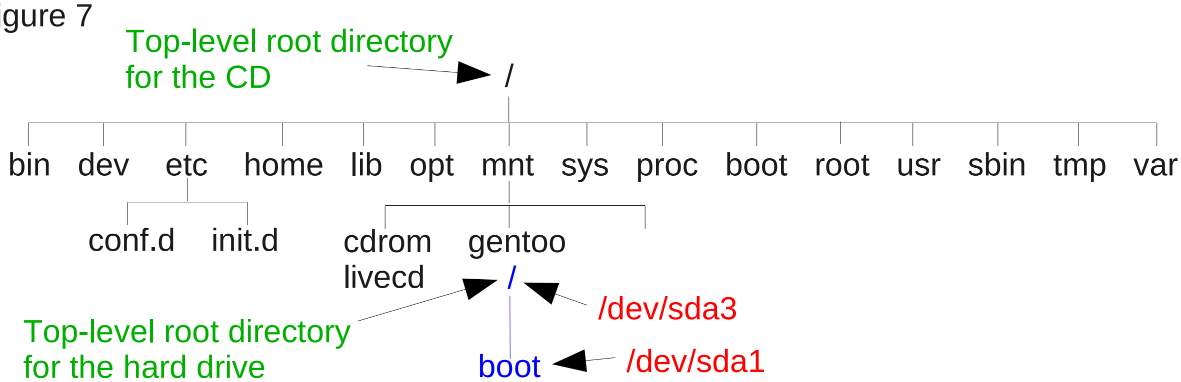
```
livecd gentoo # cd /mnt/gentoo

livecd gentoo # pwd
/mnt/gentoo

livecd gentoo # ls
lost+found
livecd gentoo # mkdir boot

livecd gentoo # ls
boot  lost+found

livecd gentoo # mount /dev/sda1 /mnt/gentoo/boot
```

22.9 The **mkdir** command creates a directory inside the current directory. In this case, a directory called **boot** was created in the **/mnt/gentoo** directory. The **/dev/sda1** boot partition was then mounted to this newly-created boot directory so it could be accessed. Let's change into the **boot** directory and see what it contains:

```
livecd gentoo # cd boot

livecd boot # pwd
/mnt/gentoo/boot

livecd boot # ls
lost+found
```

22.10 Notice that the **/mnt/gentoo/boot** directory also contains a lost+found directory which means that the **/dev/sda1** partition has been successfully mounted to it. The directory hierarchy now looks like the one shown in Figure 7.

Figure 7



23 Setting the system's date and time

23.1 Up until this point, we have not been concerned about whether or not the system's date and time were set correctly. Before we start adding files and directories to the /dev/sda3 root and /dev/sda2 boot partitions, the **date** and **time** need to be correct because each file and directory is given a **timestamp** of when it was created and incorrect timestamps will eventually cause problems.

23.2 Check the system time using the **date** command:

```
livecd boot # date
Mon Mar 21 01:50:02 UTC 2016
```

23.3 The CD is configured to use **Coordinated Universal Time** (UTC) and if your system's **date** or **time** are incorrect, they can be set using the **date** command by passing it a new date in the format **MMDDhhmmYYYY** (**M**onth, **D**ay, **h**our, **m**inute, **Y**ear). For example, to set the time to February 21st 17:32 2007, pass the parameter **022117322007** to the **date** command:

```
livecd boot # date 022117322007

Wed Feb 21 17:32:00 UTC 2007
```

23.4 The **date** command sets the operating system's copy of the date and time, but a **separate copy** of the date and time is kept on a **clock chip** on the motherboard. Each time the system boots, the time that is in the clock chip is used to initialize the operating system's date and time. If your system's time was incorrect and you used the **date** command to set it, you should also execute the following **hwclock** command to copy the operating **sys**tem's time **to** the **h**ardware **c**lock:

```
livecd boot # hwclock --systohc
```

23.5 The **hwclock** command is used to communicate with the clock chip on the motherboard, and the **--systohc** option tells the **hwclock** command to set the clock chip to the operating system's date and time.

24 Preparing to extract the standard Gentoo directory hierarchy into the /dev/sda3 top-level root partition

24.1 As shown in Figure 7, we have reached the point where we have created a top-level '**/**' **root** directory in the **/dev/sda3 root** partition, created a directory called **boot** in this **root** directory, and mounted the **/dev/sda1 boot** partition to the **boot** directory. The **boot** directory is one of the standard directories that is in a Gentoo Linux system's top-level root directory, and our next step is to place the rest of the standard Gentoo Linux directories into this **root** directory.

24.2 Since Gentoo Linux systems use a standard directory hierarchy, the Gentoo installation process has users place a **pre-created copy** of this directory hierarchy onto their hard drives during the installation process. The steps involved in this process are as follows:

1) The Gentoo developers create a standard Gentoo Linux directory hierarchy from scratch on a Gentoo development machine. The programs, configuration information and documentation that are used on most Gentoo Linux systems are placed into this directory hierarchy.
2) The Gentoo developers place the complete directory hierarchy into a single compressed file and then generate one or more digest numbers (or digital fingerprints) for it.
3) The compressed file containing the directory hierarchy, along with the file containing the digest numbers, are placed on the Gentoo servers so that users can download them.

24.3 The compressed file that contains the standard Gentoo Linux directory hierarchy we are going to use is called **stage3-amd64-20180311T214502Z.tar.xz**, and its companion file that contains this file's digest numbers is called **stage3-stage3-amd64-20180311T214502Z.tar.xz.DIGESTS**. Both files can be obtained from **http://patternmatics.org/ssu/etec1302/gentoo_2018** using a program called **wget** (web get).

24.4 We first **make sure that we are inside of the /mnt/gentoo** directory and then we can download both of these files into this directory using the **wget** program:

```
livecd / # cd /mnt/gentoo

livecd gentoo # pwd
/mnt/gentoo

livecd gentoo # ls
boot  lost+found

NOTE:THE FOLLOWNG COMMAND IS ON A SINGLE LINE.

livecd gentoo # wget -c http://patternmatics.org/ssu/etec1302/gentoo_2018/stage3-
amd64-20180311T214502Z.tar.xz
--2014-03-25 05:02:53--  http://patternmatics.org/ssu/etec1302/gentoo_2018/stage3-
amd64-20180311T214502Z.tar.xz
Connecting to 206.21.94.61:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 716 [application/x-bzip2]
Saving to: 'stage3-amd64-20180311T214502Z.tar.xz'

100%[======================================>] 716         --.-K/s   in 0s

2014-03-25 05:02:53 (24.8 MB/s) - 'stage3-amd64-20180311T214502Z.tar.xz' saved
[716/716]
```

24.5 After the **stage3-amd64-20180311T214502Z.tar.xz** file is finished downloading, make sure it is in the **/mnt/gentoo** directory:

```
livecd gentoo # pwd
/mnt/gentoo

livecd gentoo # ls -l
total 232309
drwxr-xr-x 3 root root      1024 Mar 21 01:46 boot
drwx------ 2 root root     16384 Mar 21 01:46 lost+found
-rw-r--r-- 1 root root 237628216 Mar 21 01:53 stage3-amd64-20180311T214502Z.tar.xz
```

24.6 Now download the **stage3-amd64-20180311T214502Z.tar.xz.DIGESTS** file that contains the digest numbers for the **stage3-amd64-20180311T214502Z.tar.xz** file. In order to avoid having to type the whole filename, **try pressing the up arrow on your keyboard a few times**. All the commands you have previously typed are held in the command line's history memory and they can be accessed by pressing the up arrow (pressing the down arrow moves forward through the history.) Keep going back through your command line history until you reach the **wget** command you typed earlier. Edit the filename by adding the word **DIGESTS** to the end of it and then execute the following command:

```
livecd gentoo # wget -c http://patternmatics.org/ssu/etec1302/gentoo_2018/stage3-
amd64-20180311T214502Z.tar.xz.DIGESTS
Connecting to 192.168.1.10:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 716 [application/x-bzip2]
Saving to: 'stage3-amd64-20180311T214502Z.tar.xz.DIGESTS'

stage3-i686-2016031 100%[====================>]     716  --.-KB/s   in 0s

2016-03-21 01:56:58 (54.8 MB/s) - 'stage3-amd64-20180311T214502Z.tar.xz.DIGESTS'
saved [716/716]
```

24.7 Execute the **pwd** and **ls -l** commands again to make sure that both files are in the **/mnt/gentoo** directory:

```
livecd gentoo # pwd
/mnt/gentoo

livecd gentoo # ls -l
total 169397
drwxr-xr-x 3 root root      1024 Mar 25 04:45 boot
drwx------ 2 root root     16384 Mar 25 04:46 lost+found
-rw-r--r-- 1 root root 173261744 Mar 25  2014 stage3-amd64-20180311T214502Z.tar.xz
-rw-r--r-- 1 root root       716 Mar 25  2014 stage3-amd64-
20180311T214502Z.tar.xz.DIGESTS
```

24.8 Now that both files have been successfully downloaded to the proper place, we need to calculate the digest number for the main file and check this number against the copy that is in the DIGESTS file. The GNU/Linux command that runs the **SHA512 digest algorithm** on files is sha512 and a command that will copy the contents of a file to the screen is **cat** (concatenate):

```
livecd gentoo # sha512sum stage3-amd64-20180311T214502Z.tar.xz
af849ce65244ee6dd1ef2a75deefe143933e82bce7d46bfcb24e36413cb5455e4f50f1d5cb887dc8cef
84f70c2802ca1f09664b6d71cd3f129926d3dfa922424  stage3-amd64-20180311T214502Z.tar.xz

livecd gentoo #  cat stage3-amd64-20180311T214502Z.tar.xz.DIGESTS
# SHA512 HASH
af849ce65244ee6dd1ef2a75deefe143933e82bce7d46bfcb24e36413cb5455e4f50f1d5cb887dc8cef
84f70c2802ca1f09664b6d71cd3f129926d3dfa922424  stage3-amd64-20180311T214502Z.tar.xz
# WHIRLPOOL HASH
7a1c093c2e80d380671ebfbe9795656e6f035e17b2e936f6d0140cd2f5c6307dc28a2335a7274fa0c3c
65a75384ce9aa9d3561c1790ac2b066b6e86ec6eb4588  stage3-amd64-20180311T214502Z.tar.xz
# SHA512 HASH
9b774543d26d65f2d322786ec84071b47294caf6cd057c2ab6b0f70b91a994a1796d5756e4cab5223ba
e050fd2e3ea096c132148ef1c82c23f701afe25e868b0  stage3-amd64-
20180311T214502Z.tar.xz.CONTENTS
# WHIRLPOOL HASH
9a01f37f92f698d1000f235cab2488a4e99878aee14a6b61fb7d24364d3eedaa09068c4ee391fa46ad6
```

```
a59f3b74d6f67096bc03841a00a084bcd745cd48e997b  stage3-amd64-
20180311T214502Z.tar.xz.CONTENTS90b1a9242615c034b093c9a1b71823563334163193858
stage3-amd64-20180311T214502Z.tar.xz.CONTENTS
```

24.9 If both SHA512 digest numbers match, then your **stage3-amd64-20180311T214502Z.tar.xz** file is not corrupted and we can move on to the next step.

25 Extracting the standard Gentoo directory hierarchy into the /dev/sda3 top-level root partition

25.1 As indicated earlier, the **stage3-amd64-20180311T214502Z.tar.xz** file contains the core of a standard Gentoo Linux directory hierarchy. The **.tar** part of the filename indicates that this directory hierarchy was placed into the **Tape ARchive** format. One of the earliest devices that computers used for storing information was the **magnetic tape drive**. Early UNIX machines had a utility program called **tar** that was used to copy files and directories to a single file (sometimes called a **tarball**) that could be saved on magnetic tape. The **tar** program could also take a **tar** file that was on a magnetic tape and convert it back to the original files and directories. UNIX-like operating systems, such as Gentoo Linux, still use the **tar** program, but the archive files are used for more purposes than just storing on magnetic tape. One additional purpose is to send directory structures through the Internet.

25.2 The **.bz2** part of the **stage3-amd64-20180311T214502Z.tar.xz** file indicates that the tape archive information was compressed using the **bzip2** compression algorithm. A compressed file is usually much smaller than the original. Gentoo Linux makes significant use of **.tar.bz2 tarball files** for copying directory structures, source code, and documentation to user's computers.

25.3 Our next step is to **unzip** and **untar** the **stage3-amd64-20180311T214502Z.tar.xz** file that we placed into the **/mnt/gentoo** directory. This can be done in one step by changing to the **/mnt/gentoo** directory and executing the **tar xvjpf** command:

```
livecd / # cd /mnt/gentoo

livecd gentoo # pwd
/mnt/gentoo

livecd gentoo # ls
boot  lost+found  stage3-amd64-20180311T214502Z.tar.xz  stage3-amd64-
20180311T214502Z.tar.xz.DIGESTS
```

```
livecd gentoo # tar xvJpf stage3-amd64-20180311T214502Z.tar.xz
<snip>
```

25.4 As soon as the command begins executing, a list of all of the files and directories that are being uncompressed and untared is shown on the screen. There are a significant number of files and directories in the archive, so it will take a while for the process to complete. While you are waiting, let's look at the **options** that were passed to the **tar** command.

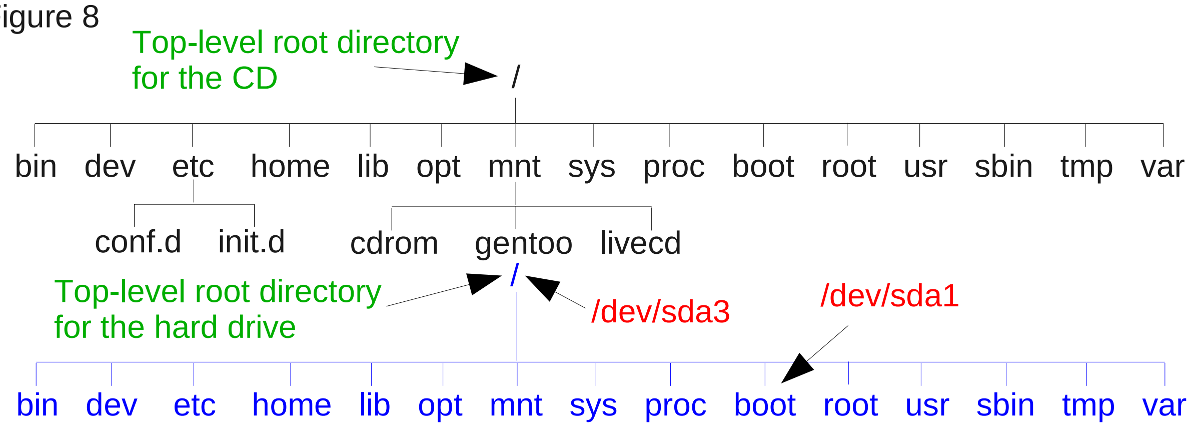
25.5 The **x** option indicates that we want to extract from an archive, not create one. The **v** option tells the tar command to be verbose with the information it prints to the screen during the extraction process. In verbose mode, the tar command will list the name of each file and directory to the screen as it is extracted. The **J** option tells the tar command that the archive has been compressed with the **xz** algorithm, and that it needs to be uncompressed before it can be untared. The **p** option indicates that the permissions for each directory and file should be preserved during the extraction process. The **f** option indicates that the archive is being extracted from a file.

