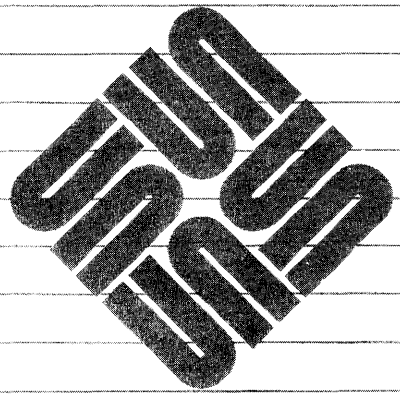
sun microsystems

System Administration *for the* Sun Workstation®



Sun Microsystems, Inc. • 2550 Garcia Avenue • Mountain View, CA 94043 • 415-960-1300

Part No: 800-1323-03
Revision A of 27 January 1989

Contents

Tables

Figures

System Administration *for the* Sun Workstation

1

Introduction And General Advice

1

Introduction And General Advice

This *System Administration Manual* explains many different procedures and types of maintenance that need to be done to keep your Sun Workstation and your Sun network up and running smoothly. It also tells you how to change the size or configuration of your system, and how to expand your system, when those are your goals. In addition, it tells you how to restore a system to its former state when disaster occurs (as it unfortunately does sometimes on all operating systems). Many of the procedures covered in this manual can and should be done on every system, even the least active standalone system. However, there are things we cover that you may never need to do at your particular site. Use the manual as a sort of almanac for consultation and verification; it gives exact procedures for many aspects of system administration.

The manual often refers you to some other manual in the Sun user documentation. Wherever possible we try to describe entire procedures and give complete examples, however it is not possible to simply reproduce all the information that is given in other sections of Sun's documentation. As you gain familiarity with this manual and with the Sun system, you will more and more be able to rely on the terse explanations that cover some long processes.

Always read sections completely before jumping in

Always read through an entire section when you are doing any procedure for the first time. Never try to leap ahead of the given instructions unless you are absolutely certain of your moves, and willing to suffer any mistakes you make. The machine has no patience, but it is not in any hurry either.

Developing Your Administration Procedures

As system administrator, you can make your work easier and get problems fixed more quickly by following the suggestions below; most of them are simple common sense. Remember, each UNIX† site should develop administrative procedures and standards to fit its situation — we encourage you to do this, it helps prevent rash action in a puzzling or unexpected situation. There are several good introductory UNIX books on the market. If you are a novice get one of these to read and keep around the machine room. Here is our introductory checklist for new systems — it covers the most obvious things. You may add your own procedures and ideas to it.

† UNIX is a trademark of AT&T Bell Laboratories.

2

Sun Network Services

2

Sun Network Services

2.1. Introduction And Terminology

This chapter introduces the Sun Network services. We describe the services currently available, and define some terms in the network environment.

Following that, we introduce and explain each of the three types of service now available for the Sun UNIX workstation: network disk service, network file system service, and yellow pages service. Within each of the three sections you will find information about periodic maintenance and trouble-shooting for the service under discussion.

While some of this material tends to be theoretical, its specific implications will be seen again and again as you become familiar with system administration. For example, if you run the yellow pages, you must understand that some typical UNIX procedures have changed in the yellow pages environment. That is also true if you use the network disk or the network file system. This chapter covers only those aspects of network services necessary for performing the duties of system administration. For a complete theoretical overview, see *The Network Services Guide.*

At the end of this chapter we explain how to add new users and new client machines at your site, one of the first and most important duties of system administration.

Networking Models

There are many ways to make computers and networks interface transparently. The two major ones are the distributed operating system approach and the network services approach.

A distributed operating system allows the network software designer to make grand assumptions about the other machines on the network. Usually two of these assumptions are: that the remote piece of hardware is identical to the local hardware, and that the remote and local machines are running identical software. These assumptions allow a quick and simple implementation of a network system in an environment limited to specific hardware and software. This type of distributed operating system is, by design, closed. That is to say, it's very difficult to integrate new hardware or software into a closed network environment, unless it comes from the vendor of that network system. A closed network system forces a customer to return to one vendor for solutions to all computing needs.

1) UNIX was never designed to yield to a higher authority (like a network authentication server) for critical information or services. As a result some UNIX semantics are hard to maintain 'over the net.' For example, trusting user id 0 (root) is not always a good idea.

2) Some UNIX execution semantics are difficult. For example, UNIX allows a user to remove an open file, yet the file does not disappear until closed by everyone. In a network environment a client UNIX machine may not own an open file. Therefore, a server may remove a client's open file.

3) When a UNIX machine crashes, it takes all its applications down with it. When a network node crashes (whether client or server), it should not drag all of its bound neighbors down. The treatment of node failure on a network raises difficulties in any system and is especially difficult in the UNIX environment. Sun has implemented a system of 'stateless' protocols to circumvent the problem of a crashing server dragging down its bound clients. Stateless here means that a client is independently responsible for completing work, and that a server need not remember anything from one call to the next. In other words, the server keeps no state. With no state left on the server, there is no state to recover when the server crashes and comes back up. So, from the client's point of view, a crashed server appears no different than a very slow server.

In implementing UNIX over the network, Sun attempted to remain compatible with UNIX whenever possible. However, certain incompatibilities have been introduced; they are typically of two kinds. First, those issues that would make a networked UNIX evolve into a distributed operating system, rather than a collection of network services. And second, those issues that would make crash recovery extremely difficult from both the implementation and administration point of view.

All incompatibilities are documented in the appropriate sections of this administration manual.

A Hint About Debugging UNIX In The Network Environment

When you cannot get something done that involves Sun network services, the problem probably lies in one of the following four areas. They are listed here with the most likely problem first.

1) The Sun network access control policies do not allow the operation, or architectural constraints prevent the operation.

2) The client software or environment is broken.

3) The server software or environment is broken.

4) The network is broken.

The following sections present specific instructions on how to check for these causes of failure in the `nd`, NFS, and `yp` environments.

```
#       venus
clear
version 1
#
user 0 0 /dev/xy0g 0 11960 -1
user bill 0 /dev/xy0g 11960 10120 0
user bill 1 /dev/xy0g 22080 20240 -1
user debby 0 /dev/xy0g 42320 10120 1
user debby 1 /dev/xy0g 52440 20240 -1
#The following are on an entirely separate
#device, /dev/xy2h
user joan 0 /dev/xy2h 0 10120 2
user joan 1 /dev/xy2h 10120 20240 -1
user mike 0 /dev/xy2h 30360 10120 3
user mike 1 /dev/xy2h 40480 20240 -1
son
```

The commands and their variables are explained below.

- **user** *client_name unit# device startblk nblks ndl#*

Where the variables have the following meaning:

client_name — The name of the client owning this partition. Incoming requests from *client_name* at *unit#* are transformed into server device *device* at the location addressed by *startblk* and *nblks*. If *client_name* is '0', this is a public partition. The special clientname 'localhost' is conventionally used for spare or locally used nd partitions.

unit# — The unit number of the partition. When a *client_name* is present, a '0' (zero) refers to the client's root partition, and a '1' refers to the client's swap partition. When the *client_name* is a '0', *unit#* refers to a public unit.

device — The `/dev` entry corresponding to the hard partition that this soft partition is on. Typically, this is `/dev/xy0g` on a Xylogics SMD controller, or `/dev/sd0g` on an Adaptec SCSI controller.

startblk — The offset from the beginning of *device* to the beginning of this soft partition. The offset is in 512 byte units.

nblks — The size of this partition in 512 byte units. If *nblks* is '-1' then this *unit#* is equivalent to the entire partition *device*; no soft partitioning of the hard partition is done.

ndl# — The `/dev/ndl*` entry corresponding to this partition, where '*' corresponds to `ndl#` on this line. There is no corresponding `/dev/ndl*` entry when `ndl#` is '-1'; this is usually the case with a swap partition or a soft partition which starts at the beginning of a hard partition — like the public partition

```
# mkdir /pub
```

4) Add the following lines to the end of /etc/rc.local:

```
/etc/nd serverat venus
/etc/mount -r /dev/ndp0 /pub &
```

Note that the mount command must specify the -r (read-only) when a client mounts a public partition from a server.

Next time omar reboots it will mount venus's public partition **0** (zero) on /pub. The mount command is run in the background so that omar can still be booted if venus is down. However, if venus goes down while omar is using files in /pub, processes on omar may hang until venus comes up again.

The Server's View Of Partitions

The public partition is the first soft partition defined by /etc/nd on the hard partition g. It is referenced by the name of the hard partition, typically /dev/xy0g. Following the public partition are the clients' root and swap partitions. The client root partitions are referenced by /dev/nd1*, where the '*' is an integer number. The appropriate device for each client is determined by the nd1# field in /etc/nd.local. For example, looking again at a sample /etc/nd.local file entry, we see the root partition for joan is /dev/nd12 (network disk local two).

```
user joan 0 /dev/xy2h 0 10120 2
user joan 1 /dev/xy2h 10120 20240 -1
```

The clients' swap partitions cannot ordinarily be referenced from the server.

See below for discussion of how to use client's root and, in special cases, swap partitions from the server.

The Client's View Of Partitions

From a client, the public partition is referenced through /dev/ndp0 (network disk public zero). The root partition is referenced through /dev/nd0 (network disk zero). The swap partition is referenced through /dev/nd1. The swap partition does not have a file system on it and should never be mounted.

Using Clients' Root Partitions from the Server

A server can access the file system on a client's root partition in the following way. First, halt the client; this **must** be done or you risk corrupting the file system. On the server, look in the /etc/nd.local file for the client's nd1 number. Now mount the desired partition with the following command:

```
# /etc/mount /dev/nd11 /mnt
```

If this were done on the system described in the sample file above, it would mount the root partition of client debby on the /mnt directory of the server. Mounting in this way would then permit access to and modification of debby's

The values for current disks with a Xylogics 450 controller are:

Table 2-1 *Sectors/Track With Xylogics Controller*

Disk Type	Sectors	Heads
D84	32	7
D168	32	10
D169	32	10
Eagle	46	20

The values for current disks with an Adaptec SCSI controller are:

Table 2-2 *Sectors/Track With Adaptec Controller*

Disk Type	Sectors	Heads
Micropolis	17	6

To get these values for any disk or controller use the `/etc/dkinfo` command. Use the proper abbreviation for controller type — `sd` for Adaptec SCSI controller and `xy` for Xylogics SMD controller — and the proper unit number that you want the information for. For a client with a Xylogics controller, unit number 0 (zero), the command would be:

```
# /etc/dkinfo xy0
```

Returning to the hypothetical system described above (in the section *The /etc/nd.local File*), let us assume a Xylogics 450 disk controller and an Eagle disk. The `/etc/mkfs` command for `bill`'s root partition is shown below. Remember, this will destroy all data on client `bill`!

```
# /etc/mkfs /dev/nd10 10120 46 20 8192 1024
```

After making a file system, the partition can be mounted and used. Typically, you want to run `/etc/fsck` on the partition and mount it as part of the boot procedure. The first reaction is to put an entry in `/etc/fstab` to do just that, however, `/etc/fsck` runs before `/etc/nd` in the boot procedure so the soft partition is not defined when `/etc/fsck` is run. It is inadvisable to run `/etc/nd` before `/etc/fsck` because that makes the public partition available to clients before its file system has been checked.

The correct approach is to add commands to the end of `/etc/rc.local`. Continuing the example from above, to make `bill`'s root partition available for use add the following lines to the end of the server's `/etc/rc.local` file:

Remember to run `/etc/nd` each time you change the `/etc/nd.local` file. Halt any affected clients before a server changes (using `/etc/halt`), and reboot clients after completing the change.

Words of Warning

The `/etc/nd` command does very little error checking. It does not check for overlapping partitions, multiple entries for the same user, unreasonable starting points, or unreasonable partition sizes.

When `/etc/nd` does find an error it prints a message saying the line was ignored. That usually means `/etc/nd` stopped processing at that point. Sometimes it does go on, but in that case gets the whole rest of the file wrong. In either case, the line ignored error message means you must fix the problem and run `/etc/nd` again.

Tabs and blank lines are not allowed in the `/etc/nd.local` file.

2.3. NFS: The Network File System Service

We begin with an explanation of some NFS terms and concepts. Then we describe how to create an NFS server that exports file systems, and how to mount and utilize remote file systems. Following that a section on debugging the NFS explains what to do when problems occur. Finally, some cautionary words about incompatibilities between NFS files and normal UNIX files.

What Is The NFS Service

In its ability to allow many clients simultaneous access to a single file system, the NFS is quite different from `nd` (Network Disk). With `nd`, each client 'owns' a fixed part of a disk partitioned for client use on an `nd` server. With NFS diskless clients still have a permanent bond to their `nd` server, but may also mount (and unmount) many other file systems when they choose. Thus, while a client has only one `nd` server, it may have many NFS servers.

The NFS enables users to share file systems over the network. A client may mount or unmount file systems from an NFS server machine. The client always initiates the binding to a server's file system by using the `mount`(8) command. Typically, a client mounts one or more remote file systems at startup time by placing lines like these in the file `/etc/fstab`, which `mount` reads when the system comes up:

```
titan:/usr2 /usr2 nfs rw,hard 0 0
venus:/usr/man /usr/man nfs rw,hard 0 0
```

See `fstab`(5) for a full description of the format.

Since clients initiate all remote mounts, NFS servers keep control over who may mount a file system by limiting named file systems to desired clients with an entry in the `/etc/exports` file. For example:

```
/usr/local                    # export to the world
/usr2      nixon ford reagan # export to only these machines
```

is an `/etc/exports` entry that speaks for itself. Note that pathnames given in `/etc/exports` must be the mount point of a local file system. See

How To Remote Mount A File System

You can mount any exported file system onto your machine, as long as you can reach its server over the network, and you are included in its `/etc/export` list for that file system. On the machine where you want to mount the file system, become super-user and type the following:

```
# mount server_name :/file_system /mount_point
```

For example, to mount the manual pages from remote machine `elvis` onto the local empty directory `/usr/man`:

```
# mount -o soft elvis:/usr/man /usr/man
```

Things like manual pages are usually mounted soft, so that if `elvis` goes down, it doesn't drag down the local machine. To make sure you have mounted a file system, and mounted it where you expected to, use either `df`(1) or `mount`(8), without an argument. Each of these displays the currently mounted file systems.

Typically, you mount frequently used file systems at startup by placing an entry for them in the file `/etc/fstab`. If you are a diskless client you can look at the `/etc/fstab` for examples. You can also see `fstab`(5).

Typical NFS Layout

To demonstrate the layout of the NFS on diskless clients, the output from `mount` commands below shows the mounted file systems on a server, and on one of its clients — notice where the client file systems are mounted from. Following that, the output from `ls` commands shows the contents of various directories on the client machine. `lenin` is a diskless client of `marx`:

```
marx% mount
/dev/xy0a on / type 4.2 (rw)
/dev/xy0e on /pub.MC68010 type 4.2 (rw)
/dev/xy0g on /usr.MC68010 type 4.2 (rw)
/dev/xy2g on /usr.MC68010/marx type 4.2 (rw)
```

```
lenin% mount
/dev/nd0 on / type 4.2 (rw)
/dev/ndp0 on /pub type 4.2 (ro)
marx:/usr.MC68010 on /usr type nfs (ro,hard)
marx:/usr.MC68010/marx on /usr/marx type nfs (rw,hard)
```

```
lenin% ls -l /usr
lrwxrwxrwx  1 root           24 Oct  7 12:20 adm -> /private.MC68010/usr/adm
drwxr-xr-x  2 bin          2560 Oct  9 14:45 bin
lrwxrwxrwx  1 root           17 Oct  9 14:44 crash -> /usr/marx/crash
drwxr-xr-x  2 root           24 Oct  8 21:29 demo
drwxr-xr-x  3 bin           512 Oct  7 12:17 dict
drwxr-xr-x  2 root           24 Oct  8 21:04 doctools
drwxr-xr-x  4 bin          1536 Oct  7 12:13 etc
drwxr-xr-x  2 bin          6144 Oct  7 18:33 hosts
drwxr-xr-x 25 bin          1536 Oct  7 12:47 include
drwxr-xr-x 17 bin          2560 Oct  7 18:33 lib
drwxrwxrwx  7 root         1024 Oct  7 18:38 local
drwxr-xr-x  2 root         8192 Oct  7 11:17 lost+found
drwxrwxrwx  2 root           24 Oct  7 12:20 man
drwxrwxr-x  2 root          512 Oct  7 12:18 mdec
lrwxrwxrwx  1 root           29 Oct  7 12:20 preserve -> /private.MC68010/usr/preserve
drwxr-xr-x  2 bin           512 Oct  7 12:17 pub
drwxr-xr-x  3 bin           512 Oct  7 12:18 sccs
lrwxrwxrwx  1 root           26 Oct  7 12:20 spool -> /private.MC68010/usr/spool
drwxrwxr-x 22 root          512 Oct  7 23:51 sys
lrwxrwxrwx  1 root           24 Oct  7 12:20 tmp -> /private.MC68010/usr/tmp
drwxr-xr-x  2 bin          1536 Oct  7 12:40 ucb
```

```
lenin% ls -l /usr/lib | grep " ->"
lrwxrwxrwx  1 root          32 Oct  7 12:20 aliases -> /private.MC68010/usr/lib/aliases
lrwxrwxrwx  1 root          36 Oct  7 12:20 aliases.dir -> /private.MC68010/usr/lib/aliases.dir
lrwxrwxrwx  1 root          36 Oct  7 12:20 aliases.pag -> /private.MC68010/usr/lib/aliases.pag
lrwxrwxrwx  1 root          32 Oct  7 12:20 crontab -> /private.MC68010/usr/lib/crontab
lrwxrwxrwx  1 root          19 Oct  7 18:33 news -> /private.MC68010/usr/lib/news
lrwxrwxrwx  1 root          36 Oct  7 12:20 sendmail.cf -> /private.MC68010/usr/lib/sendmail.cf
lrwxrwxrwx  1 root          29 Oct  7 12:20 uucp -> /private.MC68010/usr/lib/uucp
```

```
lenin% ls -l /private/usr
total 5
drwxrwxr-x  2 root          512 Oct  6 21:45 adm
drwxrwxr-x  2 root          512 Oct  6 20:31 lib
drwxrwxr-x  2 root           24 Oct  6 17:08 preserve
drwxrwxr-x 13 root          512 Oct 10 03:01 spool
drwxrwxrwx  2 root          512 Oct 10 13:09 tmp
```

not check return conditions on file system operations so you may not see this error message when accessing soft mounted files. Nevertheless, an NFS error message should be printed on the console in this case also.

If a client is having NFS trouble, check first to make sure the server is up and running. From a client you can type

```
% /usr/etc/rpcinfo -p server_name
```

to see if the server is up at all. It should print out a list of program, version, protocol, and port numbers that resembles:

```
   program vers proto   port
    100004    2   udp   1027  ypserv
    100004    2   tcp   1024  ypserv
    100004    1   udp   1027  ypserv
    100004    1   tcp   1024  ypserv
    100007    2   tcp   1025  ypbind
    100007    2   udp   1035  ypbind
    100007    1   tcp   1025  ypbind
    100007    1   udp   1035  ypbind
    100003    2   udp   2049  nfs
    100016    1   tcp   1032
    100012    1   udp   1087  sprayd
    100011    1   udp   1089  rquotad
    100005    1   udp   1091  mountd
    100008    1   udp   1093  walld
    100002    1   udp   1095  rusersd
    100002    2   udp   1095  rusersd
    100001    1   udp   1098  rstatd
    100001    2   udp   1098  rstatd
    100001    3   udp   1098  rstatd
```

If that works you can also use `rpcinfo` to check if the `mountd` server is running:

```
% /usr/etc/rpcinfo -u server_name 100005 1
```

This should come back with the response:

```
proc 100005 vers 1 ready and waiting
```

If these fail you should go login to the server's console and see if it is okay.

If the server is alive but your machine can't reach it you should check the Ethernet connections between your machine and the server (see *Ethernet Troubleshooting* in the chapter on *Communications* in this manual).

If the server is okay and the network is okay use `ps` to check your client daemons. You should have a `portmap`, `ypbind`, and several `biod` daemons

8) Krypton's `mountd` does a `getfh`(2) system call on '/usr/src' to get the `fhandle`.

9) Krypton's `mountd` returns the `fhandle`.

10) `mount` does an `nfsmount`(2) system call with the `fhandle` and '/krypton.src'.

11) `nfsmount` checks if the caller is superuser and if '/krypton.src' is a directory.

12) `nfsmount` does a `statfs`(2) call to krypton's NFS server (`nfsd`).

13) `mount` opens `/etc/mtab` and adds an entry to the end.

Any one of these steps can fail, some of them in more than one way. The sections below give detailed descriptions of the failures associated with specific error messages.

`/etc/mtab: No such file or directory`

The mounted file system table is kept in the file `/etc/mtab`(5). This file must exist before mount can succeed.

`mount: ... already mounted`

The file system that you are trying to mount is already mounted or there is a bogus entry for it in `/etc/mtab`.

`mount: ... Block device required`

You probably left off the 'krypton:' part of

```
# mount krypton:/usr/src /krypton.src
```

The `mount` command assumes you are doing a local mount unless it sees a colon in the file system name or the file system type is 'nfs' in `/etc/fstab`.

`mount: ... server not responding: RPC_PMAP_FAILURE - RPC_TIMED_OUT`

Either the server you are trying to mount from is down, or its portmapper is dead or hung. Try logging in to that machine. If you can log in, try running:

```
# rpcinfo -p hostname
```

You should get a list of registered program numbers. If you don't, you should restart the portmapper. Note that restarting the portmapper requires that you then kill and restart both `inetd` and `ypbind`. To do this, become root and do a

```
# ps ax
```

to find the process ids of `portmap`, `ypbind`, and `inetd`. Next, do a

```
# kill -9 portmap_pid ypbind_pid inetd_pid
```

to kill the daemons. Then type

```
# /etc/portmap
# /etc/inetd
# /etc/ypbind
```

to start new ones.

If you don't want to mess around with this, you can just reboot the server.

If you can't `rlogin` to the server but the server is up, you should check your Ethernet connection by trying `rlogin` to some other machine. You should also check the server's Ethernet connection.

`mount: ... server not responding: RPC_PROG_NOT_REGISTERED`

This means that mount got through to the portmapper but the NFS mount daemon (`rpc.mountd`) was not registered. Go to the server and be sure that `/usr/etc/rpc.mountd` exists and that there is an entry in `/etc/servers` exactly like:

```
rpc udp /usr/etc/rpc.mountd 100005 1
```

Finally, use `ps` to be sure that the internet daemon (`inetd`) is running. If you had to change `/etc/servers` you will have to kill `inetd` and restart it. Look in `/etc/rc.local` to see how it is started at boot time and do that by hand.

Programs can also hang if an `nd` server dies (see the section on `nd` above) or if a `yp` server dies (see `yp` below).

If your machine hangs completely, check the server(s) from which you have mounted. If one of them (or more) is down don't worry, when the server comes back up your programs will continue automatically and they won't even know the server died. No files will be destroyed.

If a soft mounted server dies, other work should not be affected. Programs that time-out trying to access soft mounted remote files will fail with `errno ETIMEDOUT`, but you should still be able to get work done on your other file systems.

If all of the servers are running go ask someone else using the server or servers that you are using if they are having trouble. If more than one machine is having problems getting service, then it is probably a problem with the server's NFS daemon (`nfsd`(4)). Log in to the server and do a `ps` to see if `nfsd` is running and accumulating cpu time. If not you may be able to kill and then restart `nfsd`. If this doesn't work you will have to reboot the server.

If other people seem to be okay you should check your Ethernet connection and the connection of the server.

Hangs Part Way Through Boot

If your machine comes part way up after a boot, but hangs where it would normally be doing remote mounts, probably one or more servers is down or your network connection is bad. See *Program Hung* and *Remote Mount Failed* above.

Everything Works But Slowly

If access to remote files seems unusually slow, type:

```
# ps aux
```

on the server to be sure it is not being clobbered by a runaway daemon, bad `tty` line, etc. If the server seems okay and other people are getting good response, make sure your block I/O daemons are running; type `ps ax` and look for `biod`. If they are not running or are hung you can kill them off by typing

```
# ps ax | grep biod
```

to find the process ids, and

```
# kill -9 pid1 pid2 pid3 pid4
```

Restart them with

```
# /etc/biod 4
```

To determine whether they are hung, do a `ps` as above, then copy a large remote file, then do another `ps`. If the `biod`s don't accumulate cpu time they are probably hung.

```
% chmod 4777 a.out
% su
Password:
# chown root a.out
a.out: Not owner
```

To change the ownership, you must login to the server as root, then make the change. Or, you can move the file to a file system owned by your machine (for example `/usr/tmp` is always owned by the local machine) and make the change there.

(NOTE: Sun does not recommend over-the-net root access as discussed in the following paragraphs.) In a very friendly network environment, you may choose to allow root access over the network. To allow this access, you must change the value of the kernel variable 'nobody' in the NFS server's kernel. To make the change you `adb`(1) the server's kernel. (NOTE: You never change the client's kernel in this procedure, only the server's.)

You can alter the copy of the kernel in memory, in which case the change will be immediate but will be lost when you reboot the machine, or you can change `/vmunix` so that every time you reboot the altered kernel will be run. If you alter `/vmunix` but do not alter the kernel in memory, the machine must be rebooted for the change to take effect. Here are the steps for each kind of change.

- To change the kernel in memory on a running system do the following. (This change will disappear when you reboot, unless you also change the binary image as explained below.) On the NFS server change the value of the kernel variable 'nobody'.

```
# adb -w /vmunix /dev/kmem
```

`adb` responds:

```
not core file = /dev/kmem
```

This is normal. Then you type:

```
nobody/D
```

and `adb` responds with:

```
_nobody:
_nobody:        -2
```

If it does not, stop and call Sun Tech Support for further help. If it does, then you enter:

File Operations Not Supported

File locking is not supported on remote file systems. Therefore, the `flock`(2) call will fail when locking a remote file.

In addition append mode and atomic writes are not guaranteed to work on remote files accessed by more than one client simultaneously.

Cannot Access Remote Devices

In the NFS you cannot access a remote mounted device or any other character or block special file — like named pipes.

Beefing Up NFS Security

The NFS currently operates under the assumption that you have a "friendly" net. That is, you can trust all of the users attached to your net. We realize that this does not apply to everyone, so here are some things you can do to improve file security.

Port Monitoring

In Berkeley UNIX, there are some Internet domain source ports to which only privileged users can attach (these are know as "privileged ports"). Currently, the NFS does not check to see if a client is bound to one of these. That is, an NFS server has no way of knowing whether a client's file request originated from the real client's kernel or from some miscreant's user-program. NFS server port checking can be turned on as follows:

```
# adb -w /vmunix
nfs_portmon?W1
_nfs_portmon:    0x0          =        0x1
<ctrl-D>
# adb -w /usr/etc/rpc.mountd
nfs_portmon?W1
_nfs_portmon:    0x0          =        0x1
<ctrl-D>
```

The next time you boot your system, the source ports will be checked. If you can trust all of the root users on your net, then just doing the above is enough. But be warned: some non-UNIX systems do not enforce the privileged port convention (in particular, PCs with 3Com boards). Another warning: all of a server's clients should be running software release 3.0 or higher for this to work (a server will reject all pre-3.0 requests).

IP Source Address Checking

If you are in a situation where you cannot trust all of the root users on your net, then there is more you can do. An NFS server has a list of machine names to which it exports file-systems. This list is know as the export list, and is defined in the file `/etc/exports`. When a client makes a mount request to an NFS server, the server checks to see if the client's machine name appears in its export list.

This checking is not very secure since one can use the `hostname` command to fake a machine-name to something in the server's export list. To improve security, you can have the server also check the client's source IP address. IP addresses can be faked by altering `/etc/hosts`, but the machine being faked will be warned repeatedly. The message "duplicate IP address! sent from ethernet address: XXXXXX" will appear on ts console, displaying the faker's

```
# date
Jan 22 15:27:31 PST 1985
# touch file3
# ls -l file*
-rw-r--r--  1 root          0 Dec 27  1983 file
-rw-r--r--  1 root          0 Jan 22 15:26 file2
-rw-r--r--  1 root          0 Jan 22  1985 file3
```

The problem is that the difference of the two times is huge:

(now) - (modification_time) =
(now) - (now + 180 seconds) =
-180 seconds = huge unsigned number,
which is greater than six months.

Thus, `ls` believes the new file was created long ago in the past. Sun modified `ls` to deal with files which are created a short time in the future.

2) `ranlib`(1) was also modified to deal with clock skew. `ranlib` timestamps a library when it is produced, and `ld` compares that timestamp with the last modified date of the library. If the last modified date occurred after the timestamp, then `ld` instructs the user to run `ranlib` again, and reload the library.

   If the library lives on a server whose clock is ahead of the client that runs `ranlib`, `ld` will always complain. Sun fixed `ranlib` to set the timestamp to the maximum value of the current time and the library's modify time.

In general remember, if your application depends upon local time and/or the file system timestamps, then it will have to deal with clock skew problems if it uses remote files.

2.4. yp: The Sun Yellow Pages Service

We begin by introducing yp related terms. Following that we list the new yp manual pages; that section also includes pointers to yp documentation beyond this chapter. Next, you will find explanations of the installation and administration of yp, a section on how to debug yp when problems occur, and finally, a discussion of file access policies and special security issues raised by the yp environment.

What Is The Yellow Pages Service?

The yellow pages is Sun's distributed network lookup service:

- yp is a distributed system; the database is fully replicated at several sites, each of which runs a server process for the database. These sites are known as yp servers. At steady state, it doesn't matter which server process answers a client request; the answer will be the same all over. This allows multiple servers per network, and gives yp service a high degree of availability and reliability.
- yp is a lookup service. It maintains a set of databases which may be queried. A client may ask for the value associated with a particular key within a database, and may enumerate every key-value pair within a database.

servers to get the information.

Most of the information describing the structure of the yp system and the commands available for that system is contained in manual pages, and is not repeated here. For quick reference, we list the man pages and an abstract of their contents here. For more information, see the *Network Services Guide.*

ypserv(8) — describes the processes which comprise the yp system. These are ypserv, the yp database server daemon, and ypbind, the yp binder daemon. ypserv must run on each yp server machine. ypbind must run on all machines which use yp services, both servers and clients.

ypfiles(8) — describes the database structure of the yp system.

ypinit(8) — many maps must be constructed from files located in /etc, such as /etc/hosts, /etc/passwd and others. The database initialization tool ypinit(8) does all such construction automatically. In addition, it constructs initial versions of maps required by the system but not built from files in /etc; an example is the map 'ypservers'. Use this tool to set up the master yp server and the slave yp servers for the first time. Do not (generally) use it as an administrative tool for running systems.

ypmake(8) — describes the use of /etc/yp/Makefile, which builds several commonly-changed components of the yp's database. These are the maps built from several ASCII files normally found in /etc: passwd, hosts, group, netgroup, networks, protocols, and services.

makedbm(8) — describes a low-level tool for building a dbm file which is a valid yp map. Databases not built from /etc/yp/Makefile may be built or rebuilt using makedbm. makedbm may also be used to 'disassemble' a map so that you can see the key-value pairs which comprise it. The disassembled form may also be modified with standard tools (such as editors, awk, grep, and cat), and is in the form required for input back into makedbm.

ypxfr(8) — moves a yp map from one yp server to another, using the yp itself as the transport medium. It can be run interactively, or periodically from crontab. In addition, ypserv uses ypxfr as its transfer agent when it is asked to transfer a map.

yppush(8) — describes a tool to administer a running yp system. It is run on the master yp server. It requests each of the ypserv processes within a domain to transfer a particular map, waits for a summary response from the transfer agent, and prints out the results for each server.

ypset(8) — tells a ypbind process (the local one, by default) to get yp services for a domain from a named yp server. This is not for casual use.

yppoll(8) — asks any ypserv for the information it holds internally about a single map.

ypcat(1) — dumps out the contents of a yp map. Use it when you don't care which server's version you are seeing. If you need to see a particular server's map, rlogin to that server (or use rsh) and use makedbm.

Altering A yp Client's Files To Use yp Services

Once the decision has been made to serve a database with the yp, it is desirable that all nodes in the net access the yp's version of the information, rather than the potentially out-of-date information in their local files. That policy is enforced by running a ypbind process on the client node (including nodes which may be running yp servers), and by abbreviating or eliminating the files which traditionally implemented the database. The files in question are: /etc/passwd, /etc/hosts, /etc/ethers, /etc/group, /etc/networks, /etc/protocols, /etc/services, /etc/netgroup, /etc/hosts.equiv, and /.rhosts. The treatment of each file is discussed in this section.

- /etc/networks, /etc/protocols, /etc/ethers, /etc/services, and /etc/netgroup need not exist at any yp client node. If you are squeamish about removing them, move them to backup names; for instance on a machine named 'ypclient':

```
ypclient%  cd /etc
ypclient%  mv networks networks-
ypclient%  <The rest of the renames, similarly>
```

- /etc/hosts.equiv is not normally served by the yp. However, you can add escape sequences to activate the yp. This reduces problems with rlogin, or rsh which are sometimes caused by different /etc/hosts.equiv files on the two machines.

  To let anyone log on to a machine, /etc/hosts.equiv may be edited to contain a single line, with only the character '+' (plus) on it. A line with only a '+' means that all further entries will be retrieved from the yp rather than the local file.

  Alternatively, more control may be exercised over logins by using lines of the form:

```
+@trusted_group1
+@trusted_group2
-@distrusted_group
```

Each of the names to the right of the '@' (at) character is assumed to be a net group name, defined in the global netgroup database. The netgroup database is served by the yp.

If none of the escape sequences is used, only the entries in /etc/hosts.equiv are used; the yp is not used.

- /.rhosts is not normally served by the yp, either. Its format is identical to that of /etc/hosts.equiv. However, since this file controls remote root access to the local machine, unrestricted access is not recommended. Make the list of trusted hosts explicit, or use net group names for the same purpose.

intended to be the yp slave server must be set up, and must be set to the same domain name as the default domain name on the machine named as the master. In addition, an entry for 'daemon' must exist in the /etc/passwd files of both slave and master, and that entry must precede any other entries which have the same uid. (setup creates /etc/passwd; make sure your password file has not been altered to change the order. Note the example shown in the section above.) You won't be prompted for a list of other servers, but you will have the opportunity to choose whether or not the procedure gives up at the first non-fatal error.

After running ypinit, make copies of /etc/passwd, /etc/hosts, /etc/group, /etc/networks, /etc/protocols, /etc/netgroup, and /etc/services. For instance on a machine named 'ypslave':

```
ypslave% cp /etc/passwd /etc/passwd-
```

Edit the original files in accordance with the section above on altering the client's database, so as to insure that processes on the slave yp server will actually make use of the yp services, rather than the local ASCII files. (That is, make sure the yp slave server is also a yp client) Make backup copies of the edited files, as well. For instance:

```
ypslave% cp /etc/passwd /etc/passwd+
```

After the yp database gets set up by ypinit, type /etc/ypserv to begin supplying yp services. On subsequent reboots, it will start automatically from /etc/rc.local

How To Set Up A yp Client

To set up a yp client, edit the local files as described in the section above on altering a yp client's file database. If /etc/ypbind is not running already, start it. With the ASCII databases of /etc abbreviated and /etc/ypbind running, the processes on the machine will be clients of the yp services. At this point, there must be a yp server available; all sorts of stuff will hang up if no yp server is available while ypbind is running. Note the possible alterations to the client's /etc database as discussed above in the section on altering the client. Because some files may not be there, or some may be specially altered, it is not always obvious how the ASCII databases are being used. The escape conventions used within those files to force inclusion and exclusion of data from the yp databases are found in the following man pages: passwd(5), hosts(5), netgroup(5), host.equiv(5), group(5). In particular, notice that changing passwords in /etc/passwd (by editing the file, or by running passwd(1)), will only affect the local client's environment. Change the yp password database by running yppasswd(1).

How To Modify Existing yp Maps After yp Installation

Databases served by the yp must be changed ON THE MASTER SERVER. The databases expected to change most frequently, like /etc/passwd, may be changed by first editing the ASCII file, and then running make(1) on /etc/yp/Makefile — also see ypmake(8).

```
ypmaster% cd /etc/yp
ypmaster% makedbm - home_domain/mymap
al ar
bl br
cl cr
<ctl D>
```

When you need to modify that map, you can use makedbm to create a temporary ASCII intermediate file which can be edited using standard tools. For instance:

```
ypmaster%  cd /etc/yp
ypmaster%  makedbm -u home_domain/mymap > mymap.temp
```

At this point 'mymap.temp' can be edited to contain the correct information. A new version of the database is created by the commands

```
ypmaster%  makedbm mymap.temp home_domain/mymap
ypmaster%  rm mymap.temp
```

The preceding paragraphs explained how to use some tools, but in reality almost everything you actually have to do can be done by ypinit(8) and /etc/yp/Makefile, unless you add non-standard maps to the database, or change the set of yp servers after the system is already up and running.

Whether you use the Makefile in /etc/yp or some other procedure — Makefile is one of many possible — the goal is the same: a new pair of well-formed dbm files must end up in the domain directory on the master yp server.

Propagation Of A yp Map

To propagate a map means to move it from place to place — in general, to move it from the master yp server to a slave yp server. Initially, ypinit(8) moves it, as described above in the section *How To Set Up A Slave* yp *Server.* After a slave yp server has been initialized, updated maps are transferred from the master server by ypxfr(8). ypxfr may be run in three different ways: periodically by cron(8); by ypserv(8); and interactively by a user. Let's look an example of each.

Maps have differing rates of change; for instance 'protocols.byname' may not change for months at a time, but 'passwd.byname' may change several times a day in a large organization. You can set up crontab(5) entries to periodically run ypxfr at a rate appropriate for any map in your yp database. ypxfr will contact the master server and transfer the map only if the master's copy is more recent than the local copy.

To avoid a crontab entry for each map, several maps with approximately the same change characteristics can be grouped in a shell script, and the shell script can be run from /usr/lib/crontab. Suggested groupings, mnemonically named, can be found in /etc/yp: ypxfr_1perhour, ypxfr_1perday, and ypxfr_2perday. If the rates of change are inappropriate for your environment, these shell scripts can be easily modified or replaced.

some yp servers, but not all, you will see unpredictable behavior from client programs.

