Chapter 5

Advanced Features

This chapter will introduce you to some of the more interesting features of Linux. This assumes that you have at least basic UNIX experience, and understand the information contained in the previous chapters.

The most important aspect of Linux that distinguishes it from other implementations of UNIX is its open design and philosophy. Linux was not developed by a small team of programmers headed by a marketing committee with a single goal in mind. It was developed by an ever-increasing group of hackers, putting what they wanted into a homebrew UNIX system. The types of software and diversity of design in the Linux world is large. Some people dislike this lack of uniformity and conformity—however, some call it one of the strongest qualities of Linux.

5.1 The X Window System

The X Window System is a large and powerful (and somewhat complex) graphics environment for UNIX systems. The original X Windows code was developed at MIT; commercial vendors have since made X the industry standard for UNIX platforms. Virtually every workstation in the world runs some variant of X Windows.

A free port of the MIT X Windows version 11, release 5 (X11R5) for 80386/80486 UNIX systems has been developed by a team of programmers headed by David Wexelblat[1]. The release, known as XFree86, is available for System V/386, 386BSD, and other i386 UNIX implementations, including Linux. It includes all of the required binaries, support files, libraries, and tools.

Configuring and using the X Window System is far beyond the scope of this book. You are encouraged to read *The X Window System User's Guide*, by Valerie Quercia and Tim O'Reilly. See Appendix A for information on this book. In this section, we'll give a general overview of installing and configuring X Windows for Linux, but it is far from complete. The man pages and README files included with the Linux X Windows distribution should be very helpful.

[1] David may be reached on the Internet at `dwex@mtgzfs3.att.com`.

The Linux XFree86 HOWTO (see Appendix A for details) contains a complete discussion of installation and configuration of X Windows for Linux. We refer all interested readers to this document.

5.1.1 Hardware requirements

As of XFree86 version 2.1, the following video chipsets are supported. The documentation included with your video adaptor should specify the chipset used.

- Tseng ET3000, ET4000AX, ET4000/W32
- Western Digital/Paradise PVGA1
- Western Digital WD90C00, WD90C10, WD90C11, WD90C24, WD90C30, WD90C31
- Genoa GVGA
- Trident TVGA8800CS, TVGA8900B, TVGA8900C, TVGA8900CL, TVGA9000, TVGA9000i, TVGA9100B, TVGA9200CX, TVGA9320, TVGA9400CX, TVGA9420
- ATI 28800-4, 28800-5, 28800-a
- NCR 77C22, 77C22E, 77C22E+
- Cirrus Logic CLGD5420, CLGD5422, CLGD5424, CLGD5426, CLGD5428
- CLGD6205, CLGD6215, CLGD6225, CLGD6235
- Compaq AVGA
- OAK OTI067, OTI077

The following accelerated chipsets are supported:

- 8514/A (and true clones)
- ATI Mach8, Mach32
- Cirrus CLGD5420, CLGD5422, CLGD5424, CLGD5426, CLGD5428
- S3 86C911, 86C924, 86C801, 86C805, 86C805i, 86C928
- Western Digital WD90C31

All of these, except for the ATI and Cirrus chipsets, are supported in both 256-color or monochrome mode. ATI and Cirrus are supported only in color. There is also a monochrome server that supports generic VGA, Hercules, Sigma LaserView, Hyundai HGC1280, and Visa cards.

This list will undoubtedly expand as time passes. The release notes for the current version of XFree86 should contain the complete list of supported video chipsets.

One problem faced by the XFree86 developers is that some video card manufacturers use non-standard mechanisms for determining clock frequencies used to drive the card. Some of these manufacturers either don't release specifications describing how to program the card, or they require developers to sign a non-disclosure statement to obtain the information. This would obviously restrict the free distribution of the XFree86 software, something that the XFree86 development team is not willing to do. Specifically, the Diamond Speedstar 24/Speedstar+, and Diamond's S3-based cards, are not supported for this reason.

Local bus cards are supported as well. The suggested setup for XFree86 under Linux is a 486 machine with at least 8 megabytes of RAM, and a video card with a chipset listed above. For optimal performance, we suggest using an S3-chipset card. You should check the documentation for XFree86 and verify that your particular card is supported before taking the plunge and purchasing expensive hardware.

I have run XFree86 on a 486/50 MHz machine with 8 megs of RAM, and it's as fast or faster than many color workstations running proprietary versions of UNIX and X. 16 megs of RAM or more is certainly a plus.

You will need at least 4 megabytes of physical RAM, and 16 megabytes of virtual RAM (for example, 8 megs physical and 8 megs swap). Remember that physical RAM is speed! A system with 4 megabytes of physical RAM will run *much* (up to 10 times) more slowly than one with 8 megs or more. Furthermore, swapping is too slow to compensate for lack of physical RAM. It is strongly suggested that you have at least 8 megs of physical RAM.

5.1.2 Installing XFree86

The Linux binary distribution of XFree86 can be found on a number of Linux FTP sites. On `sunsite.unc.edu`, it is found in the directory `/pub/Linux/X11`. (As of the time of this writing, the current version is 2.1.1; newer versions are released periodically). The binary distribution consists of a number of gzipped tar files, all of which unpack from `/`.

It's quite likely that you obtained XFree86 as part of a Linux distribution, in which case downloading the software separately is not necessary.

The XFree86 directory should contain `README` files and installation notes for the current version. Installation is covered in the XFree86-HOWTO; essentially you need to unpack the tar files (as `root`) from `/`.

After unpacking the files, you first need to link the file `/usr/bin/X11/X` to the server that you're using. For example, if you wish to use the SVGA color server, `/usr/bin/X11/X` should be linked to `/usr/bin/X11/XF86_SVGA`. If you wish to use the monochrome server instead, relink this file to `XF86_MONO` with the command

```
# ln -sf /usr/bin/X11/XF86_MONO /usr/bin/X11/X
```

The same holds true if you are using one of the other servers.