25.6 After the extraction process is complete, execute the **ls** command to see the directories that have been created inside of your **/mnt/gentoo** directory:

```
livecd gentoo # pwd
/mnt/gentoo

livecd gentoo # ls
bin   home        mnt   run                                    sys
boot  lib         opt   sbin                                   tmp
dev   lost+found  proc  stage3-amd64-20180311T214502Z.tar.xz           usr
etc   media       root  stage3-amd64-20180311T214502Z.tar.xz.DIGESTS  var
```

25.7 Your system's directory hierarchy should now look similar to the one shown in Figure 8 (keep in mind, however, that only the upper levels of both directory hierarchies are shown.) Having two almost identical directory hierarchies on a system makes it easy to become confused about which directory you are in, so be careful as you change directories and use the **pwd** command frequently to check where you are.

Figure 8



26 Downloading and extracting the portage tree

26.1 Before any more software can be installed on the hard drive, an archived copy of the Gentoo Linux **portage tree** needs to be downloaded and extracted. **Portage** is Gentoo's **software package management system**. A **software package** is what programs and data are usually placed into so that they can be easily sent over the Internet to a user's computer for installation.

26.2 Most GNU/Linux distributions use **binary** packages, which means that the software has been precompiled and is ready for execution as soon as it is downloaded to the user's system and installed. Gentoo, however, does not use binary packages by default (although it is capable of using them). Instead, **portage** downloads the **source code** for applications that the user wants to install, and **compiles** them right on the user's machine. It then installs the binary code that was generated during the compilation process into the proper directories inside the standard Gentoo directory structure.

26.3 The command that is used to download, compile, and install software packages on Gentoo system is called **emerge**, and the instructions that tell emerge how to do this for each package are contained in the **portage tree**. The word **tree** as used here is another name for **directory hierarchy**.

26.4 We will explore the contents of the **portage tree** in a moment, but first you need to **download a compressed archive of it onto your computer**, check it to make sure it is not corrupted (using **md5sum**), and then extract it. Compressed archives of the **portage tree** are also called **snapshots** because the central copy of the portage tree on the Gentoo servers is constantly being updated by people all over the world. When a compressed archive of the tree is made at a given point in time, it is like taking a picture or snapshot of it, similar to the way that a camera takes a snapshot.

26.5 Make sure you are in the **/mnt/gentoo** directory, and then obtain the **portage snapshot** that has been placed at **http://206.21.94.61** (using **wget**). Next, obtain the companion **digest file** for the snapshot, generate a **md5** digest number for the snapshot, and make sure the snapshot file is not corrupted. If the portage snapshot file is okay, then extract it using the **tar xvjf** command:

```
livecd gentoo # pwd
/mnt/gentoo

livecd gentoo # wget -c http://patternmatics.org/ssu/etec1302/gentoo_2018/portage-
20180306.tar.bz2
--2016-03-21 04:26:47--  http://192.168.1.10/portage-20180306.tar.bz2
Connecting to 192.168.1.10:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 75514727 (72M) [application/x-bzip2]
Saving to: 'portage-20180306.tar.bz2'

portage-20160315.ta 100%[====================>]  72.02M   188MB/s   in 0.4s

2016-03-21 04:26:47 (188 MB/s) - 'portage-20180306.tar.bz2' saved
[75514727/75514727]

livecd gentoo # wget -c http://patternmatics.org/ssu/etec1302/gentoo_2018/portage-
20180306.tar.bz2.md5sum
--2016-03-21 04:26:55--  http://192.168.1.10/portage-20180306.tar.bz2.md5sum
Connecting to 192.168.1.10:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 59 [application/x-bzip2]
Saving to: 'portage-20180306.tar.bz2.md5sum'

portage-20160315.ta 100%[====================>]      59  --.-KB/s   in 0s

2016-03-21 04:26:55 (5.79 MB/s) - 'portage-20180306.tar.bz2.md5sum' saved [59/59]

livecd gentoo # ls
bin                       portage-20180306.tar.bz2.md5sum
boot                      proc
dev                       root
etc                       run
home                      sbin
lib                       stage3-amd64-20180311T214502Z.tar.xz
lost+found                stage3-amd64-20180311T214502Z.tar.xz.DIGESTS
media                     sys
mnt                       tmp
opt                       usr
portage-20180306.tar.bz2  var

livecd gentoo # md5sum portage-20180306.tar.bz2
```

```
<Verify the checksum number for the file.>

(Note: the 'C' in the following tar command is a capital 'C')

livecd gentoo # tar xvjf portage-20180306.tar.bz2 -C /mnt/gentoo/usr
```

26.6 After the portage snapshot has finished being extracted, change into the **usr** directory and execute an **ls** command:

```
livecd gentoo # cd usr

livecd usr # pwd
/mnt/gentoo/usr

livecd usr # ls
bin      lib    lib64    local    sbin   src  x86_64-pc-linux-gnu
include  lib32  libexec  portage  share  tmp
```

26.7 When the portage snapshot was extracted, a directory called **portage** was created in the **usr** directory, and it contains the **portage tree**. Now, change into the **portage** directory and execute another **ls** command:

```
livecd usr # cd portage

livecd portage # ls
app-accessibility  dev-qt            mate-extra      sci-misc
app-admin          dev-ruby          media-fonts     sci-physics
app-antivirus      dev-scheme        media-gfx       sci-visualization
app-arch           dev-tcltk         media-libs      scripts
app-backup         dev-tex           media-plugins   sec-policy
app-benchmarks     dev-texlive       media-radio     skel.ChangeLog
app-cdr            dev-util          media-sound     skel.ebuild
app-crypt          dev-vcs           media-tv        skel.metadata.xml
app-dicts          eclass            media-video     sys-apps
app-doc            games-action      metadata        sys-auth
app-editors        games-arcade      net-analyzer    sys-block
app-emacs          games-board       net-dialup      sys-boot
app-emulation      games-emulation   net-dns         sys-cluster
app-forensics      games-engines     net-firewall    sys-devel
app-i18n           games-fps         net-fs          sys-firmware
app-laptop         games-kids        net-ftp         sys-freebsd
app-leechcraft     games-misc        net-im          sys-fs
app-misc           games-mud         net-irc         sys-infiniband
app-mobilephone    games-puzzle      net-libs        sys-kernel
app-office         games-roguelike   net-mail        sys-libs
app-officeext      games-rpg         net-misc        sys-power
app-pda            games-server      net-nds         sys-process
app-portage        games-simulation  net-news        virtual
app-shells         games-sports      net-nntp        www-apache
app-text           games-strategy    net-p2p         www-apps
app-vim            games-util        net-print       www-client
app-xemacs         gnome-base        net-proxy       www-misc
```

```
dev-ada            gnome-extra       net-voip         www-plugins
dev-cpp            gnustep-apps      net-wireless     www-servers
dev-db             gnustep-base      net-zope         x11-apps
dev-dotnet         gnustep-libs      perl-core        x11-base
dev-embedded       gpe-base          profiles         x11-drivers
dev-games          gpe-utils         razorqt-base     x11-libs
dev-haskell        header.txt        rox-base         x11-misc
dev-java           java-virtuals     rox-extra        x11-plugins
dev-lang           kde-base          sci-astronomy    x11-proto
dev-libs           kde-misc          sci-biology      x11-terms
dev-lisp           licenses          sci-calculators  x11-themes
dev-lua            lxde-base         sci-chemistry    x11-wm
dev-ml             mail-client       sci-electronics  xfce-base
dev-perl           mail-filter       sci-geosciences  xfce-extra
dev-php            mail-mta          sci-libs
dev-python         mate-base         sci-mathematics
```

26.8 This is the **portage tree** and it contains a significant number of directories, each of which represents a **package category**. Each **package category directory**, in turn, contains subdirectories that hold the **software packages** that belong in a given category. Let's look inside one of the **category directories**, like **games-puzzle**, to see what packages it contains:

```
livecd portage # pwd
/mnt/gentoo/usr/portage

livecd portage # cd games-puzzle

livecd games-puzzle # pwd
/mnt/gentoo/usr/portage/games-puzzle

livecd games-puzzle # ls
4stattack      ensemblist      hexalate         nightsky       tiny-and-big
amoebax        fbg             hexamine         pathological   tod
anagramarama   fish-fillets    hoh-bin          pauker         tong
angrydd        five-or-more    icebreaker       penguzzle      toppler
arrows         flobopuyo       jag              picpuz         torrent
atomix         freesweep       jools            pingus         trimines
bastet         galaxis         kiki             pipepanic      triptych-demo
biniax2        gemdropx        krosswordpuzzle  pipewalker     twindistress
braincurses    gfifteen        krystaldrop      quadra         wakkabox
brainparty     glightoff       larry            quadrapassel   wizznic
brainworkshop  gnome-klotski   lightsoff        rezerwar       wmpuzzle
bubble-chains  gnome-sudoku    lmarbles         scramble       world-of-goo
candycrisis    gnome-tetravex  lpairs           sdl-jewels     world-of-goo-demo
color-lines    gnudoku         ltris            seatris        xblockout
colorcode      gnurobbo        magiccube4d      sgt-puzzles    xbomb
concentration  gottet          meandmyshadow    shaaft         xlogical
connectagram   gpe-lights      metadata.xml     skoosh         xpired
construo       greedy          mindless         splice         xtris
cutemaze       groundhog       mirrormagic      swell-foop     xwelltris
cuyo           gtetrinet       monsterz         tanglet        xye
```

```
drod-bin       gtkballs       mures          tetrinet      zaz
einstein       gweled         neverball      textmaze
enigma         hangman        ngstar         tint
```

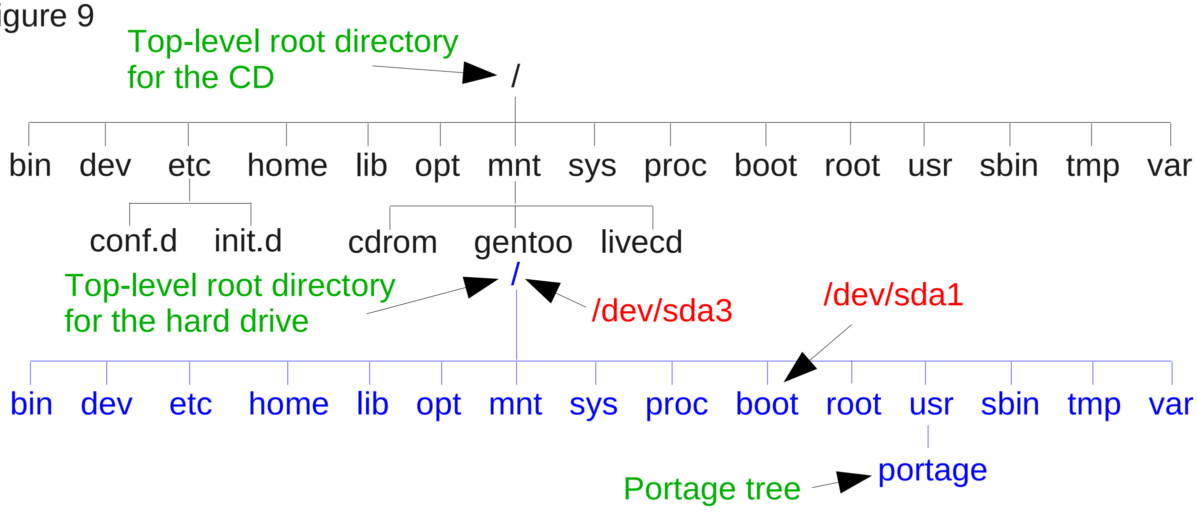
26.9 As you can see, the **games-puzzle category directory** contains quite a number of puzzle game software packages. Each game's directory holds information that tells the **emerge** command how to download the source code for the game, compile it, and install it.

26.10 Most of the **category directories** in the portage tree contain many software package directories, and the whole portage tree currently contains thousands of software packages. After you have finished installing Gentoo Linux on your system, any of the packages in the portage tree can be installed on your system simply by typing **emerge <package name>**.

26.11 Figure 9 shows that the portage tree exists within the **usr** directory that has been placed on the **/dev/sda3 root** partition.

26.12 THIS IS A GOOD STOPPING POINT.

Figure 9



27 Changing the top-level root directory from the CD to the /dev/sda3 root partition

27.1 Before proceeding, let's list the steps of the installation process we have accomplished so far:

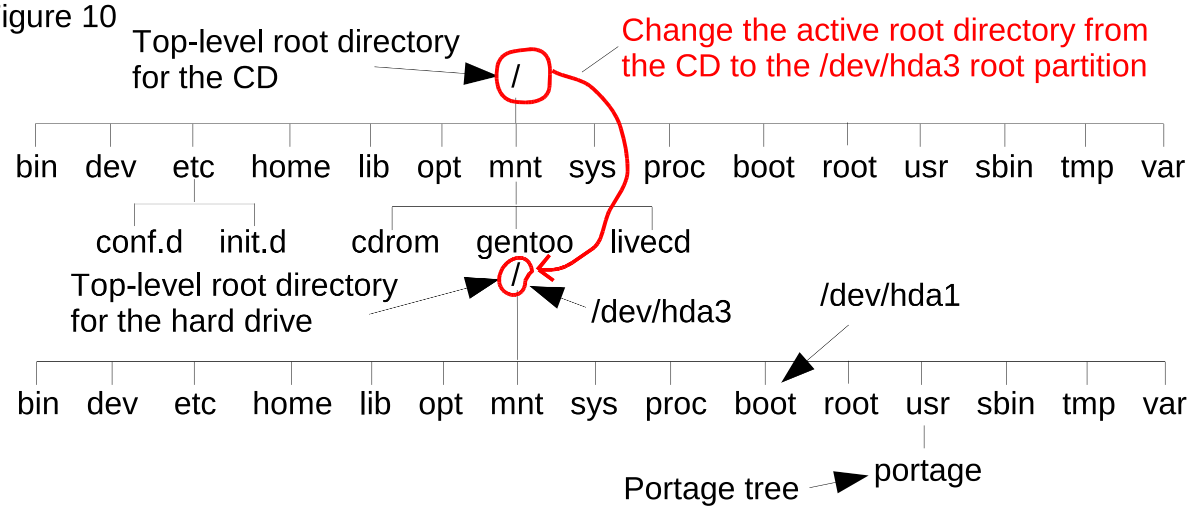
1) Downloaded the Gentoo minimal LiveCD .iso image and burned it onto a CDROM (or installed it into a virtual machine).

2) Booted a PC using this LiveCD image.
3) Partitioned the main hard drive.
4) Mounted the **/dev/sda3** root partition to the **/mnt/gentoo** directory.
5) Mounted the **/dev/sda1** boot partition to the **/mnt/gentoo/boot** directory.
6) Downloaded a compressed **tar** file that contained the core of a standard Gentoo directory structure, and extracted it into the **/mnt/gentoo** directory.
7) Downloaded a compressed **tar** file that contained a **snapshot** of the **portage tree**, and extracted it into the **/mnt/gentoo/usr/portage** directory.

27.2 We have accomplished quite a bit up to this point, and we now have most of the parts of a standard Gentoo directory structure sitting on the **/dev/sda3** root partition (which is mounted to the **/mnt/gentoo** directory). In fact, enough of a standard Gentoo directory structure exists on **/dev/sda3** root partition that we could change into the **/mnt/gentoo** directory, imagine that the CD's directory structure did not exist anymore, and pretend that the **/dev/sda3** root partition was the **new active top-level root directory**.