How To Add A New yp Server Not In The Original Set

To add a new yp slave server start by modifying some maps on the master yp server. If the new server is a host which has not been a yp server before, the host's name must be added to the map 'ypservers' in the default domain. The sequence for adding a server named 'ypslave' to domain 'home_domain' is:

```
ypmaster% cd /etc/yp
ypmaster% (makedbm -u home_domain/ypservers;\
echo ypslave ypslave) |makedbm - tmpmap
ypmaster% mv tmpmap.dir home_domain/ypservers.dir
ypmaster% mv tmpmap.pag home_domain/ypservers.pag
ypmaster% yppush ypservers
```

Note that we display some commands above on two lines. You may type these as one long command (even if the line wraps on your screen), or you may escape the return and newline with a backslash, as shown here. However, you cannot simply type in half the command, hit return, and type the second half.

The host's address should be in 'hosts.byname'. If that's not true, edit `/etc/hosts` and run `make`. In this case the commands should be:

```
ypmaster%  <edit /etc/hosts here>
ypmaster% cd /etc/yp
ypmaster% make hosts
```

The new slave yp server's databases should be set up by copying the databases from yp master server 'ypmaster'. Remote login to the new yp slave, and use `ypinit`(8) in the following way:

```
ypslave% cd /etc/yp
ypslave% ypinit -s ypmaster
```

Then complete the steps described above in the section *How To Set Up A Slave yp Server.*

How To Change The Master Server

To change a map's master, first build the map at the new master. Because the old yp master's name occurs as a key-value pair in the existing map, it is not sufficient to use an existing copy at the new master, or to send a copy there with `ypxfr`. The key must be reassociated with the new master's name. If the map has an ASCII source file, it should be present in its current version at the new master. Remake the yp map (we'll call it 'jokes.bypunchline') locally with the sequence:

```
newmaster% cd /etc/yp
newmaster% make jokes.bypunchline
```

yp server machine is so overloaded that ypserv can't get a response back to your ypbind within the timeout period. Under these circumstances, all the other yp client nodes on your net will show the same or similar problems. The condition is temporary in most cases, and the messages will usually go away when the yp server machine reboots and ypserv gets back in business; or when the load on the yp server nodes and/or the Ethernet decreases.

However, in the circumstances described below, the situation will never get better.

- The yp client has not set, or has incorrectly set, domainname on the machine. Clients must use a domain name that the yp servers know. Use domainname(1) to see the client domain name. Compare that with the domain name set on the yp servers. The domain name should be set in /etc/rc.local. When /etc/rc.local fails to set, or incorrectly sets, domainname you will have to do the following: become superuser on the machine in question, edit /etc/rc.local to fix the domainname line with a proper domain name (this assures domain name will be correct every time the machine boots), and set domainname 'by hand' so it is fixed immediately — type:

```
# domainname good_domain_name
```

- If your domain name is correct, make sure your local net has at least one yp server machine. You can only bind to a ypserv process on your local net, not on another accessible net. There must be at least one yp server for your machine's domain running on your local net. Two or more yp servers will improve availability and response characteristics for yp services.
- If your local net has a yp server, make sure it is up and running. Check other machines on your local net. If several client machines have problems simultaneously, suspect a server problem. Find a client machine behaving normally, and try the ypwhich command. If ypwhich never returns an answer, kill it and go to a terminal on the yp server machine. Type:

```
% ps ax | grep yp
```

and look for ypserv and ypbind processes. If the server's ypbind daemon is not running, start it up by typing:

```
% /etc/ypbind
```

If there is a ypserv process running, do a ypwhich on the yp server machine. If ypwhich returns no answer, ypserv has probably hung and should be restarted. Kill the existing ypserv process (you must be logged on as root), and start /etc/ypserv:

section on *Ethernet Debugging* in the *Communications* chapter of this manual.

You may be able to talk to the `portmap` on your machine from a machine operating normally. From such a machine, type:

```
flipper% rpcinfo -p your_machine_name
```

If your `portmap` is okay, the output should look like:

```
[program, version, protocol, port]:

[100005, 1, 17, 1046]
[100001, 2, 17, 1055]
[100001, 1, 17, 1055]
[100002, 1, 17, 1052]
[100008, 1, 17, 1049]
[100007, 1, 17, 1027]
[100007, 1, 6, 1026]
[100007, 2, 17, 1031]
[100007, 2, 6, 1030]
```

On your machine the port numbers will be different. The four entries that represent the `ypbind` process are:

```
[100007, 1, 17, port_#]
[100007, 1, 6, port_#]
[100007, 2, 17, port_#]
[100007, 2, 6, port_#]
```

If they are not there, `ypbind` has been unable to register its services. Reboot the machine. If they are there and they change each time you try to restart `/etc/ypbind`, reboot the system, even if the `portmap` is up. If the situation persists after reboot, call for help.

On Client: `ypwhich` Inconsistent

When you use `ypwhich` several times at the same client node, the answer you get back varies — the `yp` server changes. This is normal. The binding of `yp` client to `yp` server will change over time on a busy net, and when the `yp` servers are busy. Whenever possible, the system stabilizes at a point where all clients get acceptable response time from the `yp` servers. As long as your client machine gets `yp` service, it doesn't matter where the service comes from. Often a `yp` server machine gets its own `yp` services from another `yp` server on the net.

Debugging A Yellow Pages Server

Before trying to debug a yellow pages server, read the earlier section in this chapter on how the `yp` works.

that the map files should be owned by root, so you must change ownership of them after the transfer. Obviously, if you can do the `rcp` as root, it makes the whole thing easier.

On Server: `ypserv` Crashes

When the `ypserv` process crashes almost immediately, and won't stay up even with repeated activations, the debug process is virtually identical to that described above in the section *On Client: ypbind Crashes*. Check for the `portmap` daemon:

```
ypserver% ps ax | grep portmap
```

Reboot the server if you do not find it. If it is there, type:

```
ypserver% /usr/etc/rpcinfo -p
```

and look for output to the screen like:

```
[program, version, protocol, port]:

[100001, 2, 17, 1062]
[100001, 1, 17, 1062]
[100002, 1, 17, 1060]
[100008, 1, 17, 1058]
[100005, 1, 17, 1056]
[100007, 1, 17, 1032]
[100007, 1, 6, 1027]
[100004, 1, 6, 1026]
[100004, 1, 17, 1024]
[100004, 2, 6, 1043]
[100004, 2, 17, 1040]
```

On your machine, the port numbers will be different. The four entries that represent the `ypserv` process are:

```
[100004, 1, 6, port_#]
[100004, 1, 17, port_#]
[100004, 2, 6, port_#]
[100004, 2, 17, port_#]
```

If they are not there, `ypserv` has been unable to register its services. Reboot the machine. If they are there, and they change each time you try to restart `/etc/ypserv`, reboot the machine. If the situation persists after reboot, call for help.

global.

For example, a program that calls `/etc/passwd` (a local file) will first look in the password file on your machine; the yellow pages password file will only be consulted if your machine's password file contains '+' (plus sign) entries. The `/etc/passwd` file is local so that you can control the entries for your own machine. The only other local file is `/etc/group`. To repeat, local files are consulted first on your own machine, before looking in the yellow pages.

The remaining yellow pages files (`hosts`, `networks`, `ethers`, `services`, `protocols`, and `netgroup`) are global files. The information in these files is network wide data, and is accessed only from the yellow pages. However, when booting, each machine needs an entry in `/etc/hosts` for itself. In summary, if yellow pages is running, global files are only checked in the yellow pages; a file on your local machine is not consulted.

Two Other Files yp Consults

The files `/etc/hosts.equiv` and `/.rhosts` are not in the yellow pages database. Each machine has its own unique copy. However it is possible to put entries in your `/etc/hosts.equiv` file that refer to the yellow pages. For example a line consisting of

```
+@engineering
```

will include all members of `engineering` as it is defined in the local file `/etc/netgroup` or in the yp database. A line consisting only of '+' (a plus sign) will include everyone in your `/etc/hosts.equiv` file.

Security Implications

Recall that to be able to log into a machine without having a password, you need to be in both the `/etc/hosts.equiv` file and the `/etc/passwd` file. By having a '+' entry in `/etc/hosts.equiv`, you effectively bypass this check, and anyone in your `/etc/passwd` file will be allowed to rlogin to your machine without restriction.

An `/etc/passwd` file and `/etc/group` file may also have '+' entries. A line in an `/etc/passwd` file such as

```
+nb:::::Napoleon Bonaparte:/usr2/nb:/bin/csh
```

pulls in an entry for `nb` from the yellow pages. It gets the `uid`, `gid` and `password` from the yellow pages, and gets the gecos, home directory and default shell from the '+' entry itself. On the other hand, an `/etc/passwd` entry such as

```
+nb:
```

gets all information from the yellow pages. Finally, notice that

```
+nb::1189:10:Napoleon Bonaparte:/usr2/nb:/bin/csh
```

```
#
# Engineering: Everyone, but eric, has a machine; he has no machine.
# The machine 'testing' is used by all of hardware.
#
engineering    hardware software
hardware  (mercury,alan,sun) (venus,beth,sun) (testing,-,sun)
software  (earth,chris,sun) (mars,deborah,sun) (-,eric,sun)
#
# Marketing: Time-sharing on jupiter
#
marketing (jupiter,fran,sun) (jupiter,greg,sun) (jupiter,dan,sun)
#
# Others
#
allusers (-,,sun)
allhosts (,-,sun)
```

Based on the above, the users will be classified into groups as follows:

GROUP	USERS
hardware	alan, beth
software	chris, deborah, eric
engineering	alan, beth, chris, deborah, eric
marketing	fran, greg, dan
allusers	(every user in the yp map passwd)
allhosts	(no users)

And here is how the machines would be classified:

GROUP	HOSTS
hardware	mercury, venus, testing
software	earth, mars
engineering	mercury, venus, earth, mars, testing
marketing	jupiter
allusers	(no hosts)
allhosts	(all hosts in the yp map hosts)

Manual Pages Covering Security Issues

For more details, see the following man pages: `yppasswd`(1), `hosts.equiv`(5), `export`(5), `passwd`(5), `group`(5), `netgroup`(5), `yppasswdd`(8C).

/etc/passwd file. You must use passwd(1) while logged in as the user, or as superuser. Since anyone can login when a user has no password, you may provide a password for the new user and let him know it so he can log in and change it to whatever he prefers. When Mr. Chimp logs in for the first time, he can use the passwd utility to change his password, or yppasswd(1) to change it in the yp database.

After Mr. Chimp has a password, the entry for 'bonzo' in the password file will look something like:

```
bonzo:3u0mRdrJ4tEVs:1947:10:Mr.  Chimp:/usr2/bonzo:/bin/csh
```

Fields in the password file have the following meanings:

1) Login name — synonymous with user name.

2) Encrypted password. Tell all new users how to add, or change, their password with the passwd command and the yppasswd command. Remember, the system administrator can make this field empty when a user has forgotten his or her password, thereby enabling login without a password until such time as a new one is given. Note that an asterisk (*) in this field matches no password. The user can only log in if the machine name of the machine he is logging in from is in the /etc/hosts.equiv file on the local machine.

3) User ID. A number unique to this user. A system knows the user by ID number associated with login name, therefore a login name must have the same user ID number on all password files of machines which are networked in a local domain. Failure to keep ID's unique will prevent moving files between directories on different machines because the system will respond as if the directories are owned by two different users. In addition, file ownership may become confused when an NFS server exports a directory to an NFS client whose password file contains users with uid's that match those of different users on the NFS server.

4) Group ID. Can be used to group users together who are working on similar projects. All system staff are in group '10' for historical reasons. In this example Mr. Chimp is in the system staff group. Do not put normal users in this group. If you are not sure what group to put a new client in, see group(5) and look in the file /etc/group.

5) Information about user — usually real name, phone number, etc. An ampersand (&) here is shorthand for the user's login name.

6) The user's home directory — the directory the user logs in to.

7) Initial shell to use on login. If this field is blank the default /bin/sh is used. We recommend placing /bin/csh here, as in the example above, it gives a Berkeley 4.2 C-Shell as the user's initial shell.

After you have updated the password file and created a password for the new user, be sure to update the yellow database by running /etc/yp/make for /etc/passwd:

2.6. Adding A Client Machine To An nd Server

Each client machine of an nd server machine owns a root and a swap partition on a network disk. A public partition, holding a UNIX kernel, boot program, and other information and programs necessary for booting, is shared between the file server and all the clients. Typically, the nd server also exports file systems to its clients using the NFS.

To add a client, the nd server must have soft partitions available on the network disk allocated for client use. For a further discussion of network disks, including usage and terminology, turn to the earlier section in this chapter, *The Network Disk*.

You must be superuser to make the alterations explained below. Read through to the end of all steps before beginning the procedure.

Choose A Unique Client Name

Choose a name for the new client machine that is unique to the local network of machines. On the master yellow pages server, look in the file /etc/hosts to see what machine names are already used in your domain and choose a name that does not appear there.

If you've forgotten your master yp server you can find it out by typing:

```
% rsh `ypwhich` ypwhich
```

Edit Two Files On The Master yp Server

After you have chosen the name, stay on the master yp server and add a line for the new client to the master server's /etc/hosts file and to the /etc/ethers file. These entries enable the nd server to recognize the new client.

There are two parts to an /etc/hosts entry — the internet address and the host name. See the example below.

```
192.9.200.1 nancy
192.9.200.2 sluggo
192.9.200.3 taco
192.9.200.4 dutch
```

The last component of the internet address, the host number, must be unique for each client on the local network; assign an unused host number. Following the internet address, add the host name for the new client machine. For example, the new client machine here is 'dutch', whose internet address is 192.9.200.4; the host number is 4.

The /etc/ethers file format is similar to /etc/hosts. However, the ethernet address of the client is used instead of the internet address. For example, an /etc/ethers file looks like:

2) You may assign a previously used soft partition to the new client. That is to say, you may put the new client on the partition of a client removed from the server. In this case, find entries in `/etc/nd.local` that belonged to the former client. Change the former client machine name to the new client machine name on the `user` lines for the root and swap devices. On the `ether` line, change the name and give the new client's Ethernet address. The section below, *Preparing A Previously Used Client Partition*, more fully discusses this second case.

3) You may define a new partition in any unpartitioned space left on your network disk, and then assign it to a new client. In this case you first determine the new partition's size, then add two new `user` lines to the `/etc/nd.local` file. You also need to add an `ether` line for the new client. In many systems the root and swap partition sizes are standard for every client. In that case, use the standard *nblks* value for the new client's root and swap in your `/etc/nd.local` file — assuming of course enough room remains for a standard root and swap on the disk. The *startblks* value for any line is the sum of the *startblks* and *nblks* entries for the line preceding it. If you do not use a standard *nblks* value for partition size, you must make sure that the partition size you choose will place the partition boundary on a cylinder boundary. Follow these steps to make the determination:

   - Run `/etc/dkinfo`(8) to find out the number of sectors per track, and the number of tracks per cylinder on your disk.
   - Multiply sectors per track by tracks per cylinder to obtain the number of sectors per cylinder on your disk.
   - The number of sectors per cylinder is the minimum number of 512 byte blocks that can be allocated in the *nblks* field of the `user` line in `/etc/nd.local`. All *nblks* values greater than this minimum must be multiples of the number of sectors per cylinder.

   In order to correspond the *nblks* value to disk storage capacity, you can multiply *nblks* by 512 for an exact byte count. For an estimate in Mbytes, divide *nblks* by 2048.

   After you define a new partition on unused disk space (putting `user` lines for root and swap in `/etc/nd.local`) follow the procedures described below in the section *Preparing A Previously Used Client Partition.*

Any time you change the `/etc/nd.local` file, you must use it as input to the `/etc/nd` utility (which controls the network disk service of the kernel) to make it take effect:

```
# /etc/nd < /etc/nd.local
```

When you run this command after adding a new client, you will sometimes see an error message beginning:

```
nd user:  unknown host newclientname
```

character on it; otherwise you should move a copy of the nd server's /etc/group file to the client's partition.

3) /mnt/etc/printcap. Copy the nd server's version of this file to the client's partition:

```
# cp /etc/printcap /mnt/etc/printcap
```

4) /mnt/etc/hosts. When the client is up and running normally, the yellow pages provides this file. However, to enable the client system to come up single-user, there must be a version of the file containing an entry for *client_name* and localhost on the client's partition. For example, the minimum workable file for a client named 'pegasus' would be:

```
192.9.1.124 pegasus
127.0.0.1   localhost
```

Of course your internet numbers will be different. For details, see the section on editing the /etc/hosts file above.

5) /mnt/etc/networks. This can be provided by the yellow pages if the file contains a '+' line.

To enable the mail facility on the new client, you must copy one more file to the new client partition:

```
# cp /usr/lib/sendmail.subsidiary.cf\
/mnt/private/usr/lib/sendmail.cf
```

Note that you may enter the above command on a single line. This display wraps to a second line only because of page size limitations. If you do want to wrap your command, you must escape the carriage return with a backslash (\) as shown above.

In addition, edit the /mnt/etc/rc.boot file on the client partition, changing the line near the beginning which reads

```
/bin/hostname server_name
```

to

```
/bin/hostname new_client_name
```

Also check that the proper new client domain name is specified on the line

```
/bin/domainname domain_name
```

```
# /etc/umount/ existing_directory
# /etc/fsck -p /dev/rndl*
```

5) You have now created a partition like the one that `setup` creates during system installation. No trace of the previous client remains on the file system. Thus, you should now go to the beginning of the section above, *Preparing A New Client Partition*, and proceed as if this were a new and unused partition created by `setup`.

Booting The New Client Machine

If you have successfully completed all the steps above, you can set the new client loose on her machine. The new client will need to boot the machine; for a discussion turn to the manual *Installing UNIX On The Sun Workstation*. After booting, the client should set up the network and mail facilities; for a discussion of those procedures see the related sections in this *System Administration Manual*, and in *Installing UNIX On The Sun Workstation*.

3

Disks And File Systems

3

Disks And File Systems

This chapter begins with a brief introduction to some terms that will help you understand how the disk storage devices work and how UNIX uses them. Following that we give an overview of UNIX organization on your disk(s) — what the major file systems are and how they are known by the hardware and software components of your system. (In addition, a paper explaining the UNIX file system check program, `fsck`(8), appears at the end of this manual in the *Tutorials* section).

After the introduction and overview, we discuss some practical aspects of data storage on disks. We explain how to backup (an absolute must) and restore the files that comprise UNIX, as well as those files users create to store their own data and programs. We also explain how to free up space on disks that have become full or nearly full. There are suggestions about what files to remove and what files to keep an eye on so they won't become unacceptably large.

3.1. Disk Device Terminology

This section introduces some of the terminology of disk drives.

Platters

Most common disk devices consist of a number of *platters* mounted on a spindle spinning at a high speed. Disks store information on a thin magnetic coating. Usually, they store information serially in bits of encoded magnetic impulses — a method similar to the way magnetic tape encodes information.

Heads

To read and write the information on the surface of the disk, a number of *heads* are mounted on a common arm so they travel together. Usually, there are two heads for each platter (one for the top and one for the bottom surface). The arm moves radially in and out over the surface of the disk, positioning the heads. This radial head movement is called *seeking*. The *seek* positions are controlled by the hardware of the disk drive.

Tracks and Cylinders

The portion of a disk passing under a single stationary head during disk rotation is called a *track*. At any seek position, a track passes under each head. The set of tracks for all the heads at a single seek position is called a *cylinder*, and is in effect a stack of rings one bit 'wide', equidistant from the spindle. To change from one cylinder to another, the heads must move inward or outward, but to change from one track to another in the same cylinder, the system just switches electronically to a different head.

multibus.

The CPU makes requests for data from SMD controllers in the form: cylinder_#, track_#, sector_#. Thus, with the SMD interface, the system has to 'know about' the geometry of the disk. This allows UNIX to arrange the data on the disk so that different parts of files are positioned for efficient sequential access. On the other hand, this increases the computational load on the CPU for each disk access.

The SCSI bus treats the devices on it as idealized block devices. Requests across the SCSI bus ask for a block number from a specified device. Each device on the SCSI bus has a controller. For example, the controller for the Micropolis disk is an Adaptec, which receives commands from the CPU through the SCSI adapter and bus, and communicates with the disk across an ST-506 interface.

Because of these differences the SMD drives and the SCSI/ST-506 drives are handled differently by the system. When communicating with an SMD drive, the system makes requests of the form: cylinder_# track_# sector_#, while for the ST-506 disks, the system requests: block number nnnnnn. Then the SCSI controller translates the block number into a particular cylinder, track and sector. This idealized representation of the disk device cuts down on the CPU overhead for file transfers, but also makes it impossible for UNIX to optimize the layout of files on the disk.

SMD Disk Formatting, Mapping, Slipping

The arrangement of sectors on a track depends upon the configuration of the controller board and the disk drive. Furthermore, SMD drives distinguish between *physical sector numbers* and *logical sector numbers*. In addition to the platters which store data, a specially marked platter identifies the location of the different seek positions, enabling the heads to move to the appropriate cylinder in response to a seek command. This platter also has a timing mark which measures a full rotation of the disk platters.

Physical sector 0 follows this timing mark. It takes more time to shift from one head to another than it takes a sector to pass beneath the heads. Therefore, to facilitate sequential accesses to consecutive tracks within a cylinder, logical sector 0 of successive tracks is staggered. If the physical and logical sector numbers were not staggered, the drive might miss the beginning of the 0 sector of a track when attempting to read two sequential tracks; so logical sector 0 of track 0 of a cylinder is located at physical sector 0, logical sector 0 of track 1 is located at physical sector 1, and so on. The time required to seek from one cylinder to the next depends on the distance of the seek, but is great enough that this pattern need not be continued from one cylinder to the next.

Although the circuitry in the disk drive knows about the physical sectors to some extent, the controller writes a header on the disk identifying the beginning of a logical sector, as well as the number of the sector, the track, and the cylinder which it is in. The header also enables double checking of the disk drive electronics.

Following the header are 512 bytes of data and a trailer. The trailer has a code number derived from the contents of the data section of the sector. This code number can be used for error correction; the controller calculates it and writes

cylinder boundaries, so all of the spare sectors are grouped together at the end of the disk. SCSI/ST-506 disks are always formatted according to the defect list which `diag` currently has in memory. Thus, if the list has been edited, you must use the current copy, and if there is no current copy, you must read the list in from the disk before formatting. It also pays to check the current version of the defect list against the printed version supplied with the system.

Surface analysis passes can be done on a formatted SCSI/ST-506 just like on an SMD disk. However, with the ST-506 the bad blocks are added to the copy of the defect list which `diag` has in memory rather marked on the disk. After surface analysis, the disk should be formatted again. With ST-506 disks, the formatting process only takes about 5 minutes, and while 5 surface analysis passes take between 20 and 30 minutes, it is worth the time.

See the chapter on *Diagnostics* in this manual — the `diag` section — for further information about SCSI/ST-506 disk formatting.

Labels And Partitions

In addition to disk format organization, which is largely for the benefit of the controller, there is a further, larger scale organization for the benefit of the device driver. This organization is called disk partitioning. Partitioning divides the disk into *partitions*, each of which corresponds to one of the device entries: `/dev/`*xx#a*; where *xx* is a controller type, such as `xy` for Xylogics, *#* is the unit number of the disk attached to the controller, and *a* is an alpha value from 'a' to 'h'. These partitions are not marked as such, but are described by a *label* written on cylinder 0, track 0, sector 0 of the disk. The label contains an entry for each of the eight partitions `a - h`. Each partition entry has an offset, which is the cylinder number for the first cylinder of that partition and a size for the partition expressed in blocks. The label also contains a description of the disk drive geometry: it specifies the type of disk, the total number of cylinders, the number of heads, and the number of logical sectors per track.

`diag` supplies a default partition for the disks which Sun sells with its systems. But if you know what the particular requirements of your system will be, you may customize disk partitioning when you install your system for the first time. For example, the table below shows the default partition for a Fujitsu 130 Megabyte disk.

Table 3-1 *Sample of Fujitsu 130 MB Disk Partitions*

Partition	Starting Cylinder	Size In Sectors
a	0	15884
b	50	33440
c	0	262400
g	155	212800

The disk driver knows about partition 'a' the root partition, partition 'b' the swap partition, and partition 'g' the rest of the disk. Partition 'c' covers the entire disk and allows the disk driver to address the entire disk. For example, partition 'c' is

3.2. How UNIX Is Organized On The Disk

Some Basic Terms

The Sun UNIX operating system consists of the *kernel* and many hundreds of other files containing data and programs. At boot time the *kernel* is loaded into memory; it resides there until the system is shut down. Other files are loaded into memory as needed.

The kernel manages all of the physical resources of the workstation. Kernel functions include:

- Implements the *file system*, a tree-structured hierarchy of directories and files residing on disk or network disk or the network file system, and permits processes to create, read, write, and remove these files.
- *Virtual memory*, which allows up to 16 megabytes of unsegmented address space for each process by automatically moving 'pages' of information from disk to main memory as needed.
- The *scheduler*, which keeps track of all active processes and decides which gets to run next.
- *Device drivers*, software routines which control physical devices such as graphics display, mouse, keyboard, disk, tape, RS-232 serial ports, and Ethernet.
- *Networking software* which implements network functions such as the TCP/IP protocols and the network disk (*nd*) function which supports diskless workstations.
- *Interprocess communication* facilities such as signals and sockets.
- Facilities for creating, examining, and modifying processes.
- System management functions, such as halting, booting, and error handling.
- Miscellaneous functions which make system resources (like memory, timers, etc.) available to processes.

The kernel resides on the disk in a single file, typically known as `/vmunix`. (As you will learn, there may be other kernels with other names). We mentioned the many hundreds of other files comprising the UNIX system, most of these are *commands*, also called *utilities*, programs the user invokes directly. Other files contain *libraries*. These are collections of software routines which may be selected from the library and incorporated into another program. Libraries form an extension of the basic system features implemented by the kernel. Many files contain *data* used by the programs in other files.

The *Commands Reference Manual for the Sun Workstation* describes the commands available. Sections 1, 6, and 7 describe user commands including games and demos; section 8 describes the system administration and maintenance commands. The *System Interface Manual* describes system calls, library routines, special files, and file formats. Section 2 describes kernel calls or 'system calls;' section 3 describes library routines; section 4 describes the special files used for devices; section 5 explains some of the most important data files. The

`/stand` See `/usr/stand` for a standalone machine and `/pub/stand` for an `nd` server or client machine.

`/sys` See `/usr/sys`.

`/lost+found` `fsck`(8) uses `/lost+found` as a repository for unreferenced files it finds on the disk. Normally this directory remains empty, but some files may appear in it after running `fsck` on a damaged file system. See also `/usr/lost+found` (below) and `fsck`(8).

`/mnt` Normally empty, `/mnt` is typically used to mount file systems temporarily.

`/private` On diskless clients and `nd` servers, `/private` contains certain files which would otherwise appear in the shared file system `/pub` (see below), but which must differ from one machine to another.

For example, the directory `/usr/lib` (see below) is shared by server and clients. However, the file `/usr/lib/aliases` is itself a symbolic link to `/private/usr/lib/aliases`, and a separate copy of it exists on each machine.

`/private/usr/adm`
This directory contains system accounting files. See `ac`(8) and `sa`(8) for more details.

`/private/usr/preserve`
When one of the editors (`vi`, `ex`, `edit`, or `e` — which is actually one editor running in four different modes) is interrupted without having a chance to exit normally (for example, in a system crash) the editor attempts to save as many of the user's changes as possible. These can be recovered later using the `vi -r` command. The `/private/usr/preserve` directory is used to preserve the necessary temporary edit files, pending recovery.

`/private/usr/spool`
This contains a number of subdirectories which may be characterized as `spool directories`. Spool directories are repositories for files pending further processing, typically by a program other than the one which placed the file into the spool directory. For example, `/private/usr/spool/lpd` holds files printed by the `lpr` command pending output to the printer; `/private/usr/spool/mail` contains users' 'system mailboxes,' which contain incoming mail pending reading by the recipient; and `/private/usr/spool/uucp` contains `uucp` data files and work files in transit to or from other machines.

`/usr/hosts` Contains a script called MAKEHOSTS, which creates, for each host in `/etc/hosts`, a symbolic link to the `rsh` command which is the hostname itself. For example, after running MAKEHOSTS and adding `/usr/hosts` to your '$path' shell variable, you can just type `sundial` instead of typing `rlogin sundial`.

`/usr/include`
This directory contains all of the standard 'include files' or 'header files' used to compile C programs. These files, whose names conventionally end with `.h`, contain definitions of useful constants and macros. Particular `.h` files are often explained by entries in sections 2 and 3 of the *System Interface Manual.*

`/usr/lib` This directory contains over a hundred files used by UNIX utilities. More varied than any of the directories described here, it is kind of a 'catch-all' for UNIX utility files that didn't quite belong anywhere else.

These include libraries (names ending in `.a`, see `/lib` above for description) supporting Pascal, SunCore, Sunwindows, and other system functions; programs used by various system utilities (for example, `f77`, `lint`, `lex`, `spell`, etc.); `troff` macros (`tmac`, `me`, `ms`); Pascal and Fortran; line printer filters; and more.

`/usr/local` This directory, not on all machines, is not a standard part of UNIX, but the system administrator often creates it to hold commands, programs, or other files. These might be obtained from third parties (such as the Sun User Group) and added to the system after installation. Usually, every user adds `/usr/local` (and/or `/usr/local/bin`) to his or her '$path' shell variable to take advantage of these local commands.

`/usr/lost+found`
If `/usr` is a separate, mounted file system, this directory serves the same purpose as `/lost+found` (see above) does for the root file system.

`/usr/man` Holds the on-line manual pages. See `man`(1), as well as the manual *Installing UNIX On The Sun Workstation*, the section on installing manuals, demos, and games.

`/usr/mdec` Contains boot programs and a script for installing them. Usually not touched after system installation, except sometimes when restoring a damaged root or `/pub` file system.

`/usr/pub` The pub here stands for publishing and has nothing to do with the `/pub` directory above. Do not confuse the two. `/usr/pub` contains miscellaneous tables including a table of the ASCII characters and tables used by the `nroff` and `troff`

3.3. Backing Up File Systems With dump

The need for frequent and regular file system backups is discussed in the chapter *Periodic Maintenance* in this manual. Here we explain the mechanics of backups with dump(8). Before reading further, look at the dump page in the *Commands Reference Manual* to familiarize yourself with options mentioned below but not explained in great detail.

Reel Tape vs. Cartridge Tape

Disk files are backed up onto tape at regular intervals to preserve information that could be lost if disk problems occur. Your workstation will use either 1/2 inch reel or 1/4 inch cartridge tape, depending on the type of tape controller you have installed. A tape controller that writes high density tapes is available for 1/2 inch reel tape — but only with a Xylogics 472 controller. If you have this optional high density configuration, you can write more bits per inch (6250 vs. 1600) to tape and store more information on each tape. In addition, there are two available 1/4 inch tape controllers — the Sun archive tape controller and the SCSI tape controller. Both 1/4 inch tape controllers write at the same density. In general, the dump command works the same with every type. But, as you might expect, you must specify different device names and, in some cases, different flags for the different types of tape controller.

A generic command to dump to tape looks like:

```
# /etc/dump flags tape_device_name filesystem_to_dump
```

The *tape_device_name*'s for the controllers mentioned above are: `mt0`, 1/2 inch reel 1600 bpi; `mt8`, 1/2 inch reel 6250 bpi; `ar0`, 1/4 inch archive; and `st0`, 1/4 inch SCSI. So, specifically, the command to dump a 1/2 inch reel tape at 1600 bpi (`mt0`) looks like:

```
# /etc/dump flags /dev/rmt0 filesystem_to_dump
```

And the command to dump a 1/2 inch reel tape at 6250 (`mt8`) looks like:

```
# /etc/dump flags /dev/rmt8 filesystem_to_dump
```

While the command to dump a 1/4 inch archive tape (`ar`) looks like:

```
# /etc/dump flags /dev/rar0 filesystem_to_dump
```

And the command for a 1/4 inch SCSI tape (`st`) looks like:

```
# /etc/dump flags /dev/rst0 filesystem_to_dump
```

In each of the dump lines for 1/4 inch cartridge tape (and only for cartridge tape!) you must include the 'c' flag — which tells dump it is using a cartridge tape. Additionally, in the case of cartridge tapes, you should not use the default blocking factor; the dump would take far too long. Use the 'b' flag and give a blocking value of 126. (Note that you must `restore` a tape with the same blocking factor at which you dumped it.) For example, to do a level 9 dump on a SCSI tape of the file system `xy0g` blocked at 126, and upon completion write a message to the record file `/etc/dumpdates`, you would type the following:

```
# /etc/dump 9ucbf 126 /dev/rst0 /dev/rxy0g
```

Note that *flags* are grouped together without intervening spaces. We recommend always dumping the raw device (here `rxy0g`), it goes much faster.

device, and `rndl0` refers to the raw device. Raw devices are always dumped, see the examples below.)

```
user bill 0 /dev/xy0g 108560 39560 0
user bill 1 /dev/xy0g 148120 12880 -1
user debby 0 /dev/xy0g 161000 39560 1
user debby 1 /dev/xy0g 200560 12880 -1
user joan 0 /dev/xy0g 213440 39560 2
user joan 1 /dev/xy0g 253000 12880 -1
```

If possible, halt the client machine during incremental dumps (at least avoid dumping any heavily used machine). Always make sure the client is halted during level zero dumps and when `fsck` runs on its partition.

For more information about client partitions, see the section *The Network Disk* in the chapter *Sun Network Services*.

Sample Scripts For File System Dumps

The example shell scripts given below can be used for doing routine backup dumps of the file systems on disk servers and, where applicable, on standalone systems.

Before doing a level zero dump on any file system be sure to run `fsck`(8), the UNIX file system check program. This insures that a file system has no discoverable inconsistencies before it is dumped to tape. `fsck` is a multi-pass file system check program. Each file system pass invokes a different phase of the program. In successive passes, `fsck` checks blocks and sizes, path-names, connectivity, reference counts, and the map of free blocks (possibly rebuilding it), and performs some cleanup. For further information see `fsck`(8) in the *Commands Reference Manual*, and the *FSCK Tutorial* in the tutorial section of this manual. You should also run `fsck` whenever you have restored a complete file system (see the section below on `restore`). Remember to address the `ndl` partition properly when using `fsck`.

In the examples below we give one example for 1/2 inch tape and another for 1/4 inch tape.

In these examples, the `n` preceding the drive type indicates no rewind after the file system has been dumped. The `mt offline` rewinds the tape, and takes it off line. The `r` in the `rxy0a` (and in the other `rxy` and `rndl` file system names) indicates that the raw device should be dumped. We recommend that you always dump the raw device since it dumps considerably faster than the block device (`xy` or `ndl`).

The first example works on a server or standalone machine. It performs a full dump of a root file system, and level 9 incremental dumps of three other file systems: partitions 'e', 'f', and 'g'. On a server, partition 'g' would typically contain `/pub`; on a standalone machine, partition 'g' would typically contain `/usr`. On a server machine, under NFS, partition 'e' typically contains `usr2`. In each case, partition 'f' is locally defined. These dumps are written consecutively on a single volume of tape. This script does not take arguments; it has been determined that the dumps of these file systems will fit on the single volume of tape.

```
#
# Example 2: level zero for chosen clients on 1/2 inch tape
#
for i in $*
do
date
echo dumping ndl$i
/etc/dump 0uf /dev/ndrive /dev/rndl$i
done
echo DONE unloading tape
mt offline
```

For 1/4 inch tape drives (remember to substitute for *drive* just as above):

```
#
# Example 2: level zero for chosen clients on 1/4 inch tape
#
for i in $*
do
date
echo dumping ndl$i
/etc/dump 0ucbf 126 /dev/ndrive /dev/rndl$i
done
echo DONE unloading tape
mt -f /dev/drive offline
```

Assume this script is an executable file called 'example.2'. After mounting the tape type:

```
# example.2  n n n ...
```

The *n* arguments passed at execution are taken from the `/etc/nd.local` file on the server. In that file look at the end of the `user` lines for the clients you want to dump. For example, if `/etc/nd.local` looks like:

```
user joan 0 /dev/xy0g 213440 39560 2
user joan 1 /dev/xy0g 253000 12880 -1
user avb 0 /dev/xy0g 265880 22080 3
user avb 1 /dev/xy0g 287960 39560 -1
```

Then,

```
# example.2 2 3
```

would dump the partitions joan and avb. You must first determine that all the partitions dumped will fit on the tape volume — use `df` as mentioned above in the section *Reel Tape vs. Cartridge Tape.*

The final example performs level 9 incremental dumps for predetermined `ndl` partitions, 0 through 4 in the example script. The script takes no arguments, but you must predetermine that all the partitions will fit on one volume of tape.

For 1/2 inch tape drives (remember to substitute for *drive* just as above):

3.4. Restoring Files With `restore`

`restore`(8) restores files from tapes created by `dump`(8). You may use `restore` to reload an entire file system hierarchy from a level zero tape and incrementals that follow it, or to restore one or more single files from any `dump` tape.

Always make sure you restore from the correct tape. After you mount a tape, type `restore t` to read the tape creation date and other information.

For a complete discussion of `restore`'s functions and the options to those functions, see the `restore`(8) page in the *Commands Reference Manual*. Below we discuss three typical uses of `restore`.

Restoring One Or More Particular Files

Commonly, a user will lose a file or files through carelessness, a faulty program, or the improper use of a program. Cases like this require that the system administrator be able to restore individual files or small groups of files.

First find the tape on which the missing file(s) is likely to have been dumped in a state that most suits the user's needs. Note that this is not necessarily the most recent backed up version of the file. Sometimes a user wants last week's program that she forgot to save a copy of before she began to make this week's 'improvements.' Keep familiar with the storage and organization of backup tapes at your site so you can locate tapes months or years old. It becomes particularly important when you must restore a subdirectory containing many files worked on over a long period of time. Frequently, in a case like this, the needed versions of files are not on a single dump tape.