5.1.3 Configuring XFree86

Setting up XFree86 is not difficult in most cases. Only when you have non-standard hardware will XFree86 configuration give you any problems. However, XFree86 configuration is beyond the scope of this document; here, we'll give you a brief overview of how it works.

A complete discussion of XFree86 configuration can be found in the file `/usr/lib/X11/etc/README.Config` included with XFree86. Please read this file for complete information on setting up the system. Furthermore, you *must* read the man pages for `Xconfig`, `XFree86`, and the server that you are using (such as `XF86_SVGA`). These man pages describe all of the options that are available for the configuration files.

The main XFree86 configuration file is `/usr/lib/X11/Xconfig`. This file contains information on your mouse, video card parameters, and so on. The file `Xconfig.sample` is provided with the XFree86 distribution as an example. The `XFree86` man page explains the format of this file in detail.

As an overview: Your video card uses a number of "driving clock frequencies" (or "dot clocks") which specify the maximum rates at which the card can send video information to the monitor. Each dot clock has a resolution mode associated with it, such as 640x480 or 1024x768. (Although you are not restrained to using "standard" resolutions, as we will see). In the `Xconfig` file there exist stanzas for configuring your mouse, keyboard, and so on. There also exist stanzas for each server, headed by keywords such as `vga256` for the color SVGA server, and `vga2` for the monochrome server.

Within each server stanza are lines to set the virtual resolution, chipset type, and other parameters for your video card. There is also a `Modes` line which specifies which resolution modes are available. Modes are usually named after their resolution. For example,

```
Modes "640x480" "800x600" "1024x768"
```

Each mode on this line is an index into the `modeDB` stanza at the end of the `Xconfig` file. It is this section of the file which determines the actual video parameters for each mode.

There is also an optional `Clocks` line which you can use to set the available dot clocks for your card. By default, XFree86 will determine the clocks at startup time; however, because clock timing can be thrown off by other programs running on your system, it is usually more accurate to set the clocks explicitly in the `Xconfig` file.

The `modeDB` section of the `Xconfig` file is the important part. Each video card and monitor has its own set of timing and sync frequencies for different resolutions. The file `/usr/lib/X11/etc/modeDB.txt` contains a database of known timing numbers for several types of monitors and video cards. Many cards and monitors use the VESA standard timings included in the sample `Xconfig` file.

There are various other documents in `/usr/lib/X11/etc` which you should read. The file `VideoModes.txt` is a tutorial on hacking your own monitor frequency timings if you simply can't get any of the numbers in `modeDB.txt` to work. There is also a collection of sample `Xconfig` files on `sunsite.unc.edu` in the file `/pub/Linux/X11/Xconfig.tgz`. Also see the `XFree86` man pages for more information.

◇ **Be careful** when setting up your `Xconfig` file. If you select a clock frequency higher (or lower) than your monitor can support, you may very well damage the monitor. Check the documentation for your monitor before hacking your video timings.

If you are not sure what kind of video hardware you have, you can run the program `/usr/bin/X11/SuperProbe`, which is fairly good at detecting video hardware. However, `SuperProbe` is not always correct, so rely on your manuals and common sense before trusting it completely. Also note that `SuperProbe` will detect more hardware than XFree86-2.0 supports.

Again, we refer you to the Linux XFree86-HOWTO for complete information on setting up this beast.

5.1.4 Starting up X

After configuring the `Xconfig` file, you can start the server with the `startx` command. There are a few things to take into consideration first, however.

Make sure that the directory `/usr/bin/X11` is on your path. This directory contains all of the X binaries and the server itself.

Secondly, the X server requires a free VC to enable VC switching[2]. In other words, you must have one of your VC's available, with no `login` process running on it. The easiest way to ensure this is to edit `/etc/inittab` and delete one of the `getty` lines which starts up a login process on each VC. In my `inittab`, for example, I run `getty` on `/dev/tty1` through `/dev/tty7` (that is, VC's 1 through 7), but not on `/dev/tty8`.

When running `startx`, the file `$HOME/.xinitrc` is read. This file is a shell script which contains commands to run after the X server is started. If this file doesn't exist, the file `/usr/lib/X11/xinit/xinitrc` is used as a system-wide default instead. You can use this default file as a sample `.xinitrc` file.

Using X Windows is a large topic, and we won't try to cover it here. Read *The X Window System User's Guide*, or another book on using X, for details. See Appendix A information on this book.

5.1.5 Exiting X

Usually, the last client started in `.xinitrc` is the one used to shutdown X cleanly. For example, if the last command in `.xinitrc` is

```
exec twm
```

then killing the `twm` process will result in X shutting down.

However, if you need to immediately kill the X server for some reason, you can use the key combination `ctrl-alt-backspace`.

[2]While running X, you can switch back to your text VC's using the keys `ctrl-alt-F1` through `ctrl-alt-F12`. To return to X, simply switch to the VC reserved for XFree86.

5.2 Accessing MS-DOS Files

If, for some twisted and bizarre reason, you would have need to access files from MS-DOS, it's quite easily done under Linux.

The usual way to access MS-DOS files is to mount an MS-DOS partition or floppy under Linux, allowing you to access the files directly through the filesystem. For example, if you have an MS-DOS floppy in `/dev/fd0`, the command

```
# mount -t msdos /dev/fd0 /mnt
```

will mount it under `/mnt`. See Section 4.6.2 for more information on mounting floppies.