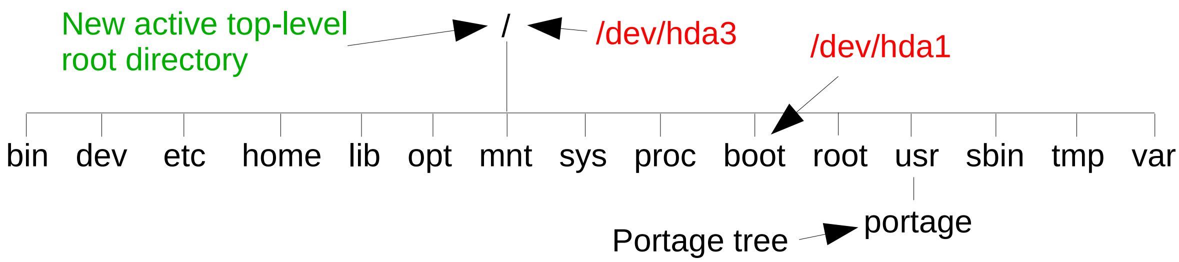
27.3 Actually, this is exactly what we are going to do using the **chroot** (Change Root) command! Figure 10 shows the complete directory hierarchy that we have been working with up to this point. The red arrow indicates that the **active top-level root directory** is about to be changed to the **/dev/sda3** root partition:

Figure 10



27.4 After executing the **chroot** command, the directory hierarchy will look like the one shown in Figure 11:

Figure 11



27.5 The CD's directory hierarchy looks like it has disappeared, and all that is left is the directory hierarchy we have been putting together on the **/dev/sda3** root partition. The CD's directory will still exist after the **chroot** command is executed, it will just be hidden temporarily.

27.6 Before executing the **chroot** command, however, we must make three items that are in the CD's directory hierarchy available within our new directory hierarchy.

27.7 The **first** item is a file called **resolv.conf** and the original copy exists in the **/etc** directory. The **resolv.conf** file contains the network information that was returned by the **DHCP** request that we talked about earlier when the machine was first booted. This network information will be needed in our new directory hierarchy and it can be copied there using the following **cp** command ('**-L**' tells the **cp** command to copy the target of any symbolic links):

```
livecd / # cp -L /etc/resolv.conf /mnt/gentoo/etc/resolv.conf
```

27.8 The **second** item that needs to be made available in the new directory hierarchy is the CD's whole **/dev** directory. Instead of copying everything in the **CD's /dev directory** to the new **/mnt/gentoo/dev directory**, however, we will use the **mount** command simply to **bind** our new **/dev directory** to the **CD's /dev directory**:

```
livecd / # mount -o bind /dev /mnt/gentoo/dev
```

27.9 The **third** and final item that needs to be made available in the new directory hierarchy is the CD's /**proc** directory. The **proc** directory is a special directory because it does not exist as a filesystem on any storage device. What the **proc** directory does is to make information that exists in the currently running kernel available in the form of **files**. Using the terminology of interfaces we discussed earlier, certain information in the kernel is made to implement the **file interface** so that this information can be accessed using the same tools that are used to access all other files. In fact, most of the resources that are contained in a UNIX-like system are made to implement the **file interface** so that they can all be treated in a uniform way. This is one of the aspects of UNIX-like systems that give them their great power.

27.10 Before we make the CD's **/proc** directory available in the new directory hierarchy, let's look inside of it to see what is there. Change into the **/proc** directory and execute the **ls** command:

```
livecd / # cd /proc

livecd proc # pwd
/proc

livecd proc # ls
1       19962  20202  4374       consoles     kpagecount    slabinfo
10      19979  20207  4377       cpuinfo      kpageflags    softirqs
13333   2      20208  442        crypto       loadavg       stat
13334   20118  20215  446        devices      locks         swaps
13335   20119  216    4561       diskstats    mdstat        sys
1400    20120  221    4564       dma          megaraid      sysrq-trigger
14642   20121  3      468        driver       meminfo       sysvipc
165     20122  315    6          execdomains  misc          timer_list
167     20123  316    7          fb           modules       tty
169     20124  326    7506       filesystems  mounts        uptime
18612   20125  330    8          fs           mpt           version
19411   20126  331    9          interrupts   mtrr          vmallocinfo
19420   20127  332    acpi       iomem        net           vmstat
19509   20128  333    asound     ioports      pagetypeinfo  zoneinfo
19510   20149  334    buddyinfo  irq          partitions
19524   20193  335    bus        kallsyms     sched_debug
19816   20197  336    cmdline    key-users    scsi
19946   20201  418    config.gz  kmsg         self
```

27.11 There is a significant amount of information about the currently running kernel present in the **/proc** directory, but we are only going to look at a few items at this time. Let's start by looking inside of the **cpuinfo** file using the **cat** command:

```
livecd proc # cat cpuinfo
processor   : 0
vendor_id   : GenuineIntel
cpu family  : 6
model       : 42
model name  : Intel(R) Core(TM) i7-2760QM CPU @ 2.40GHz
stepping    : 7
microcode   : 0x616
cpu MHz           : 2386.752
cache size  : 6144 KB
```

```
fdiv_bug    : no
hlt_bug           : no
f00f_bug    : no
coma_bug    : no
fpu         : yes
fpu_exception     : yes
cpuid level : 5
wp          : yes
flags       : fpu vme de pse tsc msr mce cx8 apic sep mtrr pge mca cmov pat pse36
clflush mmx fxsr sse sse2 rdtscp constant_tsc up pni monitor ssse3
bogomips    : 4773.50
clflush size      : 64
cache_alignment   : 64
address sizes     : 36 bits physical, 48 bits virtual
power management:
```

27.12 The **cpuinfo** file on **my computer** indicates that the CPU it contains is an **Intel(R) Core(TM) i7-2760QM** running at a frequency of **2386.752 Megahertz**. The other information in this file will become useful when you learn more about CPUs.

27.13 Next, let's look inside the **version** and **uptime** files:

```
livecd proc # cat version
Linux version 3.5.7-gentoo (root@skimmer) (gcc version 4.5.4 (Gentoo 4.5.4 p1.0,
pie-0.4.7) ) #1 SMP Thu Dec 13 04:50:11 UTC 2012
livecd proc # cat uptime
4748.77 4715.05
```

27.14 The **version** file contains information about the currently running kernel, including its version number, the version of the compiler that was used to build it, and the date it was built. The **uptime** file contains the number of seconds that the computer has been running along with how much of that time the CPU was idle.

27.15 Finally, look inside the **partitions** file and the **mounts** file:

```
livecd proc # cat partitions
major minor  #blocks  name

   7        0     110592 loop0
   8        0    8388608 sda
   8        1      32768 sda1
   8        2     524288 sda2
   8        3    7830528 sda3
  11        0     134724 sr0

livecd proc # cat mounts
rootfs / rootfs rw 0 0
proc /proc proc rw,nosuid,nodev,noexec,relatime 0 0
udev /dev devtmpfs rw,nosuid,relatime,size=10240k,nr_inodes=112219,mode=755 0 0
```

```
devpts /dev/pts devpts rw,relatime,gid=5,mode=620 0 0
sysfs /sys sysfs rw,nosuid,nodev,noexec,relatime 0 0
tmpfs / tmpfs rw,relatime 0 0
/dev/sr0 /mnt/cdrom iso9660 ro,relatime 0 0
/dev/loop0 /mnt/livecd squashfs ro,relatime 0 0
tmpfs /run tmpfs rw,nosuid,nodev,relatime,mode=755 0 0
shm /dev/shm tmpfs rw,nosuid,nodev,noexec,relatime 0 0
fusectl /sys/fs/fuse/connections fusectl rw,relatime 0 0
tmpfs /mnt/livecd/lib/firmware tmpfs rw,relatime 0 0
tmpfs /mnt/livecd/usr/portage tmpfs rw,relatime 0 0
/dev/sda3 /mnt/gentoo ext3
rw,relatime,errors=continue,user_xattr,acl,barrier=1,data=writeback 0 0
/dev/sda1 /mnt/gentoo/boot ext2 rw,relatime,errors=continue,user_xattr,acl 0 0
udev /mnt/gentoo/dev devtmpfs
rw,nosuid,relatime,size=10240k,nr_inodes=112219,mode=755 0 0
```

27.16 The **partitions file** contains a list of the **partitions** that the kernel is currently aware of. Notice that the **sda1**, **sda2, sda3** partitions are listed there. The **mounts file** contains a list of all of the currently mounted filesystems along with where in the directory hierarchy they are mounted. The **sda3** and **sda1** partitions are listed as being mounted to the **/mnt/gentoo** and **/mnt/gentoo/boot** directories because this is where we mounted them.

27.17 The **/proc** directory is also in the list and it is now time to make it available inside the new directory hierarchy. Since the **/proc** directory is really just information in the kernel that implements the file interface, we will simply mount this information a second time to the **/mnt/gentoo/proc** directory:

```
livecd proc # mount -t proc none /mnt/gentoo/proc
```

27.18 Now we are finally ready to **change the active root** from the CD's top-level root directory (**/**) to the top-level root directory of the new directory hierarchy (**/mnt/gentoo**):

```
livecd proc # chroot /mnt/gentoo /bin/bash

livecd / # pwd
/

livecd / # ls
bin                         portage-20180306.tar.bz2.md5sum
boot                        proc
dev                         root
etc                         run
home                        sbin
lib                         stage3-amd64-20180311T214502Z.tar.xz
lost+found                  stage3-amd64-20180311T214502Z.tar.xz.DIGESTS
media                       sys
```

```
mnt                       tmp
opt                       usr
portage-20180306.tar.bz2  var
```

27.19 The change of the active top-level root directory has now been accomplished. Notice that the after the **chroot** command was finished, we were placed into the **top-level directory** of the **new directory hierarchy**. In order to make sure that the active root directory was successfully transfered, execute a **cd /** command and see if we are still in the root directory of the new directory hierarchy:

```
livecd / # cd /

livecd / # pwd
/

livecd / # ls
bin                       portage-20180306.tar.bz2.md5sum
boot                      proc
dev                       root
etc                       run
home                      sbin
lib                       stage3-amd64-20180311T214502Z.tar.xz
lost+found                stage3-amd64-20180311T214502Z.tar.xz.DIGESTS
media                     sys
mnt                       tmp
opt                       usr
portage-20180306.tar.bz2  var
```

27.20 The **cd /** command did not change us to the **CD's** top-level root directory so the **chroot** command must have succeeded.

27.21 Before we can use the new directory hierarchy, however, both the **env-update** and the **source /etc/profile** commands need to be executed:

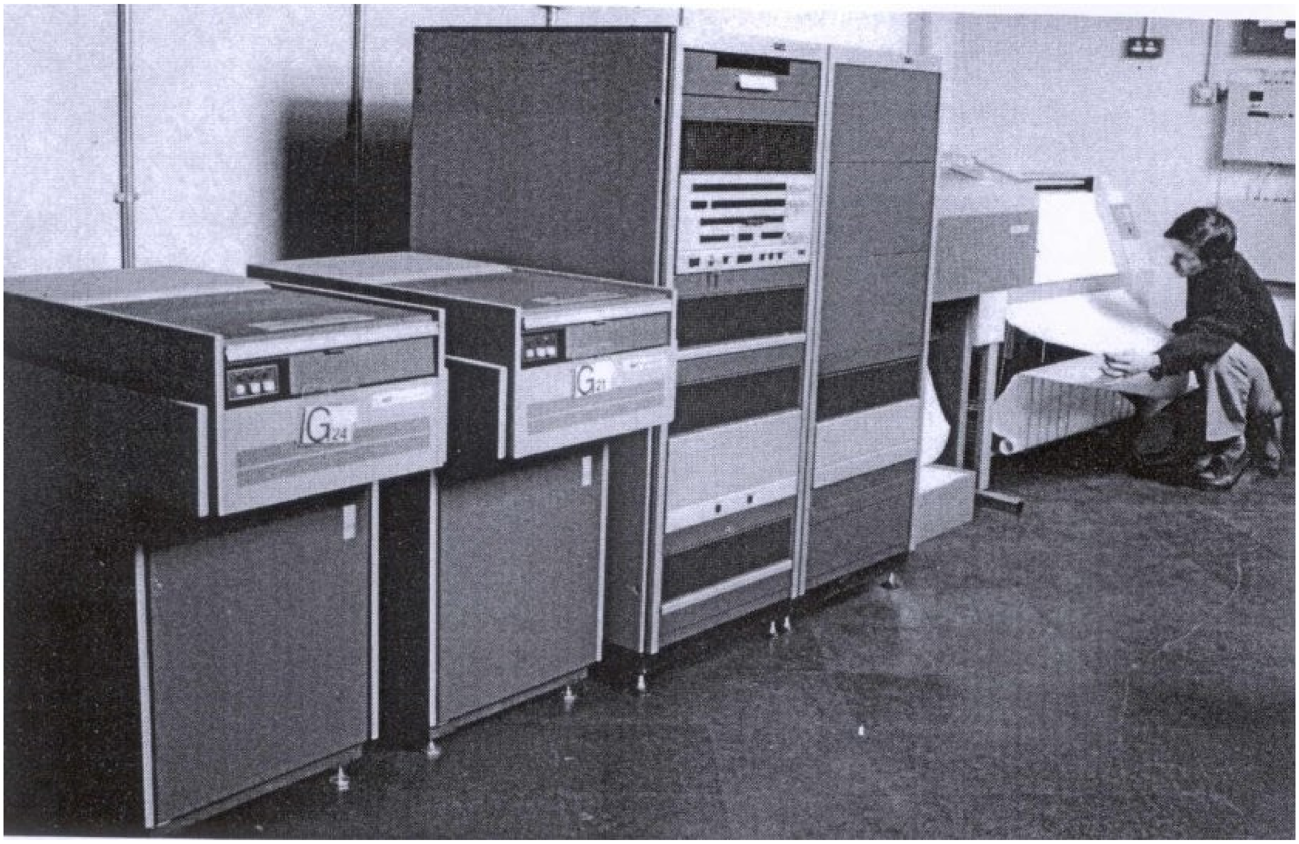
```
livecd / # env-update
>>> Regenerating /etc/ld.so.cache...

livecd / # source /etc/profile
```

27.22 It does not seem that these two commands accomplish anything, but they do and the explanation for what was accomplished is related to what a **shell** is.

28 Terminals and shells: interfaces to the operating system

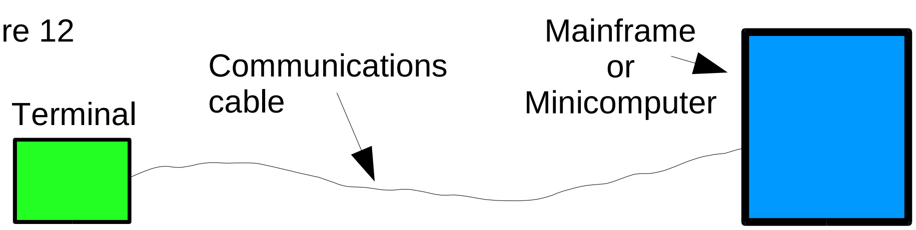
28.1 In the days before personal computers, the most common kinds of computers were **mainframe** computers and **minicomputers**. Many mainframe computers were so large that one or more rooms were required to hold them. The following picture shows a typical mainframe computer:

28.2 Minicomputers were smaller than mainframes but they were still larger than personal computers.

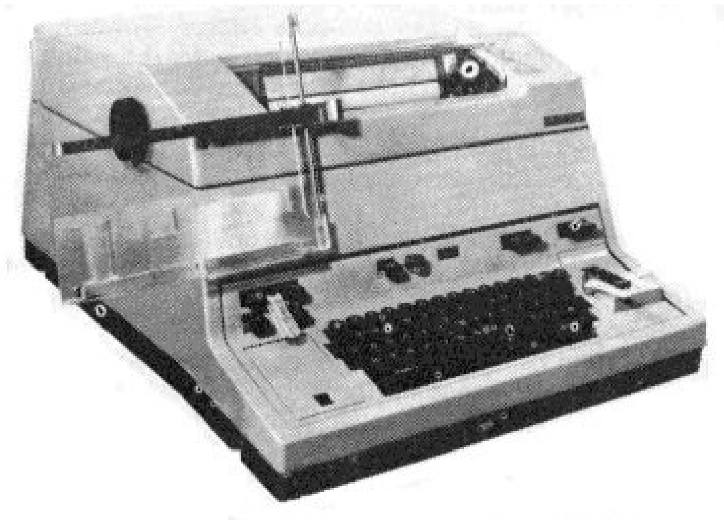


28.3 Both mainframes and minicomputers, however, often used devices called **dumb terminals** to allow humans to interact with them. The word **dumb** meant that the device did not have a computer inside of it, and it relied on the mainframe (or the minicomputer) it was attached to for the execution of CPU instructions. The word **terminal** meant that the devices were attached to the end of a cable which had its other end plugged into the computer (see Figure 12):

Figure 12



28.4 The first kind of terminals were similar in design to typewriters, and they were called **teletypes**. Whatever was typed on the keyboard was displayed on the typewriter paper, and it was also sent electronically to the computer. Output from the computer was also typed on the typewriter paper so that the user could see it:



28.5 Later, terminals were built that used CRTs (Cathode Ray Tubes) instead of typewriter paper for displaying input from the user and output from the computer:

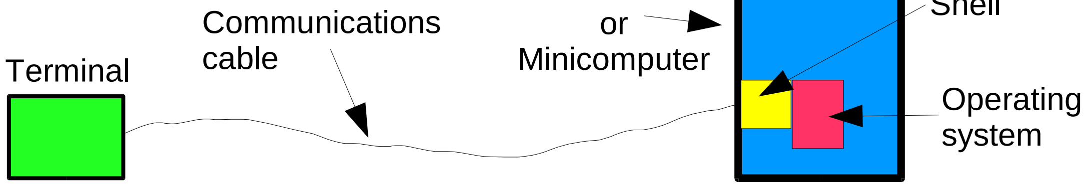
28.6 Inside the computer, a special program was needed that would perform the following two tasks:

1) Accept commands that were typed at the terminal. If these commands were not commands meant for the program itself, they were passed to the operating system.