You should almost never restore a file to its original directory. Either make a new directory to hold restored file(s) (and make sure that the user owns it), or change directory to `/usr/tmp` and restore the file(s) there. The name(s) of the restored file(s) is the name on the tape (the full pathname from the root of the dumped file system) relative to the current working directory at the time of the restore. The system administrator should probably let the user change the restored file to whatever name he or she chooses, since a currently existing file in the user's directory may bear the same name as the old restored file. The system administrator can send mail to the user, something like:

```
A copy of your file "john/prog.c"
from a dump tape of June 8, 1984
was restored as "/usr/tmp/john/prog.c"
```

After files have been restored, check to make sure ownerships and permissions have stayed as they are on the dump tape (or have been changed if you want to change them).

Below are three `restore` options particularly useful when restoring small numbers of files, or an entire directory and the subdirectories under it. See the `restore` *Commands Reference Manual* entry for all function options.

- `restore t` *filename(s)* — lists the names of specified files if they occur on the tape; useful for quickly finding out if a certain file(s) is on a particular tape. Without the *filename* argument, it lists the entire hierarchy of the tape.
- `restore x` *filename(s)* — restores the named file(s) from the tape. If *filename* is a directory, it recursively extracts the directory.

3) Mount the file system on an existing directory prior to reloading. In this example `/mnt` already existed. Use the block device, not the raw device here. **Note:** If this is a client `ndl` partition, make sure the client machine is halted.

```
# /etc/mount /dev/device /mnt
```

4) Change to the `/mnt` directory and restore the level zero tape.

For the 1/2 inch tape drives (substitute for *drive* one of the following — `rmt0` for density 1600 bpi; `rmt8` for density 6250 bpi):

```
# cd /mnt
# restore rvf /dev/rdrive
```

And for 1/4 inch tape drives (substitute for *drive* one of the following — `rar0` for archive tape controller; `rst0` for SCSI tape controller):

```
# cd /mnt
# restore rvbf  blksize /dev/rdrive
```

Continue to restore incremental tapes in the order of lowest level number first, working up to the most recent incremental tape last.

5) Remove a file that `restore` creates in the current working directory during its operation; it is called `restoresymtable`.

```
# rm restoresymtable
```

6) Unmount the file system and check it again with `fsck`. Remember to check the raw device by typing **r***device*.

```
# cd
# /etc/umount /dev/device
# /etc/fsck /dev/rdevice
```

7) Do a full dump (level zero) of the newly restored file system.

For a 1/2 inch tape drive:

```
#  /etc/dump 0uf /dev/rmt0 /dev/rdevice
```

And for 1/4 inch tape drives:

```
# /etc/dump 0ucbf 126 /dev/rdrive /dev/rdevice
```

Substitute for **r***drive* one of the following — `rar0` for archive tape controller; `rst0` for SCSI tape controller.

Restoring A Damaged 'root' Or `/pub` File System

Restoring a damaged 'root' or `/pub` file system from a dump tape presents a special case because the utilities needed are in the damaged file system. The general strategy is to load and boot the mini-UNIX file system from the boot tape, and reload the file systems from there.

Specifically, for a 'root' file system do the following:

```
# cd
# umount /dev/disk_device0a
# /etc/fsck /dev/rdisk_device0a
```

10) Do a full dump (level zero) of the newly restored root file system.

For a 1/2 inch tape drive:

```
# /etc/dump 0uf /dev/rmt0 /dev/rdevice0a
```

And for 1/4 inch tape drives:

```
# /etc/dump 0ucbf 126 /dev/rdrive /dev/rdevice0a
```

Substitute for *rdrive* one of the following — `rar0` for archive tape controller; `rst0` for SCSI tape controller.

If this is a standalone system, or a server on which the `/pub` partition does not also need to be restored, you can reboot the system at this point.

If you are on a standalone system, and `/usr` has been damaged, you can restore it after booting single-user by following steps 4 through 8 above, making sure you use the proper `/dev` device.

However, if you are on a server and `/pub` has been damaged, you must restore it before booting single-user, because not all of the utilities you need will be present. As above for 'root,' you start by loading and booting the mini-UNIX file system — step 2 above. The procedure is the same as for the 'root' file system, except that in step 4 you must create the file system with `/etc/mkfs` instead of `/etc/newfs` (`newfs` would allocate all of the 'g' partition to `/pub`, destroying the contents of the client `ndl#` partitions). The syntax for `/etc/mkfs` is explained in the section immediately above, *Restoring An Entire File System*; see step 1 there.

Once you have restored `/pub`, you need to install a new boot block for clients to boot from; use the commands:

```
# cd /usr/mdec
# installboot bootnd /dev/rxy0g
```

You can boot the system and proceed with the steps for reloading entire client partitions (that is, if any clients must be restored, for example if the whole `g` partition was wiped out). For restoring clients, follow the slightly different steps in the section above, *Restoring An Entire File System.*

Restoring Files Over Ethernet

You may restore files to a machine where you are logged in as superuser, from a tape mounted on a remote machine's tape drive. To make things work right, there must be an entry for the machine which you are logged into and restoring to, in the `/.rhosts` file on the machine with the tape drive. While superuser on the machine onto which you want to restore files, type:

```
# /etc/restore xf remote_machine:/dev/drive filename(s)
```

For 1/4 inch tape drives, remember to use the `b` flag, to duplicate the blocking factor given to the `b` flag of `dump` at dump time. There is no `c` flag for cartridge

3.5. Maintaining File Systems

Disk Capacity — Checking The Disk Resource Usage

Several commands will tell you about the current use of disk resources. Three commands will tell you the size of files in a file system, ls(1), du(1) and quot(8).

- ls gives you the size in kilobytes of the files in a directory when you pass the -s option, and recursively lists subdirectories with the -R option. To find the largest files in the current working directory, type:

  ```
  % ls -s | sort -nr | more
  ```

  To find out which files were most recently written, use ls -t. It lists files in order of most recently created, or altered, first.

- du will give the number of kilobytes contained in all files and, recursively, directories within each specified directory or file named. Note that du follows the symbolic links in a file system.

- quot must be executed by the superuser, and gives the number of blocks currently owned by each user in the named file system.

These three commands allow you to quickly find large files. If disk storage space is short, you can move unused large files to a less expensive medium like tape.

df — Show Free Space

The df(1) command returns information about specific disk partitions, including the file space occupied by each file system, the space used and the space available, and how much of the file system's total capacity has been used. When df gives the percentage of the capacity used on the disk, the number indicates the optimum capacity. When a file system is 90% full, df claims that the file system is 100% full. If the file system becomes any fuller, file system access slows down. In fact, UNIX permits file systems to become only 90% full unless you are the superuser. Ordinary users get an error if they go beyond this limit.

Please note that df will only return values for the first soft partition when a hard partition has been divided into a number of soft partitions. For example, only /pub shows up on an nd server's xy0g even though that disk also contains clients' soft partitions. This happens because df expects to read a file system, but reads instead the header for the first soft partition. Even though the header defines a file system smaller than the hard partition, df only returns values for the first soft partition.

Finally, there are two ways to find out how the disk is divided into hard and soft partitions. The verify command of diag(8) gives information about the size of the hard partitions on the disk. Run dkinfo(8) to obtain the same information without using diag. See the description of diag for more information. In addition, /etc/nd.local shows how a hard partition is divided into soft partitions, and the amount of space allocated to each soft partition. See nd(8) for a description of the contents of /etc/nd.local.

The `quotaoff(8)` command will disable quotas — see man page for `quotaon(8)`. However, when a file system is about to be unmounted by `umount(8)`, the `umount(2)` system call disables quotas just before the unmount of the specified file system; this guarantees that the quota system will not be disabled if the unmount fails because the file system is busy.

Note that while the disk quota system is typically run on an NFS server, as shown in the examples above, you may also set up quotas on a standalone or timesharing machine.

Some Implementation Detail

Disk quota usage and limits are stored in the file `quotas` on the root of the file system where the quotas are imposed. This name is not known to the system in any way. However, several user level utilities depend on it, so choosing any other name would cause problems.

The data in `quotas` is an array of structures, indexed by `uid`, with one structure for each user on the system (whether the user has a quota on this file system or not). If the `uid` space is sparse, the file may contain holes which would be lost by copying. Therefore, avoid copying the file.

The `quotactl(2)` system call informs the system of the existence of the `quotas` file. Each subsequent open of a file on the file system will be accompanied by a pairing with its quota information. In most cases this information will be retained in core, either because the user who owns the file is running some process, or because other files owned by the same user are open, or because some file was recently accessed. If the information is not in memory, the quota file is read to retrieve it. In memory, the quota information is kept hashed by `uid` and file system, and retained in an LRU chain so that recently released data can be easily reclaimed.

Each time a block is allocated or released, and each time an inode is allocated or freed, the quota system gets told about it. The quota system gets an opportunity to object to any allocation.

The quota code uses a very small percentage of the system cpu time consumed in writing a new block to disk.

Administration Of The Quota System

All quota administration commands must be run on mounted file systems. The commands will work whether or not the quota system is disabled.

The Sun disk quota system is based on the 4.2BSD quota system. Sun's quota system limits the amount of time a user can be over quota. That time-limit may be adjusted on a per-file-system basis using `edquota(8)`.

Periodically (certainly after each reboot, and when quotas are first enabled for a file system), the records retained in the quota file should be checked for consistency with the actual number of blocks and files allocated to a user. Do this using `quotacheck(8)`; it is typically run from `/etc/rc.local` before `quotaon` — see the man page for details.

The super-user may use `quota(1)` to examine the usage and quotas for any user, and `repquota(8)` to check the usages and limits for all users on a file system.

file system where it belongs.

Quotas And The NFS

Quotas on remotely mounted file systems work much the same as quotas on locally mounted file systems. The important difference is that no warnings are printed when the user goes over the soft limit. NFS clients should use `quota(1)` to find out the state of their quota allocation. Hard limit errors are treated like other hard errors in the NFS — for example exceeding actual disk capacity. When a user exceeds the hard limit she will get return code errors, not system error messages. The user may see `EDQUOT` error return codes from the system calls `write(2)`, `fsync(2)` and `close(2)`. Check these return codes to avoid disaster.

Making Room On File Systems

If you begin running out of room on a file system you need to free up some of the disk's storage space. If you get a message `filesystem full` on the system console of a running system, you should suspend or kill the currently executing programs and try to make more room on the affected file system. Of course it is better to monitor disk usage and not be caught in this situation, but that is not always possible. For systems that are filling up, or that become full suddenly, you will need to delete some files or move some files to another file system. These hints that may be useful.

1) Read the section above on *Disk Capacity*, then determine the largest files, their locations, and who owns them.

2) When UNIX crashes, it saves a core dump (roughly equal in size to the amount of physical main memory on your system) in a portion of the swap area on the disk. At the next reboot, `savecore(8)`, which runs from `/etc/rc.local`, copies this core dump into files in the directory `/usr/crash`. In the `nd` server-client environment, core dumps are put in the directory `/usr/`*`servername`*`/crash/`*`hostname`* where *servername* is the name of the `nd` server, and *hostname* is the name of the machine the core dump is from. If you have a series of system crashes, then `/usr/crash` will keep accumulating core dump files until the file system fills up. The affected file system will typically be the `usr` partition (xy0g or sd0g) on standalone systems or the `/usr/`*`servername`* partition on `nd` servers and diskless clients.

   As distributed, the section of `/etc/rc.local` which creates the core dump file is commented out. Should you wish to get the core dumps without using up too much disk space, you can create a file called `minfree` in `/usr/crash`. `savecore` reads from this file, and if the file system contains less free space in kilobytes than the ASCII number contained in `minfree`, the core file will not be saved.

3) The directory `/usr/adm` contains accounting information. (Note that in the `nd` server-client environment, `/usr/adm` is a symbolic link to `/private/usr/adm`.) Check to see if any of those files are growing inordinately large. In particular, `/usr/adm/lastlog` can become enormous if large user id numbers (> 32K) are used in `/etc/passwd`. You are advised to avoid very large user numbers in `/etc/passwd`. You can disable login accounting by removing the file `/usr/adm/wtmp`. You can

4

Communications

4

Communications

This chapter discusses communications between machines. It begins with the two types of communications networks that your workstation can be connected to.

The first is a high-speed local area network (LAN), the Ethernet; the Ethernet allows machines in close proximity to share system resources such as disks and tapes. The chapter covers Ethernet terminology for both hardware and software, and explains Ethernet installation and debugging.

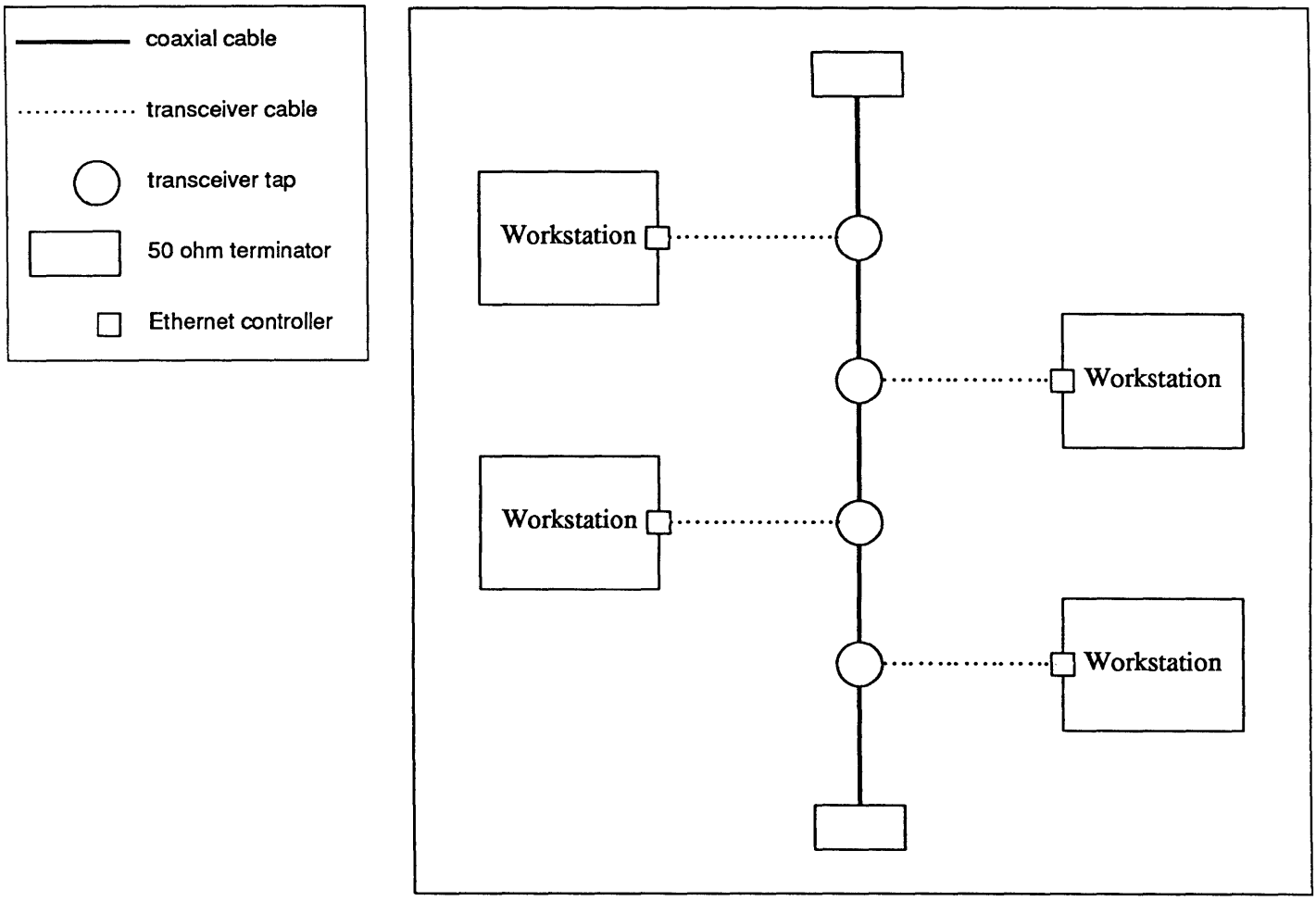
The second network type is a lower-speed wide area network that works over phone lines and allows distant machines to communicate with one another. Two aspects of wide area networks are introduced and explained in this section: `uucp`, a series of programs that allows UNIX systems to communicate with each other over dial-up or hardwired lines.

Next, a section covers the installation of `sendmail`, the electronic mail routing program for the UNIX system.

Finally we explain `tip`, which allows you connect to a remote machine as if you were logged in directly.

6) Both ends of a coaxial cable are terminated by 50 ohm terminators with insulated outside covers.

Figure 4-1 *A Local Network*

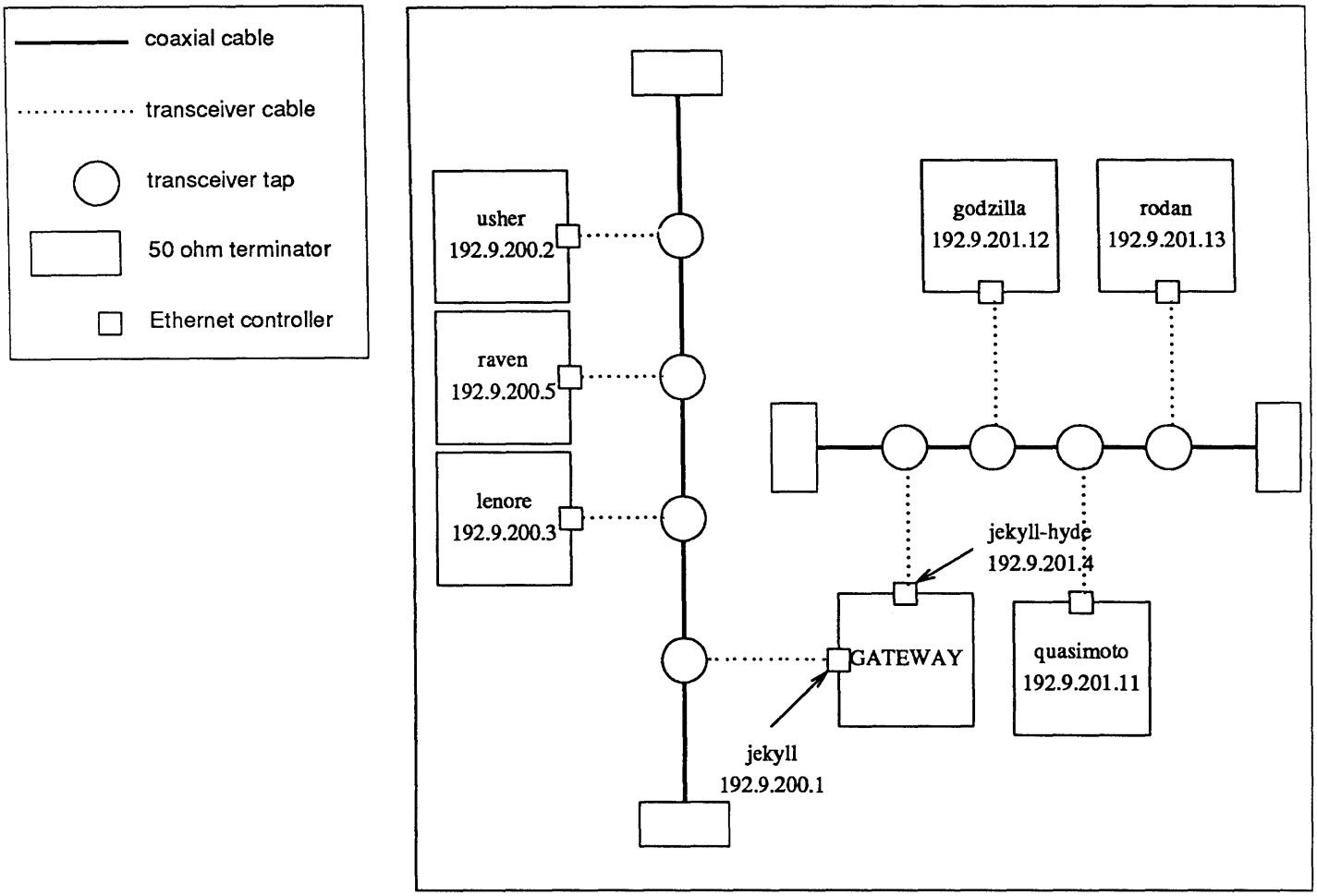


Example Of A Network Transfer

Here is a breakdown of file transfer over the Ethernet.

1) A workstation user runs a file transfer program and specifies a file to be transferred between a sending and receiving device.
2) Software maps the file's characters into device-independent virtual characters to comply with protocol specifications.
3) The mapped character stream is routed to a virtual circuit, set up between the two devices.
4) The virtual circuit software breaks the character stream into packets for transmission.
5) The packets are passed to the Ethernet driver software.

Figure 4-2 *Two Local Networks Joined By A Gateway Machine*



To set up a proper configuration for a gateway machine you must edit `/etc/hosts` on the `yp` master server and then do a `ypmake` so that all yellow pages clients will see the change. You must also edit `/etc/rc.boot` on the gateway machine. Follow the four steps below.

1. On the `yp` master server machine edit the `/etc/hosts` database. You need to add a line for the gateway machine's hostname and address on the 'second' (or other) local network. For example, say 'jekyll' will be a gateway machine between two local networks. Remember, it must have two Ethernet interfaces. Before the change to gateway status, several lines of the `/etc/hosts` database on the yellow pages master server might look like this:

numbers are unique on both networks, as they must be for an nd server that is also a gateway machine. Precautions, such as a comment in the /etc/hosts, database should be taken to make sure these unique host numbers are not inadvertently assigned in the future. For example:

```
# Local Net 192.9.200 -- 10Mb/s Ethernet -- Engineering
#
192.9.200.1       jekyll loghost datehost
192.9.200.2       usher
192.9.200.3       lenore
# host number 192.9.200.4 taken by server-gateway on
# Marketeering Local Net: DO NOT USE
192.9.200.5       raven
[ etc. ]
# Local Net 192.9.201 -- 10Mb/s Ethernet -- Marketeering
#
# host number 192.9.201.1 taken by server-gateway on
# Engineering Local Net: DO NOT USE
192.9.201.11      quasimoto
192.9.201.12      godzilla
192.9.201.13      rodan
192.9.201.4       jekyll-hyde
[ etc. ]
```

3. Finally, run ypmake on the new /etc/hosts database to propagate it to all machines. Make sure you are in the directory /etc/yp and type the following:

   ```
   # make hosts
   ```

   This will update the /etc/hosts database and yppush it to the slave databases on the network. For details see ypmake(8).

4. On the new gateway machine, edit /etc/rc.boot, and add an ifconfig line mapping the machine's second hostname and Ethernet Board. Taking our example above, here is the relevant line from the file prior to addition:

   ```
   /etc/ifconfig xx0 jekyll -trailers up
   ```

   and the way it looks after the addition:

   ```
   /etc/ifconfig xx0 jekyll -trailers up
   /etc/ifconfig xx1 jekyll-hyde -trailers up
   ```

   For *xx* above, you substitute the proper Ethernet controller type, either ec for a 3COM controller or ie for a Sun-2 controller.

Reducing Network Overhead

As you read the following section, keep in mind that we strongly discourage the use of rwhod(8C).

You can have most of the advantages of living in a network environment without running the network daemons — rwhod(8C) and routed(8C) — on every machine. For systems with only 1 MByte of local memory, we advise disabling the daemons, or running them only on specific machines, so the space consumed by them can be made available for other purposes. When running, routed and

redirect packets going to another accessible network through this gateway machine. To do this, edit the `/etc/rc.local` file on all machines **except** the gateway (*gateway_name*, in the example). Find the line that says:

```
echo -n 'local daemons:'    >/dev/console
```

Insert the following two lines just before it:

```
/usr/etc/route add 0 gateway_name 1
echo '  /usr/etc/route add 0 gateway_name 1'>/dev/console
```

Then, find the following three lines and comment them out (insert a '#' before each line) or remove them:

```
if [ -f /etc/in.routed ]; then
    /etc/in.routed & echo -n ' routed'  >/dev/console
fi
```

4. Finally, if you have only 1 MByte of memory **and** you have more than one gateway in your local network **and** you are running diskless, you can use your `nd` server's routing tables. (Of course, the server must run `routed` in this case so that clients can use its routing tables.) Edit the `/etc/rc.local` file on all clients of the server machine (*server_name*, in the example), and add the following lines:

```
/usr/etc/route add 0 server_name 1
echo '  /usr/etc/route add 0 server_name 1'>/dev/console
```

Then comment out or remove the daemon's lines:

```
if [ -f /etc/in.routed ]; then
    /etc/in.routed & echo -n ' routed'  >/dev/console
fi
```

Since a diskless node cannot run when its server is down, this always works; however, it causes packets to be sent through the server, loading it down.

In all other cases you should run `routed`.

Network Security — `/etc/hosts.equiv` and `.rhosts`

Network security is implemented at two levels: first, at the machine or node level, and, second, at the individual user level. The `/etc/hosts.equiv` and `.rhosts` files, respectively, control access at these levels. Remember that with the yellow pages `hosts.equiv` and `passwd` files are looked at in the yellow pages database on the `yp` server. If a machine does have its own copy of either of these files, the local copy will be looked at instead of the `yp` database. The security-checking process goes like this:

- When a user initiates a remote process on another machine (`rlogin`, `rsh` or `rcp`, for example), the system first checks for an entry for this user in `/etc/passwd` on the remote machine. If no entry is found, the user will be denied access: if he is trying to `rlogin` to the machine, he will be prompted for a password and then get a `Login incorrect` message; if he is attempting an `rcp` or `rsh`, he will get the `Login incorrect` message.

```
canard donald
```

Note also, that this means `donald` only has access when he's coming from `canard`; if he tries to use `gaia` from another host, he must know his password to be able to `rlogin` and can't complete remote processes. Of course, if he can `rlogin` he can always doctor his `.rhosts` file (if it is not owned by root, and I have given him a home directory). This brings up two points:

- The only way to achieve anything resembling security in a hostile environment is to exclude users from the `/etc/passwd` file; once someone knows a password, he's in. If this concerns you, you should also be especially careful to protect `/.rhosts` properly — make sure only root has write permission. Clearly, tight security has not really been an issue in the administration of previous UNIX systems; the goal has been to facilitate, rather than deny, access. See the section on *Yellow Pages Administration* for more information about how to edit the password file to exclude users when you run `yp`.
- If you have a friendlier environment, the easiest way to administer machine access at the user level is to give each 'trusted' user an account (in `/etc/passwd`) and a home directory on your machine(s), and ask each user to create his/her own `.rhosts` file in his/her home directory on the machine(s).

Finally, a caution about editing the `/etc/hosts.equiv` and `.rhosts` files. An entry in either of these files is invalid if it contains trailing white space — blanks or tabs. The presence of trailing white space may not be obvious. You can look at a file with `cat -e`, or `vi` with the 'set list' option, both of which will show a `$` at the end of each line. You may also `grep` a file searching for 'tab_character$' or 'blank_space$' patterns to get a listing of any lines ending in tabs or blanks.

Special Cursor Characters During Ether Boot

When booting over the Ethernet, you will notice that the cursor character changes from its usual solid block form to some other character. For example:

—, = The cursor alternates between '—' and '=' to indicate that the boot is proceeding normally.

`X` An Ethernet packet-transmission error has occurred.

`?` An Ethernet packet-receive error has occurred; or a packet has not been successfully received for two seconds; or the network or `nd` configuration files are incorrect.

None of these characters appears when booting from disk or tape.

The `X` or `?` characters are rare. You may see the `?` sometimes when booting a number of clients simultaneously on the same server. If the `X` or `?` appears frequently when booting, and you have ruled out errors in the configuration files, it probably indicates an Ethernet hardware malfunction which should be corrected.

2. A second solution is to retain your Class A addressing scheme from 1.0, but make sure all host numbers are smaller than 1024, so that ARP can be performed.

   Follow the `/etc/hosts` editing procedure outlined in step 1 above, and assign each machine a host number — the last component of the entire address — under 1024. You can simply retain your old network number. Remember to do the `cd` and `make` commands, and to reboot every machine whose host number you changed.

3. Finally, if you have machines that either cannot perform ARP (old Vaxen, for example), or cannot respond to ARP (older Sun systems with Class A networks and host numbers greater than 1024, for example), there is a solution for each problem.

   First, machines which cannot perform ARP may talk to a post 1.1 network by 'forcing' your post 1.1 machines to respond to an address which non-ARPers understand. Do this by adding an `ifconfig` line to the `/etc/rc.boot` file on each post 1.1 machine. The new line contains the post 1.1 machine's 6-byte hexadecimal Ethernet address:

   a) You need the Ethernet address. To get it, in most cases, power on the Sun Workstation and, as soon as the self-test completes, abort. Then read the Ethernet address from the PROM monitor's sign on messages. (Remember that the correct abort sequence depends on your keyboard, as described in the manual *Installing UNIX On The Sun Workstation.*)

      A few sites may have very old machines that require a different process. For machines with PROMs of Rev. L or earlier, or for machines without CPU ID PROMs, you obtain the Ethernet address as follows. For a 3COM Ethernet Board use the PROM Monitor to interrogate the Multibus memory-space where the address is stored. As super-user, run the following sequence of commands (type <RETURN> where indicated):

      ```
      # /etc/fasthalt
      syncing disks . . . done
      Unix Halted
      > k1
      > e fe0400
      > FE0400:   0260? <RETURN>
      > FE0402:   8C00? <RETURN>
      > FE0404:   9920? q
      ```

      Note, if your machines have Sun-2 Ethernet Boards, this simply won't work. Instead, you can fabricate 3COM addresses for these machines: any six-byte hexadecimal number will do, as long as it is **unique** on your network.

   b) When you have the Ethernet address for a post 1.1 machine, edit the `/etc/rc.boot` file and add the following line for every pre 1.1 machine you want to communicate with:

      `/etc/ifconfig` *`e_intface#`* *`ethernet_address`* `` `hostname` -trailers up``

Software Checks

1) Check `/etc/hosts` on the `yp` master server machine to make sure that the entries are correct and up to date. Make sure a correct copy has been sent to all `yp` slave servers. Check the `ether` addresses in `/etc/ethers` on the `yp` master server machine to make sure that the entries are correct and up to date. Make sure a correct copy has been sent to all `yp` slave servers. Note, different rules apply to pre 1.1 Sun software releases.

2) If you cannot boot a client, you try specifying the host number of the server in the boot command. The syntax is:

```
> b xx (0,#,0)vmunix
```

Where *xx* is either `ec` for a 3COM Ethernet controller, or `ie` for the Sun-2 Ethernet controller; and where # is the host number. If this works, while the default boot does not, the server files are not configured correctly. Check `/etc/hosts` on the `yp` master server machine and/or `/etc/nd.local`.

3) Check the accuracy of the `/etc/ifconfig` line(s) added to the `/etc/rc.boot` file.

4) On a standalone host or a server, try `rlogin` to yourself. Make sure the network daemon `inetd` is running on the machines that want to talk to each other.

5) Run `/etc/nd` 'by hand' to produce error messages if there are errors. For example,

```
# /etc/nd < /etc/nd.local
```

5) `netstat`(8), with the `-i` option, tells you how many packets a machine thinks it is transmitting and receiving on each local network. For example, on a server you may see the input packet count increasing each time a client tries to boot, while the output packet count remains steady. This suggests that the server is seeing the boot request packets from the client, but does not realize it is supposed to respond to them. This might be caused by an incorrect address in `/etc/hosts` or `/etc/nd.local`. On the other hand, if the input packet count is steady, the machine does not see the packets at all, which suggests a different type of failure, possibly a hardware problem.

Hardware Checks

1) With its host system powered on, after awhile, each transceiver should feel slightly warm to the touch. A cold transceiver probably is not receiving power. This could indicate one of several things: a loose connection on either end of the transceiver cable, a loose connection of the internal Ethernet cable to the Ethernet board, or a faulty cable, transceiver (less likely), or Ethernet board (even less likely).

2) Check the solidity of all Ethernet coaxial and transceiver cable connections. If boards were changed or moved in the workstation card cage, make sure that the internal connector (from the back panel) is plugged into the Ethernet

4.2. Wide Area Networks

An Overview Of uucp

uucp (UNIX to UNIX copy) is a series of programs designed for communication, via dial-up or hardwired lines, between two systems running UNIX. uucp may be used to transfer files between UNIX systems, and also to run commands on remote machines. For more detailed background, see the *Uucp Implementation Description* in the *Tutorials* section of this manual.

Support for uucp is located in three major directories: /usr/bin (which contains user commands), /usr/lib/uucp (operational commands), and /usr/spool/uucp (spooling area).

(PLEASE NOTE: In the nd server — diskless client environment, the directories /usr/lib/uucp and /usr/spool/uucp are exported from the nd server. Since all machines share these file systems, machine specific files cannot reside in them. Instead the directories /private/usr/lib/uucp and /private/usr/spool/uucp contain the files that would normally reside in /usr/lib/uucp and /usr/spool/uucp. During installation setup replaces the files in them by symbolic links to the files in /private versions so the files can still be referenced by their normal names. If you are working in the nd server — client environment, remember to use instead the corresponding file in /private/usr/lib/uucp or /private/usr/spool/uucp when you follow the instructions below.)

The uucp commands are:

/bin/rmail	receive remote mail	rmail(8)
/usr/bin/rnews	receive remote news	
/usr/bin/uucp	file-copy command	uucp(1C)
/usr/bin/uux	remote execution command	uux(1C)
/usr/bin/uusend	binary file transfer using mail	uusend(1C)
/usr/bin/uuencode	uusend binary file encoder	uuencode(1C)
/usr/bin/uudecode	uusend binary file decoder	uudecode(1C)
/usr/bin/uulog	scans session log files	uucp(1C)

Under normal conditions, uucp calls your designated sites at specified times and, in addition, checks to see if any files are queued on that site's machine for you. If you ever need to invoke uucp 'on command', the line:

```
# /usr/lib/uucp/uucico -r1 -ssitename
```

forces uucp to poll *sitename*, even if there is nothing waiting. If you do run uucp in this fashion, don't run it as super-user, since the suid* bit will not be honored. If you have trouble with the connection, run uucp with the debugging option, as described in installation step 7, below.

Installing uucp

A uucp network link using modems or dedicated lines may be established between two machines running UNIX. To establish a connection between two sites that both have modems, one site must have (at least) an automatic call unit (an auto-dial modem) and the other must have (at least) a dialup port (an auto-answer modem). It is better if both sites have one of each or have modems which both call and answer, like Ventel's.

If both you and the site(s) you wish to connect with have autodial units and ports, install uucp as follows. If you have only a dialup your situation will be different; procedures are described near the end of this section. For more information, read the *Uucp Implementation Description* in the tutorials section of this manual; it describes in detail the file formats and conventions, and will give you necessary context.

- The name of the host machine in /etc/rc.boot will be the name of your uucp connection to the outside world (make sure it is fewer than 8 letters). We will call this machine your 'uucp host'; its name is your 'uucp hostname'.

  If this machine is your 'main machine' (for a definition see the next section, *Setting Up the Mail System*), then your uucp hostname should be the same as your domain name.

- Change the /usr/spool/uucp/D.noname directory to your own site's /usr/spool/uucp/D.*hostname* directory with the following:

  ```
  # mv /usr/spool/uucp/D.noname /usr/spool/uucp/D.hostname
  ```

  Use your uucp hostname for *hostname*.

- Create a uucp account for each remote host in the password file on your uucp host machine. Use /etc/vipw to enter a line of the following form in /etc/passwd:

  ```
  Uhostname:*:4:4::/usr/spool/uucp:/usr/lib/uucp/uucico
  ```

  Note, make this change only on the machine handling uucp traffic, not on the yp master server. Now use the passwd command to establish a password for each remote host:

* 'suid' stands for 'set user id;' if you set this bit on an executable file, UNIX will grant or deny file access based on the permissions of the file's owner, rather than the permissions of the person who executes the file. uucp uses this facility to ensure that all the files in its spool directories are readable and writable no matter who invokes the uucp program.

```
# cd /dev
# mknod cua0 c 12 128
# chmod 600 cua0
# chown uucp cua0
# mv ttya ttyd0
```

- Edit `/etc/ttys` to include an entry for `ttyd0` (see `ttys(4)`). Insert the line: `13ttyd0`. Then type the following to initialize everything properly:

  ```
  # kill -1 1
  ```

- In some older releases, `uuxqt` may fail because it cannot open files due to permission problems. Check permissions on the following files (note that *systemname* below should be fewer than 7 characters):

  ```
  /usr/spool/uucp/C.
  /usr/spool/uucp/D.
  /usr/spool/uucp/D.systemname
  ```

  If they are not `rwx--x--x` (711), type the following:

  ```
  # chmod 711 /usr/spool/uucp/{C.,D.,D.systemname}
  ```

- Make sure your modem is hooked up properly by running `tip` with the phone number of a known machine:

  ```
  # tip 5551234
  ```

  If you get a `dialing . . . connected` response, all is well. If you get any other response, reconnect your modem.

- As a final test, run `uucp` with the debug option (`-x`), as follows:

  ```
  # /usr/lib/uucp/uucico -r1 -ssitename -x7
  ```

  With `-x`, the higher the number, the more debugging output you get; 1, 4, and 7 are reasonable choices. If you get substantial quantities of output from this command, everything is fine; you can go on to edit the files below.

- Edit the files `uucp.day`, `uucp.noon`, and `uucp.night`, in the `/usr/lib/uucp` directory. Each of these files has a 'for' statement which arranges to call sites you want to call at designated times for mail. In each file, find the line that says:

  ```
  for i in sys.name
  ```

  and change `sys.name` to the site name(s) you want to poll. For example:

  ```
  for i in ucbvax ucbarpa Shasta navajo
  ```

  will poll four sites.

- Add the `uucp.day`, `uucp.noon`, `uucp.night`, `uucp.hour`, and `uucp.week`, files to `/usr/lib/crontab` (see `cron(8)`). These arrange for the appropriate sites to be polled at the appropriate times. For example, the entries in `crontab` might look like:

4.3. Setting Up The Mail Routing System

UNIX allows you to implement a general internetwork mail routing facility called `sendmail`. `sendmail` features aliasing and forwarding, automatic routing to network gateways, and flexible configuration.