You can also mount an MS-DOS partition of your hard drive for access under Linux. If you have an MS-DOS partition on `/dev/hda1`, the command

```
# mount -t msdos /dev/hda1 /mnt
```

will mount it. Be sure to `umount` the partition when you're done using it. You can have your MS-DOS partitions automatically mounted at boot time if you include entries for them in `/etc/fstab`; see Section 4.8 for details. For example, the following line in `/etc/fstab` will mount an MS-DOS partition on `/dev/hda1` on the directory `/dos`.

```
/dev/hda1     /dos     msdos     defaults
```

The `Mtools` software may also be used to access MS-DOS files. For example, the commands `mcd`, `mdir`, and `mcopy` all behave as their MS-DOS counterparts. If you installed `Mtools`, there should be man pages available for these commands.

Accessing MS-DOS files is one thing; running MS-DOS programs from Linux is another. There is an MS-DOS Emulator under development for Linux; it is widely available, and even distributed with SLS. It can be retrieved from a number of locations, including the various Linux FTP sites (see Appendix C for details). The MS-DOS Emulator is reportedly powerful enough to run a number of applications, including Wordperfect, from Linux. However, Linux and MS-DOS are vastly different operating systems. The power of any MS-DOS emulator under UNIX is somewhat limited.

In addition, work is underway on a Microsoft Windows emulator to run under X Windows. Watch the newsgroups and FTP sites for more information.

5.3 Networking with TCP/IP

Linux supports a full implementation of the TCP/IP (Transport Control Protocol/Internet Protocol) networking protocols. TCP/IP has become the most successful mechanism for networking computers worldwide. With Linux and an Ethernet card, you can network your machine to a local area network, or (with the proper network connections), to the Internet—the worldwide TCP/IP network.

Hooking up a small LAN of UNIX machines is easy. It simply requires an Ethernet controller in each machine and the appropriate Ethernet cables and other hardware. Or, if your business or university provides access to the Internet, you can easily add your Linux machine to this network.

The current implementation of TCP/IP and related protocols for Linux is called "NET-2". This has no relationship to the so-called NET-2 release of BSD UNIX; instead, "NET-2" in this context means the second implementation of TCP/IP for Linux.

Linux NET-2 also supports SLIP—Serial Line Internet Protocol. SLIP allows you to have dialup Internet access using a modem. If your business or university provides SLIP access, you can dial in to the SLIP server and put your machine on the Internet over the phone line. Alternately, if your Linux machine also has Ethernet access to the Internet, you can set up your Linux box as a SLIP server.

For complete information on setting up TCP/IP under Linux, we encourage you to read the Linux NET-2 HOWTO, available via anonymous FTP from `sunsite.unc.edu`. The NET-2 HOWTO is a complete guide to configuring TCP/IP, including Ethernet and SLIP connections, under Linux. The Linux Ethernet HOWTO is a related document that describes configuration of various Ethernet card drivers for Linux. The *Linux Network Administrator's Guide*, from the Linux Documentation Project, is also available. See Appendix A for more information on these documents.

Also of interest is the book *TCP/IP Network Administration*, by Craig Hunt. It contains complete information on using and configuring TCP/IP on UNIX systems.

5.3.1 Hardware Requirements

You can use Linux TCP/IP without any networking hardware at all—configuring "loopback" mode allows you to talk to yourself. This is necessary for some applications and games which use the "loopback" network device.

However, if you want to use Linux with an Ethernet TCP/IP network, you need one of the following Ethernet cards: 3com 3c503, 3c503/16; Novell NE1000, NE2000; Western Digital WD8003, WD8013; Hewlett Packard HP27245, HP27247, HP27250.

The following clones are reported to work: WD-80x3 clones: LANNET LEC-45; NE2000 clones: Alta Combo, Artisoft LANtastic AE-2, Asante Etherpak 2001/2003, D-Link Ethernet II, LTC E-NET/16 P/N 8300-200-002, Network Solutions HE-203, SVEC 4 Dimension Ethernet, 4-Dimension FD0490 EtherBoard 16, and D-Link DE-600, SMC Elite 16.

See the Linux Ethernet HOWTO for a more complete discussion of Linux Ethernet hardware compatibility.

Linux also supports SLIP, which allows you to use a modem to access the Internet over the phone line. In this case, you'll need a modem compatible with your SLIP server—most servers require a 14.4bps V.32bis modem.

5.3.2 Configuring TCP/IP on your system

In this section we're going to discuss how to configure an Ethernet TCP/IP connection on your system. Note that this method should work for many systems, but certainly not all. This discussion should be enough to get you on the right path to configuring the network parameters of your machine, but there are numerous caveats and fine details not mentioned here. We direct you to the *Linux Network Administrators' Guide* and the NET-2-HOWTO for more information.[3]

First of all, we assume that you have a Linux system that has the TCP/IP software installed. This includes basic clients such as `telnet` and `ftp`, system administration commands such as `ifconfig` and `route` (usually found in `/etc`), and networking configuration files (such as `/etc/hosts`). The other Linux-related networking documents described above explain how to go about installing the Linux networking software if you do not have it already.

We also assume that your kernel has been configured and compiled with TCP/IP support enabled. See Section 4.7 for information on compiling your kernel. To enable networking, you must answer "yes" to the appropriate questions during the `make config` step, and rebuild the kernel.

Once this has been done, you must modify a number of configuration files used by NET-2. For the most part this is a simple procedure. Unfortunately, however, there is wide disagreement between Linux distributions as to where the various TCP/IP configuration files and support programs should go. Much of the time, they can be found in `/etc`, but in other cases may be found in `/usr/etc`, `/usr/etc/inet`, or other bizarre locations. In the worst case you'll have to use the `find` command to locate the files on your system. Also note that not all distributions keep the NET-2 configuration files and software in the same location—they may be spread across several directories.

The following information applies primarily to Ethernet connections. If you're planning to use SLIP, read this section to understand the concepts, and follow the SLIP-specific instructions in the following section.

5.3.2.1 Your network configuration

Before you can configure TCP/IP, you need to determine the following information about your network setup. In most cases, your local network administrator can provide you with this information.