2) Accept output from the operating system, and send it to the terminal.

28.7 The name which was given to this special type of program was "**shell**" because it can be thought to cover or hide the details of the operating system from the user. **Shells** are also known as **command line interfaces** and we have been using a **shell** to communicate with the operating system since we first booted the computer from the CD (see Figure 13):

Figure 13



28.8 If we have been communicating with a **shell** program during the installation process, what have we been using as a **terminal**? When personal computers started to become available in the late 1970s and early 1980s, people began interfacing them to mainframes and minicomputers just like dumb terminals had been. Since PCs were computers themselves, programs could be run on them that **emulated** all the functions of a **dumb terminal** and these program were called **terminal emulators**. The shell programs on the mainframes or minicomputers that the terminal emulators were communicating with could not determine if they were exchanging information with actual dumb terminals or terminal emulator programs running on PCs.

28.9 When UNIX-like operating systems began to be run on PCs during the late 1980s and early 1990s, an interesting thing happened. The **terminal emulator** programs that had previously been used to communicate with shell programs on external mainframes and minicomputers through cables were now used to communicate with shell programs that were running on the PC itself! By the way, emulated terminals are also called **virtual terminals**. This technique of using a terminal emulator to communicate with a UNIX-like operating system running on the same PC is still in use today, and we have been using a terminal emulator to communicate with the Gentoo Linux operating system that was booted from the CD.

28.10 In fact, not just one terminal emulator program was run when you booted your system from the CD, but **six** of them were! The way that you can switch between the six terminal emulator programs is by holding down the <alt> key on your keyboard and pressing either the <F1>, <F2>, <F3>, <F4>, <F5> or <F6> keys. The default terminal emulator is accessed by pressing <alt><F1>, and it is the one we have been using since the beginning of the installation process. Try switching to the second terminal emulator by pressing <alt><F2> and then execute a **pwd** command to see where it is in the directory hierarchy. Move around the directory hierarchy using the **cd** command, and use the **ls** command to see the contents of these directories. You can switch to terminal emulators 3 - 6 and experiment with them too if you would like.

28.11 Each terminal emulator can be used to view the same directory hierarchy. Terminal emulators 2 - 6 have not had the **chroot** command executed in them, and therefore they each have the **top-level root directory** of the **CD** as their **active root directory**. When you are done experimenting, press the <alt><F1> keys in order to switch back to the **chrooted** environment in the default terminal emulator.

29 Customizing the shell that is being used by the default terminal emulator

29.1 Now that we have used the **chroot** command to change the active root of the default terminal to the **/dev/sda3** root partition, it would be nice to have the command prompt indicate this. The way this is done is by changing a **variable** in the shell program that the terminal is communicating with. A **variable** is a name that has been associated with a memory location (or a set of memory locations) so that humans do not need to refer to it by its address. A shell program has a number of variables (called **environment variables)** that hold configuration data for the shell. The environment variables of the current shell can be viewed using the **set** command:

```
livecd / # set
BASH=/bin/bash
BASHOPTS=checkwinsize:cmdhist:expand_aliases:extquote:force_fignore:histappend:host
complete:interactive_comments:progcomp:promptvars:sourcepath
BASH_ALIASES=()
BASH_ARGC=()
BASH_ARGV=()
BASH_CMDS=()
BASH_LINENO=()
BASH_SOURCE=()
BASH_VERSINFO=([0]="4" [1]="2" [2]="45" [3]="1" [4]="release" [5]="i686-pc-linux-
gnu")
BASH_VERSION='4.2.45(1)-release'
COLUMNS=88
CONFIG_PROTECT=/usr/share/gnupg/qualified.txt
CONFIG_PROTECT_MASK='/etc/gentoo-release /etc/sandbox.d /etc/terminfo /etc/ca-
certificates.conf'
DIRSTACK=()
EDITOR=/bin/nano
```

```
EUID=0
GCC_SPECS=
GROUPS=()
HISTFILE=/root/.bash_history
HISTFILESIZE=500
HISTSIZE=500
HOME=/root
HOSTNAME=livecd
HOSTTYPE=i686
IFS=$' \t\n'
INFOPATH=/usr/share/info:/usr/share/gcc-data/i686-pc-linux-gnu/4.7.3/info:/usr/
share/binutils-data/i686-pc-linux-gnu/2.23.2/info
LESS='-R -M --shift 5'
LESSOPEN='|lesspipe %s'
LINES=35
LOGNAME=root
MACHTYPE=i686-pc-linux-gnu
MAIL=/var/mail/tkosan
MAILCHECK=60
MANPATH=/usr/local/share/man:/usr/share/man:/usr/share/gcc-data/i686-pc-linux-gnu/
4.7.3/man:/usr/share/binutils-data/i686-pc-linux-gnu/2.23.2/man
MULTIOSDIRS=../lib
OLDPWD=/
OPTERR=1
OPTIND=1
OSTYPE=linux-gnu
PAGER=/usr/bin/less
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/opt/bin:/usr/
i686-pc-linux-gnu/gcc-bin/4.7.3
PIPESTATUS=([0]="0")
PPID=20193
PROMPT_COMMAND='echo -ne "\033]0;${USER}@${HOSTNAME%%.*}:${PWD/#$HOME/~}\007"'
PS1='\[\033[01;31m\]\h\[\033[01;34m\] \W \$\[\033[00m\] '
PS2='> '
PS4='+ '
PWD=/
PYTHONPATH=/usr/lib/portage/pym
SHELL=/bin/bash
SHELLOPTS=braceexpand:emacs:hashall:histexpand:history:interactive-comments:monitor
SHLVL=3
SSH_CLIENT='206.21.94.232 40790 1237'
SSH_CONNECTION='206.21.94.232 40790 206.21.94.200 1237'
SSH_TTY=/dev/pts/0
SYSTEMD_LESS='FRSM --shift 5'
TERM=xterm
UID=0
USER=root
_=/etc/profile
```

29.2 As you can see, shells can contain a large number of environment variables! We are not going to discuss what all of these variables do at this time, but we will talk about some of them in a moment. First, however, notice that each environment variable is listed in the form of

VARIABLE_NAME = <VALUE>. The name of the environment variable is on the left side of the '=' sign and the data that the variable is associated with is on the right side of the sign. Here is more information about the environment variables that have been highlighted in the above list:

- **SHELL** - The name of the current shell program is called '**bash**' (which stands for **Bourne again shell)** and it is located in the **/bin** directory.
- **TERM -** The terminal emulation program that is currently being used is called **xterm**.
- **PATH** - When the name of a command is typed at the command line, the shell looks inside each of the directory paths listed in the PATH variable to locate the program that implements the command. The first path in the list is searched first, then the next path in the list is searched and so on (paths are separated by a colon ':'). If the program is found it is executed. If it is not found, the shell outputs a "command not found" error message.
- **LINES** - The number of lines or rows of information that the current terminal is configured to display.
- **PWD** - Holds the path of the current working directory.
- **PS1** - Determines what is displayed in the command prompt.

29.3 If we want to change our command prompt so that it indicates we are currently in a **chroot** environment, then the **PS1** shell environment variable needs to have information added to it. An explanation of the strange pattern of symbols that the **PS1** variable currently contains is beyond what I want to discuss at this point. Fortunately, you will not need to understand them in order to add the information needed to indicate we are now in a **chroot** environment.

29.4 The **export** command is used to change the shell's environment variables. Execute the following **export** command, and notice what happens to the command prompt (note the dollar sign '$' in front of the second PS1):

```
livecd / # export PS1="(chroot) $PS1"
(chroot) livecd / #
```

29.5 We just told the **export** command to set the **PS1 environment variable** to the characters **(chroot)** followed by the **current contents of the PS1 variable** (whenever a $ is placed in front of an environment variable name, this means to use the information that the variable is holding, not the name of the variable). If you execute another **set** command, you will find that the **PS1** variable now holds the following information:

```
PS1='(chroot) \[\033[01;31m\]\h\[\033[01;34m\] \W \$\[\033[00m\] '
```

29.6 Notice that the characters **(chroot)** have indeed been placed before the characters that were originally in the variable. Now **(chroot)** will be present on the command line's prompt until we either change the **PS1** variable again or **exit the chroot environment**.

29.7 Now that you know what shell environment variables are, we can explain what the **env-update** and **source /etc/profile** commands did that we executed earlier. Each time that a shell program is launched, it has its **environment variables** set by running the **source /etc/profile** program. The **env-update** program maintains the information that the **source /etc/profile** command uses to set the environment variables. When we executed the **chroot** command, our environment changed. We updated the environment information with the **env-update** command. We then updated the **environment variables** in our currently running shell by executing the **source /etc/profile** command.

30 The nano text editor

30.1 A significant part of installing and configuring GNU/Linux consists of editing **configuration files**. These files are usually in **text format** which means that they only hold **plain typed characters** without any additional formatting information. In contrast to this, a **word processor** file not only holds typed characters, it contains extra information about these typed characters (such as bold, indenting, font, font size, etc). The various GNU/Linux programs that read the information that is present in configuration files would not know what to do with any extra formatting information that may be present. This is why all configuration files need to be created and edited by a **text editor** and not a word processor.

30.2 The most commonly used text editor on Gentoo systems is called **nano**. Let's experiment with **nano** before we begin editing configuration files with it. Change into the **/tmp** directory in the **chrooted** environment and execute the following commands:

```
(chroot) livecd / # cd /tmp

(chroot) livecd tmp # pwd
/tmp

(chroot) livecd tmp # ls
```

30.3 Since no files or directories were listed when we executed the **ls** command, this means that the **/tmp** directory is currently empty.

30.4 Now that we are working inside the directory hierarchy which is on the hard drive, there are commands available to us that were not present on the CD. One of these commands is called **man** and it stands for **manual**. Most of the programs on a GNU/Linux system have manual pages written for them that give information about what the program does and what options can be passed to it. For example, if you want to read about what the **cat** command does, simply type **man cat** at the command prompt and use the **up and down arrow keys** to move through the document (**press the 'q' key to quit)**:

```
(chroot) livecd /tmp # man cat
Formatting page, please wait...
```

30.5 You can look at the **man pages** for any of the commands we have used up to this point (such as **ls**, **cd** and **hexdump**) and you can also read the **man page** for **nano**.

30.6 Execute **nano** with the following options:

```
(chroot) livecd tmp # nano -wc test.txt
```

30.7 The '**w**' option tells nano to allow lines that are over 80 characters wide and the '**c**' option shows the line number and column of where the cursor is currently at. The '**test.txt**' parameter tells nano to create a file named **test.txt** in the current directory when the file is saved.

30.8 When nano is executed, it shows a screen that should look similar to the one shown below. Type the sentences "**This is a test text file.**" and "**The only thing that this file contains is plain text characters.**" (which are shown in blue) into the editor window:

```
  GNU nano 1.3.11              File: test.txt                               Modified

This is a test text file.

The only thing that this file contains is plain text characters.







             [ line 3/4 (75%), col 65/65 (100%), char 91/92 (98%) ]
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Txt ^T To Spell
```

30.9 The way that commands are given to nano while it is running is by pressing the <ctrl> key and a letter. Commands are shown at the bottom of nano's window, and the <ctrl> key is indicated by the '**^**' character. After you have typed the two sentences, press the <ctrl> **O** keys to save the file. Nano will then have you confirm that you want to save the file with the name "**test.txt**" at the bottom of the screen:

```
File Name to Write: test.txt
^G Get Help         ^T To Files         M-M Mac Format      M-P Prepend
^C Cancel           M-D DOS Format      M-A Append          M-B Backup File
```

30.10 Press the <enter> key to confirm the file name, and then press <ctrl> **X** in order to exit nano and return to the shell's command line.

30.11 Execute an **ls -l** command to make sure that the file was indeed created by nano:

```
(chroot) livecd tmp # ls -l
total 4
-rw-r--r-- 1 root root 92 Feb  4 08:36 test.txt
```

30.12 The listing indicates that a file named **test.txt** is now present in the **/tmp** directory and that it is **92** bytes long. In order to verify that the sentences you typed in nano are now contained in the file, you can execute a **cat test.txt** command:

```
(chroot) livecd tmp # cat test.txt
This is a test text file.

The only thing that this file contains is plain text characters.
```

30.13 For a more detailed view of the characters in this file, execute a **hexdump -C test.txt** command (Note: the '**C**' is capitalized):

```
(chroot) livecd tmp # hexdump -C test.txt
00000000  54 68 69 73 20 69 73 20  61 20 74 65 73 74 20 74  |This is a test t|
00000010  65 78 74 20 66 69 6c 65  2e 0a 0a 54 68 65 20 6f  |ext file...The o|
00000020  6e 6c 79 20 74 68 69 6e  67 20 74 68 61 74 20 74  |nly thing that t|
00000030  68 69 73 20 66 69 6c 65  20 63 6f 6e 74 61 69 6e  |his file contain|
00000040  73 20 69 73 20 70 6c 61  69 6e 20 74 65 78 74 20  |s is plain text |
00000050  63 68 61 72 61 63 74 65  72 73 2e 0a              |characters..|
0000005c
```

30.14 The **-C** option tells the **hexdump** command to print 16 characters per line. The **hexadecimal** number that is associated with each character shown in the middle column, and the **ASCII characters** themselves are shown in the right column. **ASCII** stands for **American Code for Information Interchange**. It is a widely used standard that associates the numbers 0-127 decimal (or 0-7F hex) with the characters shown in Table 1.