This section will tell you how to install the mail system on your system. For more detail there are two papers in the *Tutorials* section at the end of this manual: *Sendmail — An Internetwork Mail Router* and *Sendmail Installation And Operation Guide.*

The mail system consists of the following commands and files:

`/bin/mail`	old standard mail program (from V7)
`/usr/ucb/mail`	UCB mail program, described in `mail`(1)
`/usr/lib/sendmail`	mail routing program
`/usr/lib/sendmail.cf`	configuration file for mail routing
`/usr/lib/sendmail.fc`	"frozen" configuration file
`/usr/lib/sendmail.main.cf`	configuration file for 'main machines' (see below)
`/usr/lib/sendmail.subsidiary.cf`	configuration file for 'subsidiary machines' (see below)
`/usr/spool/mail`	mail spooling directory for received mail
`/usr/spool/mqueue`	spool directory for mail going out over the network
`/usr/spool/secretmail`	secure mail directory
`/usr/bin/xsend`	secure mail sender
`/usr/bin/xget`	secure mail receiver
`/usr/bin/enroll`	to receive secure mail messages
`/usr/lib/aliases`	mail forwarding information
`/usr/ucb/newaliases`	command to rebuild binary forwarding database
`/usr/ucb/biff`	mail notification enabler
`/usr/etc/in.comsat`	mail notification daemon
`/usr/etc/in.syslog`	error message logger, used by `sendmail`

Note for `nd` file server configurations:

In the file server and diskless client environment, the directory `/usr/lib` is exported from the server. Since this file system is shared by all the machines, machine specific files cannot reside here. Instead on servers and clients there is a directory called `/private/usr/lib` that contains the files that would normally reside in `/usr/lib`. The `setup` program replaces the files in `/usr/lib` by symbolic links to the files in `/private/usr/lib` so the files can still be referenced by their normal names. The files (and directories) so affected are listed below:

```
/usr/lib/sendmail.cf
/usr/lib/aliases
/usr/lib/aliases.dir
/usr/lib/aliases.pag
/usr/lib/crontab
/usr/lib/uucp
/usr/lib/news
```

When the following instructions tell you to copy or edit one of the above files, instead use the corresponding file in `/private/usr/lib`.

the Arpanet Network Information Center. Otherwise, pick a name in the 'uucp' domain.

If your workstation and/or Ethernet have no phone lines or connections to other networks, the name you pick within the 'uucp' domain doesn't matter much. You may, however, have to change the name if you get other network connections and somebody else is already using that name.

If your organization already has hosts on the `uucp` network or the Usenet, ask a system administrator for help in picking a name.

If you are connected to the `uucp` network, you must be talking with at least one other computer on the network. Check with the system administrator of that machine to see if the name you have picked is already in use in the `uucp` network. If they are on the Usenet, they can look in newsgroup 'net.map'; if not, they just have to guess.

In a network of Suns, the name of the 'main machine' becomes the name of your subdomain, and each other machine can truly have any name you want. For example, a main machine at a site might be called 'fred;' various other machines on its Ethernet could be 'shirl', 'sal', etc. The *fully qualified* name of the 'fred' machine is 'fred.uucp' and must be unique in the `uucp` world; but the full name of 'shirl' is 'shirl.fred.uucp' and its uniqueness is guaranteed by the uniqueness of 'fred.uucp'.

If you only have one machine, your host name and your domain name (within the 'uucp' or 'arpa' top-level domain) are the same.

The above guidelines are valid as of this release date; however, both the Arpa Internet and the uucp network are rapidly evolving. In particular, the Arpa domain is now `.EDU`, `.GOV`, and `.COM` — educational, governmental, and commercial organizations.

Picking a 'Main Machine' for Mail Forwarding

To begin configuration, you must tell `sendmail` whether your system is the 'main machine' in a network, or a 'subsidiary machine' in a network.

If your machine is attached to an Ethernet and is also attached to phone lines, it can be a 'main machine'. Pick one such machine on the network; treat all the rest as 'subsidiary machines' for configuration purposes. Note that if you have a standalone system (a machine which is not attached to an Ethernet), treat it like the 'main machine' in a one-machine network.

If your machine is attached to an Ethernet and you don't have any phone lines, but there is an Arpanet host on your Ethernet, you can pick any workstation as your 'main machine'. If your Arpanet host runs `sendmail`, it can be your 'main machine'. All the other Suns will be subsidiary machines.

Similarly, if you have several machines on an Ethernet, and none of them have phones, pick one as the main machine and leave the rest as subsidiary machines.

A main machine can be aliased as 'mailhost' in `/etc/hosts`.

```
CV rome prussia georgia
```

This completes the `sendmail.cf` editing for the subsidiary machine. Note that if you have more than one subsidiary with no local `uucp` connections, you can edit the file just once and then copy it to all the subsidiary machines with `rcp(1)`. If your server and all clients can share the subsidiary `sendmail.cf`, then just put the file in `/usr/lib` instead of using symbolic links.

Arpanet Forwarding

On your main machine, you can make one optional change. If one of the machines with which you have a `uucp` or Ethernet connection is on the Arpanet and will relay mail for you, look for a block of lines like this:

```
# major relay mailer
DMuucp

# major relay host
DRarpa-gateway
CRarpa-gateway
```

Edit in the mailer that you connect to this host with ('uucp' or 'ether') and the name of the relay host. For example, if you share an Ethernet with a machine called 'cmu-cs-vlsi,' which is on the Arpanet, your entry might look like this:

```
# major relay mailer
DMether

# major relay host
DRcmu-cs-vlsi
CRcmu-cs-vlsi
```

On the other hand, your relay point might be `uucp` host 'ucbvax':

```
# major relay mailer
DMuucp

# major relay host
DRucbvax
CRucbvax
```

This change will let you mail to addresses like 'charlie@mit-ai.arpa' and even though you aren't on the Arpanet, the message will get through. If you don't have an Arpanet relay point, don't worry about this.

Setting up the 'Postmaster' Alias

Edit the file `/usr/lib/aliases` (or `/private/usr/lib/aliases` for a file server or diskless client workstation) and look for the entry for 'postmaster'. This is where people will send mail if they want to get in touch with someone at your site who can deal with mailer problems (you can send mail to 'postmaster's' at other network sites if you have problems with mail that comes from there). The line will look like this:

Store domain wide aliases in `/usr/lib/aliases` on the `yp` master server.

NOTE *The domain wide aliases software design is fluid as this document goes to press for the Beta release. More information will appear in this section for the First Customer Release.*

Then, while still on the `yp` master server:

```
# cd /usr/etc/yp
# make
```

This completes the mailer configuration process. (Remember to configure each individual machine!)

Testing Your Mailer Configuration

First reboot all the systems whose configuration files you have changed. Then, send test messages from various machines on the network with a command like this:

```
hal% /usr/lib/sendmail -v </dev/null addresses
```

This will send a null message to the specified address, and display messages about what it is doing. Test that:

- You can send mail to yourself, or other people, on the local machine, by addressing the message to a plain user name ('root', for example).
- If you have an Ethernet, you can send mail to someone on another machine ('root@hobo', for example). Try this in three directions — from the main machine to a subsidiary machine, *vice versa*, and from a subsidiary machine to another subsidiary machine, if you have two.
- If you have a phone line and you have set up a `uucp` connection to another host, you can send mail to someone there and they can send it back (or call you on the phone, if they receive it). Try having them send mail to you. For example, you could send to 'ucbvax!joe' if you have a connection to ucbvax. *Sendmail* won't be able to tell you whether the message really got through — since it just hands it off to `uucp` for delivery — so you have to ask a human at the other end. You might be able to get some idea of what's going on by looking in `/usr/spool/uucp/LOGFILE` (`/private/usr/spool/uucp/LOGFILE` on servers and clients); see the *Uucp Implementation Description* in the *Tutorials* section at the end of this manual.
- Mail something to Postmaster on various machines and make sure that it comes to your usual mailbox, so when other sites send you mail as Postmaster, you're sure you will see it.

Diagnosing Troubles with Mail Delivery

The best tools for diagnosis of mail problems are:

- 'Received' lines in the header of the broken message. These give a trace of which systems the message was relayed through on its way. Note that in the `uucp` network there are many sites that do not update these lines, and that in the Arpanet the lines often get rearranged. You can straighten them out by looking at the date and time in each line. Don't forget to take time zones

4.4. Tip

`Tip` establishes a full-duplex connection with another machine, giving the appearance of being logged in directly to the remote computer. You must have an account on the machine you want to connect with.

When you type the `tip` command, it reads commands from the file `.tiprc` in your home directory. This file can be used to alter the default values of `tip` variables. See `tip`(1C) for details and a discussion of the variables used in normal operation. If you use the `-v` flag on the command line `tip` displays the commands as it executes them.

Systems known by `tip`, and their attributes, are stored in the file `/etc/remote`; it describes the remote host(s) that `tip` connects with and stores information that will establish proper connections. See `remote`(5) for details and an explanation of capabilities. `remote` is an ASCII file, structured much like `termcap`(5). Each line in the file provides a description for a single remote system. Fields are separated by a colon (:); lines ending in a backslash character (\) and followed without blank space by a carriage return, are continued on the next line. Each description must end with a colon immediately following the last field.

The first entry is the name of the host system; several aliases may appear in a single name entry, separated by vertical bars (|). The name entry is followed by the fields describing capabilities. Capabilities are of three types: string, numeric, or boolean. String capabilities are given as '*capability=value*', for example: dv=/dev/harris. Numeric capabilities are given as '*capability#value*', for example: br#300. Boolean capabilities are specified simply by listing the capability.

An `/etc/remote` file might look in part like this:

```
decvax|DEC VAX-11/780:\
     :pn=6038842104%:tc=UNIX-1200:
arpavax|ucbarpa|arpa:\
     :pn=6427750%:tc=UNIX-1200:
olympics:\
     :pn=2136815931:tc=UNIX-300:
UNIX-300:\
     :el=^D^U^C^S^Q^O@:du:at=ventel:ie=#$%:oe=^D:br#300:tc=dialers:
UNIX-1200:\
     :el=^D^U^C^S^Q^O@:du:at=ventel:ie=#$%:oe=^D:br#1200:tc=dialers:
dialers:\
     :dv=/dev/cua3,/dev/cua2,/dev/cua1:
```

When editing the `/etc/remote` file remember the following rules: 1) every capability must be preceded and followed by a colon; 2) the continuation character (the \ at the end of the line) must not be followed by white space; 3) continuation lines must begin with a tab. These are the same rules you have seen in the `/etc/termcap` and `/etc/printcap` files.

You may need to look at the `remote`(5) man page to determine the meaning of all the capabilities included here.

Characters typed during a `tip` connection are normally transmitted directly to the remote machine. In addition, there is a set of recognized commands that can be given to `tip` on a line that begins with a tilde (~) escape character. These are listed below. Generally, they are for copying files or piping output between the connected machines.

be continuously displayed on the screen. A file transfer may be aborted with an interrupt. An example of the dialogue used to transfer files is given below (input typed by the user is shown in bold face).

```
arpa% tip monet
[connected]
... (assume we are talking to another UNIX system) ...
ucbmonet login: sam
Password:
monet% cat > foo.c
~> Filename: foo.c
32 lines transferred in 1 minute 3 seconds
monet%
monet% ~< Filename: reply.c
List command for remote host: cat reply.c
65 lines transferred in 2 minutes
monet%
```

Another way to accomplish the same task is shown below:

```
monet% ~p foo.c
... (actually echoes as ~[put] foo.c) ...
32 lines transferred in 1 minute 3 seconds
monet%
monet% ~t reply.c
... (actually echoes as ~[take] reply.c) ...
65 lines transferred in 2 minutes
monet%
```

To print a file locally:

```
monet% ~|Local command: pr -h foo.c | lpr
List command for remote host: cat foo.c
monet% ~^D
[EOT]
... (back on the local system) ...
```

5

Adding Hardware To Your System

5

Adding Hardware To Your System

This chapter explains many aspects of adding hardware to your system. It covers: how to add a new Multibus board to the system, how to add peripheral devices like terminals and modems to a serial port, and how to add a printer to the system.

In most cases adding new hardware to your system is a three tiered process: you connect the actual hardware piece as appropriate, if necessary you reconfigure the UNIX† kernel, and you edit files on your system to adapt it to the new device. For each kind of new hardware device, the process is explained in the appropriate section below.

The first kind of hardware device covered is a circuit board. You will probably have to reconfigure your kernel when a board is added.

Next, a section explains general procedures for adding devices to asynchronous serial ports, followed by specific instructions for terminals and modems.

Finally, we explain how to add a printer, and give some explanation of the line printer spooling system as implemented under Sun UNIX. Printers may be hooked-up on a serial port or connected directly to a controller board that drives a given model; each case is covered.

† UNIX is a trademark of AT&T Bell Laboratories.

```
# cd /sys/conf
# cp GRENDEL old.GRENDEL
```

2) Edit the kernel configuration file, adding a device driver entry for the board you have installed into the card cage. Look in *Section 4* of the *System Interface Manual* for specifications for the boards that Sun supports.

3) Run `/etc/config` on the new kernel configuration file.

```
# /etc/config GRENDEL
```

4) Build the new kernel.

```
# cd ../GRENDEL
# make depend
# make
```

5) Install the new kernel and try it out.

```
# cp vmunix /newvmunix
# /etc/halt
> b newvmunix -s
```

6) If the new kernel appears to work, save the old kernel and install the new one in `/vmunix`.

```
# cd /
# mv vmunix ovmunix
# mv newvmunix vmunix
# /etc/reboot
```

Remember to remove the `old.GRENDEL` kernel configuration file you created as a backup.

For more information about kernel building, see the manual *Installing UNIX On The Sun Workstation.*

File System Modification

You must be superuser to make the following system modifications.

The UNIX kernel communicates with devices through a special file in the directory `/dev`. When a new board is added to the system, a new entry for it must be made in the `/dev` directory. This is relatively easy to do with a shell script called MAKEDEV (8), located in `/dev`. MAKEDEV takes an argument that is the device-name of a device supported by the system, and automatically installs the files for that device in the `/dev` directory.

For example, if you are adding a Sun-1 color board, you will need to run MAKEDEV like this:

```
# cd /dev
# MAKEDEV cgone
```

what it does.

Trouble Shooting

Here is a general list of hints for debugging new boards.

1) Check the hardware. Re-seat the board to assure electrical contact. Check hardware manuals to make sure all switch settings are proper.

2) If you boot the system with a copy of the GENERIC kernel after installing a device that Sun supports, and you still have the problem, chances are good that it is a hardware problem. If the problem disappears using GENERIC, you should begin looking in the kernel or the file system for the problem.

3) Make sure the new device is included correctly in the kernel configuration file. One way to ascertain this is to look in a file called `/usr/adm/messages`, which collects all the messages generated during boot-up. By looking at the last boot-up listing in `/usr/adm/messages`, you can determine positively whether the device was included in the kernel last used to boot the system. It should appear in the boot-up listing as one of the devices the boot-up procedure found. You may also use `dmesg` (8) by hand to see recent diagnostic messages — `/etc/dmesg` is run to produce the file `/usr/adm/messages`.

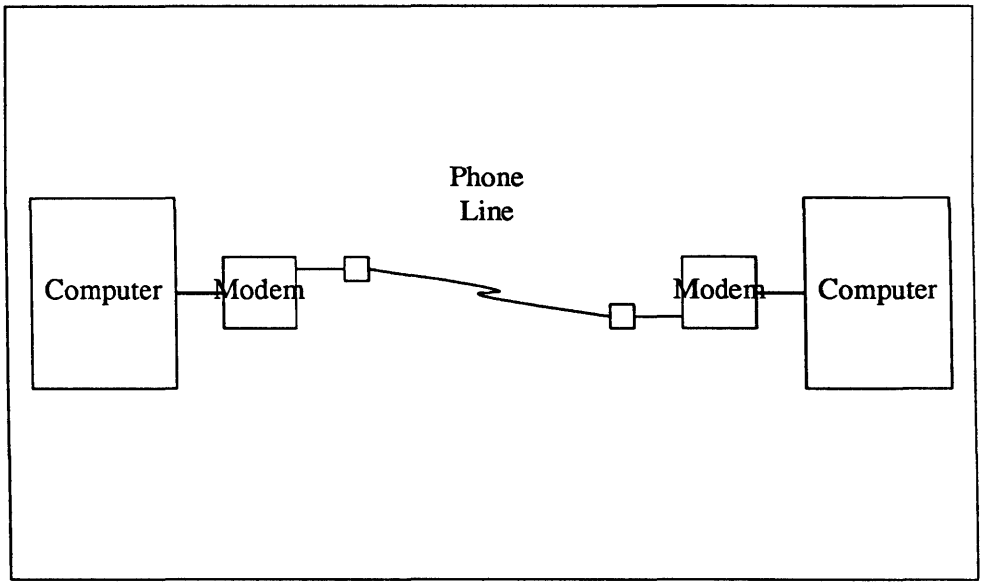
4) Do an `ls -l` in the `/dev` directory to make sure the new device is installed. The `ls -l` will also show the permissions and major and minor numbers for each device present. Check the permissions for the device against those found in the section of the MAKEDEV script that installs the device in question, they should match. Also make sure the device major and minor numbers match those in MAKEDEV.

5) Try the simplest commands to see if ANYTHING is working. The closer you come to isolating the spot where the system breaks down, the fewer places you need to look to make fixes. For example, see if a new printer board works by simply 'catting' a file to it:

```
% cat myfile > /dev/lp
```

rather than trying to pipe a job through text processors to `lpr`, where the job could fail for a reason that has nothing to do with the new board.

Figure 5-2 *Communications Through Modems*



When you connect a peripheral device to a serial port on the Workstation, make sure the cable connection between the serial port and the device is properly installed. Next make sure the appropriate serial device is configured in the UNIX kernel. Note: the generic kernel contains all the Sun supported device drivers. If you have re-configured or customized the kernel, you may need to add back the appropriate serial driver. See the section on *Kernel Configuration* in the manual *Installing UNIX On The Sun Workstation.* If specific kernel changes are necessary, they are discussed in the following sections.

You always need a special file in `/dev` for the UNIX kernel to run any device. There is an easy way to make this special file: you use the `/dev/MAKEDEV` command, which installs a special file for new UNIX devices. The command is explained in `MAKEDEV` (8).

General Debugging Hints

Here we discuss two general areas where problems occur when you have added new peripheral devices to the system.

Hardware Changes — If you have added new hardware and there are problems, check the cabling and connections first. Is everything tight? Once again verify that the new device accepts the RS-423 communication protocol, and consult the product manufacturer's manual to make sure it has been installed properly. Are the workstation and the new device set up for compatible communication? The Sun defaults to the following: 7 data bits, even parity, 9600 baud rate, flow control enabled (xon/xoff). Check which control signals the other equipment expects.

Next, check the condition of the cable. Here you may want to refer once again to your Sun Workstation *Hardware Manual* to determine what signals are sent from and received at specific ports on your board. To check for cable problems it is helpful to have a "breakout box". A breakout box plugs into the RS-232 cable, providing a patch panel that allows you to connect any pin to any other pin(s); it will often contain LEDs which display the presence or absence of a signal on

5.3. Adding A Terminal To Your System

Please read the section above called *Connecting Devices To Asynchronous Serial Ports* before proceeding further. The general discussion there will prepare you to install a new terminal on your system.

This section explains the steps necessary for adding a terminal, and gives some hints on what to do if you run into problems.

Serial Port and Cable Connection

We begin with the hardware implementation. Make sure you have an available serial port on your workstation. The number of serial ports varies from one Sun model to another. Each Sun workstation provides at least two asynchronous serial ports controlled by the Serial Communications Controller on the Sun-2 CPU board. If all the existing serial ports are in use, and you are not planning to disconnect any current peripheral devices to make a port available, you will need to add a new Multibus board to the card cage to increase the number of serial ports. The instructions for adding a Multibus board are given above in the section *Adding A Board To Your System.*

When you do have a port available, connect the terminal to the workstation with a null modem cable. A basic null modem cable swaps lines 2 and 3 so that the proper transmit and receive signals are communicated between two Data Terminal Equipment (DTE) devices. Line 7 goes straight through.† Make sure all the connections are secure and check control signals.

Kernel Modification

You must be superuser to make kernel modifications. If you are adding a first terminal, make sure that the system kernel contains a `zs0` device driver to run the CPU ports (this is standard); if you are adding terminals to a SCSI board make sure the proper `zs*` driver entry is in the kernel configuration file; if you are adding terminals to a Systech multi-terminal interface board (MTI), make sure the proper `mti0` driver entry is in the kernel configuration file.

If you had to add a new board to make ports available for your terminals, you have probably made necessary kernel changes already, as outlined above in the section *Adding A Board To Your System.* That section explains how to edit and reconfigure the system kernel.

Use `dmesg` (8) to look at system diagnostic messages to make sure the system is configured they way you want it.

Look in your kernel configuration file in the `/sys/conf` directory to find the device drivers for which your system is configured. An explanation of the device driver entries in this file is given on the appropriate page in *Section 4* of the *System Interface Manual.* For terminal connection, make sure the `flags` bit is set for a hardwired line on the port — set to on.

† The Systech MTI board requires pin 5 asserted. The simplest way to get this is to connect pins 4 and 5 at the connector going to the MTI port.

```
2|std.9600|9600-baud:\
    :sp#9600
3|std.300|300-baud:\
    :sp#300
4|std.1200|1200-baud:\
    :sp#1200
```

is a `gettytab` entry for initialization of terminals at three different baud-rates. Therefore, a '2' as the second character on the line would set the rate to 9600, an 'f' to 1200. If we were putting a terminal on /dev/ttys2 at baud-rate 9600, we would add the following line to the example `/etc/ttys` file shown above:

```
12ttys2
```

at the end of the file. This terminal will have a login process started for it at 9600 baud. After editing the `/etc/ttys` file you **must** notify `init` which then reads the new information in `/etc/ttys`. The command is:

```
# kill -1 1
```

Now edit the file, `/etc/ttytype`. This file is a database containing information about the kind of terminal attached to each tty port on the system. `/etc/ttytype` has one line per port, containing the terminal type, a blank space, and the name of the tty. For example,

```
sun console
925 ttya
925 ttyb
925 ttys0
925 ttys1
dialup ttyd0
dialup ttyd1
dialup ttyd2
dialup ttyd3
```

indicates that the terminal attached to ttys1 in the example above is treated by the system as a TVI 925, as are all other terminals attached to this system. The console is treated as a Sun Workstation. The type 'dialup' is for phone-in lines available in this example. To find the kind of terminal types your system recognizes, look in the file `/etc/termcap` (and see `termcap`(5)); this is a database describing the capabilities of terminals and the way operations are performed on them. If your terminal type is found in `termcap`, use it. If your new terminal has an emulation mode for one of the terminal types found in your `termcap` file, edit that name into `ttytype` and set the corresponding emulation mode on the terminal. If your terminal type is not in `termcap`, you may create your own `termcap` entry. The information in `ttytype` is read at login time by `tset` and `login` to initialize the TERM variable.

Finally, you will need to set some switches on the terminal so it can communicate with the Sun. Check to see if you have changed any settings on the Sun from the defaults given here: 7 data bits, even parity, flow control enabled (xon/xoff); set baud rate to the same rate you set in the `/etc/ttys` file.

5.4. Adding A Modem To Your System

Please read the section above called *Connecting Devices To Asynchronous Serial Ports* before proceeding further. The general discussion there will prepare you to install a modem on your system.

This section explains the steps necessary for adding a modem, and gives some hints on what to do if you run into problems.

Sun supports three currently manufactured modems: the Ven-Tel 1200-plus, the Ven-Tel 1200-32, and the Hayes Smartmodem 1200. Ven-Tel modems are run in Hayes compatibility mode. Sun will continue to support the discontinued Ven-Tel MD 212 series, although it alone is not discussed below.

Where this section refers to systems that have both auto-dial and auto-answer capabilities, the references are only to systems after Sun release 1.2.

Serial Ports, Cable Connection And Modem Switches

Serial Ports — Make sure you have an available serial port on your workstation. The number of serial ports varies from one Sun model to another. Each Sun workstation provides two asynchronous serial ports controlled by the Serial Communications Controller on the Sun-2 CPU board. If you also have a Sun-2 SCSI board in your system, two additional SCC's make four additional serial ports available. If all the existing serial ports are in use, and you are not planning to disconnect any current peripheral devices to make a port available, you will need to add a new Multibus board to the card cage to make more ports available. The instructions for adding a Multibus board are given above in the section *Adding A Board To Your System.*

Cable Connection — If you do have a port available, connect the modem to the workstation with an RS-232 cable that has pins 2 through 8 and pin 20 wired straight through. You may also use a 25 pin ribbon cable to connect the ribbon to the system. Make sure all the connections are secure.

The Systech MTI board requires a special cable with pins 2, 3, 4, 5, 7, and 20 wired straight through, and pin 6 on the MTI end wired to pin 8 at the modem end.

Modem Switches — There are differences between the Ven-Tel modems and the Hayes Smartmodem. However, with either brand the switch settings shown below work for use with `tip`(1) and `uucp`(1). Always read the manufacturer's manual before attempting to adjust equipment.

First the Ven-Tel 1200-plus models. (Note that this information does not apply to the old Ven-Tel MD 212.) There is one panel of external switches and one of internal switches on each of the Ven-Tel models discussed here. For the external panel, set all switches to the open or up position. When you open the modem box you will see another panel of switches, which should be set as follows:

- Switch 1 up for hardware DTR.
- Switch 2 up for WECO control signals.
- Switch 3 down to get Hayes protocol.
- Switch 4 can be set according to your preference. Set it up to disable the speaker so you will not hear the high-pitched carrier noise when connecting;

You must edit the device driver specification for the serial communications controller in your configuration file, to enable carrier detect on the line where the modem is attached. We refer to this as turning `off` the software carrier, which then allows the system to detect a hardware carrier on the line. When turned `on` the system treats each line as hard-wired with carrier always present. Turning `on` or `off` is done by specifying a parameter in the `flags` field. For example, below is a sample entry from a kernel configuration file for the serial communications controller on the standard Sun CPU board:

```
device  zs0 at mb0 csr 0xeec800 flags 0x3 priority 2
```

To make the device in this example into a modem compatible driver, the hexadecimal value '0x3' after `flags` will have to be changed. The same field will have to be changed for other device drivers if your modem is going to be attached to a board other than the Sun CPU board.

Specifically, you need to make the following changes to the device driver specification line in the kernel configuration file. The changes for each of the three boards to which a modem can be attached are given below.

The Standard Sun CPU Board — The serial communications controller on the standard Sun CPU board, device `zs0`, provides two lines with full modem control. For both dial-in and dial-out on the same serial port, the `flags` bit in the kernel corresponding to that serial port has to be set to zero. This enables hardware carrier detect so that the Sun can tell when someone dials in or hangs up. The default value of `flags` for `zs0` in the Generic kernel is 3, indicating software carrier for both ports a and b. To permit hardware carrier detect on `ttya`, `flags` should be changed to '0x2', on ttyb to '0x1', and on both to '0x0'.

A Multibus SCSI Board — Each Multibus SCSI board has two serial communications controllers identical to the one on the CPU board, for a total of four lines with modem control on each SCSI board. These are devices `zs[2,3]` for the first board and `zs[4,5]` for the second. Set the `flags` value for each driver according to the rules explained above for the CPU board. Remember that `zs2` is for the lines `ttys[0,1]` and that `zs3` is for the lines `ttys[2,3]`; if you have a second SCSI board the ttys are, by convention, `ttyt[0-3]`.

A Systech MTI Board — The Systech MTI-800 and MTI-1600 boards require a kernel configuration line for device `mti0`. As above, the `flags` value is set in hex. The default `flags` value on installation of either MTI board is '0xffff'. This value selects software carrier detect for all lines. For lines with modems, you need hardware carrier detect and must set the corresponding `flags` bit in the kernel to zero, just as in the examples above.

The following example shows how to determine the proper hexadecimal values for the `flag` bits on a Multibus board, it visualizes a Systech MTI-1600 board where ports zero through 7 are turned on, software carrier detect, and ports 8 through f are turned off, hardware carrier detect for modem operation. The logic here can be applied to all Multibus boards.

Remember to remove the `old.CHAOS` kernel configuration file you created as a backup.

For more information about kernel building, see the manual *Installing UNIX On The Sun Workstation.*

File System Modification

You must be superuser to make the following system modifications.

Using mknod — First you create an alternate device for your modem in the `/dev` directory by using `mknod`. This alternate device allows a single tty line to be connected to a modem and used for both incoming and outgoing calls.

There are several arguments which you must give to `mknod`. The first is the name of the device you are making. It will be `cua*`, where the asterisk is the logical value of this alternate device on your system — 0 for the first alternate device, 1 for the second and so forth. The second argument will always be `c`. The last two arguments are the major and minor device numbers. To determine what numbers to use here do an `ls -l` on the `/dev/tty*` device you are making the alternate device for. If you are going to use the first port on your CPU board — `ttya` — do the following:

```
# ls -l /dev/ttya
crw--w--w-  1 root       12,   0 Sep 17 18:27 /dev/ttya
```

Here the major device number is '12' and the minor device number is '0'. When running `mknod` to create an alternate device for a modem you always give the actual major device number and you always add 128 to the minor device number. Thus, in this example, the `mknod` command would take as its last two arguments '12 128'. You also change the mode and ownership of this device. The whole process would look like this:

```
# cd /dev
# mknod cua0 c 12 128
# chmod 600 cua0
# chown uucp cua0
```

In this example we created alternate device `cua0`, the special file for the first modem attached to a system. The second modem attached would be `cua1`, and so forth.

While still in the `/dev` directory, move the tty device whose software carrier you turned off in the `flags` field of your device driver specification in the kernel above, to a dialing device. This is the same one for which you created the alternate device with `mknod`. In this example we move the first port on the CPU board to the first modem device.

```
# mv ttya ttyd0
```

The second modem device would be `ttyd1`, and so forth.

Updating ttys — Now edit the `/etc/ttys` file, adding an entry for the new `ttyd1` line, and disabling logins for the no longer existing `ttya` device.

A modem suitable for use at both 1200 and 300 baud could use a `gettytab` entry similar to this:

```
sunphone:\
     :pn=7630927%:tc=UNIX-1200:
UNIX-1200:\
     :el=^D^U^C^S^Q^O@:du:at=hayes:ie=#$%:oe=^D:br#1200:tc=dialers:
dialers:\
     :dv=/dev/cua0:
```

Install `uucp` — Once your modem is installed and working, you may install `uucp`. See the appropriate sections in the *Communications* chapter of this manual.

Set Up The Mail System — Finally you must check to make sure that your machine's mailer configuration is set up properly. See *Setting Up The Mail System* in the *Communications* chapter of this manual for a complete discussion of the mail system. If you are adding a modem to connect a main machine to phone lines, or if you are a standalone machine not on a local network, install the `main` file in `sendmail.cf`. If you are a subsidiary machine on a local network (standalone or client) and planning to stay that way, despite the attachment of phone lines, install the `subsidiary` file in `sendmail.cf`.

Hayes Specific Considerations

`tip` **Support** — To use a Hayes modem with `tip`, you should specify the modem type in the `/etc/remote` file as either `at=hayes` or `at=at`. The phone number attribute (`pn`) can contain any valid dial commands; see your modem manual for details. The most common commands to the phone number attribute are:

- A phone number of numeric characters 0-9.
- A comma (,) will cause a 2 second pause to wait for a secondary dial tone. For example to dial an outside line from a local phone network.
- A `P` or `T` will switch to pulse or tone dialing. By default tip will use tone dialing. Typically, a rotary phone has pulse dialing and a push button has tone.
- You may set parameters for the dialer by starting the phone number with an 'S' (for set) flag. Read the Hayes manual for an explanation.
- Note that `tip` can cycle through a set of phone numbers, dialing each one until a connection is made. Previously the numbers were often separated with a comma, although this feature was never documented. Now, since comma is a valid dial character, phone numbers must be separated with a vertical bar (|), just like the separator in the name field. For example, the `/etc/remote` line for `pn` would look something like:

```
:pn=2138896565|2138896564|2138896563:
```

`uucp` **Support** — To use a Hayes (or AT) modem with `uucp` the modem type in the `/usr/lib/uucp/L.sys` file should be `ACUHAYES`.

- The `uucp` default is for tone dialing.
- The phone number may contain any valid dial commands, however `uucp` uses any alphabetic prefix of a phone number to look up a translation in the

running, no one is dialed in, and there is no lock file, try unplugging the modem cable and/or power cycling the modem. Finally, you can do `ps ax` and look for a process tying up the port.

- If you get a `tip: /dev/cua0: Permission denied` or `link down` error message, make sure you have `:dv=/dev/cua0:` in `/etc/remote`. Check for a lock file in `/usr/spool` or `/usr/spool/uucp` called `LCK.*` where `*` is the name of the dial out destination. Make sure permission modes on `/dev/cua0` are 622, and it is owned by uucp. Try turning off the modem, unplugging it for a minute, and plugging it back in again. Also check permissions on the `/usr/spool/uucp` directory. It must exist and be readable, writable, and executable by everybody (777 mode).

that proper transmit and receive signals are communicated between two DTE devices. If the printer is a DCE device, then connect 2, 3, and 7 straight through. Make sure all the connections are tight.

When you have connected the printer as explained above, you should verify that cabling and hardware are performing before proceeding. Obviously, if there is a problem with connection or faulty hardware, none of the later software installation will work.

- Consult the operation manual for the printer and run the printer's stand-alone diagnostics. If these do not run, call the printer manufacturer. Remember to set the switch on the printer back to operate mode after the self-test.
- Verify that the printer switch settings are correct. Check for the correct settings in the printer operation manual, and make sure they correspond with the parameters given above for the Sun.
- Send something over the tty line to the printer. To do this, set the characteristics of the printer using `stty` (1), and then `cat` (1) a file to the `/dev/tty*` serial port where the printer is attached. If the printer is attached to the first serial port on the CPU board, type:

  ```
  # (stty 1200; cat /etc/passwd) > /dev/ttya
  ```

  to print a copy of the password file.

  If the printer is 'dead', your cable may be bad, or you may need a null modem cable as described above.

  If something all garbled prints, `stty` may not have set all the terminal characteristics properly for the printer. Read the manufacturer's manual and check the switches on the printer. In particular check the baud rate, 1200 in this example. Make sure the printer and the `stty` setting match. You may have to set additional options with `stty` to match those on your printer. The most likely are: to set parity to `even`, `odd` or neither (`space` parity); to set tab expansion with `-tabs`; and to allow carriage return for newline, and output CR-LF for carriage return or newline with `-nl`. These parameters are set for the line printer spooling system software in the `/etc/printcap` file by the `fs` capability. See below for an explanation. Also see `printcap` (4) in *The System Interface Manual.*

File System Modification

You must be superuser to make the following file system modifications.

The Sun software includes the necessary programs to run the line printer system, and a default line printer queue is included. You need to create a `/etc/printcap` file to make the printer work properly.

Editing The `printcap` File

`printcap`(5) is a database describing printers. It describes line printers directly attached to a machine and printers accessible across a network. Each printer should have an entry, and it is a good idea to remove entries for printers not in your system. Typically, a system will have just a single printer. The syntax of entries in `printcap` can seem torturous; check carefully after you have modified the file. We give explanations and examples below.

- `fs` – Clears flag bits. Similar to `fs`, but turns off bits instead of turning them on. The `fc#0300` turns off the even and odd parity bits, resulting in space parity.
- `tr` – A trailer string, to be sent to the printer at the end of a series of print jobs. In this example the trailer is a form feed.
- `xs` – Sets local mode bits. If you want 8 bits out, no parity, set `xs` to 40. Use the `xc` capability to clear local mode bits. Setting `litout`, however, disables regular output processing such as tab expansion and CR/LF processing.
- `of` – This is the name of an output filtering program, a standard post-processor for nroff in this example. It is supposed to make underlined text come out properly on line printers. It may or may not be appropriate on your printer. Try running your printer without it. If the format looks fine, you probably do not need the filter.
- `lf` – This is the name of the file where spooler errors will be logged. It can be created anywhere, but must exist with write permissions before errors can be sent to it. To assure that the spooler daemon can write on the error log file, become superuser, create the log file, and do a `chmod 666` on it.

Other File System Modifications

After your `printcap` file has been edited for your printer(s), you will need to make a few more adjustments to the UNIX file system.

- Check to make sure the proper permissions and ownerships exist on the files `/usr/lib/lpd`, `/usr/ucb/lpr`, and on the directory, `/usr/spool/lpd`. Note that you may have given a different name to your `/usr/spool/lpd` directory, and that you will have one spooling directory with a unique name for each printer accessible on your system. For the files named above type `ls -lg` to check permissions, ownership and group. To check the same things on the spooling directory, type `ls -lgd`. In addition, check the files in the spooling directory with `ls -lg`. The permissions, ownership, and group should match those shown below:

```
# ls -lg /usr/lib/lpd /usr/ucb/lpr
-rws--s--x  1 root    daemon   53248 Oct 14 09:19 /usr/lib/lpd
-rws--s--x  1 root    daemon   30720 Oct 14 09:19 /usr/ucb/lpr
# ls -lgd /usr/spool/lpd
drwxrwx---  2 daemon  daemon     512 Nov 09 11:00 /usr/spool/lpd
# ls -lg /usr/spool/lpd
-rw-r--r--  1 root     daemon         22 Mar  1 18:25 lock
-rw-rw-r--  1 root     daemon         29 Mar  1 17:28 status
```

- Make sure that `init`(8) does not create a login process on the port you are using for your printer. To do this, edit the `/etc/ttys` file, putting a zero (0) in the first column of the entry for the tty port that your printer is attached to. The example `printcap` attaches a serial printer to the first CPU serial port, `ttya`; `/etc/ttys` has an entry like:

```
02ttya
```

If it was necessary to edit the `/etc/ttys` file, you must notify `init` to

Kernel Modification

You must be superuser to make kernel modifications.