- IP address. This is the unique machine address in dotted-decimal format. An example is 128.253.153.54. Your network admins will provide you with this number.

  If you're only configuring loopback mode (i.e. no SLIP, no ethernet card, just TCP/IP connections to your own machine) then your IP address is 127.0.0.1.

- Your network mask ("netmask"). This is a dotted quad, similar to the IP address, which determines which portion of the IP address specifies the subnetwork number, and which portion specifies the host on that subnet. (If you're shaky on these TCP/IP networking terms, we suggest reading some introductory material on network administration.) The network mask is a pattern of bits, which when overlayed onto an address on your network, will tell you which

[3] Some of this information is adapted from the NET-2-HOWTO by Terry Dawson and Matt Welsh.

subnet that address lives on. This is very important for routing, and if you find, for example, that you can happily talk to people outside your network, but not to some people within your network, there is a good chance that you have an incorrect mask specified.

Your network administrators will have chosen the netmask when the network was designed, and therefore they should be able to supply you with the correct mask to use. Most networks are class C subnetworks which use 255.255.255.0 as their netmask. Other Class B networks use 255.255.0.0. The NET-2 code will automatically select a mask that assumes no subnetting as a default if you do not specify one.

This applies as well to the loopback port. Since the loopback port's address is always 127.0.0.1, the netmask for this port is always 255.0.0.0. You can either specify this explicitly or rely on the default mask.

- Your network address. This is your IP address masked bitwise-ANDed the netmask. For example, if your netmask is 255.255.255.0, and your IP address is 128.253.154.32, your network address is 128.253.154.0. With a netmask of 255.255.0.0, this would be 128.253.0.0.

  If you're only using loopback, you don't have a network address.

- Your broadcast address. The broadcast address is used to broadcast packets to every machine on your subnet. Therefore, if the host number of machines on your subnet is given by the last byte of the IP address (netmask 255.255.255.0), your broadcast address will be your network address ORed with 0.0.0.255.

  For example, if your IP address is 128.253.154.32, and your netmask is 255.255.255.0, your broadcast address is 128.253.154.255.

  Note that for historical reasons, some networks are setup to use the network address as the broadcast address, if you have any doubt, check with your network administrators. (In many cases, it will suffice to duplicate the network configuration of other machines on your subnet, substituting your own IP address, of course.)

  If you're only using loopback, you don't have a broadcast address.

- Your gateway address. This is the address of the machine which is your "gateway" to the outside world (i.e. machines not on your subnet). In many cases the gateway machine has an IP address identical to yours but with a ".1" as its host address; e.g., if your IP address is 128.253.154.32, your gateway might be 128.253.154.1. Your network admins will provide you with the IP address of your gateway.

  In fact, you may have multiple gateways. A *gateway* is simply a machine that lives on two different networks (has IP addresses on different subnets), and routes packets between them. Many networks have a single gateway to "the outside world" (the network directly adjacent to your own), but in some cases you will have multiple gateways—one for each adjacent network.

  If you're only using loopback, you don't have a gateway address. The same is true if your network is isolated from all others.

- Your nameserver address. Most machines on the net have a name server which translates hostnames into IP addresses for them. Your network admins will tell you the address of your name server. You can also run a server on your own machine by running `named`, in which case the nameserver address is 127.0.0.1. Unless you absolutely *must* run your own name server, we suggest using the one provided to you on the network (if any). Configuration of `named` is another issue altogether; our priority at this point is to get you talking to the network. You can deal with name resolution issues later.

If you're only using loopback, you don't have a nameserver address.

SLIP users: You may or may not require any of the above information, except for a nameserver address. When using SLIP, your IP address is usually determined in one of two ways: Either (a) you have a "static" IP address, which is the same every time you connect to the network, or (b) you have a "dynamic" IP address, which is allocated from a pool available addresses when you connect to the server. In the following section on SLIP configuration this is covered in more detail.

NET-2 supports full routing, multiple routes, subnetworking (at this stage on byte boundaries only), the whole nine yards. The above describes most basic TCP/IP configurations. Yours may be quite different: when in doubt, consult your local network gurus and check out the man pages for `route` and `ifconfig`. Configuring TCP/IP networks is very much beyond the scope of this book; the above should be enough to get most people started.

5.3.2.2 The networking `rc` files

`rc` files are systemwide configuration scripts executed at boot time by `init`, which start up all of the basic system daemons (such as `sendmail`, `cron`, etc.) and configure things such as the network parameters, system hostname, and so on. `rc` files are usually found in the directory `/etc/rc.d` but on other systems may be in `/etc`.

Here, we're going to describe the `rc` files used to configure TCP/IP. There are two of them: `rc.inet1` and `rc.inet2`. `rc.inet1` is used to configure the basic network parameters (such as IP addresses and routing information) and `rc.inet2` fires up the TCP/IP daemons (`telnetd`, `ftpd`, and so forth).

Many systems combine these two files into one, usually called `rc.inet` or `rc.net`. The names given to your `rc` files doesn't matter, as long as they perform the correct functions and are executed at boot time by `init`. To ensure this, you may need to edit `/etc/inittab` and uncomment lines to execute the appropriate `rc` file(s). In the worst case you will have to create the `rc.inet1` and `rc.inet2` files from scratch and add entries for them to `/etc/inittab`.

As we said, `rc.inet1` configures the basic network interface. This includes your IP and network address, and the routing table information for your network. The routing tables are used to route outgoing (and incoming) network datagrams to other machines. On most simple configurations, you have three routes: One for sending packets to your own machine, another for sending packets to other machines on your network, and another for sending packets to machines outside of your network (through the gateway machine). Two programs are used to configure these parameters: `ifconfig` and `route`. Both of these are usually found in `/etc`.

`ifconfig` is used for configuring the network device interface with the parameters that it requires to function, such as the IP address, network mask, broadcast address and the like. `route` is used to create and modify entries in the routing table.