ASCII (American Standard Code for Information Interchange) Chart

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
10	0a	Linefeed/Newline	63	3F	?	96	60	`
13	0d	Carriage Return	64	40	@	97	61	a
32	20	Space	65	41	A	98	62	b
33	21	!	66	42	B	99	63	c
34	22	"	67	43	C	100	64	d
35	23	#	68	44	D	101	65	e
36	24	$	69	45	E	102	66	f
37	25	%	70	46	F	103	67	g
38	26	&	71	47	G	104	68	h
39	27	'	72	48	H	105	69	i
40	28	(	73	49	I	106	6A	j
41	29	)	74	4A	J	107	6B	k
42	2A	*	75	4B	K	108	6C	l
43	2B	+	76	4C	L	109	6D	m
44	2C	,	77	4D	M	110	6E	n
45	2D	-	78	4E	N	111	6F	o
46	2E	.	79	4F	O	112	70	p
47	2F	/	80	50	P	113	71	q
48	30	0	81	51	Q	114	72	r
49	31	1	82	52	R	115	73	s
50	32	2	83	53	S	116	74	t
51	33	3	84	54	T	117	75	u
52	34	4	85	55	U	118	76	v
53	35	5	86	56	V	119	77	w
54	36	6	87	57	W	120	78	x
55	37	7	88	58	X	121	79	y
56	38	8	89	59	Y	122	7A	z
57	39	9	90	5A	Z	123	7B	{
58	3A	:	91	5B	[	124	7C	|
59	3B	;	92	5C	\	125	7D	}
60	3C	<	93	5D	]	126	7E	~
61	3D	=	94	5E	^			
62	3E	>	95	5F	_			

Table 1

30.15 In the above **HexDump** output, the first word of the first sentence that was typed into the **test.txt** file is "**This**". Notice that the hex number that is associated with the letter '**T**' is **54**. The letter '**h**' is associated with **68**, the letter '**i**' is associated with **69** and the letter '**s**' is associated with **73**. The spaces between the words are represented by the number **20**, **newlines** are represented by **0a**, and periods are represented by **2e**.

30.16 Now that you know what a text file is and how to use the nano editor, we will use nano in the next section to edit the configuration file that holds the main compile options for a Gentoo Linux system.

31 USE flags

31.1 Earlier I indicated that "**a GNU/Linux distribution is usually put together by a group of experienced developers who copy the software needed to create a GNU/Linux distribution into one place, compile and configure it, and then make the result available to others**". Gentoo is unique, however, because instead of its experienced developers devoting their time to creating a distribution, they created a **software system** (called **portage**) that generates a **customized** GNU/Linux distribution **automatically** on the user's machine.

31.2 As we discussed previously, each package in the **portage tree** contains instructions that tell the **emerge** program how to obtain the package's source code, compile it, configure it, and install the files that are generated into the proper places in the directory hierarchy. The way that portage enables users to customize how packages are built and configured is through a mechanism called **USE flags**. A common definition for a **flag** is "**a piece of cloth used as a signaling device**." **USE flags** are also a kind of signaling device except they use **keywords** instead of pieces of cloth, and they **allow users to communicate their package building and configuration choices to portage**.

31.3 Portage uses two kinds of USE flags, which are **global USE flags** and **local USE flags**. **Global USE flags** are flags that are used by multiple packages, and **local USE flags** are only used by a single package.

31.4 Examples of USE flag keywords include **gtk, gnome**, **dvd** and **cdr.** A list of the **global** USE flags and their descriptions can be found in the **/usr/portage/profiles/use.desc** file. A list of the **local** USE flags can be found in the **/usr/portage/profiles/use.local.desc** file. Instead of using the **more** command to look at the contents of these files, however, you can use an improved version of **more** called **less**. One of the improvements that **less** has is the ability to scroll up and down through a file using the **up and down arrow keys**. Change into the **/usr/portage/profiles** directory and use the **less** command to look at the **use.desc** file (press the '**q**' key to exit the less command):

```
(chroot) livecd / # cd /usr/portage/profiles/

(chroot) livecd profiles # pwd
/usr/portage/profiles

(chroot) livecd profiles # ls
arch             ChangeLog-2010  desc        license_groups  thirdpartymirrors
arch.list        ChangeLog-2011  eapi        package.mask    uclibc
base             ChangeLog-2012  embedded    prefix          updates
```

```
categories      ChangeLog-2013  features   profiles.desc   use.desc
ChangeLog-2007  ChangeLog-2014  hardened   releases        use.local.desc
ChangeLog-2008  ChangeLog-2015  info_pkgs  repo_name
ChangeLog-2009  default         info_vars  targets
```

```
(chroot) livecd profiles # less use.desc
```

31.5 You can also look through the **use.local.desc** file if you would like. At this point you do not need to fully understand what all of these USE flags actually do, but you should at least understand that they are used to customize your Gentoo installation. (Note: type the 'q' key to exit the 'less' command.)

32 The /etc/portage/make.conf file

32.1 Now that you know what USE flags are, we next need to cover where they are placed so that portage can find them. Inside the **/etc** directory is a file called **make.conf**, and it is the main place that portage looks for **USE flags**. Portage also looks in other places for USE flags, but we are not going to discuss these other places at this time.

32.2 Let's look at what the **make.conf** file currently contains, and then we will add a small number of **USE flags** to this file for portage to use. Change into the **/etc** directory, and execute a **cat make.conf** command:

```
(chroot) livecd etc # cd /etc/portage

(chroot) livecd etc/portage # pwd
/etc/portage

(chroot) livecd etc/portage # cat make.conf
# These settings were set by the catalyst build script that automatically
# built this stage.
# Please consult /usr/share/portage/config/make.conf.example for a more
# detailed example.
CFLAGS="-O2 -pipe"

# NOTE: This stage was built with the bindist Use flag enabled
PORTDIR="/usr/portage"
DISTDIR="/usr/portage/distfiles"
PKGDIR="/usr/portage/packages"

# This sets the language of build output to English.
# Please keep this setting intact when reporting bugs.
LC_MESSAGES=C
```

32.3 CFLAGS is a variable which contains configuration information that portage will use to build packages. The CFLAGS variable allows portage to configure the compiler that will be used to compile the packages on your system. For more information on the make.conf file, look at the **man page** for the **make.conf** file.

32.4 Our next step is to **add USE flags to a variable called USE that is in the make.conf file**. This variable contains the **USE flag** information we want to pass to **portage**. **Edit the make.conf file with nano** , add the flags that are in the line highlighted in green below to the USE variable, and then save the file (make sure you remember to type the quotes at the beginning and end of the list):

```
# These settings were set by the catalyst build script that automatically
# built this stage.
# Please consult /usr/share/portage/config/make.conf.example for a more
# detailed example.
CFLAGS="-O2 -pipe"

# NOTE: This stage was built with the bindist Use flag enabled
USE="bindist mmx sse sse2 gtk gnome -kde X dvd alsa cdr"
PORTDIR="/usr/portage"
DISTDIR="/usr/portage/distfiles"
PKGDIR="/usr/portage/packages"

# This sets the language of build output to English.
# Please keep this setting intact when reporting bugs.
LC_MESSAGES=C
```

32.5 The **gtk** and **gnome** flags will be explained when we add GUI and desktop software to our system. Any flag with a negative sign in front of it means that support for the capabilities that flag indicates should not be added to packages. In this case, we do not want support for the **kde** desktop. Descriptions for the rest of the flags in this list can be found in the **/usr/portage/profiles/use.desc** file.

33 Emerging the kernel's source code

33.1 We are now about 2/3 of the way through the base installation process and quickly approaching its climax (which is the configuration, compilation, and installation of the kernel). Before we proceed, however, I would like to take a few moments to reflect on the material we have covered up to this point.

33.2 If you have made it this far, then you have seen for yourself that there is a significant amount of detailed information that is associated with manually installing a UNIX-like operating system. You are not going to fully understand all of this information by just going through the installation process one time. I was not truly comfortable with this material myself until after I had installed Gentoo at least 5 times, and I bet it will take you this many times to be comfortable with it too.

33.3 Therefore, if you are concerned that you are not completely understanding all of the information we have covered so far, my advice to you is to not worry too much about this. Continue to work hard, understand as much as you can, and then rest easy knowing that much of this material will sink in only after you have gone through the installation process a number of times.

33.4 Now that you have had a small pep talk, let's proceed with configuring and installing the Linux kernel. Properly configuring and installing the Linux kernel is not easy, but it is not that difficult either. The good news is that after the kernel has been installed, we will be nearing the end of the installation process.

33.5 Before the kernel can be installed, however, the computer system needs to be told which **timezone** it is in. Change into the **/usr/share/zoneinfo** directory, list its contents, and locate your timezone:

```
(chroot) livecd / # cd /usr/share/zoneinfo

(chroot) livecd zoneinfo # pwd
/usr/share/zoneinfo

(chroot) livecd zoneinfo # ls
Africa      Chile    Factory    Iceland      MET       posixrules  UTC
America     CST6CDT  GB         Indian       Mexico    PRC         WET
Antarctica  Cuba     GB-Eire    Iran         MST       PST8PDT     W-SU
Arctic      EET      GMT        iso3166.tab  MST7MDT   ROC         zone1970.tab
Asia        Egypt    GMT0       Israel       Navajo    ROK         zone.tab
Atlantic    Eire     GMT-0      Jamaica      NZ        Singapore   Zulu
Australia   EST      GMT+0      Japan        NZ-CHAT   Turkey
Brazil      EST5EDT  Greenwich  Kwajalein    Pacific   UCT
Canada      Etc      Hongkong   Libya        Poland    Universal
CET         Europe   HST        localtime    Portugal  US
```

33.6 I am located in the eastern united states so my timezone is **EST** (Eastern Standard Time). In order to set the timezone for your machine, all you need to do is to copy the correct **timezone file** from the **/usr/share/zoneinfo** directory into the **/etc** directory and then give it the name "**localzone**". This can be done in one step with the **cp** (copy) command:

```
(chroot) livecd zoneinfo # cp EST /etc/localtime
```

33.7 The first parameter that is passed to the **cp** command is the name and location of the **source file**, and the second parameter is the name and location of the **destination file**.

33.8 We can now use the **emerge** command to download the package that contains the kernel's source code to our machines (**Note: your PC needs to be authenticated with the network before executing the “emerge” command. The reason for this is that the “emerge” command accesses servers that are not on the Atlas network.**):

```
(chroot) livecd zoneinfo # cd /
```

```
(chroot) livecd / # pwd
/
```

```
(chroot) livecd / # USE="-doc" emerge sys-kernel/gentoo-sources
```

33.9 As soon as you execute this **emerge** command, the **gentoo-sources** information in the **portage tree** (which exists in the **/usr/portage/sys-kernel/gentoo-sources** directory) will be found and the instructions it contains for downloading the files required to configure and build the kernel will be followed. (**Note: emerging gentoo-sources will take awhile.**) I had indicated earlier that there was more than one way to specify **USE flags**, and the above command uses one of them. In this case, we do not want the **emerge** command to download any extra documentation files when it downloads the source code for the kernel.

33.10 By default, **emerge** uses the **wget** command (which we used earlier) to download files from the Internet to the local machine, and this is what is happening when you see **wget's progress bar**:

```
100%[=====================================>] 153
```

33.11 The **emerge** command displays information about what it is doing step-by-step. The full listing of the **emerge** command we just executed is too long to include in this document.

33.12 Lines that look like the following indicate that files are being copied into the specified places in the directory hierarchy:

```
        >>> /usr/src/linux-2.6.19-gentoo-r5/include/keys/user-type.h
```

33.13 If you watched the **emerge** listing as it scrolled by the screen, you will have noticed that most of the files that were being copied into the directory hierarchy were being placed in the **/usr/src** directory. In the next section we will change into this directory in order to configure the kernel. Before we do, however, let's finish talking about what happened during the emerge process.

33.14 The bottom part of the **emerge listing** usually contains messages from the Gentoo developers who are responsible for maintaining that specific package. You should always read these messages in case they contain important information.

33.15 The final thing we will do before configuring the kernel is to see where the **wget** command downloaded the files to that contain the kernel's source code. Change into the **/usr/portage/distfiles** directory and list its contents:

```
(chroot) livecd / # cd /usr/portage/distfiles

(chroot) livecd distfiles # pwd
/usr/portage/distfiles

(chroot) livecd distfiles # ls
bc-1.06.95.tar.bz2              genpatches-4.1-20.extras.tar.xz
genpatches-4.1-20.base.tar.xz   linux-4.1.tar.xz
```

33.16 Emerge downloaded four compressed files that contain the information needed to configure and build version 3.4 of the Linux kernel. The main file is called **linux-3.4.tar.xz** and the two **genpatches** files contain what are called **patches**. A **patch** contains information that is used to modify existing files. In this case, the Gentoo developers are taking the official 3.4 version of the Linux kernel's source code and using **patches** to make adjustments to it. Sometimes this is done to fix bugs in the original source code while other times it is done to make improvements to it.

33.17 By default, emerge downloads the **compressed files** for all packages into the **/usr/portage/distfiles** directory. If a given file already exists in this directory, then **emerge** does not bother download it again.

34 Configuring the kernel

34.1 It is now time to configure the Linux kernel. As we discussed earlier, a kernel accesses a computer's hardware through special programs called **devices drivers** (see Figures 1 and 2). A significant amount of the work required to configure a Linux kernel involves determining the **types**, **manufacturers** and **model numbers** of the electronic chips on the motherboard (and expansion cards), and matching these with the kernel modules needed to access them.

34.2 CPUs communicate with the chips that are on the motherboard and expansion cards using a **communications bus**. A **communications bus** usually consists of a set of wires that are run from one point to another (or to multiple points) in a circuit. The most common bus that is used today in most PCs and servers is called the **PCI** (Peripheral Component Interconnect) bus.

34.3 The first step that needs to be done when configuring a kernel is to obtain a list of all the **electronic chips** that are connected to the **PCI bus** of the computer you are installing Gentoo on. There is a program named **lspci** (list PCI) that is capable of doing this, and it is included on the CD. The **lspci** program, however, has not yet been installed into the **chrooted environment** on the hard drive. This means that if you switch to any of the virtual terminals other than the default one (using <alt><F2-F6>) you can run **lspci**, but if you want to run **lspci** in the **chrooted environment**, you must first **emerge** it. The **lspci** program is contained within the **sys-apps/pciutils** package, and it can be emerged by executing an **emerge pciutils** command:

```
(chroot) livecd / # emerge pciutils
```

34.4 After the **pciutils** package is done emerging, execute the **lspci** command and see which electronic chips are listed for your computer. :

```
(chroot) livecd / # lspci
```

```
00:00.0 Host bridge: Intel Corporation 440FX - 82441FX PMC [Natoma] (rev 02)
00:01.0 ISA bridge: Intel Corporation 82371SB PIIX3 ISA [Natoma/Triton II]
00:01.1 IDE interface: Intel Corporation 82371AB/EB/MB PIIX4 IDE (rev 01)
00:02.0 VGA compatible controller: InnoTek Systemberatung GmbH VirtualBox Graphics
Adapter
00:03.0 Ethernet controller: Intel Corporation 82540EM Gigabit Ethernet Controller
(rev 02)
00:04.0 System peripheral: InnoTek Systemberatung GmbH VirtualBox Guest Service
00:05.0 Multimedia audio controller: Intel Corporation 82801AA AC'97 Audio
Controller (rev 01)
00:06.0 USB controller: Apple Inc. KeyLargo/Intrepid USB
00:07.0 Bridge: Intel Corporation 82371AB/EB/MB PIIX4 ACPI (rev 08)
00:0b.0 USB controller: Intel Corporation 82801FB/FBM/FR/FW/FRW (ICH6 Family) USB2
EHCI Controller
00:0d.0 SATA controller: Intel Corporation 82801HM/HEM (ICH8M/ICH8M-E) SATA
Controller [AHCI mode] (rev 02)
```

34.5 The above chips are the ones that are listed for the VirtualBox virtual PC that I am installing Gentoo on in order to provide the examples for this document. In this list, I have highlighted the function of each chip in **blue**, its manufacturer in **green,** and its model number in **red**. Only the chips that are needed to successfully boot the machine have been highlighted at this time.