Here we give a brief outline of the steps for building a new kernel after installing a Systech VPC-2200 printer interface board. For a general discussion of software modifications after adding a new board, see the section *Adding A Board To Your System* in this manual. (If you use a different board and write your own device driver, the process will be slightly different; see the section *Writing Device Drivers For The Sun Workstation* in the *System Internals Manual.*)

Follow the steps shown here to reconfigure the kernel after the installation of a Systech printer controller board. These steps explain the process on a system named GRENDEL:

1) Change to `/sys/conf` directory and make a copy of the current kernel configuration file to keep until the new one is installed and running.

```
# cd /sys/conf
# cp GRENDEL old.GRENDEL
```

2) Edit the kernel configuration file, GRENDEL in this example, adding an entry for the Systech board you have installed into the card cage. Look in *Section 4* of the *System Interface Manual* for a description of this device. The entry you make in the kernel configuration file looks like this:

```
device vpc0 at mb0 csr 0x480 priority 2
```

3) Run `/etc/config` on the new kernel configuration file.

```
# /etc/config GRENDEL
```

4) Build the new kernel.

```
# cd ../GRENDEL
# make depend
# make
```

5) Install the new kernel and try it out.

```
# cp vmunix /newvmunix
# /etc/halt
> b newvmunix -s
```

6) If the new kernel appears to work, save the old kernel and install the new one in `/vmunix`.

```
# cd /
# mv vmunix ovmunix
# mv newvmunix vmunix
# /etc/reboot
```

Remember to remove the `old.GRENDEL` kernel configuration file you created as a backup.

Now, let's turn to the fields within each entry. The `printcap` manual page gives a table of all the capabilities available. Those shown here will run a Versatec 80 printer.

As mentioned above, the first field of each entry gives the names the printer is known by. One of the names must be `lp`; on a machine with more than one printer, one printer must use `lp` as one of its names — but only one printer.

Notice that each subsequent field is introduced by a two character code. Numeric capabilities take the form: *character_code#number_value*; for example, `px#2112`. String capabilities take the form: *character_code=sequence*; for example, `lf=/usr/spool/lpd-errs`. The capabilities shown in this entry are:

- `lp` – Specifies the device name to be opened for output.
- `sd` – Specifies the name of a spooling directory. Make sure the directory exists with proper permissions before attempting to run the printer.
- `pl` – Sets the page length in lines.
- `px` – Sets the page width in pixels.
- `py` – Sets the page length in pixels.
- `tr` – Specifies a trailer character, in this case a form feed, to be sent to the printer at the end of a series of print jobs, making the paper easier to remove.
- `of` – This is the name of an output filtering program to set up the data for the Versatec printer. When `if` is specified, `of` is used only for the banner page. See the section below *Output Filters*.
- `if` – This is the name of an output filtering program which can do accounting.
- `tf` – A troff output filter for the Versatec printer.
- `cf` – A plot output filter for the Versatec printer.
- `vf` – A raster image and screen dump filter for the Versatec printer.
- `lf` – This is the name of the file where spooler errors will be logged. It can be created anywhere, but must exist with write permissions before errors can be sent to it. To assure that the spooler daemon can write on the error log file, become superuser, create the file, and do a `chmod 666` on it.

Other File System Modifications

After your `printcap` file has been edited for your printer(s), you will need to make a few more adjustments to the UNIX file system.

- Check to make sure the proper permissions and ownerships exist on the files `/usr/lib/lpd`, `/usr/ucb/lpr`, and on the directory, `/usr/spool/lpd`. Note that you may have given a different name to your `/usr/spool/lpd` directory, and that you will have one spooling directory with a unique name for each printer accessible on your system.

- rp – indicates the name of the printer on the remote machine is 'versatec'; the default for this capability is the 'lp' printer on remote.
- sd – specifies /usr/spool/vpd is the spooling directory instead of the default value of /usr/spool/lpd.

Output Filters

The printcap examples above show various types of output filters. This section gives more details on their uses and specifications.

Filters are used to handle device dependencies and to perform accounting functions. The output filter of is used to filter text data to the printer device when accounting is not used or when all text data must be passed through a filter. It is not intended to perform accounting since it is started only once, all text files are filtered through it, and no provision is made for passing owners' login name, identifying the beginning and ending of jobs, etc. The other filters, such as if, (if specified) are started for each job printed and perform accounting if there is an af entry. The af entry designates the file where if puts its accounting informations — see the second example below. If entries for both of and one of the other filters are specified, the output filter is used only to print the banner page; it is then stopped to allow other filters access to the printer. An example of a printer which requires output filters is the Benson-Varian.

```
va|varian|Benson-Varian:\
     :lp=/dev/va0:sd=/usr/spool/vad:of=/usr/lib/vpf:\
     :tf=/usr/lib/rvcat:mx#2000:pl#66:tr=\f:
```

The tf filter (invoked with lpr -t) takes a troff output file and converts it to Versatec output. It is used by vtroff(1). Note that the page length is set to 66 lines by the pl entry for 8.5" by 11" fan-fold paper. To enable accounting, the varian entry would be augmented with an af file as shown below.

```
va|varian|Benson-Varian:\
     :lp=/dev/va0:sd=/usr/spool/vad:of=/usr/lib/vpf:\
     :if=/usr/lib/vpf:tf=/usr/lib/rvcat:af=/usr/adm/vaacct:\
     :mx#2000:pl#58:tr=\f:
```

Output Filter Specifications

Sun software provides several filters which are listed as ?f under CAPABILITIES in printcap(5). For many devices or accounting methods, it is probably necessary to create a new filter.

Filters are spawned by lpd with their standard input the data to be printed, and standard output the printer. The standard error is attached to the lf file for logging errors. A filter must return a '0' exit code if there were no errors, '1' if the job should be reprinted, and '2' if the job should be thrown away. When lprm sends a kill signal to the lpd process controlling printing, it sends a SIGINT signal to all filters and descendents of filters. This signal can be trapped by filters which need to perform cleanup operations such as deleting temporary files.

Arguments passed to a filter depend on its type. The of filter is called with the following arguments.

ofilter -w*width* -l*length*

The *width* and *length* values come from the pw and pl entries in the printcap database. The if filter is passed the following parameters.

lpc — Line Printer Control Program

The `lpc` (8) program is used by the system administrator to control the operation of the line printer system. For each line printer configured in `/etc/printcap`, `lpc` may be used to:

- disable or enable a printer,
- disable or enable a printer's spooling queue,
- rearrange the order of jobs in a spooling queue,
- find the status of printers, and their associated spooling queues and printer dameons.

The major commands and their intended use are described here. The command format and remaining commands are described in `lpc` (8).

`abort` and `start`

`Abort` terminates an active spooling daemon on the local host immediately and then disables printing (preventing new daemons from being started by `lpr`). This is normally used to force a hung line printer daemon to restart (i.e., `lpq` reports that there is a daemon present but nothing is happening). It does not remove any jobs from the queue (use the `lprm` command instead). `Start` enables printing and requests `lpd` to start printing jobs.

`enable` and `disable`

`Enable` and `disable` allow spooling in the local queue to be turned on/off. This will allow/prevent `lpr` from putting new jobs in the spool queue. It is frequently convenient to turn spooling off while testing new line printer filters since the `root` user can still use `lpr` to put jobs in the queue but no one else can. The other main use is to prevent users from putting jobs in the queue when the printer is expected to be unavailable for a long time.

`restart`

`Restart` allows ordinary users to restart printer daemons when `lpq` reports that there is no daemon present.

`stop`

`Stop` is used to halt a spooling daemon after the current job completes; this also disables printing. This is a clean way to shutdown a printer in order to perform maintenance, etc. Note that users can still enter jobs in a spool queue while a printer is stopped.

`topq`

`Topq` places jobs at the top of a printer queue. This can be used to reorder high priority jobs since `lpr` only provides first-come-first-serve ordering of jobs.

- The printer server, `lpd`, uses the same verification procedures as `rshd`(8C) in authenticating remote clients. The host on which a client resides must be present in the file /etc/hosts.equiv, used to create clusters of machines under a single administration.

In practice, none of `lpd`, `lpq`, or `lprm` would have to run as user `root` if remote spooling were not supported. In previous incarnations of the printer system `lpd` ran setuid `daemon`, setgid `spooling`, and `lpq` and `lprm` ran setgid `spooling`.

Problems

If you have problems after installation, check the cabling first, as outlined above in the section *Connecting Devices To Asynchronous Serial Ports*.

For serial printers, check the information in the `/etc/ttys` file, and make sure you have used `kill` to signal `init` as explained above.

If the printer does not work or behaves strangely, check the `/etc/printcap` file. You may have unexpected whitespace, or you may have forgotten to escape the newline character in your entry: a backslash (\) followed immediately by a newline (carriage return) will continue that line on the next. If there any output filters specified in the `printcap` entry, they could be the source of problems; make sure they have not introduced characters the printer does not know about.

Error Messages From The Line Printer System

There are a number of messages which may be generated by the line printer system. This section categorizes the most common and explains the cause for their generation. The messages are grouped under the command which generates them. Where the message indicates a failure, directions are given to remedy the problem.

In the examples below, the name `printer` is the name of the printer. This would be one of the names from the `printcap` database.

lpr

`lpr:` *printer* `: unknown printer`

The `printer` specified by the `-P` option or the `lp` default, was not found in the `printcap` database. Usually this is a typing mistake; however, it may indicate a missing or incorrect entry in the /etc/printcap file.

`lpr:` *printer* `: jobs queued, but cannot start daemon.`

The connection to `lpd` on the local machine failed. This usually means the printer server started at boot time has died or is hung. Check the local socket /dev/printer to be sure it still exists (if it does not exist, there is no `lpd` process running). Use

```
# ps ax | fgrep lpd
```

to get a list of process identifiers of running lpd's. The `lpd` to kill is the one which is not listed in any of the 'lock' files (the lock file is contained in the spool directory of each printer). Kill the master daemon using the following command.

```
# kill pid
```

The `lpd` process overseeing the spooling queue, as indicated in the 'lock' file in that directory, does not exist. This normally occurs only when the daemon or output filter has unexpectedly died. The error log file for the printer should be checked for a diagnostic from the deceased process. To restart an `lpd`, use

```
# lpc restart printer
```

lprm

`lprm: printer : cannot restart printer daemon`

This case is the same as when `lpr` prints that the daemon cannot be started.

lpc

`couldn't start printer`

This case is the same as when `lpr` reports that the daemon cannot be started.

`cannot examine spool directory`

Error messages beginning with `cannot ...` are usually due to incorrect ownership and/or protection mode of the lock file, spooling directory or the `lpc` program.

lpd

The `lpd` program can write many different messages to the error log file (the file specified in the `lf` entry in `printcap`). Most of these messages are about files which can not be opened and usually indicate the `printcap` file or the protection modes of the files are not correct. Files may also be inaccessible if people manually manipulate the line printer system (i.e. they bypass the `lpr` program).

In addition to messages generated by `lpd`, any of the filters that `lpd` spawns may also log messages to this file.

6

Periodic Maintenance

6

Periodic Maintenance

This chapter discusses many of the tasks that the system administrator will have to perform. Some of these things will happen according to the schedule you make. Others are infrequent, even random, but they will have to be done eventually.

First we talk about the need for backing up the file system. Then we cover booting and shutting down the system. Next, a large section explains what to do when your system crashes. There are short sections explaining the system log configuration and system performance. Following that are some remarks about accounting, local modifications on your system, and files that need periodic attention.

- For example you could do level 9's daily and keep the daily tapes for one week.
- Once a week you could do a level 5 and keep the weekly tapes for a month.
- Once a month you could do a level 3 and keep the monthly tapes for one year.

In this way you would not **need** to do a level 0 for one year, yet would still have all the modified files backed up. This is simply an example; **we recommend** doing level 0's more frequently. Remember to keep them, and the incremental tapes, around for a good long time. There is no telling when someone will come to the system administrator and ask for a copy of a file that existed for one week back in the beginning of the year.

Each site will decide on the back-up schedule and frequency that work best for it. The most important thing is to settle upon some schedule and make sure the dump tapes are made. In addition, make sure stored tapes are kept cool and clean and are well labeled. Tapes that cannot be read due to deterioration or lack of a label are useless.

For an explanation of how to back up and restore files, see the section *Backing Up And Restoring File Systems* in Chapter 2 of this manual.

error code in the LEDs.

- With a Sun-1/150U or Sun-2/170, you can see the LEDs by opening the front door of the cardcage enclosure.
- With a Sun-2/120, pull off the front plastic panel of the pedestal; the row of 8 LEDs is on the front edge of the CPU card in the far left slot of the cardcage.
- With a Sun-2/50, the LEDs are marked 'Diag Leds' at the left edge of the panel covering the CPU board.
- With a Sun-2/160, the LEDs are at the top edge of the leftmost (the CPU) board stacked vertically in the rack.

When power is first applied to the workstation, all eight LEDs light, then each lights quickly in sequence. Following this 'lamp test', the lights blink rapidly as each test is passed. The lights slow down as memory is tested; each of the two memory tests takes a few seconds per megabyte. Finally, three LEDs on the end light momentarily, then all the LEDs go off except for a middle LED which blinks about once a second. If your workstation follows this sequence, self-test has not found a critical problem. (Once UNIX or other programs have gained control of the system, they can use the LEDs in other ways. This description only applies to the power-on sequence.)

If at some point in the above sequence, the lights freeze (keep the same pattern for more than a minute), or the sequence restarts from the beginning, there is a critical hardware problem with the workstation. The appropriate thing to do in this case is to contact Sun Microsystems Field Service or your local Field Service organization. Copy down the pattern of lights (as well as you can, if it is repeating over and over); they contain important diagnostic information for Field Service.

When Non-Critical Errors Are Found In Self Test

Non-critical errors result in a display like the following:

```
Self Test found a problem in something

Wrote wdata at address addr, but read rdata.
Damage found, damages

--> Give the above information to your service-person

Sun Workstation, Model model_number, type_of_keyboard .
ROM Rev N, some_number_MB memory installed
Serial #some_number, Ethernet address n:n:n:n:n:n

Auto-boot in progress . . .
```

something shows what part of the system was most recently found to be malfunctioning. (If more than one error occurs, a summary of all errors is given in the *damages* section, and details about the last error are reported here.)

Booting From Specific Devices

The Sun Workstation can be booted from:

- Any logical partition of a local disk.
- Any publicly available network disk partition on your Ethernet.
- The first file of a local tape drive.

As mentioned above, the monitor automatically attempts to boot `vmunix` from a default disk. If you want to boot a different program, or from a different device, you must stop the automatic boot process by aborting. The specific abort sequence depends on your keyboard type. For a Sun-1 Model 100U or 150U, the sequence is either 'SET-UP-A' (hold down the 'SET-UP' key while typing the 'A' key) or 'ERASE-EOF-A'. For a Sun-2 keyboard, the abort sequence is 'L1-A'. For a standard terminal keyboard, the BREAK key generates an abort. When you abort, the monitor displays the address where it aborted and a **>** prompt, and waits for you to type a command.

The monitor's boot command looks like:

```
> b device(parameters)pathname args
```

where *device* is the type of hardware to boot from, *parameters* specify the address or partitioning of the device, *pathname* is the name of the actual file (in a UNIX file system on that device) to boot into memory, and *args* are optional arguments to the program.

To determine which devices your monitor PROM is able to boot from, you can use the command:

```
> b ?
```

The devices are shown in order from 'best' to 'worst', as used by the automatic boot procedure to select a boot device.

To boot from the monitor command level on the default device (the first device the monitor PROM can find to boot from), type:

```
> b
```

This is the normal command to give when you have interrupted automatic reboot and want to proceed to boot from `/vmunix`.

If you want to come up in single-user mode, type:

```
> b -s
```

IF you are up and running single-user, you can bring the system up to multi-user by typing the 'CTRL' key and the 'd' key simultaneously:

```
# ^D
```

Different ways of booting UNIX are used in various circumstances. Here are some of the ways; they are explained in the sections below.

- When booting multi-user fails, booting single-user will sometimes work. It may then be possible to fix the system from single-user mode so it will work again in multi-user mode. For instance, occasionally the `/etc/passwd`

```
192.9.1.1      winkin
192.9.1.2      blinkin
192.9.1.3      nod
192.9.1.24     henry
```

The last component of the complete internet address is the host number. Henry's host number here is 24. Note that you must supply the hostnumber to the monitor in hexadecimal; if the numbers in the `/etc/hosts` database are in decimal, you will have to convert.

Using zero as *hostnumber* is valid here, and means 'whichever host is my net disk server.'

partition is the desired public partition number on the specified server. The correspondence between this number and a real disk partition is defined in `/etc/nd.local` on the server machine.

pathname is the name of the file to boot.

args are optional arguments.

Booting From Tape

The Sun Workstation can be booted from 1/2-inch nine-track magnetic tape, from a 1/4-inch cartridge tape controlled by a Sun 1/4-inch tape controller, or from a 1/4-inch tape controlled by a SCSI tape controller, using the following command:

> **b** *tape* (*controller, unit, filenum*)

tape is the device abbreviation for your tape controller: **mt** for 1/2-inch tape with a Tapemaster controller, **xt** for 1/2-inch tape with a Xylogics controller, **ar** for a Sun 1/4-inch tape controller, or **st** for a SCSI tape controller.

controller is a small number indicating the *n*th standard magnetic tape controller in the system, or is the Multibus address of the controller.

unit specifies which tape drive on the controller is to be used.

filenum specifies which file of the tape is to be booted. By convention, boot commands number the first file on the tape file #0, the second #1, and so on.

The monitor ignores the supplied value of *filenum* and can only boot the first file on a tape. To boot a file further down the tape, use the monitor to boot the 'boot' program. Sun-supplied UNIX distribution tapes always have the 'boot' program on the first file of the tape.

Booting Files From The Default Device

To boot any file from the default device, enter:

> **b** *pathname args*

In this manner you boot an alternate version of the kernel by supplying its name at *pathname* in the example above. This is also useful for booting standalone utility programs, like `diag`, after your disk or network disk is set up.

Messages from the Monitor and the Boot Program

Abort at *aaaaaa*

The monitor has aborted execution of the current program because you entered the 'abort sequence' (upper left key held while pressing 'A') from the Sun keyboard, or pressed BREAK on a serial console. *aaaaaa* is the address of the next instruction. You can continue the program from there by entering the c command.

Address Error, addr: *xxxxxx* at *aaaaaa*

The current program has stopped because it made an invalid memory access. *xxxxxx* is the (invalid) address; *aaaaaa* is an address near the instruction which failed (typically two to ten bytes beyond). There is no general way to recover from this error, except to debug the program.

ar: cartridge is write protected

The current program is trying to write on an Archive tape cartridge, but the 'Safe' switch at the top left corner of the cartridge is set to prevent writing on the tape.

ar: *xxxx* error

The monitor or boot program is trying to boot from an Archive tape, and encountered an unexpected error. The status bytes *xxxx* can be decoded by looking under 'Read Status Command' in the *Archive Product Manual.* This error could be caused by incorrect cables, a bad tape, or other problems.

ar: drive not responding

The monitor is trying to boot from an Archive tape, but can get no response from the tape drive. This can occur if your system contains an Archive controller board but no tape drive, or if the tape drive's cable is loose or disconnected, or if the tape drive's power is not on.

ar: invalid state *xx*

This message indicates that the standalone I/O system has a bug in its Archive driver.

ar: no cartridge in drive

The monitor or boot program is trying to boot from an Archive tape, but there is no cartridge in the tape drive.

ar: no drive

The monitor or boot program is trying to boot from an Archive tape, but the specified drive does not exist. Typical Archive configurations include only drive 0.

ar: RDST gave Exception, retrying

The current program is trying to use the Archive tape drive, and encountered an error. The error is probably caused by hardware. Check the cable(s) that connect the tape drive to the system.

ar: triggered at idle *xx*

This message indicates that the standalone I/O system has a bug in its Archive driver.

in general, except to debug the program.

Can't write files yet...Sorry
The current program is trying to write to a disk or network disk file thru the standalone I/O system. Writing on files (as opposed to writing on devices) is not supported when running standalone (that is, before booting the UNIX kernel).

Corrupt label

Corrupt label on head *h*
The monitor or boot program is trying to boot from a disk. The first sector of the disk appears to be a label (as it ought to be), but the checksum on the label is wrong. Try again a few times; if the problem recurs, you should probably relabel your disk. See sections *Using the Diag Utility* and *Labeling the Disk* in the chapter, *Installing UNIX for the First Time*. Before trying to relabel your disk, make sure that you know what ought to be in the label — writing the wrong label on the disk is highly likely to cause destruction of some or all files on the disk.

count=*ddd*?
A standalone program is trying to write to a device and has specified a block size which is not a multiple of 512. The write proceeds anyway, but may cause incorrect results.

Damage found, *damage...*
As part of the power-on self test procedure, the monitor has found damage in one or more parts of the system. This message should be reported to your local service representative or Sun Microsystems Field Service. *Damage* is a list of subsystem names, such as 'memory' or 'timer'.

Exception *ee* at *aaaaaa*
The current program has stopped because it got an interrupt. The interrupt could have been caused either by hardware or software. *ee* is the hexadecimal address of the interrupt vector used; you can look it up on a Motorola 68010 or 68000 reference card or CPU manual to see what kind of interrupt has occurred. *aaaaaa* is the address of the instruction where the interrupt occurred. If *ee* is `2c`, the partition you are trying to boot from is probably missing its boot track.

Extra chars in command
Your previous 'u' command had extra, unrecognized characters on the end.

FC*n* space
The address space being accessed by the monitor's memory reference commands is defined by Function Code number *n*. See the Motorola 68010 CPU manual for more information. This message is printed by the 's' command.

For phys part *p*, No label found.
The boot program is trying to boot from a non zero 'physical partition' on a disk, and can't find a label. Physical partitions are used for disk drives part of which are fixed and part of which are removable.

mt: controller does not initialize
The monitor is trying to boot from nine-track tape, and could not get the tape controller to complete its initialization sequence. This might indicate a possible defect in the controller, or incorrect configuration of the controller board.

mt: error 0*xxx*
The monitor is trying to boot from nine-track tape, and encountered an unexpected error. The error number *xx* can be decoded by looking in *Appendix C* of the *Tapemaster Product Specification*, which is supplied with your tape drive.

mt: unit not ready
The monitor is trying to boot from nine-track tape, but the tape drive is not ready. Check to see that the drive is on-line.

nd: no file server, giving up.
The monitor or boot program is trying to boot from a network disk server over the Ethernet. It has been retrying for a long time and there is no response from the server. Check the Ethernet address in the boot command; if it is zero, make sure your machine's Ethernet address is recorded in the server's `/etc/nd.local` file. If that's OK, check your Ethernet cable connection, see whether the server is running correctly, and/or see whether other machines on the network can communicate.

No controller at mbio *xxxx*
The monitor is trying to boot, but it can't find a device controller where you asked it to look. You should try another boot command, or make sure that your controller board is plugged in and has all its jumpers and switches set properly.

No default boot devices
The monitor is trying to boot but it can't find a disk or Ethernet interface to boot from. To boot from a tape, you must specify the device name explicitly, as in '`b ar()`'.

No label found -- attempting boot anyway.
The monitor or boot program is trying to boot from a disk, and can't find a valid label on the disk. This is best fixed by booting a copy of `diag`(8S) from a different device (for example, network disk or tape) and using the 'verify label' and 'label' commands. See the warning under 'Corrupt label' above. This error might also be caused by missing or bad disk cables.

No more file slots
The current program is using the standalone I/O library and has opened too many devices or files.

not a directory
The current program (possibly the boot program) has tried to open a disk or network disk file with a pathname, but one of the names in the path is not a directory.

other), which makes it much more likely to succeed.

Seek not from beginning of file
The current program is using the standalone I/O library and has tried to do an unsupported seek operation.

SegMap *aaaaaa*: *xx*?
The monitor is examining or changing the segment map in response to your recent 'm' command. You can enter a space and type 'RETURN' to get back to command mode.

Self Test completed successfully
The monitor has completed its power-on self test without finding any hardware problems.

Self Test found a problem in *something*
The monitor has completed its power-on self test and found a problem in some subsystem. *Something* describes the general location of the error. Further messages give more details; see "Wrote ..." and "Damage found...".

Serial #*some_number*, Ethernet address *n:n:n:n:n:n*
The monitor is identifying your machine's serial number and hardware Ethernet address as part of the power-on sequence. The hardware Ethernet address is taken from the ID PROM on the Sun-2 CPU board, and is given as a 6-byte hexadecimal value with a colon between each byte. A typical Ethernet address might be `8:0:20:1:1:A3`.

Short read
The boot program is trying to boot a program from disk or net disk. It has located the program, but encountered an error while reading it into memory.

Size: *text* +*data* +*bss* bytes
The boot program is loading in the program you requested. *Text*, *data*, and *bss* are the sizes of the three sections of the program; they are printed as each is read into memory. After finishing display of this message, the boot program begins execution of your program; further messages can come from it instead of from the boot program or monitor.

Sun Workstation, Model Sun-1/100U or Sun-1/150U, *keyb* keyboard
The Model 100U or 150U workstation has just been powered on, or you entered a 'kb' command, and the monitor is identifying its configuration. *Keyb* is either VT100 or Two-tone, depending which keyboard your monitor ROMs support.

Sun Workstation, Model Sun-2/120 or Sun-2/170, Sun-2 keyboard
The Model 120 or 170 workstation has just been powered on, or you entered a 'kb' command, and the monitor is identifying its configuration.

Timeout Bus Error ...
See "Bus Error...". The attempted access was invalid because no device responded at the addressed location. This most often happens for Multibus references. The program was probably trying to access a device or section of memory which does not exist, or which has gotten into a hung state. If

Watchdog reset!
The current program has stopped executing with a 'double bus fault.' This is explained in detail in the Motorola 68010 manual; the two most common causes are that low memory (interrupt vectors) has been overwritten, or the system stack pointer is pointing to an invalid address. There is a serious problem, possibly in the kernel you are running, more likely it is in the hardware.

What?
You typed a command that the monitor does not recognize. Try again.

Wrote *wdata* at address *addr*, but read *rdata*
The monitor has completed its power-on self test and found a problem in some subsystem. The preceding "Self Test found a problem..." message describes which part of the system was in error. This message gives more details about the error. *wdata* is the data that was written into part of the system, or which was expected to be there if the system was functioning normally. *addr* is the address where the data was read and/or written. For memory errors, this is a physical memory address; for other errors, the interpretation of this field depends on what subsystem was being tested. *rdata* is the data that was read back from *addr* and was found to be invalid because it was not the same as *wdata*. This information should be written down and reported to your local Field Service organization, or to Sun Microsystems Field Service. See the section *Non-Critical Errors From Self Test* above.

xt: error *nn* during config of ctlr *cc*

xt hard err *nn*

xt: no response from ctlr *cc*
A standalone program (possibly the boot program) is trying to use the Tapemaster nine-track tape drive, and has encountered an error. This could be caused by a bad or missing tape, loose or misplugged cables, incorrect jumpers on the Tapemaster controller board, or hardware errors. *Nn* can be decoded by looking in the Xylogics Product Specification.

xy: error *nn* cmd *xx*

xy: error *nn* bno *bbbbb*

xy: init error *xx*
The monitor is trying to boot from the Xylogics disk and has encountered an error. The command being executed at the time is defined by the hexadecimal value *xx* (if present); the block number is *bbbbb* (if present), and the particular error is encoded as *nn*. The error and command can be decoded by looking in the Xylogics manual.

xy: no bad block info
The boot program is trying to read from the Xylogics disk, but can't find the information about bad blocks on the disk. It continues, but if the program attempts to read any bad blocks (which have been remapped to elsewhere on the disk), the attempt will fail.

6.3. When The System Crashes

Sooner or later, every system will crash or become hung in such a way that it no longer responds to commands. This section discusses some of the ways a system administrator can respond to system crashes and hung systems.

When the system crashes it prints out a short message telling why it crashed, attempts to preserve a core image, and invokes an automatic reboot procedure. If the reboot finds no unexpected inconsistency in the file systems due to software or hardware failure, it will resume multi-user operations. Below we discuss the error messages generated by a crash. In addition we discuss how the system core dump is saved, how to get rid of unwanted core dumps that can accumulate on your system, how to force a core dump, what steps can be taken to analyze a core dump, what to do if the automatic reboot fails, and when to call for help.

In general, a bad program should never crash the system. If it does, there is probably a bug in the kernel. Sometimes, however, a user program may crash and 'hang' something in the system — a user process, a user's window, or even the whole system — even though UNIX continues to run. Some aspects of a hung system and how to overcome them are discussed below. As the section *User Program Crashes* explains below, you are sometimes forced to reboot after a crashed program, even though UNIX has not crashed.

Crash Error Messages

When the system crashes, it prints out a message like:

```
panic: what I think went wrong
```

Less frequently you might see the message `Watchdog reset!` in place of the `panic`. A list of the most likely messages, along with a short explanation, can be found on the manual page `crash(8)`. Where there are repeated crashes it is extremely important to keep an exact copy of each message — including punctuation and upper or lower case lettering. These will be helpful to a doctor trying to heal a sick system.

Two files automatically store messages about crashes: `/usr/adm/messages`, keeps a record of system messages, and `/usr/adm/shutdownlog` tells how the system was shut down each time.

If you have not been able to obtain a copy of a crash message from the console, look in these files for a history of system behavior. They might also be helpful for determining patterns of problems over time.

System Core Dumps

As currently distributed, the system will not attempt to write a core dump. However, you can enable core dumps by editing the `/etc/rc.local` file on your machine. The lines to do the core dump are there, but are commented out for distribution. To enable core dumps look for the following lines in `/etc/rc.local`:

Analyzing System Core Dumps

It is not recommended that novices attempt to debug UNIX from the the core dump. Sophisticated users can `analyze(8)` the dump, or run `adb(1)` with the **-k** flag to poke around in it. See *Using ADB to Debug the UNIX Kernel* in the *System Internals Manual* for more details.

User Program Crashes

User program crashes, as distinguished from UNIX system crashes, are almost always due to programmer error — a bug in the program. The most common messages from a program crash are: `Segmentation Fault`, `User Bus Error`, `Floating Point Exception`. These often indicate an illegal pointer in the program, perhaps the result of using a value where a pointer to a value was expected. System error messages are documented in the introduction to *System Calls*, Section 2 of the *System Interface Manual*. A crashing program will usually dump core in its current directory if it has write permission. If it does, you will see the message `Core Dumped`.

Sometimes, the system will appear hung or dead; that is to say it will not respond to keyboard input. Before making the, rather drastic, assumption that your program has crashed, check the items below to make sure that there is not some other simpler reason for the system's non-responsiveness. (When we use the circumflex ^ below it signifies holding down the CTRL key while typing the key shown. For example, **^Q** means to hold down the CTRL key and simultaneously strike **Q**.)

1) Type **^Q** (CTRL Q) in case **^S** was accidentally hit, freezing the screen.
2) The `tty` mode may be fouled up. Try typing the linefeed character, **^J** (CTRL J), instead of the RETURN key, to force a linefeed. If the system responds, type **^J /usr/ucb/reset ^J** to reset the `tty` modes.
3) If you are running *Suntools*, make sure the mouse cursor resides in the window where you are trying to type commands
4) Type **^** (CTRL backslash). This should force a 'quit' in the running program, and probably the writing of a 'core' file.
5) Type **^C** (CTRL C) to interrupt the program that may be running.
6) If possible try logging into the same CPU from another terminal, or `rlogin` from another system on the network. Type **ps -ax** and look for the hung process. If you can identify it, try to kill it. You will have to be superuser or be logged in as the same user running the process. Type **kill** *<pid number>*. If that does not work try **kill -9** *<pid>*. (A quick way to see if a **kill** has worked is to repeat it. If the response is `no such process`, it was killed.)
7) If all of the above fail, the system has probably gone to another world. Abort and reboot.
8) If even that fails, call Tech Support for help.

6.4. Kernel Configuration

Every UNIX site should configure the kernel when it is installed. You will find a detailed explanation and notes about the configuration process in the manual *Installing* UNIX *On The Sun Workstation.* This section supplements that discussion with information less frequently needed. It is not intended to be used without expert knowledge of the procedures discussed in that installation manual.

In addition, various sections in this *System Administration For The Sun Workstation* manual give brief, walkthrough explanations of common types of kernel reconfiguration. If you need to supplement those with broader context, you should look first at the kernel configuration section in *Installing* UNIX *On The Sun Workstation*, since the sub-sections that follow here are mostly rather esoteric and specific.

Notes to Step 3 of Kernel Configuration

This section contains four additional notes about the procedures in Step 3 of kernel configuration, as they are given in the manual *Installing* UNIX *On The Sun Workstation.* Briefly, the notes cover:

1) System configuration on systems without source code
2) Adding new device drivers
3) Sharing object modules
4) Building profiled kernels.

Note 1: Configuring Systems Without Source

Object-only releases have binaries for standard system modules in the directory `/sys/OBJ`. Using these binaries you can create new configurations and add new device drivers to the kernel. The following lines from the GENERIC config file must be in every config file for object-only distributions:

```
machine sun
cpu "SUN2"
options "INET"

pseudo-device   inet
pseudo-device   ether
pseudo-device   loop
controller  mb0 at nexus ?
```

If you include these lines you can make any changes you wish to the configuration file, provided you do not configure in more devices of a particular type than are allowed by the distributed object code in `/sys/OBJ`. Attempting to do so will not be detected and may cause the kernel to appear to work but have only occasional failures. Double check the `.h` files in `/sys/OBJ` if you change the number of devices configured for any standard drivers.

Note 2: Adding New Device Drivers

New device drivers require entries in the files `/sys/sun/conf.c`, `/sys/conf/files.sun`, and possibly `/sys/sun/swapgeneric.c` and `sys/conf/devices.sun`. New devices also require one or more new special files to be added to the `/dev` directory. See the *Device Driver Tutorial* in the Sun *System Internals Manual.*

Rules for Defaulting System Devices

This section covers the rules the `config` program uses to define underspecified locations of system devices.

When the `config` program processes a `config` line which does not fully specify the location of the root file system, swap or paging area(s), device for system dumps, and device for argument list processing it applies a set of rules to define those values left unspecified. The following list of rules is used in defaulting system devices.

1) If a root device is not specified, the swap specification must indicate a 'generic' system is to be built.
2) If the root device does not specify a unit number, it defaults to unit 0.
3) If the root device does not include a partition specification, it defaults to the `a` partition.
4) If no swap area is specified, it defaults to the `b` partition of the root device.
5) If no device is specified for processing argument lists, the first swap partition is selected.
6) If no device is chosen for system dumps, the first swap partition is selected (see below to find out where dumps are placed within the partition).

The following table summarizes the default partitions selected when a device specification is incomplete, e.g. `xy0`.

Type	Partition
root	a
swap	b
args	b
dumps	b

In addition, if/when multiple swap partitions are specified, the system treats the first specified as a 'primary' swap area which is always used. The remaining partitions are then interleaved into the paging system at the time a `swapon` (2) system call is made. This is normally done at boot time with a call to `swapon` (8) from the `/etc/rc` file.

Finally, system dumps are automatically taken after a system crash, provided the device driver for the 'dumps' device supports this. The dump contains the contents of memory, but not the swap areas. Normally the dump device is a disk in which case the information is copied to a location near the back of the partition. The dump is placed in the back of the partition because the primary swap and dump device are commonly the same device and this allows the system to be rebooted without immediately overwriting the saved information. When a dump has occurred, the system variable `dumpsize` is set to a non-zero value indicating the size (in bytes) of the dump. The `savecore` (8) program then copies the information from the dump partition to a file in a 'crash' directory and also makes a copy of the system which was running at the time of the crash (usually `/vmunix`).

```
Configuration ::=  [ Spec ; ]*

Spec ::= Config_spec
    | Device_spec
    | trace
    | /* lambda */

/* configuration specifications */

Config_spec ::=  machine ID
    | cpu ID
    | options Opt_list
    | ident ID
    | System_spec
    | timezone [ - ] NUMBER [ dst [ NUMBER ] ]
    | timezone [ - ] FPNUMBER [ dst [ NUMBER ] ]
    | maxusers NUMBER

/* system configuration specifications */

System_spec ::= config ID System_parameter [ System_parameter ]*

System_parameter ::=  swap_spec | root_spec | dump_spec | arg_spec

swap_spec ::=  swap [ on ] swap_dev [ and swap_dev ]*

swap_dev ::=  dev_spec [ size NUMBER ]

root_spec ::=  root [ on ] dev_spec

dump_spec ::=  dumps [ on ] dev_spec

arg_spec ::=  args [ on ] dev_spec

dev_spec ::=  dev_name | major_minor

major_minor ::=  major NUMBER minor NUMBER

dev_name ::=  ID [ NUMBER [ ID ] ]

/* option specifications */

Opt_list ::=  Option [ , Option ]*

Option ::=  ID [ = Opt_value ]

Opt_value ::=  ID | NUMBER

/* device specifications */

Device_spec ::= device Dev_name Dev_info Int_spec
    | disk Dev_name Dev_info
```

Data Structure Sizing Rules

This section explains the rules for sizing kernel data structures, both those calculated at compile time and those calculated at boot time.

Certain kernel data structures are sized at compile time according to the maximum number of simultaneous users expected, while others are calculated at boot time based on the physical resources present, such as memory. This section lists both sets of rules and also includes some hints on changing built-in limitations on certain data structures.