For most configurations, an `rc.inet1` file that looks like the following should work. You will, of course, have to edit this for your own system. Do *not* use the sample IP and network addresses listed here for your own system; they correspond to an actual machine on the Internet.

```
#!/bin/sh
# This is /etc/rc.d/rc.inet1 -- Configure the TCP/IP interfaces

# First, configure the loopback device

HOSTNAME=`hostname`

/etc/ifconfig lo 127.0.0.1      # uses default netmask 255.0.0.0
/etc/route add 127.0.0.1        # a route to point to the loopback device

# Next, configure the ethernet device. If you're only using loopback or
# SLIP, comment out the rest of these lines.

# Edit for your setup.
IPADDR="128.253.154.32"         # REPLACE with YOUR IP address
NETMASK="255.255.255.0"         # REPLACE with YOUR netmask
NETWORK="128.253.154.0"         # REPLACE with YOUR network address
BROADCAST="128.253.154.255"     # REPLACE with YOUR broadcast address, if you
                                # have one. If not, leave blank and edit below.
GATEWAY="128.253.154.1"         # REPLACE with YOUR gateway address!

/etc/ifconfig eth0 ${IPADDR} netmask ${NETMASK} broadcast ${BROADCAST}

# If you don't have a broadcast address, change the above line to just:
# /etc/ifconfig eth0 ${IPADDR} netmask ${NETMASK}

/etc/route add ${NETWORK}

# The following is only necessary if you have a gateway; that is, your
# network is connected to the outside world.
/etc/route add default gw ${GATEWAY} metric 1

# End of Ethernet Configuration
```

Again, you may have to tweak this file somewhat to get it to work. The above should be sufficient for the majority of simple network configurations, but certainly not all.

`rc.inet2` starts up various servers used by the TCP/IP suite. The most important of these is `inetd`. `inetd` sits in the background and listens to various network ports. When a machine tries to make a connection to a certain port (for example, the incoming `telnet` port), `inetd` forks off a copy of the appropriate daemon for that port (in the case of the `telnet` port, `inetd` starts `in.telnetd`). This is simpler than running many separate, standalone daemons (e.g., individual copies of `telnetd`, `ftpd`, and so forth)—`inetd` starts up the daemons only when they are needed.

`syslogd` is the system logging daemon—it accumulates log messages from various applications and stores them into log files based on the configuration information in `/etc/syslogd.conf`. `routed` is a server used to maintain dynamic routing information. When your system attempts to send packets to another network, it may require additional routing table entries in order to do so. `routed` takes care of manipulating the routing table without the need for user intervention.

Our example `rc.inet2`, below, only starts up the bare minimum of servers. There are many other servers as well—many of which have to do with NFS configuration. When attempting to setup TCP/IP on your system, it's usually best to start with a minimal configuration and add more complex pieces (such as NFS) when you have things working.

Note that in the below file, we assume that all of the network daemons are held in `/etc`. As usual, edit this for your own configuration.

```
#! /bin/sh
# Sample /etc/rc.d/rc.inet2

# Start syslogd
if [ -f /etc/syslogd ]
then
      /etc/syslogd
fi

# Start inetd
if [ -f /etc/inetd ]
then
      /etc/inetd
fi

# Start routed
if [ -f /etc/routed ]
then
      /etc/routed -q
fi

# Done!
```

Among the various additional servers that you may want to start in `rc.inet2` is `named`. `named` is a name server—it is responsible for translating (local) IP addresses to names, and vice versa. If you don't have a nameserver elsewhere on the network, or want to provide local machine names to other machines in your domain, it may be necessary to run `named`. (For most configurations it is not necessary, however.) `named` configuration is somewhat complex and requires planning; we refer interested readers to a good book on TCP/IP network administration.

5.3.2.3 `/etc/hosts`

`/etc/hosts` contains a list of IP addresses and the hostnames that they correspond to. In general,

`/etc/hosts` only contains entries for your local machine, and perhaps other "important" machines (such as your nameserver or gateway). Your local name server will provide address-to-name mappings for other machines on the network, transparently.

For example, if your machine is `loomer.vpizza.com` with the IP address 128.253.154.32, your `/etc/hosts` would look like:

```
127.0.0.1              localhost
128.253.154.32         loomer.vpizza.com loomer
```

If you're only using loopback, the only line in `/etc/hosts` should be for 127.0.0.1, with both `localhost` and your hostname after it.

5.3.2.4 /etc/networks

The `/etc/networks` file lists the names and addresses of your own, and other, networks. It is used by the `route` command, and allows you to specify a network by name, should you so desire.

Every network you wish to add a route to using the `route` command (generally called from `rc.inet1`—see above) *must* have an entry in `/etc/networks`.

As an example,

```
default 0.0.0.0 # default route    - mandatory
loopnet 127.0.0.0 # loopback network - mandatory
mynet 128.253.154.0 # Modify for your own network address
```

5.3.2.5 /etc/host.conf

This file is used to specify how your system will resolve hostnames. It should contain the two lines:

```
order hosts,bind
multi on
```

These lines tell the resolve libraries to first check the `/etc/hosts` file for any names to lookup, and then ask the nameserver (if one is present). The `multi` entry allows you to have multiple IP addresses for a given machine name in `/etc/hosts`.

5.3.2.6 /etc/resolv.conf

This file configures the name resolver, specifying the address of your name server (if any) and your domain name. Your domain name is your fully-qualified hostname (if you're a registered machine on the Internet, for example), with the hostname chopped off. That is, if your full hostname is `loomer.vpizza.com`, your domain name is just `vpizza.com`.

For example, if your machine is `goober.norelco.com`, and has a nameserver at the address 128.253.154.5, your `/etc/resolv.conf` would look like:

```
domain      norelco.com
nameserver  127.253.154.5
```

You can specify more than one nameserver—each must have a `nameserver` line of its own in `resolv.conf`.