34.6 As we discussed earlier, when we **emerged** the **gentoo-sources** package, most of the source code for the Linux kernel was placed into the

/usr/src directory. Change into the **/usr/src** directory now and let's see what it contains by executing a **ls -l** command:

```
(chroot) livecd / # cd /usr/src

(chroot) livecd src # pwd
/usr/src

(chroot) livecd src # ls -l
total 4
lrwxrwxrwx  1 root root   22 Mar 21 00:02 linux -> linux-4.1.15-gentoo-r1
drwxr-xr-x 24 root root 4096 Mar 21 00:02 linux-4.1.15-gentoo-r1
```

34.7 The **/usr/src** directory contains one directory called **linux-3.4.83-gentoo** and a **symbolic link** to this directory called **linux**. The source code for the Gentoo version 3.4.83 Linux kernel is in the **linux-3.4.83-gentoo** directory. The purpose of the symbolic link is to allow multiple versions of the Linux kernel to exist in this directory and the active kernel will be pointed to by the symbolic link. Since we currently only have one version of the kernel's source code installed on the machine, it is the active kernel by default and the symbolic link points to it.

34.8 Use the symbolic link to change into the directory where the Linux source code is held:

```
(chroot) livecd src # cd linux

(chroot) livecd linux # pwd
/usr/src/linux

(chroot) livecd linux # ls -l
total 30900
-rw-r--r--   1 root root    18693 May 20  2012 COPYING
-rw-r--r--   1 root root    94984 May 20  2012 CREDITS
drwxr-xr-x  96 root root    12288 Mar 25 01:51 Documentation
-rw-r--r--   1 root root     2536 May 20  2012 Kbuild
-rw-r--r--   1 root root      277 Mar 25 01:49 Kconfig
-rw-r--r--   1 root root   210362 Mar 25 01:49 MAINTAINERS
-rw-r--r--   1 root root    53560 Mar 25 01:49 Makefile
total 628
drwxr-xr-x  32 root root   4096 Mar 21 00:01 arch
drwxr-xr-x   3 root root   4096 Mar 21 00:01 block
-rw-r--r--   1 root root  18693 Jun 22  2015 COPYING
-rw-r--r--   1 root root  96960 Jun 22  2015 CREDITS
drwxr-xr-x   4 root root   4096 Mar 21 00:01 crypto
drwxr-xr-x   2 root root   4096 Mar 21 00:01 distro
drwxr-xr-x 108 root root  12288 Mar 21 00:01 Documentation
drwxr-xr-x 122 root root   4096 Mar 21 00:02 drivers
drwxr-xr-x  36 root root   4096 Mar 21 00:01 firmware
drwxr-xr-x  76 root root   4096 Mar 21 00:01 fs
```

```
drwxr-xr-x  28 root root   4096 Mar 21 00:01 include
drwxr-xr-x   2 root root   4096 Mar 21 00:01 init
drwxr-xr-x   2 root root   4096 Mar 21 00:01 ipc
-rw-r--r--   1 root root   2163 Jun 22  2015 Kbuild
-rw-r--r--   1 root root    277 Mar 21 00:00 Kconfig
drwxr-xr-x  16 root root   4096 Mar 21 00:01 kernel
drwxr-xr-x  11 root root  12288 Mar 21 00:01 lib
-rw-r--r--   1 root root 310415 Jun 22  2015 MAINTAINERS
-rw-r--r--   1 root root  54442 Mar 21 00:00 Makefile
drwxr-xr-x   3 root root   4096 Mar 21 00:01 mm
drwxr-xr-x  59 root root   4096 Mar 21 00:01 net
-rw-r--r--   1 root root  18593 Jun 22  2015 README
-rw-r--r--   1 root root   7485 Jun 22  2015 REPORTING-BUGS
drwxr-xr-x  15 root root   4096 Mar 21 00:01 samples
drwxr-xr-x  14 root root   4096 Mar 21 00:01 scripts
drwxr-xr-x   9 root root   4096 Mar 21 00:01 security
drwxr-xr-x  23 root root   4096 Mar 21 00:01 sound
drwxr-xr-x  21 root root   4096 Mar 21 00:01 tools
drwxr-xr-x   2 root root   4096 Mar 21 00:01 usr
drwxr-xr-x   3 root root   4096 Mar 21 00:01 virt
```

34.9 The Linux kernel's source code exists within an organized directory structure. Fortunately, we will not need to study the contents of this directory structure in order to configure the kernel because a **kernel configuration utility program** (called **menuconfig**) is included with the source code. We will need to enter the **arch** directory, however, after the kernel has been built because this is where the compiled binary file for the kernel will be placed after the build process is finished.

34.10 We will now verify that we are in the **/usr/src/linux** directory and then we will execute the **menuconfig** program using the command **make menuconfig**:

```
(chroot) livecd linux # pwd
/usr/src/linux

(chroot) livecd linux # make menuconfig
```

34.11 As soon as the menuconfig program is executed, it shows something similar to the following text graphics display:

```
 .config - Linux/i386 3.4.83-gentoo Kernel Configuration
 ────────────────────────────────────────────────────────────────────────────────
  ┌─────────────── Linux/i386 3.4.83-gentoo Kernel Configuration ───────────────┐
  │  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, </> for Search.          │
  │  Legend: [*] built-in  [ ] excluded  <M> module  < > module capable          │
  │                                                                              │
  │ ┌──────────────────────────────────────────────────────────────────────────┐ │
  │ │        Gentoo Linux  --->                                                │ │
```

```
            General setup  --->
        [*] Enable loadable module support  --->
        -*- Enable the block layer  --->
            Processor type and features  --->
            Power management and ACPI options  --->
            Bus options (PCI etc.)  --->
            Executable file formats / Emulations  --->
        [*] Networking support  --->
            Device Drivers  --->
            Firmware Drivers  --->
            File systems  --->
            Kernel hacking  --->
            Security options  --->
        -*- Cryptographic API  --->
        [*] Virtualization  --->
            Library routines  --->
        ---
            Load an Alternate Configuration File
        v(+)

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

34.12 As the instructions say at the top of the display, the arrow keys on the keyboard move the blue selection bar around the display. Press the **up** and **down** arrow keys in order to see how they move the selection bar.

34.13 The arrows (--->) next to various menu items indicate that these menus have submenus. The way that you enter a menu's submenu is by placing the selection bar over the menu and pressing the <enter> key. Let's experiment with this by entering the submenu of the **Processor type and features --->** menu:

```
.config - Linux/i386 3.4.83-gentoo Kernel Configuration

                Linux/i386 3.4.83-gentoo Kernel Configuration
    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, </> for Search.
    Legend: [*] built-in  [ ] excluded  <M> module  < > module capable

            Gentoo Linux  --->
            General setup  --->
        [*] Enable loadable module support  --->
        -*- Enable the block layer  --->
            Processor type and features  --->
            Power management and ACPI options  --->
            Bus options (PCI etc.)  --->
            Executable file formats / Emulations  --->
        [*] Networking support  --->
            Device Drivers  --->
            Firmware Drivers  --->
```

```
            File systems  --->
            Kernel hacking  --->
            Security options  --->
        -*- Cryptographic API  --->
        [*] Virtualization  --->
            Library routines  --->
        ---
            Load an Alternate Configuration File
     ────v(+)──────────────────────────────────────────────────────────

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

```
.config - Linux/i386 3.4.83-gentoo Kernel Configuration
──────────────────────────────────────────────────────────────────────────────
  ───────────────────────── Processor type and features ─────────────────────
    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, </> for Search.
    Legend: [*] built-in  [ ] excluded  <M> module  < > module capable

        [*] Tickless System (Dynamic Ticks)
        [*] High Resolution Timer Support
        [*] Symmetric multi-processing support
        [*] Enable MPS table
        [ ] Support for big SMP systems with more than 8 CPUs
        [*] Support for extended (non-PC) x86 platforms
        [ ] Intel MID platform support
        [ ] RDC R-321x SoC
        [ ] Support non-standard 32-bit SMP architectures
        < > Eurobraille/Iris poweroff module
        [*] Single-depth WCHAN output
        [ ] Paravirtualized guest support  --->
        [ ] Memtest
            Processor family (Core 2/newer Xeon)  --->
        [*] Generic x86 support
        [*] HPET Timer Support
        (8) Maximum number of CPUs
        [*] SMT (Hyperthreading) scheduler support
        [*] Multi-core scheduler support
     ────v(+)──────────────────────────────────────────────────────────

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

34.14 The first thing you will notice when you enter the **Processor type and features** submenu is that there are a significant number of options available. Most menus are going to contain many options, but fortunately we will only need to deal with a small number of these options in order to configure the kernel to the point where it will boot the system. Later, after your system is successfully booting from the hard drive, you can return to the menuconfig program and study the options it contains in more depth.

34.15 For now, though, I want to show you how to **exit a submenu**. As the instructions indicate at the top of the display, pressing the **escape key** twice (<Esc><Esc>) will **exit** the current submenu. What the instructions **do not say**, however, is that if you press the escape key once and then wait for a second or two, this will also exit the current submenu. Therefore, you have two ways you can exit a submenu. You can either **1) quickly press the escape key twice** or **2) only press the escape key once and then wait a second or two**.

34.16 Press the escape key twice now in order to exit the **Processor type and features** submenu. When you exit this menu you are placed back in the initial top-level menu.

35 How to select kernel options

35.1.1 Now that you have relaunched the menuconfig program, enter the **Processor type and features** submenu again and let's take a closer look at it:

```
.config - Linux/i386 3.4.83-gentoo Kernel Configuration
────────────────────────────────────────────────────────────────────────────────
 ┌──────────────────────── Processor type and features ────────────────────────┐
 │  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, </> for Search.          │
 │  Legend: [*] built-in  [ ] excluded  <M> module  < > module capable          │
 │ ┌──────────────────────────────────────────────────────────────────────────┐ │
 │ │    [*] Tickless System (Dynamic Ticks)                                   │ │
 │ │    [*] High Resolution Timer Support                                     │ │
 │ │    [*] Symmetric multi-processing support                                │ │
 │ │    [*] Enable MPS table                                                  │ │
 │ │    [ ] Support for big SMP systems with more than 8 CPUs                 │ │
 │ │    [*] Support for extended (non-PC) x86 platforms                       │ │
 │ │    [ ] Intel MID platform support                                        │ │
 │ │    [ ] RDC R-321x SoC                                                    │ │
 │ │    [ ] Support non-standard 32-bit SMP architectures                     │ │
 │ │    < > Eurobraille/Iris poweroff module                                  │ │
 │ │    [*] Single-depth WCHAN output                                         │ │
 │ │    [ ] Paravirtualized guest support  --->                               │ │
 │ │    [ ] Memtest                                                           │ │
 │ │        Processor family (Core 2/newer Xeon)  --->                        │ │
 │ │    [*] Generic x86 support                                               │ │
 │ │    [*] HPET Timer Support                                                │ │
 │ │    (8) Maximum number of CPUs                                            │ │
 │ │    [*] SMT (Hyperthreading) scheduler support                            │ │
 │ │    [*] Multi-core scheduler support                                      │ │
 │ └────v(+)──────────────────────────────────────────────────────────────────┘ │
```

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

35.1.2 In order to understand how to select kernel options with the menuconfig program, you must first understand that a kernel can be built in the following two ways:

1) As one monolithic binary file which contains the core kernel code along with all of the device drivers.
2) As a smaller binary file (which may include some device drivers) along with **external** device drivers called "**kernel modules**". A **kernel module** is a device driver that has been built separately from the kernel and is not placed within the kernel's binary file. Instead, it is stored somewhere in the filesystem, and it can be loaded into memory and plugged into a running kernel as needed. Kernel modules can also be detached from the kernel and removed from memory if they are not needed any longer.

35.1.3 Of these two ways to build the kernel, we are going to use technique #1 (and build one big monolithic kernel) because I think it is the easier of the two ways for beginners to understand. Technique #2, however, is more flexible than technique #1, and you may explore building a kernel that uses kernel modules at a later date. An example of a system that uses technique #2 is the CD that we booted the machine with. The CD contains a large number of kernel modules. As the system boots, it scans the system's hardware and determines which kernel modules are needed to access this hardware. Only the modules that are needed are loaded into memory and attached to the kernel. This uses the system's memory resources more efficiently than one huge monolithic kernel would.

35.1.4 Now that you know about the two ways to build a Linux kernel, we can discuss how the menuconfig program allows you to select **kernel options**. The square braces [] and arrowed braces <> that are next to each kernel option allow you to either select that option or to deselect it. Options are selected by moving the selection bar over the option and pressing the '**Y**' key. Pressing the '**N**' key will deselect the option and pressing the space bar will toggle the selection. A **selected** option will have an **asterisk** '*' placed within the braces and a **deselected** option will have **empty** braces.

35.1.5 Experiment with this now by moving the selection bar to one of the options and pressing '**Y**', '**N**', and the **space bar**. If you want more information about what an option does, press the '**H**' key and a **help** window will appear. Do this now with an option to read about what it does and then exit the window either by pressing the **escape key twice** or by pressing the **<enter>** key. When you are done experimenting, set the option to its original setting.

35.1.6 The difference between options that have square braces [] and options that have arrowed braces <> is that options that have arrowed braces <> can also be built as **kernel modules**. Highlight an option that has <> braces with the selection bar and, in addition to pressing the '**Y**' key, '**N**' key, and the **space bar**, try pressing the '**M**' key to select this option as a **module**. When you are done experimenting, set the option back to its original setting.

36 Selecting your CPU's processor family

36.1.1 The last option we will set in the **Processor type and features** submenu is the **Processor Family** option. Move the selection bar over this option and then enter its submenu by pressing the <enter> key. A **Processor family** window will be shown and you now need to decide which processor family the CPU in your system belongs to. Earlier, you looked inside of the **/proc/cpuinfo** file to determine what CPU your system had and you can do this again by temporarily switching to another virtual terminal by pressing the <alt><Fx> keys (where 'x' is between 2 and 6). You can switch back to the original terminal by typing <alt><F1>.

36.1.2 After you have determined the type of your CPU, switch back to the default terminal, find its family in the **Processor Family** window and select it by pressing the **space bar**. You will then be automatically sent back to the **Processor type and features** menu.

36.1.3 We are done selecting options in the **Processor type and features** menu so press the **escape** key twice to **exit** this menu and go back to the top-level menu.

37 Network device driver

37.1 The next option we are going to select is a device driver for our system's **Ethernet network interface chip** and it exists within the **Device Drivers ---> Network device support** submenu:

```
.config - Linux/i386 3.4.83-gentoo Kernel Configuration
──────────────────────────────────────────────────────────────────────
 ┌──────────── Linux/i386 3.4.83-gentoo Kernel Configuration ─────────┐
 │  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, </> for Search.         │
  │  Legend: [*] built-in  [ ] excluded  <M> module  < > module capable         │
  │ ┌─────────────────────────────────────────────────────────────────────────┐ │
  │ │        Gentoo Linux  --->                                               │ │
  │ │        [*] 64-bit kernel                                                │ │
  │ │        General setup  --->                                              │ │
  │ │    [*] Enable loadable module support  --->                             │ │
  │ │    -*- Enable the block layer  --->                                     │ │
  │ │        Processor type and features  --->                                │ │
  │ │        Power management and ACPI options  --->                          │ │
  │ │        Bus options (PCI etc.)  --->                                     │ │
  │ │        Executable file formats / Emulations  --->                       │ │
  │ │    [*] Networking support  --->                                         │ │
  │ │        Device Drivers  --->                                             │ │
  │ │        Firmware Drivers  --->                                           │ │
  │ │        File systems  --->                                               │ │
  │ │        Kernel hacking  --->                                             │ │
  │ │        Security options  --->                                           │ │
  │ │    -*- Cryptographic API  --->                                          │ │
  │ │    [*] Virtualization  --->                                             │ │
  │ │        Library routines  --->                                           │ │
  │ │    ---                                                                  │ │
  │ │        Load an Alternate Configuration File                             │ │
  │ └────v(+)─────────────────────────────────────────────────────────────────┘ │
  ├─────────────────────────────────────────────────────────────────────────────┤
  │                      <Select>    < Exit >    < Help >                       │
  └─────────────────────────────────────────────────────────────────────────────┘
```