Compile Time Rules

The file `/sys/conf/param.c` contains the definitions of almost all data structures sized at compile time. This file is copied into the directory of each configured kernel to allow configuration-dependent rules and values to be maintained. The rules implied by its contents are summarized below (here MAXUSERS refers to the value defined in the configuration file in the `maxusers` rule).

nproc
The maximum number of processes which may be running at any time. It is defined to be 10 + 16 * MAXUSERS and referred to in other calculations as NPROC.

ntext
The maximum number of active shared text segments. Defined as 24 + MAXUSERS.

ninode
The maximum number of files in the file system which may be active at any time. This includes files in use by users, as well as directory files being read or written by the system and files associated with bound sockets in the UNIX ipc domain. This is defined as (NPROC + 16 + MAXUSERS) + 32.

nfile
The number of 'file table' structures. One file table structure is used for each open, unshared, file descriptor. Multiple file descriptors may reference a single file table entry when they are created through a dup call, or as the result of a `fork`. This is defined to be

```
16 * (NPROC + 16 + MAXUSERS) / 10 + 32
```

ncallout
The number of 'callout' structures. One callout structure is used per internal system event handled with a timeout. Timeouts are used for terminal delays, watchdog routines in device drivers, protocol timeout processing, etc. This is defined as 16 + NPROC.

nclist
The number of 'c-list' structures. C-list structures are used in terminal i/o. This is defined as 100 + 16 * MAXUSERS.

nmbclusters
The maximum number of pages which may be allocated by the network. This is defined as 256 (a half megabyte of memory) in /sys/h/mbuf.h. In practice, the network rarely uses this much memory. It starts off by

6.5. System Log Configuration

Various system daemons and programs record information in the system log to aid in problem analysis. They send this information as Internet datagrams directed to the 'syslog' daemon on host 'loghost'. The daemon receives these datagrams and records the information or notifies users of problems. See `syslog(8)` for more details on this process.

The default configuration runs a `syslog` daemon on each machine, and also keeps all datagrams on the local machine (by maintaining the 'loghost' alias in the `/etc/hosts` entry on the local machine, or in the NFS environment, in the `/etc/hosts` entry exported by the `yp` master server machine for your domain). If you are running standalone, the default is for the `syslog` daemon to keep all datagrams on your machine. However, in a network environment it's much easier to track problems if all machines log their information in a single place. Therefore, during first time UNIX installation, the `setup` program reconfigures things so that only the designated server is the 'loghost': `setup` strips the 'loghost' alias from the `/etc/hosts` entries for the clients, and adds the alias for the server machine. This means that, for example, if the machine named krypton is your network server, the beginning of your machines' `/etc/hosts` files might look like:

```
192.9.1.1   krypton loghost
192.9.1.2   wally
192.9.1.3   beaver
192.9.1.4   june
192.9.1.5   eldridge
```

Now all datagrams sent to 'loghost' (from wally, beaver, etc.) are sent to krypton. There might also be other aliases on the same line of the `/etc/hosts` entry, like 'lprhost' or 'mailhost' — that's OK. Note also that, since the `syslog` daemon only starts up when messages must be handled, only the 'loghost' runs the daemon.

If you want to change this configuration — for example, if you have more than one server, and you want only one 'loghost' — simply change the placement of the 'loghost' alias, and then re-copy `/etc/hosts` to all machines. Test your system log configuration by running:

```
% tail -f /usr/spool/log/syslog
```

on the loghost machine, then sending any kind of mail on the various other machines. Each message sent will generate four or five lines of output if things are working.

6.7. Resource Control

Resource control in the current version of UNIX is rather primitive. The resources consumed by any single process can be voluntarily limited by the mechanisms of `setrlimit(2)`. This can be done at the command level in `csh` by using the 'limit' command — see `csh(1)`. Disk space usage can be monitored by `quot(8)` or `du(1)`. No system-enforced procedure for controlling a user's disk space usage is implemented under the current system, although a modicum of control can be obtained by dividing user groups between different disk partitions.

6.9. Making Local Modifications

Locally written commands are typically kept in `/usr/src/local` and their binaries in `/usr/local`. This allows `/usr/bin`, `/usr/ucb`, and `/bin` to correspond to the distribution tape (and to the system manuals). People wishing to use `/usr/local` commands should be made aware that they aren't in the base manual.

A `/usr/junk` directory to throw garbage into, as well as binary directories `/usr/old` and `/usr/new` are useful. The `man` command supports manual directories such as `/usr/man/manj` for junk and `/usr/man/manl` for local manual page entries to make this or something similar practical.

7

Upgrading System Software

7

Upgrading System Software

This chapter outlines procedures for upgrading software to a new release. It tells you what files to save when upgrading, and explains how to merge old files into the new system.

your tape controller as shown in Table 7-2. And for # substitute the correct unit number of the tape device.

Table 7-2 *UNIX Tape Device Abbreviations*

Abbreviation	*Device*
mt	Tapemaster half-inch magnetic tape
xt	Xylogics half-inch magnetic tape
st	SCSI quarter-inch tape
ar	Archive quarter-inch tape

For example, if you are using your first (or only) SCSI tape drive, enter `/dev/rst0`. If your system doesn't have a tape drive, you'll have to use slightly different commands, shown below, which access a tape drive elsewhere on the network. We'll call the machine with the tape drive your 'remote host'. Thus, before you begin, you'll have to know the device abbreviation for the remote host's tape drive (abbreviations are the same as in table above), and the remote host's hostname.

1. First, `tar` off the system files.

   For a machine with its own tape drive:

```
# cd /
# tar cf /dev/rtape# .??* dev/MAKEDEV.local etc lib usr/include usr/lib
```

   For a machine without a local tape drive, use the commands below. Substitute your remote host's hostname for *remote_host*. Use 126 for *block_size* on a quarter-inch tape, and 20 for a half-inch tape. Note that the command should be typed as a single line; be sure to escape the newline with a backslash (\), as shown, before you type <RETURN>:

```
# cd /
# tar cfb - block_size .??* dev/MAKEDEV.local etc lib usr/include\
usr/lib | rsh remote_host dd of=/dev/tape# obs=block_sizeb
```

   Either sequence makes a tape containing the system files you will need to set up your system after the upgrade.

2. NOW CHANGE TO ANOTHER BLANK TAPE, and run the following command. As described above, substitute the correct device specification for *tape#*. Also, replace *usera . . .usern* with the names of all users on your system. (You can check all names by looking in `/etc/passwd` or by doing `ls usr`.)

   For a machine with a tape drive:

```
# cp passwd group fstab ttys ttytype /etc
# cp crontab /usr/lib
```

Remember to remove the old copies after you're **sure** your system completeness/consistency is confirmed. If you wish, you can `mv(1)` the files instead of copying them.

Other files must be merged into the distributed versions by hand (`diff(1)` is often useful here). In particular, be careful with `/etc/termcap`.

3. The spooling and local directories, and the user files saved on the second `tar` tape may be restored in their eventual resting places without too much concern. Be sure to use the **p** option to `tar` so that files are recreated with the same file modes (you may want to add other options, as described in the previous section):

   For a machine with a drive:

```
# cd /
# tar xfbp /dev/rtape# block_size
```

   For a machine without a drive:

```
# cd /
# rsh remote_host dd if=/dev/tape# ibs=block_sizeb | tar xBfp -
```

4. Whatever else is left is likely to be site-specific or require careful scrutiny before you place it in its eventual resting place. Refer to the documentation (`hier(7)`, for example) before arbitrarily overwriting a file.

8

Diag — A Disk Maintenance Program

8

Diag — A Disk Maintenance Program

This chapter describes `diag`, the Sun Microsystems standalone disk utility program. It describes how to start `diag`, how to prepare a new disk for operation, and how to fix problems on a disk.

8.1. Architecture

A disk consists of a stack of spinning platters, each with its own **head**. The heads, which all move together, travel back and forth across the surface of the platters between the outer rim of the stack and the center.

Each platter consists of a series of **tracks**, which are concentric rings that run 360 degrees around the platter. Each track is divided into **sectors**, and each sector is marked with a unique header. Sectors are the basic storage units; each sector contains 512 bytes of usable data. Sectors are also referred to as `blocks`.

A **cylinder** is a vertical stack of tracks; for example, the third track on all the platters in the stack is cylinder 3. Because the heads move together, they are always on the same cylinder.

Cylinder, Head and Sector Numbers

`Diag` understands disk addresses in two forms: cylinder/head/sector (CC/HH/SS), or absolute decimal block numbers (NNNNNN). `Diag` accepts either, but the CC/HH/SS form resembles the physical architecture of the disk more closely. Both forms identify one particular sector.

A CC/HH/SS address in the form CC/00/00 identifies a cylinder boundary; the cylinder starts at sector 0 in track 0. Similarly, each track starts at sector 0, so an address in the form CC/HH/00 identifies a track boundary.

Logical vs. Physical

An address can be logical or physical. A logical address represents a count of usable sectors from a partition boundary, and a physical address represents an actual location on the surface of the disk.

When UNIX reports disk errors, it provides the file system name and a logical block count, which is the number of usable sectors between the partition boundary and the starting sector of the transfer where the error occurred. That is, one of the sectors in a particular transfer had an error, but UNIX reported the address of the first sector in the transfer.

CAUTION **Do not use `format` on any SMD disk purchased from Sun or you will erase valuable mapping information.**

Disks configured for sector slipping contain a spare data sector at the end of each track. When `diag` attempts to slip a sector, it removes the defective sector from service by identifying it with a special header, then it bumps subsequent logical sectors forward until the last sector moves to where the spare was.

If the disk is not configured for sector slipping, or if a track contains a second bad sector, you must rely of mapping. `Diag` writes the defective sector's address in the bad sector map and marks the sector with a header identifying it as a mapped sector. When the access to the mapped sector fails, the bad sector map redirects UNIX to use another sector on the spares track.

Sector slipping improves disk efficiency; it allows the disk to operate without the additional overhead of mapping.

To identify slipped sectors, mapped sectors, and to tell if the disk is configured for sector slipping, use the `rhdr` command, described later in this chapter.

SCSI Bad Sectors

To deal with SCSI disk bad sectors, the controller marks the bad sector and resumes its sector count on the next sector. It then slips the physical locations of all subsequent sectors towards the end of the disk.

As defects add up, the sectors get skewed across cylinder and track boundaries. To deal with the difference between where a sector should be and where it has slipped to, the controller does a quick scan of the disk when it comes up. It divides the disk into zones and keeps a table of the amount of slippage for each zone. When it does a seek, it adds the amount of slippage for each zone it has to go through, so that it ends up looking somewhere near the desired sector.

Because slipping requires moving all subsequent sectors on the disk, all of diag's strategies for dealing with defective sectors involve reformatting the disk. When it's finished formatting and slipping sectors, it writes a list of the slipped sectors at the end of the disk.

8.2. Starting Diag

`Diag` is a standalone program that lives in `stand/diag`. You must boot it standalone (without UNIX) from the system monitor. After booting, you must then configure `diag` to match your controller/disk configuration. This section provides instructions for both.

User Interface

`diag` responds to commands typed on the workstation keyboard. It contains a number of subsystems, where the command sets are different, and it changes its prompt to remind you of this. The normal prompt is:

```
diag>
```

When you are in the `format` subsystem, it changes to:

```
format>
```

```
>b stand/diag
Boot: disc(0,0,0)stand/diag
Load: disc(0,0,0)boot
Boot: disc(0,0,0)stand/diag
Size: 34816+20480+1160 bytes          [varies by version]

where disc =
xy for a Xylogics controller,
sd for a SCSI controller,
ip for an Interphase controller,[1] or
ec for 3Com, ie for Sun ethernet boards or le
  for Lance ethernet boards.
```

Booting `diag` from a tape requires loading the boot block from the tape, then loading `diag` from the tape. Normally `diag` is the file immediately after the boot block on the tape (file number 1).

To do this, type:

```
>b tape()
Boot: tape(0,0,0)
Boot: tape(0,0,1)
Size: 34816+20480+1160 bytes          [varies by version]
```

where *tape* is **mt** for 1/2" tape with Tapemaster controller, **xt** for 1/2" tape with Xylogics controller, **ar** for 1/4" Archive tape, and **st** for 1/4" SCSI tape.

When `diag` first starts up, it displays a sign on message:

```
Version 2.2 84/08/10                  [varies with different versions]
Disk Initialization and Diagnosis

When asked if you are sure, respond with 'y' or 'Y'
```

Earlier versions of `diag` may not display the version message. These are outdated; use a newer version.

Configuring *diag*

When `diag` starts, it automatically enters the `diag` command subsystem. It prompts for required hardware-specific configuration information, then returns to the command level. To change the configuration later from the command level, enter the command `diag`.

`Diag` needs detailed information about the disk and controller it is working with. It has information about standard configurations built in, and it has a facility for

[1] Sun Microsystems no longer officially supports Interphase 2180 controllers.

0 is the correct response for the first (or only) disk drive connected to the selected controller; **1** is the correct response for the second, and so on. SMD disks can be set for unit numbers of 0 to 3 while SCSI disks are either unit 0 or unit 1.

Next `diag` displays a menu of the different disks for which it has built-in configuration information, then asks for the disk drive type. If you select one of these (except for `other`), `diag` prints some physical data about that disk, including the number of cylinders, number of alternate cylinders, number of heads, and number of sectors per track. If you select `other`, it prompts for this information. Then it initializes the controller, issues a `status` command to the device, and displays results. This works like the `status` command issued to the `diag>` prompt.

The following example shows a Fujitsu M2312K (84–Mbyte unformated) disk controlled by a Xylogics 450 controller with Revision "C" PROMs.

```
Specify drive:
          0 - Fujitsu-M2312K
          1 - Fujitsu-M2284/M2322
          2 - Fujitsu-M2351 Eagle
          3 - Other
which one? 0
ncyl 586 acyl 3 nhead 7 nsect 32
status: ready
drive status: ready
Xylogics PROM Rev 'C'
```

If you have an older Xylogics board with Rev A PROMs, `diag` issues a warning message stating that these boards should be upgraded. They perform certain operations incorrectly, which causes the software to be unable to detect some ECC errors.

Now `diag` has the information it needs about the disk. It returns to the command mode and displays its prompt.

```
diag>
```

If the sequence fails before this point, check the hardware cabling and the information already given. Then use the `diag` command to reenter the data, or reboot `diag`.

Other Disks

If you select the *other* drive type `diag` asks for information about the drive. Once you have defined the new drive type, `diag` adds this entry to the list of drive types, enabling you to use it over and over. `Diag` allows you to define a total of 4 `other` drive types.

NOTE *Consult the disk drive manual for information about* `other` *disk drives.*

The following two examples show how to configure an `other` disk drive for an SMD, then a SCSI controller.

```
Specify drive:
          0 - Micropolis-1304
          1 - Micropolis-1325
          2 - Maxtor-1050
          3 - Fujitsu-2243AS
          4 - Vertex-185
          5 - Other
which one? 5
# of data cylinders? 576                    [total minus alternates]
# of alternate cylinders? 2                 [bad sector spaces]
buffered seek? (usually 2) 2                [from drive manual]
cyl # to start write precomp? 0             [from drive manual]
# of heads? 5
ASCII identification? Phony-4321            [For labeling disk]
# of sectors? 17                           [sectors per track]
ncyl 576 acyl 2 nhead 5 nsect 17
status: 6 Word mode Dma ena
```

8.3. Preparing a New Disk

This section describes how to use `diag` to prepare new SCSI and SMD disks for operation These discussions assume that `diag` is loaded and configured as described earlier.

For SCSI disks, this process includes formatting the disk, performing surface analysis, then writing a label on the disk. The format divides the disk into blocks and sectors, the surface analysis checks for and repairs surface defects, and the label writes important information on the disk, including the disk name, and the partition map.

For SMD disks, this process includes using `fix` to perform surface analysis, then writing a label on the disk.

CAUTION **Do not format any SMD disk you receive from Sun. This will destroy mapping information that you cannot replace. To perform the surface analysis of your SMD disk, use the `fix` command in the SMD**

`format` subsystem, as described later.

The partition table defines the disk partition boundaries, which UNIX uses for file system boundaries. You should have a partition table ready to write when you start this procedure. If you are going to use one of `diag`'s built-in partition tables, this is no problem, but if you are going to use any other partition table, see the instructions for the `partition` command later in this chapter.

NOTE *If you are installing a new system, we recommend that you use the instructions in Installing UNIX on the Sun Workstation to prepare your disks.*

`diag` provides 2 different format subsystems; one for SCSI disks and one for SMD. This section provides separate sections for each.

```
format> r
format> p
Defect list
Defect  Cylinder  Head   Bytes from Index
  N         NN       N          NNNN
etc.
```

4. Make sure this either matches the hardcopy disk defect list. It may have a few more defects on it, but not fewer.

   a) If they match, continue with step 4.

   b) If the hardcopy list shows defects that are not displayed on the screen list, use the a command to add to the list in RAM (it will get written on the disk when you format it):

```
format> a
cylinder? number
head? number
bytes from index? number
```

   c) If you cannot read the list from the disk, use the a command as shown above to type in the entire hardcopy list.

   d) After making any changes to the list on the disk, use the p command to display the changes on the screen. Check again to make sure the copy on the screen matches the hardcopy list.

5. When you have verified the defect listing, format the disk with the format (f) command. After you type this command, the system displays a warning, then asks for confirmation:

```
format> format
format/verify, DESTROYS ALL DISK DATA!
are you sure? y
```

The formatting process takes three minutes or more. At the end, you should see a message:

```
Defect list written on disk
```

If you see any other message (SCSI reset, for example), the formatting process did not succeed, and the defect list was not recorded on the disk. **You must format the disk again.**

6. When you have successfully formated the disk, do a surface analysis using the (s) sub-command. This analyzes the entire disk surface, then displays a list of any defective sectors it found. For a new disk, you should ask for five

11. After labeling the disk, `diag` automatically verifies the label it has just written. The following example shows the verify for a Fujitsu M2322:

```
[ This is an example only; do not enter this information. ]
verify label
id: <Fujitsu-M2322 cyl 821 alt 2 hd 10 sec 32 interleave 1>
        Partition a: starting cyl=0, # blocks=15884 [#'s vary]
        Partition b: starting cyl=50, # blocks=33440
        Partition c: starting cyl=0, # blocks=262720
        Partition g: starting cyl=155, # blocks=213120
diag>
```

12. If you confirm, `diag` partitions your disk according to the default maps shown below, then exits:

```
diag>
```

The following table shows the default partitions for Sun-supplied SCSI disks[3]:

Table 8-2 *Default Partition Sizes for SCSI Disk Subsystems*

SCSI Disk	*Raw*	*Partition Sizes (MBytes)*							
		"a"	"b"	"c"	"d"	"e"	"f"	"g"	"h"
Micropolis 1304	50	8.1	8.4	43.1	*unused*	*unused*	*unused*	26.5	*unused*
Micropolis 1325	85	8.1	17.1	70.9	*unused*	*unused*	*unused*	45.6	*unused*
Maxtor XT-1050	50	8.1	8.4	44.4	*unused*	*unused*	*unused*	27.9	*unused*
Fujitsu M2243AS	86	8.1	17.1	70.8	*unused*	*unused*	*unused*	45.6	*unused*
Vertex V185	85	8.1	17.1	70.9	*unused*	*unused*	*unused*	45.5	*unused*

Using fix to Format SMD Disks

CAUTION **Using `format` destroys the slipping and mapping information on the SMD disk. Since you cannot replace this information, do not use `format` on any SMD disk unless you are SURE you wish to erase this information.**

This section provides instructions for preparing formatting, checking, and labeling an SMD disk. It includes two paths for formatting and testing; one for disks purchased from Sun, and one for disks not purchased from Sun. If you got your disk from Sun, use Step 1A; it uses the `fix` command to check the surface and

[3] Note that the numbers in this table are approximate: formatted capacity depends on the type of controller. Also, note that a 'Megabyte' of disk capacity is defined as one million bytes, and that UNIX file storage capacity is substantially smaller.

the mapping and slipping information on your disk.

a. Enter the `format` command. `Diag` warns you that this destroys disk data, then asks for confirmation:

```
diag> format
format/verify.  DESTROYS ALL DISK DATA
are you sure? y
```

b. Next, `diag` asks for the number of surface analysis passes. We recommend a minimum of 5 surface analysis passes:

```
# of surface analysis passes (5 recommended)? 5
```

c. The rest is automatic: `diag` tests the surface of the disk, slips or maps any defective sectors it finds, then formats the entire disk. During the format, it displays the number of each cylinder it formats. When it's done, it displays a message:

```
cyl nnn format complete - 0 bad sector
Use the label command to label the disk
diag>
```

d. Continue on to Step 2.

NOTE *From here on, this procedure assumes you have are either going to use a default partition table or you have written or modified one using the* `partition` *command. The default partitions are designed for standalone use only; if you are installing an NFS server, or have any other reason for not using the default partition, construct or modify a partition with the* `partition` *command, then return here.*

2. Enter the command `label`:

```
diag> label
```

`Diag` now asks if you want to use default partition map, then asks for confirmation before proceeding:

```
diag> label
label this disk...
OK to use logical partition map 'disk type'? y
Are you sure you want to write?  y
```

NOTE `diag` *remembers if you recently used the partition command; if so, it uses the last partition you accessed in place of 'disk type' above. If you just created a*

Checking and Fixing a Bad Sector (SMD) — Describes how to go to the sector indicated by a UNIX error report, determine correct action, and repair the defect. This procedure provides a reasonable chance of salvaging most of the data on the defective sector.

Electronic Problems — Describes how to set up `diag` to test the swapping area of the disk, so you can perform electronic troubleshooting while using `diag` to check the disk functionality without destroying critical data on the disk.

Checking and Fixing a Bad Sector (SCSI)

Use this procedure to add a bad sector reported by UNIX to the bad sector list on the disk.

NOTE *Because of the nature of the SCSI interface, repairing any surface defect requires reformatting the disk, which erases all the data on it. To ensure the safety of your data, take a full dump before repairing any SCSI disk repair.*

1. Take full dumps of your entire system.
2. Boot and configure `diag` as described earlier in this chapter.
3. Use the `verify` command to find the number of the first cylinder on the partition having the error. Note that in the error report, the last letter in the device name identifies the partition.

   For example:

```
diag> verify
verify label
id: <Micropolis 1304 cyl 825 alt 5 hd 6 sec 177
  Partition a: Starting cyl = 0, # of blocks = 15884
  Partition b: Starting cyl = 156, # of blocks = 16422
  Partition c: Starting cyl = 0, # of blocks = 84150
  Partition g: Starting cyl = 317, # of blocks = 51816
```

4. Using the `add` command, add the block number reported by UNIX to the first cylinder of the partition. For example, if UNIX reported an error in `sd0g`:

```
diag> +
Number: NNN/      [Starting cylinder of g]
  plus: nnnn      [The block number]
= nnnnn = 0xnnnn = CCC/HH/SS (logical)
```

5. Do a `read` of the track to obtain the exact address of the failed block:

```
format> format
DISK FORMAT - DESTROYS ALL DISK DATA!
are you sure? y
Formatting... done              [Takes a while]
Defect list written on disk.
format>
```

11. Restore the system from the dump tape made earlier.

Checking and Fixing a Bad Sector (SMD)

To repair a bad sector on an SMD disk, use the following procedure:

1. Obtain the file system and block number from UNIX error message. Remember the last letter of the *device* identifies the partition.
2. Boot and configure `diag` as described earlier.
3. Using the block number and file system name provided by the UNIX error message, calculate the CC/HH/SS address of the defect as follows:

   a) Enter the `verify` command:

```
diag> v
verify label                    This is an example only
id: <Fujitsu-M2351 Eagle cyl 840 alt 2 hd 20 sec 46>
partition a: starting cyl # 0       # blocks 15884
partition b: starting cyl # 18      # blocks 16442
partition c: starting cyl # 0       # blocks 772800
parititon g: starting cyl # 55      # blocks 722200
```

   b) Use the `add` command to add the starting cylinder number of the file system with the error to the offset provided by the UNIX error message:

```
diag> +             Start addition
number? CC/00/00    Cyl boundary (head and sector = 0)
plus: nnnnnn        Add the offset to it
=dec =hex =CC/HH/SS     Finally, the answer
diag>
```

4. Using the CC/HH/SS results of the above calculation, do a `read` to test the entire track containing the suspected sector (remember, SS = 00). Tell it to test 16 tracks, using one block per transfer:

NOTE *It's a good idea to toggle the 'errors' flag so you can see retrys. Sometimes it is the only way to spot the error. To toggle the flag, enter:*

```
diag> e.
```

```
diag> write
write
starting block? NN
# of blocks? 1
increment? 1
# of blocks per transfer? 1
CC/HH/SS diag>
```

7. Now repeat the `read` from above:

```
diag> read
read
Starting block? CC/HH/00
number of blocks? 16
Increment? 1
# of blocks per transfer? 1
CC/HH/NN diag>
```

8. If the sector checks out OK, the problem is solved. You may want to repeat the `read` a few times to make sure the sector is not marginal. If the read still reports an error, repeat the `map` done earlier, only when it asks for permission to map the sector, answer **`yes`**.

Electronic Problems

Many disk errors are caused by problems with the disk cables or the electronics. These problems tend to generate multiple disk error messages at random locations.

The following procedure shows how to set up `diag` to "bang" on the swap area of the disk. This is where the most expendable data lives; destroying the data here should cause minimum harm. This procedure does not describe how to troubleshoot the electronics; it only describes how to set up `diag` so that the electronics troubleshooting causes the least damage.

For electronics troubleshooting instructions, see the appropriate field service manual.

1. Start up and configure `diag` for the disk with the problem.
2. Enter the `verify` command. For a Micropolis 1304, the display goes:

```
diag> v
verify label
Micropolis 1304 cyl = 824, alt 5, hd 6, sec 17>
partition a: starting cyl # 0      # blocks 15884
partition b: starting cyl # 156    # blocks 16442
partition d: starting cyl # 0      # blocks 84150
parititon g: starting cyl # 317    # blocks 51856
```

the disk, and checks to see if any sector in the range tested has been mapped. If so, it reports this fact. Since `diag` reports errors when trying to read a mapped sector, the mapcheck message may explain an otherwise mysterious error report (default = ON).

Miscellaneous Commands

Use the following commands to help you use `diag`:

`help` or `?` — Display current list of commands available.

`quit` — Exit `diag`.

`clear` — Clear drive faults.

`status` — Fetch and display current controller and drive status.

`diag` — Reset configuration information (described earlier in this manual).

`translate` — Take a disk address and display it in CC/HH/SS, decimal, and hex, translated for the currently configured disk.

addition and subtraction — Entering a + or − on the command line causes a prompt asking you for two numbers (to be added or subtracted). `Diag` provides the result in decimal, hex, and cylinder/sector/head form, translated for the currently configured disk.

`version` — Displays all the *sccs* identifiers in the program.

`verify` — Reads the labels on the disk, prints out the partition map, and if necessary, asks if you want to restore the primary label.

Tests

The following commands perform non-interactive tests:

`position` — This non-destructive test checks the ability to seek and read sectors. It selects and reads single sectors at random, continuing until the user types ^C. If it encounters an error, if `mapcheck` is *on*, it checks to see if the sector is mapped; if so, it reports that fact.

`test` — This **destructive** test writes, then reads groups of sectors continuously, testing for ability to seek, write, and read. It selects sectors at random, and tests a block of sectors starting there. It prompts for the size of the block, and it continues until the user enters ^C. If it finds an error, and if `mapcheck` is ON, it checks the map to see if any sectors in the group are mapped. If so, it reports that fact.

`seek` — This non-destructive test does an hourglass seek over all cylinders, then reports the time it took.

`read/write` — This test checks for ability to read and write data. Write is **destructive** but read is not.

`dmatest` (Xylogics 450 controller only) — This test checks the Xylogics 450 controller using BUFLOAD and BUFDUMP commands. If `abortdma` is ON, it exits when it finds an error; otherwise it continues until the user types ^C.

map

CAUTION **This command can destroy disk data if you add a new mapping.** Backup disk data before proceeding.

The `map` command (SMD disks only) displays the current map table and allows you to add a new mapping.

After you enter `map`, it displays the current map table, then prompts for additional information. When it asks if you want to add a mapping, if you enter `n`, it exits without changing anything. If you enter `y` to add a mapping, it asks if you want to attempt to preserve the data. If you answer `y`, it writes the data to the alternate sector before the mapping is actually done. A typical session goes:

```
diag> map
Current mapping:
Sector CC/HH/SS mapped to CC/HH/SS
Sector CC/HH/SS mapped to CC/HH/SS
Do you wish to add a mapping? y
mapping may be removed only by complete format of the disk
cylinder to be mapped? 704
track to be mapped? 9
sector to be mapped? 7
Attempt to preserve data? y
Data transfer successful!
OK to map 704/9/7? y
mapped sector 704/7/7 to 822/8/31
```

After it maps the sector, `map` adds the new sector to the map, marks the old sector bad, and rewrites the new map table.

If a read error occurs during attempt to preserve data, `map` reports the transfer unsuccessful. This may be only partially true; with fixed ECC errors, the data is still intact. With a hard CRC error, some data may survive, but with other errors the data is usually lost.

You can map a sector on a disk set up for slip sectoring, but slipping a bad sector is preferable to mapping it.

fix

CAUTION **This command damages disk data.** Backup disk data before proceeding.

The `fix` command formats and verifies user-specified sections of SMD disks. Use it to verify a section of the disk without doing the entire thing.

`Fix` requires a starting and ending track address. It also requires a number of surface analysis passes. After it obtains this information, it asks for permission to continue.

A typical session might go:

```
diag> rhdr
read track header
starting track (dd/dd): 00

0/0
sec 0  8000  8100  8200  8300  8400  8500  8600  8700
sec 8  8800  8900  8A00  8B00  8C00  8D00  8E00  8F00
sec 16 9000  9100  9200  9300  9400  9500  9600  9700
sec 24 9800  9900  9A00  9B00  9C00  9D00  9E00  9F00
stop? <ret>
0/1
sec 0  9F01  8001  8101  8201  8301  8401  8501  8601
sec 8  8701  8801  8901  8A01  8B01  8C01  8D01  8E01
sec 16 8F01  9001  9101  9201  9301  9401  9501  9601
sec 24 9700  9801  9901  9A01  9B01  9C01  9D01  9E01
stop? y
```

Reading headers can help locate anomalies in sector headers, and locate spare, mapped, slipped and runt sectors. This in turn can help identify a disk with slip sectoring. The following headers identify unusual sectors:

FFFFFFFF — Mapped sector. These headings identify mapped sectors. UNIX redirects accesses to these sectors to spare sectors as specified by the map. On a disk with slip sectoring, mapping occurs only after a second sector on a track goes bad (very rarely).

FEFEFEFE — Slipped sector. This header marks a place where a logical sector was slipped from.

DDDDDDDD — Spare data sector. Each track on a slip sectored disk starts life with a spare data sector; if a bad sector on that track is slipped, this spare gets used up.

EEEEEEEE — Runt sector. This is the designation given to the extra space at the end of a track; it is usually too short for a data sector but too long to ignore.

Use `rhdr` to help identify mapped or slipped sectors, or to discover if a disk is setup for slip sectoring. Also, if any error message identifies a track or sector as the source of a problem, use `rhdr` to check the headers in that area for consistency.

Understanding a normal header number requires translating it from hex to binary and reading the contents of the fields. The bits are:

A typical session goes:

```
diag> label
label this disk
OK to use logical partition map disk_type? y
Are you sure you want to write? y
```

After it writes the label, it displays the label it has just written. For example, if you had a Fujitsu M2322, it would display:

```
verify label
id: <fujitsu-M2322 cyl 821 alt 2 hd 10 sec 32 interleave 1>
    Partition a: starting cyl=0, # blocks=15884
    Partition b: starting cyl=50, # blocks=33440
    Partition c: starting cyl=0, # blocks=262720
    Partition g: starting cyl=155, # blocks=213120
diag>
```

Note that the numbers in the above display differ depending on disk type.

To use a label other than the 'built in' defaults, see *partition*, and *Installing UNIX on the Sun Workstation*.

partition

The `partition` command selects a table to use when labeling the disk. `Diag` maintains default partition tables for all standard disks; it asks you to select a disk, then prints the default partition table. Then it asks you if you want to modify this table. A typical session goes:

scan

The `scan` command performs repeated sector scans over a specified range of the disk. It is typically used to find additional disk errors after the disk is formated but before UNIX is installed.

CAUTION **This command destroys disk data. Backup disk data before proceeding.**

`scan` looks for new bad sectors and doesn't look at look at mapped sectors. Unlike `fix`, when it does a mapping, it writes the new mapping table to the disk immediately. It runs continuously until interrupted.

`scan` includes a number of options, all of which it prompts for. These are:

scan entire disk? — If you answer `y`, it scans the entire disk; if you answer `n`, it asks for beginning and ending addresses. If the area to scan includes primary and secondary label areas, it displays a message to this effect.

NOTE *If you overwrite the disk label, you will have to re-label the disk before use.*

use random bit patterns? — `y` causes it to use random patterns (better for a small intensive disk scans); `n` causes it to use 5 standard patterns.

perform corrections when defects are found?— If `diag` is configured for a Xylogics controller, `scan` asks if it should do corrections to bad sectors. If `y`, it tries to slip bad sectors and if it can't, it tries to map them. If `n`, it only reports newly found bad sectors. If the controller is not a Xylogics, it displays a message that it cannot fix defective sectors.

Using `scan` without fixing bad sectors is handy for tracking cable problems, or for other cases where you don't want to fix a sector every time you find an error. It is also useful for SCSI surface analysis.

A typical session might go:

```
diag> scan
scan - continuous scan for defective sectors
DESTROYS DISK DATA
scan entire disk? n
starting block? 700
ending block? 702
use random bit patterns? y
perform corrections when errors are found? y
OK to scan from 2/1/25 to 2/1/26? y
type control-C to quit

[pass 1 - bit pattern #1: 0xnnnnnnnn]
2/1/n
```

Scan continues until the user aborts with `control C`. Then it prints:

```
Command aborted
diag>
```

A

File System Check Program

A

File System Check Program

For Release 3.0 the file system blocksize has been changed to 8192. This document has not yet been updated to reflect that change.

When a UNIX† operating system is brought up, a consistency check of the file systems should always be performed. This precautionary measure helps to insure a reliable environment for file storage on disk. If an inconsistency is discovered, corrective action must be taken. `fsck` runs in two modes. Normally it is run non-interactively by the system after a normal boot. When running in this mode, it will only make changes to the file system that are known to always be correct. If an unexpected inconsistency is found `fsck` will exit with a non-zero exit status, leaving the system running single-user. Typically the operator then runs `fsck` interactively. When running in this mode, each problem is listed followed by a suggested corrective action. The operator must decide whether or not the suggested correction should be made.

The purpose of this memo is to dispel the mystique surrounding file system inconsistencies. It first describes the updating of the file system (the calm before the storm) and then describes file system corruption (the storm). Finally, the set of deterministic corrective actions used by `fsck` (the Coast Guard to the rescue) is presented.

A.1. Overview of the File System

The file system is discussed in detail in [Mckusick83]; this section gives a brief overview.

Superblock

A file system is described by its *super-block*. The super-block is built when the file system is created (see *newfs* (8)) and never changes. The super-block contains the basic parameters of the file system, such as the number of data blocks it contains and a count of the maximum number of files. Because the super-block contains critical data, *newfs* replicates it to protect against catastrophic loss. The *default super block* always resides at a fixed offset from the beginning of the file system's disk partition. The *redundant super blocks* are not referenced unless a head crash or other hard disk error causes the default super-block to be unusable. The redundant blocks are sprinkled throughout the disk partition.

This document reflects the use of `fsck` with the revised file system organization implemented in release 0.1 of the Sun UNIX operating system. This is a revision of the original paper written by T. J. Kowalski.

† UNIX is a trademark of AT&T Bell Laboratories.

information stores data.

Fragments

To avoid waste in storing small files, the file system space allocator divides a single file system block into one or more *fragments*. The fragmentation of the file system is specified when the file system is created; each file system block can be optionally broken into 2, 4, or 8 addressable fragments. The lower bound on the size of these fragments is constrained by the disk sector size; typically 512 bytes is the lower bound on fragment size. The block map associated with each cylinder group records the space availability at the fragment level. Aligned fragments are examined to determine block availability.

On a file system with a block size of 4096 bytes and a fragment size of 1024 bytes, a file is represented by zero or more 4096 byte blocks of data, and possibly a single fragmented block. If a file system block must be fragmented to obtain space for a small amount of data, the remainder of the block is made available for allocation to other files. For example, consider an 11000 byte file stored on a 4096/1024 byte file system. This file uses two full size blocks and a 3072 byte fragment. If no fragments with at least 3072 bytes are available when the file is created, a full size block is split yielding the necessary 3072 byte fragment and an unused 1024 byte fragment. This remaining fragment can be allocated to another file, as needed.

Updates to the File System

Every working day hundreds of files are created, modified, and removed. Every time a file is modified, the operating system performs a series of file system updates. These updates, when written on disk, yield a consistent file system. The file system stages all modifications of critical information; modification can either be completed or cleanly backed out after a crash. Knowing the information that is first written to the file system, deterministic procedures can be developed to repair a corrupted file system. To understand this process, the order that the update requests were being honored must first be understood.

When a user program does an operation to change the file system, such as a *write*, the data to be written is copied into an internal *in-core* buffer in the kernel. Normally, the disk update is handled asynchronously; the user process is allowed to proceed even though the data has not yet been written to the disk. The data, along with the inode information reflecting the change, is eventually written out to disk. The real disk write may not happen until long after the *write* system call has returned. Thus at any given time, the file system, as it resides on the disk, lags behind the state of the file system represented by the in-core information.

The disk information is updated to reflect the in-core information when the buffer is required for another use, when a *sync* (2) is done (at 30 second intervals) by */etc/update* (8), or by manual operator intervention with the *sync* (8) command. If the system is halted without writing out the in-core information, the file system on the disk will be in an inconsistent state.

If all updates are done asynchronously, several serious inconsistencies can arise. One inconsistency is that a block may be claimed by two inodes. Such an inconsistency can occur when the system is halted before the pointer to the block in the old inode has been cleared in the copy of the old inode on the disk, and after

Super-block Checking

The most commonly corrupted item in a file system is the summary information associated with the super-block. The summary information is prone to corruption because it is modified with every change to the file system's blocks or inodes, and is usually corrupted after an unclean halt.

The super-block is checked for inconsistencies involving file-system size, number of inodes, free-block count, and the free-inode count. The file-system size must be larger than the number of blocks used by the super-block and the number of blocks used by the list of inodes. The file-system size and layout information are the most critical pieces of information for `fsck`. While there is no way to actually check these sizes, since they are statically determined by *newfs*, `fsck` can check that these sizes are within reasonable bounds. All other file system checks require that these sizes be correct. If `fsck` detects corruption in the static parameters of the default super-block, `fsck` requests the operator to specify the location of an alternate super-block.