5.3.2.7 Setting your hostname

You should set your system hostname with the `hostname` command. This is usually called from `/etc/rc` or `/etc/rc.local`; simply search your system `rc` files to determine where it is invoked. For example, if your (full) hostname is `loomer.vpizza.com`, edit the appropriate `rc` file to execute the command:

```
/bin/hostname loomer.vpizza.com
```

Note that the `hostname` executable may not be found in `/bin` on your system.

5.3.2.8 Trying it out

Once you have all of these files set up, you should be able to reboot your new kernel and attempt to use the network. There are many places where things can go wrong, so it's a good idea to test individual aspects of the network configuration (e.g., it's probably not a good idea to test your network configuration by firing up Mosaic over a network-based X connection).

You can use the `netstat` command to display your routing tables; this is usually the source of the most trouble. The `netstat` man page describes the exact syntax of this command in detail. In order to test network connectivity, we suggest using a client such as `telnet` to connect to machines both on your local subnetwork and external networks. This will help to narrow down the source of the problem. (For example, if you're unable to connect to local machines, but can connect to machines on other networks, more than likely there is a problem with your netmask and routing table configuration). You can also invoke the `route` command directly (as `root`) to play with the entries in your routing table.

You should also test network connectivity by specifying IP addresses directly, instead of hostnames. For example, if you have problems with the command

```
$ telnet shoop.vpizza.com
```

the cause may be incorrect nameserver configuration. Try using the actual IP address of the machine in question; if that works, then you know that your basic network setup is (more than likely) correct, and the problem lies in your specification of the name server address.

Debugging network configurations can be a difficult task, and we can't begin to cover it here. If you are unable to get help from a local guru we strongly suggest reading the *Linux Network Administrators' Guide* from the LDP.

5.3.3 SLIP Configuration

SLIP (Serial Line Internet Protocol) allows you to use TCP/IP over a serial line, be that a phone line, with a dialup modem, or a leased asynchronous line of some sort. Of course, to use SLIP you'll need access to a dial-in SLIP server in your area. Many universities and businesses provide SLIP access for a modest fee.

There are two major SLIP-related programs available—`dip` and `slattach`. Both of these programs are used to initiate a SLIP connection over a serial device. It is *necessary* to use one of these programs in order to enable SLIP—it will not suffice to dial up the SLIP server (with a communications program such as `kermit`) and issue `ifconfig` and `route` commands. This is because `dip` and `slattach` issue a special *ioctl()* system call to seize control of the serial device to be used as a SLIP interface.

`dip` can be used to dial up a SLIP server, do some handshaking to login to the server (exchanging your username and password, for example) and then initate the SLIP connection over the open serial line. `slattach`, on the other hand, does very little other than grab the serial device for use by SLIP. It is useful if you have a permanent line to your SLIP server and no modem dialup or handshaking is necessary to initiate the connection. Most dialup SLIP users should use `dip`, on the other hand.

`dip` can also be used to configure your Linux system as a SLIP server, where other machines can dial into your own and connect to the network through a secondary Ethernet connection on your machine. See the documentation and man pages for `dip` for more information on this procedure.

SLIP is quite unlike Ethernet, in that there are only two machines on the "network"—the SLIP host (that's you) and the SLIP server. For this reason, SLIP is often referred to as a "point-to-point" connection. A generalization of this idea, known as PPP (Point to Point Protocol) has also been implemented for Linux.

When you initiate a connection to a SLIP server, the SLIP server will give you an IP address based on (usually) one of two methods. Some SLIP servers allocate "static" IP addresses—in which case your IP address will be the same every time you connect to the server. However, many SLIP servers allocate IP addresses dynamically—in which case you receive a different IP address each time you connect. In general, the SLIP server will print the values of your IP and gateway addresses when you connect. `dip` is capable of reading these values from the output of the SLIP server login session and using them to configure the SLIP device.

Essentially, configuring a SLIP connection is just like configuring for loopback or ethernet. The main differences are discussed below. Read the previous section on configuring the basic TCP/IP files, and apply the changes described below.

5.3.3.1 Static IP address SLIP connections using `dip`

If you are using a static-allocation SLIP server, you may want to include entries for your IP address and hostname in `/etc/hosts`. Also, configure these files listed in the above section: `rc.inet2`, `host.conf`, and `resolv.conf`.

Also, configure `rc.inet1`, as described above. However, only want to execute `ifconfig` and route commands for the loopback device. If you use dip to connect to the SLIP server, it will execute the appropriate ifconfig and route commands for the SLIP device for you. (If you're using slattach, on the other hand, you *will* need to include ifconfig/route commands in rc.inet1 for the SLIP device—see below.)

dip *should* configure your routing tables appropriately for the SLIP connection when you connect. In some cases, however, dip's behavior may not be correct for your configuration, and you'll have to run ifconfig or route commands by hand after connecting to the server with dip (this is most easily done from within a shell script that runs dip and immediately executes the appropriate configuration commands). Your gateway is, in most cases, the address of the SLIP server. You may know this address before hand, or the gateway address will be printed by the SLIP server when you connect. Your dip chat script (described below) can obtain this information from the SLIP server.

ifconfig may require use of the pointopoint argument, if dip doesn't configure the interface correctly. For example, if your SLIP server address is 128.253.154.2, and your IP address is 128.253.154.32, you may need to run the command

```
ifconfig sl0 128.253.154.32 pointopoint 128.253.154.2
```

as root, after connecting with dip. The man pages for ifconfig will come in handy.

Note that SLIP device names used with the ifconfig and route commands are sl0, sl1 and so on (as opposed to eth0, eth1, etc. for Ethernet devices).

In Section 5.3.4, below, we explain how to configure dip to connect to the SLIP server.