```
 .config - Linux/i386 3.4.83-gentoo Kernel Configuration
 ─────────────────────────────────────────────────────────────────────────────────
  ┌────────────────────────────── Device Drivers ───────────────────────────────┐
  │  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, </> for Search.         │
  │  Legend: [*] built-in  [ ] excluded  <M> module  < > module capable         │
  │ ┌─────────────────────────────────────────────────────────────────────────┐ │
  │ │        Generic Driver Options  --->                                     │ │
  │ │    <*> Connector - unified userspace <-> kernelspace linker  --->       │ │
  │ │    < > Memory Technology Device (MTD) support  --->                     │ │
  │ │    < > Parallel port support  --->                                      │ │
  │ │    -*- Plug and Play support  --->                                      │ │
  │ │    [*] Block devices  --->                                              │ │
  │ │        Misc devices  --->                                               │ │
  │ │    < > ATA/ATAPI/MFM/RLL support (DEPRECATED)  --->                     │ │
  │ │        SCSI device support  --->                                        │ │
  │ │    <*> Serial ATA and Parallel ATA drivers  --->                        │ │
  │ │    [*] Multiple devices driver support (RAID and LVM)  --->             │ │
  │ │    < > Generic Target Core Mod (TCM) and ConfigFS Infrastructure  --->  │ │
  │ │    [ ] Fusion MPT device support  --->                                  │ │
  │ │        IEEE 1394 (FireWire) support  --->                               │ │
  │ │    < > I2O device support  --->                                         │ │
  │ │    [*] Macintosh device drivers  --->                                   │ │
  │ │    [*] Network device support  --->                                     │ │
  │ │    [ ] ISDN support  --->                                               │ │
```

```
 │ │        Input device support  --->                               │ │
 │ └────v(+)──────────────────────────────────────────────────────────┘ │
 ├──────────────────────────────────────────────────────────────────────┤
 │                      <Select>    < Exit >    < Help >                │
 └──────────────────────────────────────────────────────────────────────┘
```

37.1.1 Inside the **Network device support** submenu there is a submenu named **Ethernet driver support**. Select it.

```
.config - Linux/i386 3.4.83-gentoo Kernel Configuration
──────────────────────────────────────────────────────────────────────────
 ┌────────────────────────── 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, </> for Search.
 │  Legend: [*] built-in  [ ] excluded  <M> module  < > module capable  │
 │ ┌──────────────────────────────────────────────────────────────────┐ │
 │ │    --- Network device support                                    │ │
 │ │    -*-   Network core driver support                             │ │
 │ │    < >     Bonding driver support                                │ │
 │ │    < >     Dummy net driver support                              │ │
 │ │    < >     EQL (serial line load balancing) support              │ │
 │ │    [ ]     Fibre Channel driver support                          │ │
 │ │    -*-     Generic Media Independent Interface device support    │ │
 │ │    < >     Intermediate Functional Block support                 │ │
 │ │    < >     Ethernet team driver support (EXPERIMENTAL)  --->     │ │
 │ │    < >     MAC-VLAN support (EXPERIMENTAL)                       │ │
 │ │    <*>     Network console logging support                       │ │
 │ │    [ ]     Netpoll traffic trapping                              │ │
 │ │    < >     Universal TUN/TAP device driver support               │ │
 │ │    < >     Virtual ethernet pair device                          │ │
 │ │    < >   ARCnet support  --->                                    │ │
 │ │          *** CAIF transport drivers ***                          │ │
 │ │    [*]   Ethernet driver support  --->                           │ │
 │ │    <*>   FDDI driver support                                     │ │
 │ │    < >     Digital DEFTA/DEFEA/DEFPA adapter support             │ │
 │ └────v(+)──────────────────────────────────────────────────────────┘ │
 ├──────────────────────────────────────────────────────────────────────┤
 │                      <Select>    < Exit >    < Help >                │
 └──────────────────────────────────────────────────────────────────────┘
```

37.1.2 The **Ethernet driver support** submenu looks like this:

```
 .config - Linux/i386 3.4.83-gentoo Kernel Configuration
──────────────────────────────────────────────────────────────────────────
 ┌────────────────────────── Ethernet driver 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, </> for Search.
 │  Legend: [*] built-in  [ ] excluded  <M> module  < > module capable  │
 │ ┌──────────────────────────────────────────────────────────────────┐ │
 │ │    --- Ethernet driver support                                   │ │
```

```
│ │     [*]   3Com devices                                                  │ │
│ │     < >     3Com 3c574 PCMCIA support                                   │ │
│ │     < >     3Com 3c589 PCMCIA support                                   │ │
│ │     < >     3c590/3c900 series (592/595/597) "Vortex/Boomerang" support │ │
│ │     < >     3cr990 series "Typhoon" support                             │ │
│ │     [*]   Adaptec devices                                               │ │
│ │     < >     Adaptec Starfire/DuraLAN support                            │ │
│ │     [*]   Alteon devices                                                │ │
│ │     < >     Alteon AceNIC/3Com 3C985/NetGear GA620 Gigabit support      │ │
│ │     [*]   AMD devices                                                   │ │
│ │     < >     AMD 8111 (new PCI LANCE) support                            │ │
│ │     < >     AMD PCnet32 PCI support                                     │ │
│ │     < >     New Media PCMCIA support                                    │ │
│ │     [*]   Atheros devices                                               │ │
│ │     < >     Atheros L2 Fast Ethernet support                            │ │
│ │     < >     Atheros/Attansic L1 Gigabit Ethernet support                │ │
│ │     < >     Atheros L1E Gigabit Ethernet support (EXPERIMENTAL)         │ │
│ │     < >     Atheros L1C Gigabit Ethernet support (EXPERIMENTAL)         │ │
│ └─────v(+)────────────────────────────────────────────────────────────────┘ │
├─────────────────────────────────────────────────────────────────────────────┤
│                      <Select>    < Exit >    < Help >                       │
└─────────────────────────────────────────────────────────────────────────────┘
```

37.1.3 My system has an **Advanced Micro Devices PCnet32** Ethernet chip in it and therefore this is the option I selected.

37.1.4 For VirtualBox users, make sure the following drivers are selected:

```
│ │     [*]   Intel devices                                                 │ │
│ │     <*>     Intel(R) PRO/100+ support                                   │ │
│ │     <*>     Intel(R) PRO/1000 Gigabit Ethernet support                  │ │
│ │     <*>     Intel(R) PRO/1000 PCI-Express Gigabit Ethernet support
```

37.1.5 After you have selected your Ethernet chip, escape back to the **Device Drivers** menu.

38 Saving the kernel configuration file and viewing it

38.1.1 We are done configuring the kernel for now and the last thing we need to do is to carefully escape back to the top-level menu and then **escape out of the menuconfig application** itself. Before exiting to the command prompt, the **menuconfig** application will ask if we want to save our new kernel configuration. Select the <Yes> option by pressing the <enter> key:

```
.config - Linux/i386 3.4.83-gentoo Kernel Configuration
─────────────────────────────────────────────────────────────────────────

           ┌───────────────────────────────────────────────────────┐
```

```
Do you wish to save your new configuration ? <ESC><ESC>
to continue.

                 < Yes >      <  No  >
```

38.1.2 Now that the kernel configuration file has been saved, let's look at it. At the command prompt, execute an **ls** command:

```
(chroot) livecd linux # ls
arch     crypto         firmware  ipc      lib          net             scripts
usr
block    distro         fs        Kbuild   MAINTAINERS  README          security
virt
COPYING  Documentation  include   Kconfig  Makefile     REPORTING-BUGS  sound
CREDITS  drivers        init      kernel   mm           samples         tools
```

38.1.3 The file is in this directory but we cannot see it because it was saved as a **hidden file**! In order to have the **ls** command show all of the hidden files in a directory, pass a **-a** option to it (which means list all):

```
(chroot) livecd linux # ls -a
.             COPYING        firmware    Kbuild       Makefile        scripts
..            CREDITS        fs          Kconfig      mm              security
arch          crypto         .gitignore  kernel       net             sound
block         distro         include     lib          README          tools
.config       Documentation  init        .mailmap     REPORTING-BUGS  usr
.config.old   drivers        ipc         MAINTAINERS  samples         virt
```

38.1.4 In GNU/Linux, the way to make a file hidden is to place a period '.' before the file name. The name of the Linux kernel's configuration file is **.config**, and it is shown above highlighted in blue. It is a text file, so you can use the **cat** command to view its contents or you can page through it using the **less** command (the file is quite long so I am only going to show the beginning part of it):

```
(chroot) livecd linux # cat .config
#
# Automatically generated file; DO NOT EDIT.
# Linux/i386 3.4.83-gentoo Kernel Configuration
#

#
# Gentoo Linux
#
CONFIG_GENTOO_LINUX=y
CONFIG_GENTOO_LINUX_UDEV=y

#
# Support for init systems, system and service managers
#
```

```
CONFIG_GENTOO_LINUX_INIT_SCRIPT=y
# CONFIG_GENTOO_LINUX_INIT_SYSTEMD is not set
# CONFIG_64BIT is not set
CONFIG_X86_32=y
# CONFIG_X86_64 is not set
CONFIG_X86=y
CONFIG_INSTRUCTION_DECODER=y
CONFIG_OUTPUT_FORMAT="elf32-i386"
CONFIG_ARCH_DEFCONFIG="arch/x86/configs/i386_defconfig"
CONFIG_GENERIC_CMOS_UPDATE=y
CONFIG_CLOCKSOURCE_WATCHDOG=y
CONFIG_GENERIC_CLOCKEVENTS=y
CONFIG_GENERIC_CLOCKEVENTS_BROADCAST=y
CONFIG_LOCKDEP_SUPPORT=y
CONFIG_STACKTRACE_SUPPORT=y
CONFIG_HAVE_LATENCYTOP_SUPPORT=y
CONFIG_MMU=y
# CONFIG_NEED_DMA_MAP_STATE is not set
CONFIG_INIT_ENV_ARG_LIMIT=32
<snip>
```

39 Compiling the kernel

39.1.1 It is now time to compile the kernel! In order to compile the kernel, simply execute the **make && make modules_install** command at the command line (the output from the compile process is also long so I will only show a small part of it):

```
(chroot) livecd linux # make && make modules_install
  HOSTLD  scripts/kconfig/conf
scripts/kconfig/conf -s arch/i386/Kconfig
  CHK     include/linux/version.h
  UPD     include/linux/version.h
  CHK     include/linux/utsrelease.h
  UPD     include/linux/utsrelease.h
  SYMLINK include/asm -> include/asm-i386
  CC      arch/i386/kernel/asm-offsets.s
  GEN     include/asm-i386/asm-offsets.h
  CC      scripts/mod/empty.o
  HOSTCC  scripts/mod/mk_elfconfig
  MKELF   scripts/mod/elfconfig.h
  HOSTCC  scripts/mod/file2alias.o
  HOSTCC  scripts/mod/modpost.o
  HOSTCC  scripts/mod/sumversion.o
  HOSTLD  scripts/mod/modpost
  HOSTCC  scripts/kallsyms
<snip>
```

39.1.2 The double ampersand symbol '&&' is called a "**logical and**" symbol, and it allows more than one command to be executed on a single command line. For example, if we wanted to execute a **date** command

and then a **ls** command on the same command line, we could enter **date && ls**. The **date** command would be executed first, and then the **ls** command would be executed. While the kernel is compiling you can switch to another virtual terminal and type **date && ls** to see what happens:

```
(chroot) livecd / # date && ls
Sun Mar 30 09:22:24 EST 2014
bin   lost+found                       proc                                     sys
boot  media                            root                                     tmp
dev   mnt                              run                                      usr
etc   opt                              sbin                                     var
home  portage-20150322.tar.bz2         stage3-amd64-20180311T214502Z.tar.xz
lib   portage-20150322.tar.bz2.md5sum  stage3-amd64-20180311T214502Z.tar.xz.DIGESTS
kosan1 / # date
Sun Mar 30 06:37:12 EST 2014
```

39.1.3 Notice that the **date** command was executed (its output is highlighted in blue) followed by the **ls** command.

39.1.4 Going back to the command line we used to compile the kernel, **make** is a program that is used to build software that consists of many source files. We used the **logical and** symbol '&&' to run the **make** program twice. The first time **make** is run it **compiles the main kernel**. The second time **make** is run it **compiles** any options that were configured as **kernel modules**, and installs them somewhere in the directory hierarchy.

39.1.5 The kernel will take a while to build, and here is the last part of the output that is generated after it is finished:

```
<...>
  LD      arch/x86/boot/setup.elf
  OBJCOPY arch/x86/boot/setup.bin
  OBJCOPY arch/x86/boot/vmlinux.bin
  BUILD   arch/x86/boot/bzImage
Setup is 15008 bytes (padded to 15360 bytes).
System is 4851 kB
CRC 795a7650
Kernel: arch/x86/boot/bzImage is ready  (#6)
  Building modules, stage 2.
  MODPOST 7 modules
  CC      arch/x86/platform/iris/iris.mod.o
  LD [M]  arch/x86/platform/iris/iris.ko
  INSTALL arch/x86/kernel/test_nx.ko
  INSTALL arch/x86/platform/iris/iris.ko
  INSTALL drivers/char/kcopy/kcopy.ko
  INSTALL drivers/hid/hid-logitech-dj.ko
  INSTALL drivers/scsi/scsi_wait_scan.ko
  INSTALL net/netfilter/xt_LOG.ko
```

```
  INSTALL net/netfilter/xt
```

39.1.6 The first line of this output that is highlighted in blue indicates that the kernel has been compiled into one large file called **bzImage** and it has been placed into the **/usr/src/linux/arch/i386/boot** directory. In the next section we will copy the kernel into the **/boot** directory (which is the directory that the **sda1 boot partition** is attached to) so that it can be located during boot time. The second line that is highlighted in blue indicates that the kernel option that we selected as a **kernel module** has been installed into our directory hierarchy. The directory it was installed into is /lib/modules/3.4.83-gentoo/kernel/arch/x86/platform, but this is not shown.

39.1.7 Assuming that you are still in the **/usr/src/linux** directory, change into the **arch/i386/boot** directory by executing a **cd arch/i386/boot** command and then execute an **ls -l** command to see the **bzImage** file:

```
(chroot) livecd linux # cd arch/x86_64/boot

(chroot) livecd boot # pwd
/usr/src/linux/arch/x86_64/boot

(chroot) livecd boot # ls -l
total 0
lrwxrwxrwx 1 root root 22 Mar 21 02:52 bzImage -> ../../x86/boot/bzImage
```

39.1.8 The listing indicates that the directory contains a symbolic link to the **bzImage** file. Before we copy **bzImage** into the **/boot** directory, we should make sure that the **/dev/sda1 boot partition** is still mounted to it. This can be done by cating /proc/mounts:

```
(chroot) livecd boot # cat /proc/mounts
/dev/sda3 / ext3
rw,relatime,errors=continue,user_xattr,acl,barrier=1,data=writeback 0 0
/dev/sda1 /boot ext2 rw,relatime,errors=continue,user_xattr,acl 0 0
udev /dev devtmpfs rw,nosuid,relatime,size=10240k,nr_inodes=127142,mode=755 0 0
none /proc proc rw,relatime 0 0
```

39.1.9 The parts of this listing that are highlighted in blue indicate that the **/dev/sda1** partition is indeed mounted to the **/boot** directory. We can now copy the **bzImage** file into the **/boot** directory by issuing a **cp bzImage /boot** command:

```
(chroot) livecd boot # cp bzImage /boot
```

39.1.10 Finally, let's make sure that the **bzImage** file was correctly copied to the **/boot** directory by changing to this directory and then executing an

ls command:

```
(chroot) livecd boot # cd /boot

(chroot) livecd boot # ls
bzImage  lost+found
```

40 Configuring the system

40.1 We are almost done with the base installation process and our next step is to configure the system.

41 Configuring the /etc/fstab file

41.1.1 During the Gentoo Linux installation process, we manually mounted filesystems to various directories in the directory hierarchy. When the system is ready to use, however, we do not want to be required to manually mount filesystems each time the system is booted. Luckily, the process of mounting filesystems at boot time can be automated by defining which filesystems should be mounted where in the **/etc/fstab** file. The name **fstab** stands for **file system table** and here are the contents of this file before it has been configured:

```
(chroot) livecd / # cd /etc

(chroot) livecd etc # pwd
/etc

(chroot) livecd etc # nano -wc fstab
# /etc/fstab: static file system information.
#
# noatime turns off atimes for increased performance (atimes normally aren't
# needed); notail increases performance of ReiserFS (at the expense of storage
# efficiency).  It's safe to drop the noatime options if you want and to
# switch between notail / tail freely.
#
# The root filesystem should have a pass number of either 0 or 1.
# All other filesystems should have a pass number of 0 or greater than 1.
#
# See the manpage fstab(5) for more information.
#

# <fs>                  <mountpoint>    <type>          <opts>          <dump/
pass>

# NOTE: If your BOOT partition is ReiserFS, add the notail option to opts.
#
# NOTE: Even though we list ext4 as the type here, it will work with ext2/ext3
```

```
#       filesystems.  This just tells the kernel to use the ext4 driver.
#
# NOTE: You can use full paths to devices like /dev/sda3, but it is often
#       more reliable to use filesystem labels or UUIDs. See your filesystem
#       documentation for details on setting a label. To obtain the UUID, use
#       the blkid(8) command.