Free Block Checking

`fsck` checks that all the blocks marked as free in the cylinder group block maps are not claimed by any files. When all the blocks have been initially accounted for, `fsck` checks that the number of free blocks plus the number of blocks claimed by the inodes equals the total number of blocks in the file system.

If anything is wrong with the block allocation maps, `fsck` will rebuild them, based on the list it has computed of allocated blocks.

The summary information associated with the super-block counts the total number of free blocks within the file system. `fsck` compares this count to the number of free blocks it found within the file system. If the two counts do not agree, then `fsck` replaces the incorrect count in the summary information by the actual free-block count.

The summary information counts the total number of free inodes within the file system. `fsck` compares this count to the number of free inodes it found within the file system. If the two counts do not agree, then `fsck` replaces the incorrect count in the summary information by the actual free-inode count.

Checking the Inode State

An individual inode is not as likely to be corrupted as the allocation information. However, because of the great number of active inodes, a few of the inodes are usually corrupted.

The list of inodes in the file system is checked sequentially starting with inode 2 (inode 0 marks unused inodes; inode 1 is saved for future generations) and progressing through the last inode in the file system. The state of each inode is checked for inconsistencies involving format and type, link count, duplicate blocks, bad blocks, and inode size.

Each inode contains a mode word. This mode word describes the type and state of the inode. Inodes must be one of six types: regular inode, directory inode, symbolic link inode, special block inode, special character inode, or socket inode. Inodes may be found in one of three allocation states: unallocated, allocated, and neither unallocated nor allocated. This last state suggests an incorrectly formated inode. An inode can get in this state if bad data is written into the inode list. The only possible corrective action is for `fsck` is to clear the

Checking the Data Associated with an Inode

An inode can directly or indirectly reference three kinds of data blocks. All referenced blocks must be the same kind. The three types of data blocks are: plain data blocks, symbolic link data blocks, and directory data blocks. Plain data blocks contain the information stored in a file; symbolic link data blocks contain the path name stored in a link. Directory data blocks contain directory entries. `fsck` can only check the validity of directory data blocks.

Each directory data block is checked for several types of inconsistencies. These inconsistencies include directory inode numbers pointing to unallocated inodes, directory inode numbers that are greater than the number of inodes in the file system, incorrect directory inode numbers for "." and "..", and directories that are not attached to the file system. If the inode number in a directory data block references an unallocated inode, then `fsck` will remove that directory entry. Again, this condition can only arise when there has been a hardware failure.

If a directory entry inode number references outside the inode list, then `fsck` will remove that directory entry. This condition occurs if bad data is written into a directory data block.

The directory inode number entry for "." must be the first entry in the directory data block. The inode number for "." must reference itself; e.g., it must equal the inode number for the directory data block. The directory inode number entry for ".." must be the second entry in the directory data block. Its value must equal the inode number for the parent of the directory entry (or the inode number of the directory data block if the directory is the root directory). If the directory inode numbers are incorrect, `fsck` will replace them with the correct values.

File System Connectivity

`fsck` checks the general connectivity of the file system. If directories are not linked into the file system, then `fsck` links the directory back into the file system in the *lost+found* directory. This condition only occurs when there has been a hardware failure.

A.3. Fsck Error Conditions

Conventions

`fsck` is a multi-pass file system check program. Each file system pass invokes a different Phase of the `fsck` program. After the initial setup, `fsck` performs successive Phases over each file system, checking blocks and sizes, path-names, connectivity, reference counts, and the map of free blocks, (possibly rebuilding it), and performs some cleanup.

Normally `fsck` is run non-interactively to *preen* the file systems after an unclean halt. While preening a file system, it will only fix corruptions that are expected to occur from an unclean halt. These actions are a proper subset of the actions that `fsck` will take when it is running interactively. Throughout this appendix many errors have several options that the operator can take. When an inconsistency is detected, `fsck` reports the error condition to the operator. If a response is required, `fsck` prints a prompt message and waits for a response. When preen'ing most errors are fatal. For those that are expected, the response taken is noted. This appendix explains the meaning of each error condition, the possible responses, and the related error conditions.

file is not a block or character device; OK
You have given `fsck` a regular file name by mistake. Check the type of the file specified.

Possible responses to the OK prompt are:

YES
Ignore this error condition.

NO
ignore this file system and continues checking the next file system given.

One of the following messages will appear:
MAGIC NUMBER WRONG
NCG OUT OF RANGE
CPG OUT OF RANGE
NSECT < 1
NTRAK < 1
SPC DOES NOT JIVE w/NTRAK*NSECT
INODES NOT MULTIPLE OF A BLOCK
IMPLIES MORE INODE THAN DATA BLOCKS
NCYL DOES NOT JIVE WITH NCG*CPG
FPG DOES NOT JIVE WITH CPG & SPC
SIZE PREPOSTEROUSLY SMALL
SIZE PREPOSTEROUSLY LARGE
CGSIZE INCORRECT
CSSIZE INCORRECT

and will be followed by the message:
F*: BAD SUPER BLOCK: *B
USE -b OPTION TO FSCK TO SPECIFY LOCATION OF AN ALTERNATE
SUPER-BLOCK TO SUPPLY NEEDED INFORMATION; SEE fsck(8).
The super block has been corrupted. An alternative super block must be selected from among those listed by *newfs* (8) when the file system was created. For file systems with a blocksize less than 32K, specifying –b32 is a good first choice.

CAN NOT SEEK: BLK *B* (CONTINUE)
`fsck`'s request for moving to a specified block number *B* in the file system failed. This should never happen. See a guru.

Possible responses to the CONTINUE prompt are:

YES
attempt to continue to run the file system check. Often, however the problem will persist. This error condition will not allow a complete check of the file system. A second run of `fsck` should be made to re-check this file system. If the block was part of the virtual memory buffer cache, `fsck` will terminate with the message "Fatal I/O error".

Possible responses to the CLEAR prompt are:

YES
de-allocate inode *I* by zeroing its contents. This will always invoke the UNALLOCATED error condition in Phase 2 for each directory entry pointing to this inode.

NO
ignore this error condition.

LINK COUNT TABLE OVERFLOW (CONTINUE)
An internal table for `fsck` containing allocated inodes with a link count of zero has no more room. Recompile `fsck` with a larger value of MAXLNCNT.

Possible responses to the CONTINUE prompt are:

YES
continue with the program. This error condition will not allow a complete check of the file system. A second run of `fsck` should be made to re-check this file system. If another allocated inode with a zero link count is found, this error condition is repeated.

NO
terminate the program.

B* BAD I=*I
Inode *I* contains block number *B* with a number lower than the number of the first data block in the file system or greater than the number of the last block in the file system. This error condition may invoke the **EXCESSIVE BAD BLKS** error condition in Phase 1 if inode *I* has too many block numbers outside the file system range. This error condition will always invoke the **BAD/DUP** error condition in Phase 2 and Phase 4.

EXCESSIVE BAD BLKS I=*I* (CONTINUE)
There is more than a tolerable number (usually 10) of blocks with a number lower than the number of the first data block in the file system or greater than the number of last block in the file system associated with inode *I*.

Possible responses to the CONTINUE prompt are:

YES
ignore the rest of the blocks in this inode and continue checking with the next inode in the file system. This error condition will not allow a complete check of the file system. A second run of `fsck` should be made to re-check this file system.

NO
terminate the program.

B* DUP I=*I
Inode *I* contains block number *B* which is already claimed by another inode.

YES
replace the block count of inode *I* with *Y*.

NO
ignore this error condition.

Phase 1B: Rescan for More Dup's

When a duplicate block is found in the file system, the file system is rescanned to find the inode which previously claimed that block. This section lists the error condition when the duplicate block is found.

B DUP I=*I*
Inode *I* contains block number *B* that is already claimed by another inode. This error condition will always invoke the **BAD/DUP** error condition in Phase 2. You can determine which inodes have overlapping blocks by examining this error condition and the DUP error condition in Phase 1.

Phase 2 – Check Pathnames

This phase concerns itself with removing directory entries pointing to error conditioned inodes from Phase 1 and Phase 1b. This section lists error conditions resulting from root inode mode and status, directory inode pointers in range, and directory entries pointing to bad inodes. All errors in this phase are fatal if the file system is being preen'ed.

ROOT INODE UNALLOCATED. TERMINATING.
The root inode (usually inode number 2) has no allocate mode bits. This should never happen. The program will terminate.

NAME TOO LONG *F*
An excessively long path name has been found. This is usually indicative of loops in the file system name space. This can occur if the super user has made circular links to directories. The offending links must be removed (by a guru).

ROOT INODE NOT DIRECTORY (FIX)
The root inode (usually inode number 2) is not directory inode type.

Possible responses to the FIX prompt are:

YES
replace the root inode's type to be a directory. If the root inode's data blocks are not directory blocks, a VERY large number of error conditions will be produced.

NO
terminate the program.

DUPS/BAD IN ROOT INODE (CONTINUE)
Phase 1 or Phase 1b have found duplicate blocks or bad blocks in the root inode (usually inode number 2) for the file system.

T, and directory name *F* are printed.

Possible responses to the REMOVE prompt are:

YES
the directory entry *F* is removed.

NO
ignore this error condition.

DUP/BAD I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* FILE=*F* (REMOVE)
Phase 1 or Phase 1b have found duplicate blocks or bad blocks associated with directory entry *F*, inode *I*. The owner *O*, mode *M*, size *S*, modify time *T*, and file name *F* are printed.

Possible responses to the REMOVE prompt are:

YES
the directory entry *F* is removed.

NO
ignore this error condition.

Phase 3 – Check Connectivity

This phase concerns itself with the directory connectivity seen in Phase 2. This section lists error conditions resulting from unreferenced directories, and missing or full *lost+found* directories.

DIRECTORY *D* CORRUPTED (SALVAGE)
A directory with an inconsistent internal state has been found. This error is fatal if the file system is being preen'ed.

Possible responses to the SALVAGE prompt are:

YES
throw away all entries up to the next 512-byte boundary. This rather drastic action can throw away up to 42 entries, and should be taken only after other recovery efforts have failed.

NO
Skip up to the next 512-byte boundary and resume reading, but do not modify the directory.

UNREF DIR I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* (RECONNECT)
The directory inode *I* was not connected to a directory entry when the file system was traversed. The owner *O*, mode *M*, size *S*, and modify time *T* of directory inode *I* are printed. When preen'ing, the directory is reconnected if its size is non-zero, otherwise it is cleared.

Possible responses to the RECONNECT prompt are:

YES
reconnect inode *I* to the file system in the directory for lost files (usually *lost+found*). This may invoke the *lost+found* error condition in Phase 4 if there are problems connecting inode *I* to *lost+found.*

NO
ignore this error condition. This will always invoke the CLEAR error condition in Phase 4.

(CLEAR)
The inode mentioned in the immediately previous error condition can not be reconnected. This cannot occur if the file system is being preen'ed, since lack of space to reconnect files is a fatal error.

Possible responses to the CLEAR prompt are:

YES
de-allocate the inode mentioned in the immediately previous error condition by zeroing its contents.

NO
ignore this error condition.

SORRY. NO lost+found DIRECTORY
There is no *lost+found* directory in the root directory of the file system; `fsck` ignores the request to link a file in *lost+found.* This will always invoke the CLEAR error condition in Phase 4. Check access modes of *lost+found*. This error is fatal if the file system is being preen'ed.

SORRY. NO SPACE IN lost+found DIRECTORY
There is no space to add another entry to the *lost+found* directory in the root directory of the file system; `fsck` ignores the request to link a file in *lost+found.* This will always invoke the CLEAR error condition in Phase 4. Check size and contents of *lost+found.* This error is fatal if the file system is being preen'ed.

LINK COUNT FILE I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* COUNT=*X* SHOULD BE *Y* (ADJUST)
The link count for inode *I* which is a file, is *X* but should be *Y*. The owner *O*, mode *M*, size *S*, and modify time *T* are printed. When preen'ing the link count is adjusted.

Possible responses to the ADJUST prompt are:

YES
replace the link count of file inode *I* with *Y*.

NO
ignore this error condition.

NO
ignore this error condition.

BAD/DUP FILE I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* (CLEAR)
Phase 1 or Phase 1b have found duplicate blocks or bad blocks associated with file inode *I*. The owner *O*, mode *M*, size *S*, and modify time *T* of inode *I* are printed. This error cannot arise when the file system is being preen'ed, as it would have caused a fatal error earlier.

Possible responses to the CLEAR prompt are:

YES
de-allocate inode *I* by zeroing its contents.

NO
ignore this error condition.

BAD/DUP DIR I=*I* OWNER=*O* MODE=*M* SIZE=*S* MTIME=*T* (CLEAR)
Phase 1 or Phase 1b have found duplicate blocks or bad blocks associated with directory inode *I*. The owner *O*, mode *M*, size *S*, and modify time *T* of inode *I* are printed. This error cannot arise when the file system is being preen'ed, as it would have caused a fatal error earlier.

Possible responses to the CLEAR prompt are:

YES
de-allocate inode *I* by zeroing its contents.

NO
ignore this error condition.

FREE INODE COUNT WRONG IN SUPERBLK (FIX)
The actual count of the free inodes does not match the count in the super-block of the file system. When preen'ing, the count is fixed.

Possible responses to the FIX prompt are:

YES
replace the count in the super-block by the actual count.

NO
ignore this error condition.

Phase 5 - Check Cyl Groups

This phase concerns itself with the free-block maps. This section lists error conditions resulting from allocated blocks in the free-block maps, free blocks missing from free-block maps, and the total free-block count incorrect.

cg *C*: bad magic number
The magic number of cylinder group *C* is wrong. This usually indicates that the cylinder group maps have been destroyed. When running manually the cylinder group is marked as needing to be reconstructed. This error is fatal if the file

YES
replace the count in the super-block by the actual count.

NO
ignore this error condition.

Phase 6 - Salvage Cylinder Groups

This phase concerns itself with the free-block maps reconstruction. No error messages are produced.

Cleanup

Once a file system has been checked, a few cleanup functions are performed. This section lists advisory messages about the file system and modify status of the file system.

***V* files, *W* used, *X* free (*Y* frags, *Z* blocks)**
This is an advisory message indicating that the file system checked contained *V* files using *W* fragment sized blocks leaving *X* fragment sized blocks free in the file system. The numbers in parenthesis breaks the free count down into *Y* free fragments and *Z* free full sized blocks.

******* REBOOT UNIX *******
This is an advisory message indicating that the root file system has been modified by `fsck`. If UNIX is not rebooted immediately, the work done by `fsck` may be undone by the in-core copies of tables UNIX keeps. When preen'ing, `fsck` will exit with a code of 4. The auto-reboot script interprets an exit code of 4 by issuing a reboot system call.

******* FILE SYSTEM WAS MODIFIED *******
This is an advisory message indicating that the current file system was modified by `fsck`. If this file system is mounted or is the current root file system, `fsck` should be halted and UNIX rebooted. If UNIX is not rebooted immediately, the work done by `fsck` may be undone by the in-core copies of tables UNIX keeps.

A.4. References

[Dolotta78] Dolotta, T. A., and Olsson, S. B. eds., *UNIX User's Manual, Edition 1.1* (January 1978).

[Joy83] Joy, W., Cooper, E., Fabry, R., Leffler, S., McKusick, M., and Mosher, D. *System Interface Overview*, University of California at Berkeley, Computer Systems Research Group Technical Report #4, 1982.

[McKusick83] McKusick, M., Joy, W., Leffler, S., and Fabry, R. *A Fast File System for UNIX*, University of California at Berkeley, Computer Systems Research Group Technical Report #7, 1982.

[Ritchie78] Ritchie, D. M., and Thompson, K., The UNIX Time-Sharing System, *The Bell System Technical Journal* 57,

B

UNIX File System

B

UNIX File System

For release 3.0 the file system blocksize has been changed to 8192. This document has not yet been updated to reflect that change.

This document describes a reimplementation of the UNIX file system. The reimplementation provides substantially higher throughput rates by using more flexible allocation policies, that allow better locality of reference and that can be adapted to a wide range of peripheral and processor characteristics. The new file system clusters data that is sequentially accessed and provides two block sizes to allow fast access for large files while not wasting large amounts of space for small files. File access rates of up to ten times faster than the traditional UNIX file system are experienced. Long needed enhancements to the user interface are discussed. These include a mechanism to lock files, extensions of the name space across file systems, the ability to use arbitrary length file names, and provisions for efficient administrative control of resource usage.

Also described are the changes between the original 512 byte UNIX file system to the file system implemented with the first Berkeley-compatible release of Sun's version of the UNIX system. It presents the motivations for the changes, the methods used to affect these changes, the rationale behind the design decisions, and a description of the new implementation. This discussion is followed by a summary of the results that have been obtained, directions for future work, and the additions and changes that have been made to the user visible facilities. The paper concludes with a history of the software engineering of the project.

The original UNIX system that runs on the PDP-11[1] has simple and elegant file system facilities. File system input/output is buffered by the kernel; there are no alignment constraints on data transfers and all operations are made to appear synchronous. All transfers to the disk are in 512 byte blocks, which can be placed arbitrarily within the data area of the file system. No constraints other than available disk space are placed on file growth [Ritchie74], [Thompson79].

When used together with other UNIX enhancements, the original 512 byte UNIX file system is incapable of providing the data throughput rates that many applications require. For example, applications that need to do a small amount of processing on a large quantities of data such as VLSI design and image processing, need to have a high throughput from the file system. High throughput rates are also needed by programs with large address spaces that are constructed by mapping files from the file system into virtual memory. Paging data in and out of the file system is likely to occur frequently. This requires a file system providing

[1] DEC, PDP, VAX, MASSBUS, and UNIBUS are trademarks of Digital Equipment Corporation.

The allocation of data blocks to files is also suboptimum. The traditional file system never transfers more than 512 bytes per disk transaction and often finds that the next sequential data block is not on the same cylinder, forcing seeks between 512 byte transfers. The combination of the small block size, limited read-ahead in the system, and many seeks severely limits file system throughput.

The first work at Berkeley on the UNIX file system attempted to improve both reliability and throughput. The reliability was improved by changing the file system so that all modifications of critical information were staged so that they could either be completed or repaired cleanly by a program after a crash [Kowalski78]. The file system performance was improved by a factor of more than two by changing the basic block size from 512 to 1024 bytes. The increase was because of two factors; each disk transfer accessed twice as much data, and most files could be described without need to access through any indirect blocks since the direct blocks contained twice as much data. The file system with these changes will henceforth be referred to as the *old file system.*

This performance improvement gave a strong indication that increasing the block size was a good method for improving throughput. Although the throughput had doubled, the old file system was still using only about four percent of the disk bandwidth. The main problem was that although the free list was initially ordered for optimal access, it quickly became scrambled as files were created and removed. Eventually the free list became entirely random causing files to have their blocks allocated randomly over the disk. This forced the disk to seek before every block access. Although old file systems provided transfer rates of up to 175 kilobytes per second when they were first created, this rate deteriorated to 30 kilobytes per second after a few weeks of moderate use because of randomization of their free block list. There was no way of restoring the performance an old file system except to dump, rebuild, and restore the file system. Another possibility would be to have a process that periodically reorganized the data on the disk to restore locality as suggested by [Maruyama76].

B.2. New file system organization

As in the old file system organization each disk drive contains one or more file systems. A file system is described by its super-block, that is located at the beginning of its disk partition. Because the super-block contains critical data it is replicated to protect against catastrophic loss. This is done at the time that the file system is created; since the super-block data does not change, the copies need not be referenced unless a head crash or other hard disk error causes the default super-block to be unusable.

To ensure that it is possible to create files as large as 2↑32 bytes with only two levels of indirection, the minimum size of a file system block is 4096 bytes. The size of file system blocks can be any power of two greater than or equal to 4096. The block size of the file system is maintained in the super-block so it is possible for file systems with different block sizes to be accessible simultaneously on the same system. The block size must be decided at the time that the file system is created; it cannot be subsequently changed without rebuilding the file system.

The new file system organization partitions the disk into one or more areas called *cylinder groups*. A cylinder group is comprised of one or more consecutive cylinders on a disk. Associated with each cylinder group is some bookkeeping

Table B-1 *Wasted Space as a function of Block Size*

Space used	% waste	Organization
775.2 Mb	0.0	Data only, no separation between files
807.8 Mb	4.2	Data only, each file starts on 512 byte boundary
828.7 Mb	6.9	512 byte block UNIX file system
866.5 Mb	11.8	1024 byte block UNIX file system
948.5 Mb	22.4	2048 byte block UNIX file system
1128.3 Mb	45.6	4096 byte block UNIX file system

The space wasted is measured as the percentage of space on the disk not containing user data. As the block size on the disk increases, the waste rises quickly, to an intolerable 45.6% waste with 4096 byte file system blocks.

To be able to use large blocks without undue waste, small files must be stored in a more efficient way. The new file system accomplishes this goal by allowing the division of a single file system block into one or more *fragments*. The file system fragment size is specified at the time that the file system is created; each file system block can be optionally broken into 2, 4, or 8 fragments, each of which is addressable. The lower bound on the size of these fragments is constrained by the disk sector size, typically 512 bytes. The block map associated with each cylinder group records the space availability at the fragment level; to determine block availability, aligned fragments are examined. Figure 1 shows a piece of a map from a 4096/1024 file system.

Bits in map	XXXX	XXOO	OOXX	OOOO
Fragment numbers	0-3	4-7	8-11	12-15
Block numbers	0	1	2	3

Figure B-1 *Example layout of blocks and fragments in a 4096/1024 file system*

Each bit in the map records the status of a fragment; an "X" shows that the fragment is in use, while a "O" shows that the fragment is available for allocation. In this example, fragments 0–5, 10, and 11 are in use, while fragments 6–9, and 12–15 are free. Fragments of adjoining blocks cannot be used as a block, even if they are large enough. In this example, fragments 6–9 cannot be coalesced into a block; only fragments 12–15 are available for allocation as a block.

On a file system with a block size of 4096 bytes and a fragment size of 1024 bytes, a file is represented by zero or more 4096 byte blocks of data, and possibly a single fragmented block. If a file system block must be fragmented to obtain space for a small amount of data, the remainder of the block is made available for allocation to other files. As an example consider an 11000 byte file stored on a 4096/1024 byte file system. This file would uses two full size blocks and a 3072 byte fragment. If no 3072 byte fragments are available at the time the file is

systems in that it uses the same amount of space for small files while requiring less indexing information for large files. This savings is offset by the need to use more space for keeping track of available free blocks. The net result is about the same disk utilization when the new file systems fragment size equals the old file systems block size.

In order for the layout policies to be effective, the disk cannot be kept completely full. Each file system maintains a parameter that gives the minimum acceptable percentage of file system blocks that can be free. If the the number of free blocks drops below this level only the system administrator can continue to allocate blocks. The value of this parameter can be changed at any time, even when the file system is mounted and active. The transfer rates to be given in section 4 were measured on file systems kept less than 90% full. If the reserve of free blocks is set to zero, the file system throughput rate tends to be cut in half, because of the inability of the file system to localize the blocks in a file. If the performance is impaired because of overfilling, it may be restored by removing enough files to obtain 10% free space. Access speed for files created during periods of little free space can be restored by recreating them once enough space is available. The amount of free space maintained must be added to the percentage of waste when comparing the organizations given in Table 1. Thus, a site running the old 1024 byte UNIX file system wastes 11.8% of the space and one could expect to fit the same amount of data into a 4096/512 byte new file system with 5% free space, since a 512 byte old file system wasted 6.9% of the space.

File system parameterization

Except for the initial creation of the free list, the old file system ignores the parameters of the underlying hardware. It has no information about either the physical characteristics of the mass storage device, or the hardware that interacts with it. A goal of the new file system is to parameterize the processor capabilities and mass storage characteristics so that blocks can be allocated in an optimum configuration dependent way. Parameters used include the speed of the processor, the hardware support for mass storage transfers, and the characteristics of the mass storage devices. Disk technology is constantly improving and a given installation can have several different disk technologies running on a single processor. Each file system is parameterized so that it can adapt to the characteristics of the disk on which it is placed.

For mass storage devices such as disks, the new file system tries to allocate new blocks on the same cylinder as the previous block in the same file. Optimally, these new blocks will also be well positioned rotationally. The distance between "rotationally optimal" blocks varies greatly; it can be a consecutive block or a rotationally delayed block depending on system characteristics. On a processor with a channel that does not require any processor intervention between mass storage transfer requests, two consecutive disk blocks often can be accessed without suffering lost time because of an intervening disk revolution. For processors without such channels, the main processor must field an interrupt and prepare for a new disk transfer. The expected time to service this interrupt and schedule a new disk transfer depends on the speed of the main processor.

The physical characteristics of each disk include the number of blocks per track and the rate at which the disk spins. The allocation policy routines use this

the "list directory" command often accesses the inode for each file in a directory. The layout policy tries to place all the files in a directory in the same cylinder group. To ensure that files are allocated throughout the disk, a different policy is used for directory allocation. A new directory is placed in the cylinder group that has a greater than average number of free inodes, and the fewest number of directories in it already. The intent of this policy is to allow the file clustering policy to succeed most of the time. The allocation of inodes within a cylinder group is done using a next free strategy. Although this allocates the inodes randomly within a cylinder group, all the inodes for each cylinder group can be read with 4 to 8 disk transfers. This puts a small and constant upper bound on the number of disk transfers required to access all the inodes for all the files in a directory as compared to the old file system where typically, one disk transfer is needed to get the inode for each file in a directory.

The other major resource is the data blocks. Since data blocks for a file are typically accessed together, the policy routines try to place all the data blocks for a file in the same cylinder group, preferably rotationally optimally on the same cylinder. The problem with allocating all the data blocks in the same cylinder group is that large files will quickly use up available space in the cylinder group, forcing a spill over to other areas. Using up all the space in a cylinder group has the added drawback that future allocations for any file in the cylinder group will also spill to other areas. Ideally none of the cylinder groups should ever become completely full. The solution devised is to redirect block allocation to a newly chosen cylinder group when a file exceeds 32 kilobytes, and at every megabyte thereafter. The newly chosen cylinder group is selected from those cylinder groups that have a greater than average number of free blocks left. Although big files tend to be spread out over the disk, a megabyte of data is typically accessible before a long seek must be performed, and the cost of one long seek per megabyte is small.

The global policy routines call local allocation routines with requests for specific blocks. The local allocation routines will always allocate the requested block if it is free. If the requested block is not available, the allocator allocates a free block of the requested size that is rotationally closest to the requested block. If the global layout policies had complete information, they could always request unused blocks and the allocation routines would be reduced to simple bookkeeping. However, maintaining complete information is costly; thus the implementation of the global layout policy uses heuristic guesses based on partial information.

If a requested block is not available the local allocator uses a four level allocation strategy:

1) Use the available block rotationally closest to the requested block on the same cylinder.

2) If there are no blocks available on the same cylinder, use a block within the same cylinder group.

3) If the cylinder group is entirely full, quadratically rehash among the cylinder groups looking for a free block.

Table B-2 *Reading Rates of the Old and New UNIX File Systems*

Type of File System	Processor and Bus Measured	Read Speed	Read Bandwidth	% CPU
old 1024	750/UNIBUS	29 Kbytes/sec	29/1100 3%	11%
new 4096/1024	750/UNIBUS	221 Kbytes/sec	221/1100 20%	43%
new 8192/1024	750/UNIBUS	233 Kbytes/sec	233/1100 21%	29%
new 4096/1024	750/MASSBUS	466 Kbytes/sec	466/1200 39%	73%
new 8192/1024	750/MASSBUS	466 Kbytes/sec	466/1200 39%	54%

Table B-3 *Writing rates of the old and new UNIX file systems*

Type of File System	Processor and Bus Measured	Write Speed	Write Bandwidth	% CPU
old 1024	750/UNIBUS	48 Kbytes/sec	48/1100 4%	29%
new 4096/1024	750/UNIBUS	142 Kbytes/sec	142/1100 13%	43%
new 8192/1024	750/UNIBUS	215 Kbytes/sec	215/1100 19%	46%
new 4096/1024	750/MASSBUS	323 Kbytes/sec	323/1200 27%	94%
new 8192/1024	750/MASSBUS	466 Kbytes/sec	466/1200 39%	95%

Unlike the old file system, the transfer rates for the new file system do not appear to change over time. The throughput rate is tied much more strongly to the amount of free space that is maintained. The measurements in Table 2 were based on a file system run with 10% free space. Synthetic work loads suggest the performance deteriorates to about half the throughput rates given in Table 2 when no free space is maintained.

The percentage of bandwidth given in Table 2 is a measure of the effective utilization of the disk by the file system. An upper bound on the transfer rate from the disk is measured by doing 65536[7] byte reads from contiguous tracks on the disk. The bandwidth is calculated by comparing the data rates the file system is able to achieve as a percentage of this rate. Using this metric, the old file system is only able to use about 3-4% of the disk bandwidth, while the new file system uses up to 39% of the bandwidth.

In the new file system, the reading rate is always at least as fast as the writing rate. This is to be expected since the kernel must do more work when allocating blocks than when simply reading them. Note that the write rates are about the same as the read rates in the 8192 byte block file system; the write rates are slower than the read rates in the 4096 byte block file system. The slower write

[7] This number, 65536, is the maximal I/O size supported by the VAX hardware; it is a remnant of the system's PDP-11 ancestry.

B.4. File system functional enhancements

The speed enhancements to the UNIX file system did not require any changes to the semantics or data structures viewed by the users. However several changes have been generally desired for some time but have not been introduced because they would require users to dump and restore all their file systems. Since the new file system already requires that all existing file systems be dumped and restored, these functional enhancements have been introduced at this time.

Long file names

File names can now be of nearly arbitrary length. The only user programs affected by this change are those that access directories. To maintain portability among UNIX systems that are not running the new file system, a set of directory access routines have been introduced that provide a uniform interface to directories on both old and new systems.

Directories are allocated in units of 512 bytes. This size is chosen so that each allocation can be transferred to disk in a single atomic operation. Each allocation unit contains variable-length directory entries. Each entry is wholly contained in a single allocation unit. The first three fields of a directory entry are fixed and contain an inode number, the length of the entry, and the length of the name contained in the entry. Following this fixed size information is the null terminated name, padded to a 4 byte boundary. The maximum length of a name in a directory is currently 255 characters.

Free space in a directory is held by entries that have a record length that exceeds the space required by the directory entry itself. All the bytes in a directory unit are claimed by the directory entries. This normally results in the last entry in a directory being large. When entries are deleted from a directory, the space is returned to the previous entry in the same directory unit by increasing its length. If the first entry of a directory unit is free, then its inode number is set to zero to show that it is unallocated.

File locking

The old file system had no provision for locking files. Processes that needed to synchronize the updates of a file had to create a separate "lock" file to synchronize their updates. A process would try to create a "lock" file. If the creation succeeded, then it could proceed with its update; if the creation failed, then it would wait, and try again. This mechanism had three drawbacks. Processes consumed CPU time, by looping over attempts to create locks. Locks were left lying around following system crashes and had to be cleaned up by hand. Finally, processes running as system administrator are always permitted to create files, so they had to use a different mechanism. While it is possible to get around all these problems, the solutions are not straight-forward, so a mechanism for locking files has been added.

The most general schemes allow processes to concurrently update a file. Several of these techniques are discussed in [Peterson83]. A simpler technique is to simply serialize access with locks. To attain reasonable efficiency, certain applications require the ability to lock pieces of a file. Locking down to the byte level has been implemented in the Onyx file system by [Bass81]. However, for the applications that currently run on the system, a mechanism that locks at the granularity of a file is sufficient.

Symbolic links

The 512 byte UNIX file system allows multiple directory entries in the same file system to reference a single file. The link concept is fundamental; files do not live in directories, but exist separately and are referenced by links. When all the links are removed, the file is deallocated. This style of links does not allow references across physical file systems, nor does it support inter-machine linkage. To avoid these limitations *symbolic links* have been added similar to the scheme used by Multics [Feiertag71].

A symbolic link is implemented as a file that contains a pathname. When the system encounters a symbolic link while interpreting a component of a pathname, the contents of the symbolic link is prepended to the rest of the pathname, and this name is interpreted to yield the resulting pathname. If the symbolic link contains an absolute pathname, the absolute pathname is used, otherwise the contents of the symbolic link is evaluated relative to the location of the link in the file hierarchy.

Normally programs do not want to be aware that there is a symbolic link in a pathname that they are using. However certain system utilities must be able to detect and manipulate symbolic links. Three new system calls provide the ability to detect, read, and write symbolic links, and seven system utilities were modified to use these calls.

In future Berkeley software distributions it will be possible to mount file systems from other machines within a local file system. When this occurs, it will be possible to create symbolic links that span machines.

Rename

Programs that create new versions of data files typically create the new version as a temporary file and then rename the temporary file with the original name of the data file. In the old UNIX file systems the renaming required three calls to the system. If the program were interrupted or the system crashed between these calls, the data file could be left with only its temporary name. To eliminate this possibility a single system call has been added that performs the rename in an atomic fashion to guarantee the existence of the original name.

In addition, the rename facility allows directories to be moved around in the directory tree hierarchy. The rename system call performs special validation checks to ensure that the directory tree structure is not corrupted by the creation of loops or inaccessible directories. Such corruption would occur if a parent directory were moved into one of its descendants. The validation check requires tracing the ancestry of the target directory to ensure that it does not include the directory being moved.

Quotas

The UNIX system has traditionally attempted to share all available resources to the greatest extent possible. Thus any single user can allocate all the available space in the file system. In certain environments this is unacceptable. Consequently, a quota mechanism has been added for restricting the amount of file system resources that a user can obtain. The quota mechanism sets limits on both the number of files and the number of disk blocks that a user may allocate. A separate quota can be set for each user on each file system. Each resource is given both a hard and a soft limit. When a program exceeds a soft limit, a warning is printed on the users terminal; the offending program is not terminated

the new one and from the new one to the old one. Having the file system interpretation implemented in user code had several major benefits. These included being able to use the standard system tools such as the debuggers to set breakpoints and single step through the code. When bugs were discovered, the offending problem could be fixed and tested without the need to reboot the machine. There was never a period where it was necessary to maintain two concurrent file systems in the kernel. Finally it was not necessary to dedicate a machine entirely to file system development, except for a brief period while the new file system was boot strapped.

The final step was to merge the new file system back into the UNIX kernel. This was done in less than two weeks, since the only bugs remaining were those that involved interfacing to the synchronization routines that could not be tested in the simulated system. Again the simulation system proved useful since it enabled files to be easily copied between old and new file systems regardless of which file system was running in the kernel. This greatly reduced the number of times that the system had to be rebooted.

The total design and debug time took about one man year. Most of the work was done on the file system utilities, and changing all the user programs to use the new facilities. The code changes in the kernel were minor, involving the addition of only about 800 lines of code.

B.6. Acknowledgements

We thank Robert Elz for his ongoing interest in the new file system, and for adding disk quotas in a rational and efficient manner. We also acknowledge Dennis Ritchie for his suggestions on the appropriate modifications to the user interface. We appreciate Michael Powell's explanations on how the DEMOS file system worked; many of his ideas were used in this implementation. Special commendation goes to Peter Kessler and Robert Henry for acting like real users during the early debugging stage when files were less stable than they should have been. Finally we thank our sponsors, the National Science Foundation under grant MCS80-05144, and the Defense Advance Research Projects Agency (DoD) under Arpa Order No. 4031 monitored by Naval Electronic System Command under Contract No. N00039-82-C-0235.

B.7. References

[Accetta80] Accetta, M., Robertson, G., Satyanarayanan, M., and Thompson, M. "The Design of a Network-Based Central File System", Carnegie-Mellon University, Dept of Computer Science Tech Report, #CMU-CS-80-134

[Almes78] Almes, G., and Robertson, G. "An Extensible File System for Hydra" Proceedings of the Third International Conference on Software Engineering, IEEE, May 1978.

[Bass81] Bass, J. "Implementation Description for File Locking", Onyx Systems Inc, 73 E. Trimble Rd, San Jose, CA 95131 Jan 1981.

[Dion80] Dion, J. "The Cambridge File Server", Operating Systems Review, 14, 4. Oct 1980. pp 26-35

Review, 15, 4. Oct 1981. pp 39-54

[Sturgis80] Sturgis, H., Mitchell, J., and Israel, J. "Issues in the Design and Use of a Distributed File System", Operating Systems Review, 14, 3. pp 55-79

[Symbolics81a] "Symbolics File System", Symbolics Inc, 9600 DeSoto Ave, Chatsworth, CA 91311 Aug 1981.

[Symbolics81b] "Chaosnet FILE Protocol". Symbolics Inc, 9600 DeSoto Ave, Chatsworth, CA 91311 Sept 1981.

[Thompson79] Thompson, K. "UNIX Implementation", Section 31, Volume 2B, UNIX Programmers Manual, Bell Laboratory, Murray Hill, NJ 07974. Jan 1979

[Thompson80] Thompson, M. "Spice File System", Carnegie-Mellon University, Dept of Computer Science Tech Report, #CMU-CS-80-???

[Trivedi80] Trivedi, K. "Optimal Selection of CPU Speed, Device Capabilities, and File Assignments", Journal of the ACM, 27, 3. July 1980. pp 457-473

[White80] White, R. M. "Disk Storage Technology", Scientific American, 243(2), August 1980.

UUCP Implementation Description

C

UUCP Implementation Description

C

UUCP Implementation Description

Uucp is a series of programs designed such that UNIX systems can communicate with each other using either dial-up or hardwired communication lines. *Uucp* transfers files between UNIX systems, and can also run commands on remote machines. This document gives a detailed description of the current implementation of *uucp*. It is designed for use by an administrator or installer of the system. It is not meant as a user's guide.

Uucp is a batch operation. Files are created in a spool directory for processing by the *uucp* daemons. There are three types of files used for the execution of work: *Data files* contain data for transfer to remote systems; *work files* contain directions for file transfers between systems; *execute files* are scripts for UNIX commands that involve the resources of one or more systems.

There are four primary programs involved in *uucp*'s operation:

uucp	builds *work files* and gathers *data files* in the spool directory for data transmission.
uux	creates *work files* and *execute files*, and gathers *data files* for the remote execution of UNIX commands.
uucico	executes the work files for data transmission.
uuxqt	executes the scripts for UNIX command execution.