5.3.3.2 Static IP address SLIP connections using slattach

If you have a leased line or cable running directly to your SLIP server, then there is no need to use dip to initiate a connection. slattach can be used to configure the SLIP device instead.

In this case, your /etc/rc.inet1 file should look something like the following:

```
#!/bin/sh
IPADDR="128.253.154.32"        # Replace with your IP address
REMADDR="128.253.154.2" # Replace with your SLIP server address

# Modify the following for the appropriate serial device for the SLIP
# connection:
slattach -p cslip -s 19200 /dev/ttyS0
/etc/ifconfig sl0 $IPADDR pointopoint $REMADDR up
/etc/route add default gw $REMADDR
```

slattach allocates the first unallocated SLIP device (sl0, sl1, etc.) to the serial line specified.

Note that the first parameter to slattach is the SLIP protocol to use. At present the only valid values are slip and cslip. slip is regular SLIP, as you would expect, and cslip is SLIP with datagram header compression. In most cases you should use cslip; however, if you seem to be having problems with this, try slip.

If you have more than one SLIP interface then you will have routing considerations to make. You will have to decide what routes to add, and those decisions can only be made on the basis of the actual layout of your network connections. A book on TCP/IP network configuration, as well as the man pages to `route`, will be of use.

5.3.3.3 Dynamic IP address SLIP connections using `dip`

If your SLIP server allocates an IP address dynamically, then you certainly don't know your address in advance—therefore, you can't include an entry for it in `/etc/hosts`. (You should, however, include an entry for your host with the loopback address, 127.0.0.1.)

Many SLIP servers print your IP address (as well as the server's address) when you connect. For example, one type of SLIP server prints a string such as,

```
Your IP address is 128.253.154.44.
Server address is 128.253.154.2.
```

`dip` can capture these numbers from the output of the server and use them to configure the SLIP device.

See Section 5.3.3.1, above, for information on configuring your various TCP/IP files for use with SLIP. Below, we explain how to configure `dip` to connect to the SLIP server.

5.3.4 Using `dip`

`dip` can simplify the process of connecting to a SLIP server, logging in, and configuring the SLIP device. Unless you have a leased line running to your SLIP server, `dip` is the way to go.

To use `dip`, you'll need to write a "chat script" which contains a list of commands used to communicate with the SLIP server at login time. These commands can automatically send your username/password to the server, as well as get information on your IP address from the server.

Here is an example `dip` chat script, for use with a dynamic IP address server. For static servers, you will need to set the variables `$local` and `$remote` to the values of your local IP address and server IP address, respectively, at the top of the script. See the `dip` man page for details.

```
main:
  # Set Maximum Transfer Unit. This is the maximum size of packets
  # transmitted on the SLIP device. Many SLIP servers use either 1500 or
  # 1006; check with your network admins when in doubt.
  get $mtu 1500

  # Make the SLIP route the default route on your system.
  default

  # Set the desired serial port and speed.
  port cua03
```

```
  speed 38400

  # Reset the modem and terminal line. If this causes trouble for you,
  # comment it out.
  reset

  # Prepare for dialing. Replace the following with your
  # modem initialization string.
  send ATT&C1&D2\\N3&Q5%M3%C1N1W1L1S48=7\r
  wait OK 2
  if $errlvl != 0 goto error
  # Dial the SLIP server
  dial 2546000
  if $errlvl != 0 goto error
  wait CONNECT 60
  if $errlvl != 0 goto error

  # We are connected.  Login to the system.
login:
  sleep 3
  send \r\n\r\n
  # Wait for the login prompt
  wait login: 10
  if $errlvl != 0 goto error

  # Send your username
  send USERNAME\n

  # Wait for password prompt
  wait ord: 5
  if $errlvl != 0 goto error

  # Send password.
  send PASSWORD\n

  # Wait for SLIP server ready prompt
  wait annex: 30
  if $errlvl != 0 goto error

  # Send commands to SLIP server to initate connection.
  send slip\n
  wait Annex 30

  # Get the remote IP address from the SLIP server. The ‘get...remote’
  # command reads text in the form xxx.xxx.xxx.xxx, and assigns it
  # to the variable given as the second argument (here, $remote).
  get $remote remote
  if $errlvl != 0 goto error
```

```
  wait Your 30

  # Get local IP address from SLIP server, assign to variable $local.
  get $local remote
  if $errlvl != 0 goto error

  # Fire up the SLIP connection
done:
  print CONNECTED to $remote at $rmtip
  print GATEWAY address $rmtip
  print LOCAL address $local
  mode SLIP
  goto exit
error:
  print SLIP to $remote failed.

exit:
```

`dip` automatically executes `ifconfig` and `route` commands based on the values of the variables `$local` and `$remote`. Here, those variables are assigned using the `get`...`remote` command, which obtains text from the SLIP server and assigns it to the named variable.

If the `ifconfig` and `route` commands that `dip` runs for you don't work, you can either run the correct commands in a shell script after executing `dip`, or modify the source for `dip` itself. Running `dip` with the `-v` option will print debugging information while the connection is being set up, which should help you to determine where things might be going awry.

Now, in order to run `dip` and open the SLIP connection, you can use a command such as:

```
/etc/dip/dip -v /etc/dip/mychat 2>&1
```

Where the various `dip` files, and the chat script (`mychat.dip`), are stored in `/etc/dip`.

The above discussion should be enough to get you well on your way to talking to the network, either via Ethernet or SLIP. Again, we strongly suggest looking into a book on TCP/IP network configuration, especially if your network has any special routing considerations, other than those mentioned here.