#LABEL=boot       /boot       ext4        noauto,noatime    1 2
#UUID=58e72203-57d1-4497-81ad-97655bd56494           /           ext4  noatime
      0 1
#LABEL=swap       none        swap        sw          0 0
#/dev/cdrom       /mnt/cdrom  auto        noauto,ro   0 0
```

41.1.2 The **fstab** file is organized into columns with the **filesystem device** listed in the **first** column <fs>, the **point in the directory hierarchy where it should be mounted** listed in the **second** column <mountpoint> and the **type** of the filesystem listed in the **third** column <type>. If you would like to know what the <opts> and <dump/pass> columns do, you can look at the man page for the **fstab** file.

41.1.3 Add the following lines to the end of the fstab file:

```
...
/dev/sda1          /boot             ext2              noauto,noatime  1 2
/dev/sda3          /                 ext3              noatime         0 1
/dev/sda2          none              swap              sw              0 0
```

42 Configuring the network

42.1.1 The system's network software now needs to be configured and the first step in this process is to **give your machine a name**. The machine's **name** is held in a variable called **HOSTNAME** which is in the **/etc/conf.d/hostname** file. The default name that the machine is currently set to is **localhost** and this can be seen in the following listing:

```
(chroot) livecd / # cd /etc/conf.d

(chroot) livecd conf.d # pwd
/etc/conf.d

(chroot) livecd conf.d # nano -wc hostname

# Set to the hostname of this machine
hostname="localhost"
```

42.1.2 The name I am giving my machine for now is "**kosan1**" but I will probably change it later (edit the **hostname** file to change the machine's host name):

```
hostname="kosan1"
```

42.1.3 The next step in configuring the network is to set the name of the **Internet domain** it is inside of. We will discuss what an Internet domain name is later so for now **use nano to create a file in /etc/conf.d named "net"**, and place the following line into it:

```
dns_domain_lo="hostname"
```

42.1.4 The last network-oriented file that needs to be edited with nano is the **/etc/hosts** file. The **hosts** file allows the system administrator to manually describe the local network the machine is attached to by associating **machine names** with **IP addresses**. We are assuming that our systems will use **DHCP** to automatically be configured by the network so we will not be adding much information to the **hosts** file. The one line in this file that we will edit, however, currently looks like this:

```
127.0.0.1       localhost
```

42.1.5 Earlier we discussed that the 127.0.0.1 IP address is a special address which is associated with the **loopback** interface. This address refers to the local machine and we need to add the following (colored) information to the line that contains this address in the **/etc/hosts** file:

```
127.0.0.1       kosan1.homenetwork kosan1 localhost
```

42.1.6 Keep in mind that instead of using the name '**kosan1**' here you should use the name that you gave your own machine.

43 Services and the network service

43.1.1.1 GNU/Linux operating systems have special programs called **system services** that can be started either manually or automatically. Another name for a **system service** is a "**daemon**" and this name is used because in Greek mythology, daemons were entities that performed various tasks for the Greek gods.

43.1.1.2 Many of these services will run continuously until they are stopped or the machine is shut down. The following is a list of services that are often run on GNU/Linux machines:

- Network service - Maintains the machine's connection to the network.
- Logging service - Accepts messages from all of the pieces of software that are running on the system and saves them into a log file.
- Cron service - Runs commands at times that are set by the system administrator.
- Secure shell service - Allows users to remotely log into the system using a secure connection.

43.1.1.3 A network service has already been installed on our machines, but it is not currently configured to start automatically at boot time. In order to have this service start automatically at boot time, execute a **rc-update add net.enp0s3 default** command:

```
(chroot) livecd conf.d # cd /etc/init.d

(note, 'l' is a lower case L in the following line):
(chroot) livecd init.d # ln -s net.lo net.enp0s3

(chroot) livecd init.d # rc-update add net.enp0s3 default
 * net.enp0s3 added to runlevel default
```

43.1.1.4 The name of the network service is **net.enp0s3** and we have configured it to start when the machine enters what is called the **default runlevel**. GNU/Linux machines usually have a number of runlevels and a **runlevel** is a way to define a set of system services that should be running at the same time. The **default runlevel** is the normal runlevel a Gentoo system operates at after it has booted.

44 Emerging the DHCP client

44.1.1.1 Since we are going to use DHCP to have the network automatically configure our machines, we need to install the DHCP client program (a client is software that uses a service that is provided by a server). This is done by executing an **emerge dhcp** command:

```
(chroot) livecd conf.d # emerge dhcp
```

45 Installing additional services

45.1 Earlier we talked about system services and we will now install a **logging** service and a **cron** service. To install a logging service called **syslog-ng**, execute an **emerge syslog-ng** command. After it is done emerging, add it to the **default runlevel** by executing an **rc-update add syslog-ng default** command:

```
(chroot) livecd / # emerge syslog-ng
```

```
(chroot) livecd / # rc-update add syslog-ng default
```

45.2 To install a **cron** service called **vixie-cron**, execute an **emerge vixie-cron** command and after it is done emerging, add it to the **default runlevel** by executing an **rc-update add vixie-cron default** command:

```
(chroot) livecd / # emerge vixie-cron
```

```
(chroot) livecd / # rc-update add vixie-cron default
```

46 Installing additional software

46.1 When you eventually boot your virtual machine from its hard drive, you will need to authenticate it with the Atlas network using a browser. Lynx is a text-based web browser that can be used for this purpose.

```
(chroot) livecd / # emerge lynx
```

47 Installing and configuring the bootloader

47.1 Before the **/dev/sda1** boot partition can be used to boot the system, **boot loader** software needs to be installed on it and configured. There are various bootloader options available, but the one we will be using is an older bootloader named “lilo” which stands for Linux Loader. Emerge lilo:

```
(chroot) livecd / # emerge sys-boot/lilo
```

47.2 After **lilo** has been **emerged**, a file needs to be placed into the **/etc** directory named **lilo.conf**:

```
(chroot) livecd boot # nano -wc /etc/lilo.conf
```

47.3 Place the following information into lilo.conf (make sure the '**I**' in **bzImage** is capitalized):

```
boot=/dev/sda
prompt
timeout=20
default=gentoo

image=/boot/bzImage
 label=gentoo
 read-only
```

```
root=/dev/sda3
```

47.4 The **default** keyword selects which **boot option** to accept by **default**. In our case, we only have one boot option (named “gentoo”) so this will be the default. The **timeout** keyword selects how long the boot menu will wait for input from the keyboard before moving forward with the boot process using the default boot option.

47.5 The **label** keyword determines what is displayed in the boot menu for a given boot option. The **root** keyword tells lilo what device contains the boot partition and which partition it is. Finally, the **image** keyword indicates where the kernel image has been placed on the boot partition.

47.6 The final step that need to be done to install the boot loader is to execute the **/sbin/lilo** command:

```
(chroot) livecd / # /sbin/lilo
```

47.7 If the **lilo** command did not list any errors (warnings are okay), then the **boot loader** was installed correctly.

48 Setting the root password and adding a user account

48.1 UNIX-like systems are known as **multiuser** systems because they can have more than one person logged into the system and using it at the same time. Each user is given what is called an **account** on the system and each account has a **username** and a **password** associated with it. Most users are also given their own **directory** to use which is called their **home directory**. All user home directories are placed within the **/home** directory.

48.2 Each user account has various **permissions** associated with it which determine which resources on the machine the user has access to. There is always **at least one account** on all UNIX-like systems which is called the **root account** and the user that has access to this account is called the **root user** or the **super user**. The **root account** has **permission** to **access all of the resources in the system** and the **home directory** for this account is the **/root** directory.

48.3 It is just a **coincidence** that the **top-level directory is called a root directory**, the **super user of a system is also called the root user** and the **root user's home directory is called /root**. This can be confusing at first but one becomes used to it over time.

48.4 During the whole time we have been working from the CD, we have been using the super user's account. In the **bash shell**, a person can tell if they are currently using the **super user's account** if a pound sign '**#**' is displayed at the end of the command prompt.

48.5 Before we **reboot** the system, we need to **set the password for the root user**. If you do not set the root user's password now, when you reboot the machine you will not be able to log into your system! What you will then be forced to do is to reboot from the CD, mount all of the partitions, chroot into the /dev/sda3 partition, and then set the password. Let's set the password now in order to avoid this extra work. You can set the root user's password for your machine by executing a **passwd root** command:

```
(chroot) livecd / # passwd root
New UNIX password:
BAD PASSWORD: it is too short
Retype new UNIX password:
passwd: password updated successfully
```

48.6 The **passwd** command does not show the password you typed so that someone cannot look at your screen and steal your password. It also asks you to type the password twice in case you made a spelling mistake while typing. The **passwd** command will also inform you if it thinks you have chosen an **unwise** password (which it calls a BAD password). In my case, it thinks the password I typed is too short. However, if the command also indicates that the **password was updated successfully**, then you can still use this password if you would like.

48.7 Now that you have set the root user's password, it is time to create a normal user account for yourself on the machine. It is considered to be a bad idea to use the root user's account for doing normal work on a system for a number of reasons. One reason is that if you make a mistake (like accidentally deleting the /etc directory) you can lose data or crash the machine. It is safer to do normal work in your own user account and just switch into the root user's account only when you need that account's extra permissions to do something.

48.8 You can create a user account for yourself using the **useradd** command. The following example shows the creation of a user account which will have the username **tkosan** associated with it (**note: the username must have all lowercase letters**):

```
(chroot) livecd / # useradd -m -G users,wheel -s /bin/bash tkosan
```

48.9 The **-m** option tells **useradd** to **create a home directory** for the user, the **-G** option **indicates what groups the user should belong to** (we will cover groups later) and the **-s** option **determines which shell the user will use when they log in**. After the account has been created, a password needs to be set for it using the **passwd** command:

```
(chroot) livecd / # passwd tkosan
New UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

48.10 You will now be able to log into your new account when you reboot the machine.

49 Rebooting the machine

49.1 The base Gentoo Linux installation process is now complete! Before rebooting the machine, however, we need to **exit from the chroot environment**, **change into the top-level root directory,** and **unmount all of the filesystems** that we mounted earlier. The chroot environment is exited using the **exit** command, and filesystems are unmounted using the **umount** command:

```
(chroot) livecd / # exit
exit

livecd / # cd /
livecd / # umount /mnt/gentoo/dev
livecd / # umount /mnt/gentoo/proc
livecd / # umount /mnt/gentoo/boot
livecd / # umount /mnt/gentoo
```

49.2 You should also exit from each **virtual terminal** you logged into using the **exit** command.

49.3 Finally, execute the **halt** command to halt the system

```
livecd / # halt
```

49.4 Before rebooting, either **1) remove the CD** or **2) make sure that the motherboard is configured to have the hard drive as the first boot device and not the CDROM drive using the motherboard setup utility**. After the system reboots, you will be presented with a **login prompt**. Type the **username** for the root account (which is "**root**") and press the <enter> key. Type the root account's password, press the <enter> key again and you will be given a standard command prompt.

49.5 You can now start exploring your new system. The first thing you may want to do is to **remove** the **stage3** and **portage** files from the **top-level root directory** using the **rm** (remove) command since you will not be needing them anymore.

49.6 When you are ready to shut your system down, you can use the **halt** command. You must be using the super user's account in order to halt the machine.

50 Miscellaneous procedures

50.1 Shutting down the system before chroot is executed

50.1.1 If you need to shut down the virtual machine (and not just close it and save its state) before the chroot command is executed, execute the following commands:

```
cd /
umount /mnt/gentoo/boot
umount /mnt/gentoo
halt
```

50.1.2 When you reboot the VM, you must remount /dev/sda3 to /mnt/gentoo and /dev/sda1 to /mnt/gentoo/boot before continuing.

50.2 Shutting down the system after chroot is executed

50.2.1 If you need to shut down the virtual machine (and not just close it and save its state) after the chroot command is executed, execute the following commands:

```
exit
cd /
umount /mnt/gentoo/proc
umount /mnt/gentoo/dev
umount /mnt/gentoo/boot
umount /mnt/gentoo
halt
```

50.2.2 When you reboot the VM, you must remount sda3, sda1, proc, and dev and chroot again before continuing.

50.3 Procedure for reentering the chroot environment.

```
livecd / # swapon /dev/sda2
livecd / # mount /dev/sda3 /mnt/gentoo
livecd / # mount /dev/sda1 /mnt/gentoo/boot
livecd / # cp -L /etc/resolv.conf /mnt/gentoo/etc/resolv.conf
livecd / # mount -o bind /dev /mnt/gentoo/dev
livecd / # mount -t proc none /mnt/gentoo/proc
livecd / # chroot /mnt/gentoo /bin/bash
livecd / # env-update
livecd / # source /etc/profile
```

50.4 Procedure for when grub> comes up when booting from the hard drive.

50.5 When grub> come up when trying to boot from the hard drive, this usually means that the grub.conf file has a typo in it. Do the following to fix it:

1) Boot off of the .iso CD image.
2) Mount /dev/hda1 to /mnt/gentoo.
3) Use the cd command to change into /mnt/gentoo/boot/grub.
4) Edit the grub.conf file to fix it.
5) Change back into the root directory with cd /.
6) Unmount /mnt/gentoo.
7) Try rebooting using the hard drive again.

51 NOTES FOR THE PROFESSOR

```
- Start with NAT networking.
- chroot.
- emerge lynx.
- Switch to bridged networking. VirtualBox seems to have trouble using bridged
networking with wlan0, so have the host use eth0.
- Use lynx to authenticate the VM with atlas.
Atom
```