5.4 Networking with UUCP

UUCP (UNIX-to-UNIX Copy) is an older mechanism used to transfer information between UNIX systems. Using UUCP, UNIX systems dial each other up (using a modem) and transfer mail messages, news articles, files, and so on. If you don't have TCP/IP or SLIP access, you can use UUCP to communicate with the world. Most of the mail and news software (see Sections 5.5 and 5.6) can be configured to use UUCP to transfer information to other machines. In fact, if there is an Internet site nearby, you can arrange to have Internet mail sent to your Linux machine via UUCP from that site.

The *Linux Network Administrator's Guide* contains complete information on configuring and using UUCP under Linux. Also, the Linux UUCP HOWTO, available via anonymous FTP from `sunsite.unc.edu`, should be of help. Another source of information on UUCP is the book *Managing UUCP and USENET*, by Tim O'Reilly and Grace Todino. See Appendix A for more information.

5.5 Electronic Mail

Like most UNIX systems, Linux provides a number of software packages for using electronic mail. E-mail on your system can either be local (that is, you only mail other users on your system), or networked (that is, you mail, using either TCP/IP or UUCP, users on other machines on a network). E-mail software usually consists of two parts: a *mailer* and a *transport*. The mailer is the user-level software which is used to actually compose and read e-mail messages. Popular mailers include `elm` and `mailx`. The transport is the low-level software which actually takes care of delivering the mail, either locally or remotely. The user never sees the transport software; they only interact with the mailer. However, as the system administrator, it is important to understand the concepts behind the transport software and how to configure it.

The most popular transport software for Linux is `Smail`. This software is easy to configure, and is able to send both local and remote TCP/IP e-mail. The more powerful `sendmail` transport is used on most UNIX systems, however, because of its complicated setup mechanism, most Linux systems don't use it.

The Linux Mail HOWTO gives more information on the available mail software for Linux and how to configure it on your system. If you plan to send mail remotely, you'll need to understand either TCP/IP or UUCP, depending on how your machine is networked (see Sections 5.3 and 5.4). The UUCP and TCP/IP documents listed in Appendix A should be of help there.

Most of the Linux mail software can be retrieved via anonymous FTP from `sunsite.unc.edu` in the directory `/pub/Linux/system/Mail`.

5.6 News and USENET

Linux also provides a number of facilities for managing electronic news. You may choose to set up a local news server on your system, which will allow users to post "articles" to various "newsgroups" on the system... a lively form of discussion. However, if you have access to a TCP/IP or UUCP network, then you will be able to participate in USENET—a worldwide network news service.

There are two parts to the news software—the *server* and the *client*. The news server is the software which controls the newsgroups and handles delivering articles to other machines (if you are on a network). The news client, or *newsreader*, is the software which connects to the server to allow users to read and post news.

There are several forms of news servers available for Linux. They all follow the same basic protocols and design. The two primary versions are "C News" and "INN". There are many types of newsreaders, as well, such as `rn` and `tin`. The choice of newsreader is more or less a matter of taste; all newsreaders should work equally well with different versions of the server software. That is, the newsreader is independent of the server software, and vice versa.

If you only want to run news locally (that is, not as part of USENET), then you will need to run a server on your system, as well as install a newsreader for the users. The news server will store the articles in a directory such as `/usr/spool/news`, and the newsreader will be compiled to look in this directory for news articles.

However, if you wish to run news over the network, there are several options open to you. TCP/IP network-based news uses a protocol known as NNTP (Network News Transmission Protocol). NNTP allows a newsreader to read news over the network, on a remote machine. NNTP also allows news servers to send articles to each other over the network—this is the software upon which USENET is based. Most businesses and universities have one or more NNTP servers set up to handle all of the USENET news for that site. Every other machine at the site runs an NNTP-based newsreader to read and post news over the network via the NNTP server. This means that only the NNTP server actually stores the news articles on disk.

Here are some possible scenarios for news configuration.

- You run news locally. That is, you have no network connection, or no desire to run news over the network. In this case, you need to run C News or INN on your machine, and install a newsreader to read the news locally.
- You have access to a TCP/IP network and an NNTP server. If your organization has an NNTP news server set up, you can read and post news from your Linux machine by simply installing an NNTP-based newsreader. (Most newsreaders available can be configured to run locally or use NNTP). In this case, you do not need to install a news server or store news articles on your system. The newsreader will take care of reading and posting news over the network. Of course, you will need to have TCP/IP configured and have access to the network (see Section 5.3).
- You have access to a TCP/IP network but have no NNTP server. In this case, you can run an NNTP news server on your Linux system. You can install either a local or an NNTP-based newsreader, and the server will store news articles on your system. In addition, you can configure the server to communicate with other NNTP news servers to transfer news articles.
- You want to transfer news using UUCP. If you have UUCP access (see Section 5.4), you can participate in USENET as well. You will need to install a (local) news server and a news reader. In addition, you will need to configure your UUCP software to periodically transfer news articles to another nearby UUCP machine (known as your "news feed"). UUCP does not use NNTP to transfer news; simply, UUCP provides its own mechanism for transferring news articles.

The one downside of most news server and newsreader software is that it must be compiled by hand. Most of the news software does not use configuration files; instead, configuration options are determined at compile time.

Most of the "standard" news software (available via anonymous FTP from `ftp.uu.net` in the directory `/news`) will compile out-of-the box on Linux. Necessary patches can be found on `sunsite.unc.edu` in `/pub/Linux/system/Mail` (which is, incidentally, also where mail software for Linux is found). Other news binaries for Linux may be found in this directory as well.

For more information, refer to the Linux News HOWTO from `sunsite.unc.edu` in `/pub/Linux/docs/HOWTO`. Also, the LDP's *Linux Network Administrator's Guide* contains complete information on configuring news software for Linux. The book *Managing UUCP and Usenet*, by Tim O'Reilly and Grace Todino, is an excellent guide to setting up UUCP and news software. Also of interest is the USENET document "How to become a USENET site," available from `ftp.uu.net`, in the directory `/usenet/news.announce.newusers`.