There are two administrative programs:

uulog	gathers temporary log files that may occur due to lockout of the *uucp* log file and reports some information such as copy requests and completion status.
uuclean	removes old files from the spool directory.

The remaining sections of this manual describe the operation of each program, security, installation details, files required for execution, and administration of the *uucp* system.

C.1. UUCP — UNIX to UNIX File Copy

The *uucp* command was is designed to look like *cp* (1) to the user. The syntax is:

uucp [option] ... source ... destination

where the source and destination may contain the prefix *system-name!*, which indicates the system where the file or files reside or where they will be copied.

sets up the transfer of files whose names end with *.h* in dan's login directory on system *usg* to dan's local login directory.

For each source file, *uucp* checks the source and destination file-names, the system-part of each argument, and the options to classify the work into several types:

[1] Copy source to destination on local system.

[2] Receive files from other systems.

[3] Send files to a remote system.

[4] Send files from remote systems to another remote system.

[5] Receive files from remote systems when the source contains special shell characters as mentioned above.

[6] Request that the *uucp* command be executed by a remote system.

After the work has been set up in the spool directory, the *uucico* program is started to try to contact the other machine and execute the work (unless the –r option was specified).

Type 1 — Local Copy

The copy is done locally. The *–m* and *–n* options are not honored in this case.

Type 2 — Receive Files

A *work file* is created or appended with a one line entry for each request. The upper limit to the number of files per *work file* is set in *uucp.h* The default setting is 20. After the limit has been reached, a new *work file* is created.

All *work files* and *execute files* use a blank as the field separator. The fields for these entries are given below.

[1] R

[2] The full path-name of the source or a ~something/path-name. The *~something* part is expanded on the remote system.

[3] The full path-name of the destination file. If the *~something* notation is used, it is immediately expanded.

[4] The user's login name.

[5] A "–" followed by an option list. The options –m and –d may appear.

Type 3 — Send Files

Each source file is copied into a *data file* in the spool directory. (A –c option on the *uucp* command prevents the *data file* from being made. In this case, the file is transmitted from the indicated source.) The fields for these entries are given below.

[1] S

[2] The full-path name of the source file.

[3] The full-path name of the destination or ~something/file-name.

[4] The user's login name.

–x*num* *Num* is a level number between 1 and 9; higher numbers give more debugging output.

The command:

```
pr  abc  |  uux  -  usg!lpr
```

sets up the output of the

```
pr abc
```

command as standard input to an *lpr* command to be executed on system *usg*.

Uux generates an *execute file* containing the names of the files required for execution (including standard input), the user's login name, the destination of the standard output, and the command to be executed. This *execute file* file is either put in the spool directory for local execution or sent to the remote system using a send command (*uucp* type 3 command, described previously).

For required files that are not on the execution machine, *uux* generates receive command files (*uucp* type 2 command, described previously). The *uucico* program puts these command-files on the execution machine for execution.

The *execute file* contains a script that is processed by the *uuxqt* program. It is made up of several lines, each of which contains an identification character and one or more arguments. The lines are described in the subsections below. Here is a summary of the types of lines that appear in the file. They are described in detail in the sections following.

User Line Identifies the requestor's login name and system.

File Line Identifies a filename for transmission.

Standard Input Line
Specifies a standard input file.

Standard Output Line
Specifies a standard output file.

Command Line
Identifies a UNIX system command for *uuxqt* to execute.

User Line

U user system

where the *user* and *system* are the requester's login name and system.

Required File Line

F file-name real-name

where *file-name* is a unique name used for file transmission and *real-name* is the last part of the actual file name (contains no path information).

Zero or more of these lines may be present. The *uuxqt* program checks for the existence of all these files before the command is executed.

- Execute all requests from both systems.
- Log work requests and work completions.

Uucico may be started in several ways:

a. by a system daemon specified in a crontab entry,

b. by one of the *uucp, uux, uuxqt* or *uucico* programs,

c. directly by the user (this is usually for testing),

d. by a remote system. The uucico program should be specified as the "shell" field in the */etc/passwd* file for the logins used by remote systems to access uucp.

When started by method a, b or c, *uucico* is considered to be in *MASTER* mode. In this mode, a connection is made to a remote system. If started by a remote system (method d), *uucico* is considered to be in *SLAVE* mode.

The *MASTER* mode operates in one of two ways. If no system name is specified (–s option not specified) *uucico* scans the spool directory for systems to call. If a system name is specified, that system is called, and work is only done for that system.

Uucico is generally started by another program. There are several options used for execution:

–r1 Start *uucico* in *MASTER* mode. This is used when *uucico* is started by a program or "cron" shell.

–s*sys* Do work only for system *sys*. If *–s* is specified, a call to the specified system is made even if there is no work for system *sys* in the spool directory. This is useful for polling systems that do not have the hardware to initiate a connection.

The following options are used primarily for debugging:

–d*dir* Use directory *dir* for the spool directory.

–x*num* *Num* is a level number between 1 and 9; higher numbers give more debugging output.

The next part of this section describes the major steps within the *uucico* program.

Scan For Work

The names of the work related files in the spool directory have format

type . system-name grade number

type is an upper case letter (*C* – copy command file, *D* – data file, *X* – execute file),

system-name
is the remote system, *truncated to seven characters.*

The calling program checks *proto-list* for a letter corresponding to an available line protocol and returns a *use-protocol* message. The *use-protocol* message is

U*code*

where *code* is either a one character protocol letter or "N", which means there is no common protocol. The only protocol which is currently implemented is "g", which uses the packet driver.

Work Processing

The *MASTER* program does a work search similar to the one used in the *Scan For Work* section described above (the *MASTER* has been specified by the "–r1" uucico option). Each message used during the work processing is specified by the first character of the message:

S send a file,

R receive a file,

C copy complete,

X execute a *uucp* command,

H hangup.

The *MASTER* sends *R*, *S* or *X* messages until all work for the remote system is complete, at which point an *H* message is sent. The *SLAVE* replies with *SY*, *SN*, *RY*, *RN*, *HY*, *HN*, *XY*, or *XN*, corresponding to *yes* or *no* for each request.

An *N* response can be followed by a number giving the reson for the failure:

N0
Copy failed (reason not given by remote system).

N1
Local access to file denied.

N2
Remote access to path or file denied.

N3
System error – bad *uucp* command generated.

N4
Remote system cannot create temporary file.

N5
Cannot copy to file or directory – file left in *pubdir/user/file*.

N6
Cannot copy to file or directory – file left in *pubdir/user/file*.

The send and receive replies are based on permission to access the requested file or directory using the *USERFILE* and read/write permissions of the file or directory.

After each file is copied into the spool directory of the receiving system, a copy-complete message is sent by the receiver of the file. The message *CY* is sent if the file has successfully been moved from the spool directory to the destination.

–p*pre* Examine files with prefix *pre* for deletion. Up to 10 of these options may be specified.

–x*num* This is the level of debugging output desired.

C.7. Security

CAUTION **The *uucp* system, left unrestricted, will let any outside user execute any commands and copy out/in any file that is readable/writable by a *uucp* login user. It is up to the individual sites to be aware of this and apply the protections that they feel are necessary.**

There are several security features available aside from the normal file mode protections. These must be set up by the administrator of the *uucp* system.

- The login for *uucp* does not get a standard shell. Instead, the *uucico* program is started so that all work is done through *uucico*.
- The owner of the *uucp* programs should be an administrative login. It should not be one of the logins used for remote system access to *uucp*.
- All *uucp* logins should have passwords.
- A path check is done on file names that are to be sent or received. The *USERFILE* supplies the information for these checks. The *USERFILE* can also be set up to require call-back for certain login-ids. See the "Files Required For Execution" section for the file description.
- A conversation sequence count can be set up so that the called system can be more confident of the caller's identity.
- The *uuxqt* program reads a file containing a list of commands that it will execute. A "PATH" shell statement is prepended to the command line as specified in the *uuxqt* program. The installer may modify the list or remove the restrictions as desired.
- The *L.sys* file should be owned by the *uucp* administrative login and have mode 0400 to protect the phone numbers and login information for remote sites.
- The programs *uucp*, *uucico*, *uux*, *uuxqt*, *uulog*, and *uuclean* should be owned by the *uucp* administrative login, have the setuid bit set, and have only execute permissions.

C.8. UUCP Installation

It is assumed that the *login name* used by a remote computer to call into a local computer is not the same as the login name of a normal user or the *uucp* administrative login. However, several remote computers may use the same login name. It is suggested that the installer follow the convention of using the letter "U" followed by the system name as the login name for each system. For example, use login name *Uusg* for the *usg* system.

Each computer should be given a unique *system name* that is transmitted at the start of each call. This name identifies the calling machine to the called machine. The *login/system* names are used for security as described later in the *USERFILE*

PUBDIR	is a public directory for remote access. This is also the login directory for remote *uucp* users. It should be the same as that defined in *uucp.h*.
SPOOL	is the uucp spool directory. This should be the same as that defined in *uucp.h*.
XQTDIR	is the directory for uuxqt to use for command execution. It is also defined in *uucp.h*.
OWNER	is the administrative login for *uucp*.
LIBS	should include *syskludge/syskludge.a if the syskludge library is used. UUDIR should be defined in uucp.h*.
CFLAGS	add –DVMUNIX if on a VMUNIX system.

Compiling the System

The command:

```
make install
```

makes the required directories, compiles all programs, sets the proper file modes, and copies the programs to the proper directories. This command should be run as *root*. The command:

```
make
```

compiles the entire system.

The programs *uucp*, *uux*, and *uulog* should be put in */usr/bin*. The programs *uuxqt*, *uucico*, and *uuclean* should be put in the *program* directory.

Files Required for Execution

Six files are required for execution. They should reside in the *program* directory. The field separator for all files is a space. The required files are summarized here, and the following subsections describe them in detail.

L-devices	Contains call unit information.
L-dialcodes	Contains dialcode abbreviations.
USERFILE	Contains user accessibility and constraint information.
L.sys	Contains information about the systems which local *uucp* programs can call.
L.cmds	Contains commands which *uuxqt* is allowed to execute.
SYSTEMNAME	Contains the name of the system.

L-devices — Call Unit Information File

L-devices contains call-unit device and hardwired connection information. The special device files are assumed to be in the */dev* directory. The format for each entry in the *L-devices* file is:

type line call-unit speed

login,sys [c] path-name [path-name] ...

login is the login name for a user or the remote computer,

sys is the system name for a remote computer,

c is the optional *call-back required* flag,

path-name is a path-name prefix that is acceptable for *sys*.

The constraints are implemented as follows.

[1] When the program is obeying a command stored on the local machine (*MASTER* mode) the path-names allowed are those given on the first line in the *USERFILE* that has the login name of the user who entered the command. If no such line is found, the first line with a *null* login name is used.

[2] When the program is responding to a command from a remote machine (*SLAVE* mode) the path-names allowed are those given on the first line in the file that has the system name that matches the remote machine. If no such line is found, the first one with a *null* system name is used.

[3] When a remote computer logs in, the login name that it uses *must* appear in the *USERFILE*. There may be several lines with the same login name but one of them must either have the name of the remote system or must contain a *null* system name.

[4] If the line matched in ([3]) contains a "c", the remote machine is called back before any transactions take place.

Examples

The line:

```
u,m  /usr/xyz
```

allows machine *m* to login with name *u* and request the transfer of files whose names start with */usr/xyz*.

The line:

```
dan,  /usr/dan
```

allows the ordinary user *dan* to issue commands for files whose name starts with */usr/dan*. (Note that this type of restriction is seldom used.)

The lines:

```
u,m /usr/xyz  /usr/spool
u,  /usr/spool
```

allows any remote machine to login with name *u*. If its system name is not *m*, it can only ask to transfer files whose names start with */usr/spool*. If it is system *m*, it can send files from path */usr/xyz* as well as */usr/spool*.

The lines:

device — This fields either starts with *ACU*, or is the hardwired device to be used for the call. For the hardwired case, the last part of the special file name is used (tty0, for instance).

class — This is usually the line speed for the call (for example, 300). The exception is when the C library routine "dialout" is available in which case this is the dialout class.

phone — The phone number is made up of an optional alphabetic abbreviation and a numeric part. The abbreviation should be one that appears in the *L-dialcodes* file (for example, mh5900, boston995–9980). For the hardwired devices, this field contains the same string as used for the *device* field.

login — The login information is given as a series of fields and subfields in the format

```
[ expect  send ]  . . .
```

where *expect* is the string expected to be read and *send* is the string to be sent when the *expect* string is received.

The expect field may be made up of subfields of the form

```
expect[-send-expect] . . .
```

where the *send* is sent if the prior *expect* is *not* successfully read and the *expect* following the *send* is the next expected string. For example:

```
login--login
```

expects to see the word *login*; if it gets it, the program proceeds to the next field; if it does not get *login*, it sends *null* followed by a new line, then expects *login* again.

There are two special names available to be sent during the login sequence. The string *EOT* sends an EOT character and the string *BREAK* tries to send a BREAK character. The *BREAK* character is simulated using line speed changes and null characters and may not work on all devices and/or systems. A number from 1 to 9 may follow the *BREAK*. For example, *BREAK1* sends 1 null character instead of the default of 3. Note that *BREAK1* usually works best for 300/1200 baud lines.

The following escape sequences are also recognized:

\r — send a carriage-return.

\n — send a newline (linefeed) character.

\d — delay for 1 second.

L.sys file.

Note that the telephone numbers specified in the *L.sys* file will have a format dependent on the ACU device type. This is a deficiency which may be corrected in the future.

C.9. Administration

This section indicates some events and files that must be administered for the *uucp* system. Some administration can be accomplished by *shell files* initiated by *crontab* entries. Others may require manual intervention. Some sample *shell files* are given toward the end of this section.

SQFILE — Sequence Check File

SQFILE is set up in the *program* directory and contains an entry for each remote system with which you agree to perform conversation sequence checks. The initial entry is just the system name of the remote system. The first conversation adds the conversation count and the date/time of the most resent conversation. These items are updated with each conversation. If a sequence check fails, the entry will have to be adjusted manually. Note that this feature is rarely used.

TM — Temporary Data Files

These files are created in the *spool* directory while a file is being copied from a remote machine. Their names have the form

```
TM.pid.ddd
```

where *pid* is a process-id and *ddd* is a sequential three digit number starting at zero. After the entire file is received, the *TM* file is moved/copied to the requested destination. If processing is abnormally terminated the file remains in the spool directory. The leftover files should be periodically removed; the *uuclean* program is useful in this regard. The command

```
program/uuclean  -pTM
```

removes all *TM* files older than three days.

LOG — Log Entry Files

During execution, log information is appended to the *LOGFILE*. If the *LOGFILE* is locked by another process, the log information is placed in individual log files with a with a *LOG* prefix. These individual files should be combined into the *LOGFILE* by using the *uulog* program. *Uulog* appends the contents of the individual log files onto the end of the *LOGFILE*. The command:

```
uulog
```

accomplishes the merge. Options are available to print some or all the log entries after the files are merged. The *LOGFILE* should be removed periodically.

The *LOG* files are created initially with mode 0222. If the program that creates the file terminates normally, it changes the mode to 0666. Aborted runs may leave the files with mode 0222 and the *uulog* program will not read or remove them. To remove them, either use *rm*, *uuclean*, or change the mode to 0666 and let *uulog* merge them into the *LOGFILE*.

```
cp spool/LOGFILE    spool/o.LOGFILE
rm spool/LOGFILE
```

can be used.

The shell files in *program*/uucp.* do a more extensive job than that described here. They should be started by entries in *crontab*. Read the shell files for more information.

Login Entry

Two or more logins should be set up for *uucp*. One should be an administrative login: the owner of all the uucp programs, directories and files. All others are used by remote systems to access the uucp system. Each of the */etc/passwd* entries for the *access* logins should have *program*/uucico as the shell to be executed. The login directory should be the public directory (usually */usr/spool/uucppublic*) for both the administrative login and the access logins. The various *access* login names are used in the *USERFILE* to restrict file access.

File Modes

The programs *uucp*, *uux*, *uucico*, *uulog*, *uuclean*, and *uuxqt* should be owned by the *uucp* administrative login with the "setuid" bit set and only execute permissions (mode 04111). The *L.sys*, *SQFILE*, and the *USERFILE*, which are put in the *program* directory should be owned by the *uucp* administrative login and set with mode 0400. The mode of *spool* should be 0755. The mode of *xqtdir* should be 0777. The *L-dialcodes* and the *L-devices* files should have mode 0444.

D

Sendmail Installation and Operation

D

Sendmail Installation and Operation

`sendmail` implements a general purpose internetwork mail routing facility under the UNIX operating system. It is not tied to any one transport protocol – its function may be likened to a crossbar switch, relaying messages from one domain into another. In the process, it can do a limited amount of message header editing to put the message into a format that is appropriate for the receiving domain. All of this is done under the control of a configuration file.

Due to the requirements of flexibility for `sendmail`, the configuration file can seem somewhat unapproachable. However, there are only a few basic configurations for most sites, for which standard configuration files have been supplied. Most other configurations can be built by adjusting an existing configuration file incrementally.

Although `sendmail` is intended to run without the need for monitoring, it has a number of features that may be used to monitor or adjust the operation under unusual circumstances. These features are described.

Section D.1 describes how to do a basic sendmail installation. Section D.2 explains the day-to-day information you should know to maintain your mail system. If you have a relatively normal site, these two sections should contain sufficient information for you to install `sendmail` and keep it happy. Section D.4 describes some parameters that may be safely tweaked. Section D.3 has information regarding the command line arguments. Section D.5 contains the nitty-gritty information about the configuration file. This section is for masochists and people who must write their own configuration file. Sections D.6 through D.9 give a brief but detailed explanation of a number of features not described in the rest of the paper.

The references in this paper are actually found in the companion paper *Sendmail — An Internetwork Mail Router*. Read that paper before this one to gain a basic understanding of how the pieces fit together.

D.1. Basic Installation

There are two basic steps to installing `sendmail`. The first, and hardest, part is to build the configuration file. The configuration file describes the mailers it knows about, how to parse addresses, how to rewrite the message header, and the settings of various options. When `sendmail` starts up it reads the file. Although the configuration file is quite complex, a configuration can usually be built by adjusting an existing off-the-shelf configuration. The second step is actually doing the installation, which means creating the necessary files, etc.

/etc/rc

It will be necessary to start up the `sendmail` daemon when your system reboots. This daemon performs two functions: it listens on the SMTP socket for connections (to receive mail from a remote system) and it processes the queue periodically to insure that mail gets delivered when hosts come up.

The following lines should be in */etc/rc* (or */etc/rc.local* as appropriate) in the area where it is starting up the daemons:

```
if [ -f /usr/lib/sendmail ]; then
    (cd /usr/spool/mqueue; rm -f [lnx]f*)
    /usr/lib/sendmail -bd -q30m &
    echo -n ' sendmail' >/dev/console
fi
```

The *cd* and *rm* commands insure that all lock files have been removed; extraneous lock files may be left around if the system goes down in the middle of processing a message. The line that actually invokes `sendmail` has two flags: `-bd` causes it to listen on the SMTP port, and `-q30m` causes it to run the queue every half hour.

/usr/lib/sendmail.fc

If you intend to install the frozen version of the configuration file (for quick startup) you should create the file */usr/lib/sendmail.fc* and initialize it. This step may be safely skipped.

```
% cp /dev/null /usr/lib/sendmail.fc
% /usr/lib/sendmail -bz
```

/usr/lib/sendmail.hf

This is the help file used by the SMTP **HELP** command. The file is already installed in the distribution.

/usr/lib/sendmail.st

If you wish to collect statistics about your mail traffic, create the file */usr/lib/sendmail.st*:

```
% cp /dev/null /usr/lib/sendmail.st
% chmod 666 /usr/lib/sendmail.st
```

This file does not grow. It is printed with the program *aux/mailstats*.

/usr/etc/in.syslog

You may want to run the *syslog* program on one host in your network (to collect log information about `sendmail`). This program normally resides in */usr/etc/in.syslog*, with support files */etc/syslog.conf* and */etc/syslog.pid*. The file */etc/syslog.conf* describes the file(s) that `sendmail` will log in. For a complete description of *syslog*, see the manual page for *syslog*(8).

Format of Queue Files

All queue files have the form *xfAA99999* where *AA99999* is the *id* for this file and the *x* is a type. The types are:

d The data file. The message body (excluding the header) is kept in this file.

l The lock file. If this file exists, the job is currently being processed, and a queue run will not process the file. For that reason, an extraneous *lf* file can cause a job to apparently disappear (it will not even time out!).

n This file is created when an id is being created. It is a separate file to insure that no mail can ever be destroyed due to a race condition. It should exist for no more than a few milliseconds at any given time.

q The queue control file. This file contains the information necessary to process the job.

t A temporary file. These are an image of the *qf* file when it is being rebuilt. It should be renamed to a *qf* file very quickly.

x A transcript file, existing during the life of a session showing everything that happens during that session.

The *qf* file is structured as a series of lines each beginning with a code letter. The lines are as follows:

D The name of the data file. There may only be one of these lines.

H A header definition. There may be any number of these lines. The order is important: they represent the order in the final message. These use the same syntax as header definitions in the configuration file.

R A recipient address. This will normally be completely aliased, but is actually realiased when the job is processed. There will be one line for each recipient.

S The sender address. There may only be one of these lines.

T The job creation time. This is used to compute when to time out the job.

P The current message priority. This is used to order the queue. Higher numbers mean lower priorities. The priority increases as the message sits in the queue. The initial priority depends on the message class and the size of the message.

M A message. This line is printed by using `sendmail` with the **–bp** flag, and is generally used to store status information. It can contain any text.

As an example, the following is a queue file sent to "mckusick@calder" and "wnj :"

When the queue is finally emptied, you can remove the directory:

```
% rmdir /usr/spool/omqueue
```

You can also run a subset of the queue at any time with the -R *string* (run queue where any recipient address matches *string*) or with -M *nnnnn* (run just one message, with queue id *nnnnn*).

The Local Alias Database

The local alias database exists in two forms. (See the section below for information about domain-wide aliases with the yellow pages.) One is a text form, maintained in the file */usr/lib/aliases*. The aliases are of the form

```
name: name1, name2, ...
```

Only local names may be aliased; for example,

```
eric@mit-xx: eric@berkeley
```

will not have the desired effect. Aliases may be continued by starting any continuation lines with a space or a tab. Blank lines and lines beginning with a sharp sign (''#'') are comments.

The second form is processed by the *dbm*(3) library. This form is in the files */usr/lib/aliases.dir* and */usr/lib/aliases.pag*. This is the form that `sendmail` actually uses to resolve aliases. This technique is used to improve performance.

Rebuilding the Alias Database

The DBM version of the database may be rebuilt explicitly by executing the command

```
% newaliases
```

This is equivalent to giving `sendmail` the `-bi` flag:

```
% /usr/lib/sendmail -bi
```

If the **D** option is specified in the configuration, `sendmail` will rebuild the alias database automatically if possible when it is out of date. The conditions under which it will do this are:

1. The DBM version of the database is mode 666. --or--
2. `sendmail` is running setuid to root.

Auto-rebuild can be dangerous on heavily loaded machines with large alias files; if it might take more than five minutes to rebuild the database, there is a chance that several processes will start the rebuild process simultaneously.

Potential Problems

There are a number of problems that can occur with the alias database. They all result from a `sendmail` process accessing the DBM version while it is only partially built. This can happen under two circumstances: One process accesses the database while another process is rebuilding it, or the process rebuilding the database dies (due to being killed or a system crash) before completing the rebuild.

`sendmail` has two techniques to try to relieve these problems. First, it ignores interrupts while rebuilding the database; this avoids the problem of someone

domain-wide aliases. Similarly, domain-wide aliases that do not resolve to an address with a host name are forwarded to the domain-name host. For these reasons, it is a good idea to have the domain name be a synonym for a host that handles mail for the domain.

Per-User Forwarding (.forward Files)

As an alternative to the alias database, any user may put a file with the name *.forward* in his or her home directory. If this file exists, `sendmail` redirects mail for that user to the list of addresses listed in the *.forward* file. For example, if the home directory for user "mckusick" has a *.forward* file with contents:

```
mckusick@ernie
kirk@calder
```

then any mail arriving for "mckusick" will be redirected to the specified accounts.

Special Header Lines

Several header lines have special interpretations defined by the configuration file. Others have interpretations built into `sendmail` that cannot be changed without changing the code. These builtins are described here.

Return-Receipt-To:

If this header is sent, a message will be sent to any specified addresses when the final delivery is complete, if the mailer has the **I** flag (local delivery) set in the mailer descriptor.

Errors-To:

If errors occur anywhere during processing, this header will cause error messages to go to the listed addresses rather than to the sender. This is intended for mailing lists.

To:

If a message comes in with no recipients listed in the message (in a To:, Cc:, or Bcc: line) then `sendmail` will add a "To:" header line for each recipient specified on the `sendmail` command line.

At least one recipient line is required under RFC 822.

D.3. Arguments

The complete list of arguments to `sendmail` is described in detail in section D.6. Some important arguments are described here.

Queue Interval

`sendmail`" The amount of time between forking a process to run through the queue is defined by the `-q` flag. If you run in mode **f** or **a** this can be relatively large, since it will only be relevant when a host that was down comes back up. If you run in **q** mode it should be relatively short, since it defines the maximum amount of time that a message may sit in the queue.

Daemon Mode

If you allow incoming mail over an IPC connection, you should have a daemon running. This should be set by your */etc/rc* file using the `-bd` flag. The `-bd` flag and the `-q` flag may be combined in one call:

```
% /usr/lib/sendmail -bd -q30m
```

Timeouts

All time intervals are set using a scaled syntax. For example, "10m" represents ten minutes, whereas "2h30m" represents two and a half hours. The full set of scales is:

```
s   seconds
m   minutes
h   hours
d   days
w   weeks
```

Queue Interval

The argument to the `-q` flag specifies how often a subdaemon will run the queue. This is typically set to between fifteen minutes and one hour.

Read Timeouts

It is possible to time out when reading the standard input or when reading from a remote SMTP server. Technically, this is not acceptable within the published protocols. However, it might be appropriate to set it to something large in certain environments (such as an hour). This will reduce the chance of large numbers of idle daemons piling up on your system. This timeout is set using the **r** option in the configuration file.

Message Timeouts

After sitting in the queue for a few days, a message will time out. This is to insure that at least the sender is aware of the inability to send a message. The timeout is typically set to three days. This timeout is set using the **T** option in the configuration file.

The time of submission is set in the queue, rather than the amount of time left until timeout. As a result, you can flush messages that have been hanging for a short period by running the queue with a short message timeout. For example,

```
% /usr/lib/sendmail -oT1d -q
```

will run the queue and flush anything that is one day old.

Delivery Mode

There are a number of delivery modes that `sendmail` can operate in, set by the **d** configuration option. These modes specify how quickly mail will be delivered. Legal modes are:

```
i   deliver interactively (synchronously)
b   deliver in background (asynchronously)
q   queue only (don't deliver)
```

There are tradeoffs. Mode "i" passes the maximum amount of information to the sender, but is hardly ever necessary. Mode "q" puts the minimum load on your machine, but means that delivery may be delayed for up to the queue interval. Mode "b" is probably a good compromise. However, this mode can cause large numbers of processes if you have a mailer that takes a long time to deliver a message.

16 Verbose information regarding the queue.

File Modes

There are a number of files that may have a number of modes. The modes depend on what functionality you want and the level of security you require.

To Suid or not to Suid?

`sendmail` can safely be made setuid to root. At the point where it is about to *exec* (2) a mailer, it checks to see if the userid is zero; if so, it resets the userid and groupid to a default (set by the **u** and **g** options). (This can be overridden by setting the **S** flag to the mailer for mailers that are trusted and must be called as root.) However, this will cause mail processing to be accounted (using *sa* (8)) to root rather than to the user sending the mail.

Temporary File Modes

The mode of all temporary files that `sendmail` creates is determined by the **F** option. Reasonable values for this option are 0600 and 0644. If the more permissive mode is selected, it will not be necessary to run `sendmail` as root at all (even when running the queue), but will allow users to read mail in the queue.

Should my Alias Database be Writable?

At Sun Microsystems, we provide the alias database (*/usr/lib/aliases**) with mode 666. There are some dangers inherent in this approach: any user can add him-/her-self to any list, or can cause any user's mail to beforwarded elsewhere. However, we have found users to be basically trustworthy, and the cost of having a read-only database greater than the expense of finding and eradicating the rare nasty person.

D.5. The Whole Scoop on the Configuration File

This section describes the configuration file in detail, including hints on how to write one of your own if you have to.

There is one point that should be made clear immediately: the syntax of the configuration file is designed to be reasonably easy to parse, since parsing can be done every time `sendmail` starts up, rather than easy for a human to read or write.

`sendmail` uses single letters for several different functions:

- Command-line flags, see section D.6.
- Configuration options, see section D.7.
- Queue file line types, see the section above, *Format Of Queue Files*.
- Configuration file line types, see the section below, *The Syntax*.
- Mailer field names, see the section below, *M Define Mailer*.
- Mailer flags, see section D.8.
- Macro names, see the section below, *Special Macros, Conditionals*.
- Class names, see the configuration file.

An overview of the configuration file is given first, followed by details of the semantics.

standard output of the command. It is permissible to split them among multiple lines; for example, the two forms:

```
CHmonet ucbmonet
```

and

```
CHmonet
CHucbmonet
```

are equivalent.

M — Define Mailer

Programs and interfaces to mailers are defined in this line. The format is:

M*name* , {*field*=*value* } *

where *name* is the name of the mailer (used internally only) and the "field=name" pairs define attributes of the mailer. Fields are:

```
Path        The pathname of the mailer
Flags       Special flags for this mailer
Sender      A rewriting set for sender addresses
Recipient   A rewriting set for recipient addresses
Argv        An argument vector to pass to this mailer
Eol         The end-of-line string for this mailer
Maxsize     The maximum message length to this mailer
Length      The maximum length of the Argv for this mailer
```

Only the first character of the field name is checked.

H — Define Header

The format of the header lines that `sendmail` inserts into the message are defined by the **H** line. The syntax of this line is:

H [**?***mflags***?**] *hname* **:***htemplate*

Continuation lines in this spec are reflected directly into the outgoing message. The *htemplate* is macro expanded before insertion into the message. If the *mflags* (surrounded by question marks) are specified, at least one of the specified flags must be stated in the mailer definition for this header to be automatically output. If one of these headers is in the input it is reflected to the output regardless of these flags.

Some headers have special semantics that will be described below.

O — Set Option

There are a number of 'random' options that can be set from a configuration file. Options are represented by single characters. The syntax of this line is:

O*ovalue*

This sets option *o* to *value*. Depending on the option, *value* may be a string, an integer, a boolean (with legal values "t," "T," "f," or "F" — the default is TRUE), or a time interval. See D.7 for the list of options.

```
De$j Sendmail $v ready at $b
DnMAILER-DAEMON
DlFrom $g  $d
Do.:%@!^=/
Dq$g$?x ($x)$.
Dj$H.$D
```

An acceptable alternative for the **$q** macro is "$?x$x $.<$g>". These correspond to the following two formats:

```
eric@Berkeley (Eric Allman)
Eric Allman <eric@Berkeley>
```

Some macros are defined by `sendmail` for interpolation into argv's for mailers or for other contexts. These macros are:

a	The origination date in Arpanet format
b	The current date in Arpanet format
c	The hop count
d	The date in UNIX (ctime) format
f	The sender (from) address
g	The sender address relative to the recipient
h	The recipient host
i	The queue id
p	Sendmail's pid
r	Protocol used
s	Sender's host name
t	A numeric representation of the current time
u	The recipient user
v	The version number of sendmail
w	The hostname of this site
x	The full name of the sender
z	The home directory of the recipient

There are three types of dates that can be used. The **$a** and **$b** macros are in Arpanet format; **$a** is the time as extracted from the "Date:" line of the message (if there was one), and **$b** is the current date and time (used for postmarks). If no "Date:" line is found in the incoming message, **$a** is set to the current time also. The **$d** macro is equivalent to the **$a** macro in UNIX (ctime) format.

The **$f** macro is the id of the sender as originally determined; when mailing to a specific host the **$g** macro is set to the address of the sender *relative to the recipient*. For example, if I send to "bollard@matisse" from the machine "ucbarpa" the **$f** macro will be "eric" and the **$g** macro will be "eric@ucbarpa."

The **$x** macro is set to the full name of the sender. This can be determined in several ways. It can be passed as flag to `sendmail`. The second choice is the value of the "Full-name:" line in the header if it exists, and the third choice is the comment field of a "From:" line. If all of these fail, and if the message is being originated locally, the full name is looked up in the */etc/passwd* file.

The Right Hand Side

When the left hand side of a rewriting rule matches, the input is deleted and replaced by the right hand side. Tokens are copied directly from the RHS unless they begin with a dollar sign. Metasymbols are:

$*n*	Substitute indefinite token *n* from LHS
$>*n*	Call ruleset *n*
$#*mailer*	Resolve to *mailer*
$@*host*	Specify *host*
$:*user*	Specify *user*
$[*host*$]	Map to primary hostname

The **$*n*** syntax substitutes the corresponding value from a **$+**, **$–**, **$***, **$=**, or **$~** match on the LHS. It may be used anywhere.

The **$>*n*** syntax causes the remainder of the line to be substituted as usual and then passed as the argument to ruleset *n*. The final value of ruleset *n* then becomes the substitution for this rule.

The **$#** syntax should *only* be used in ruleset zero. It causes evaluation of the ruleset to terminate immediately, and signals to `sendmail` that the address has completely resolved. The complete syntax is:

$#*mailer*$@*host*$:*user*

This specifies the {mailer, host, user} 3-tuple necessary to direct the mailer. If the mailer is local the host part may be omitted. The *mailer* and *host* must be a single word, but the *user* may be multi-part.

A RHS may also be preceeded by a **$@** or a **$:** to control evaluation. A **$@** prefix causes the ruleset to return with the remainder of the RHS as the value. A **$:** prefix causes the rule to terminate immediately, but the ruleset to continue; this can be used to avoid continued application of a rule. The prefix is stripped before continuing.

The **$@** and **$:** prefixes may preceed a **$>** spec; for example:

```
R$+     $:$>7$1
```

matches anything, passes that to ruleset seven, and continues; the **$:** is necessary to avoid an infinite loop.

Semantics of Rewriting Rule Sets

There are five rewriting sets that have specific semantics. These are related as depicted by Figure 1.

Mailer Flags, etc.

There are a number of flags that may be associated with each mailer, each identified by a letter of the alphabet. Many of them are assigned semantics internally. These are detailed in D.8. Any other flags may be used freely to conditionally assign headers to messages destined for particular mailers.

The "error" Mailer

The mailer with the special name "error" can be used to generate a user error. The (optional) host field is a numeric exit status to be returned, and the user field is a message to be printed. For example, the entry:

```
$#error$:Host unknown in this domain
```

on the RHS of a rule will cause the specified error to be generated if the LHS matches. This mailer is only functional in ruleset zero.

Semantics of Mailer Descriptions

Each mailer has an internal name. This can be arbitrary, except that the names "local" and "prog" must be defined. Ruleset zero will resolve addresses to this mailer name (and a host and user name).

The pathname of the mailer must be given in the P field. If this mailer should be accessed via a TCP connection, use the string "[IPC]" instead.

The F field defines the mailer flags. You should specify an "f" or "r" flag to pass the name of the sender as a `-f` or `-r` flag respectively. These flags are only passed if they were passed to `sendmail`, so that mailers that give errors under some circumstances can be placated. If the mailer is not picky you can just specify "–f $g" in the argv template. If the mailer must be called as **root** the "S" flag should be given; this will not reset the userid before calling the mailer.[9] If this mailer is local (that is, will perform final delivery rather than another network hop) the "l" flag should be given. Quote characters (backslashes and " marks) can be stripped from addresses if the "s" flag is specified; if this is not given they are passed through. If the mailer is capable of sending to more than one user on the same host in a single transaction the "m" flag should be stated. If this flag is on, then the argv template containing **$u** will be repeated for each unique user on a given host. The "e" flag will mark the mailer as being 'expensive,' which will cause `sendmail` to defer connection until a queue run.[10]

An unusual case is the "C" flag. This flag applies to the mailer that the message is received from, rather than the mailer being sent to; if set, the domain spec of the sender (that is, the "@host.domain" part) is saved and is appended to any addresses in the message that do not already contain a domain spec. For example, a message of the form:

```
From: eric@ucbarpa
To: wnj@monet, mckusick
```

will be modified to:

[9] `sendmail` must be running setuid to root for this to work.

[10] The c configuration option must be given for this to be effective.

message and ruleset twenty should be applied to recipient addresses; the argv to send to a message will be the word "mail," the word "–d," and words containing the name of the receiving user. If a `-r` flag is inserted it will be between the words "mail" and "–d." The second mailer is called "ether," it should be connected to via an IPC connection, it can handle multiple users at once, connections should be deferred, and any domain from the sender address should be appended to any receiver name without a domain; sender addresses should be processed by ruleset eleven and recipient addresses by ruleset twenty-one. There is a 100,000 byte limit on messages passed through this mailer.

Building a Configuration File from Scratch

Building a configuration table from scratch is an extremely difficult job. Fortunately, it is almost never necessary to do so; nearly every situation that may come up may be resolved by changing an existing table. In any case, it is critical that you understand what it is that you are trying to do and come up with a philosophy for the configuration table. This section is intended to explain what the real purpose of a configuration table is and to give you some ideas for what your philosophy might be.

What you are Trying to do

The configuration table has three major purposes. The first and simplest is to set up the environment for `sendmail`. This involves setting the options, defining a few critical macros, etc. Since these are described in other places, we will not go into more detail here.

The second purpose is to rewrite addresses in the message. This should typically be done in two phases. The first phase maps addresses in any format into a canonical form. This should be done in ruleset three. The second phase maps this canonical form into the syntax appropriate for the receiving mailer. `sendmail` does this in three subphases. Rulesets one and two are applied to all sender and recipient addresses respectively. After this, you may specify per-mailer rulesets for both sender and recipient addresses; this allows mailer-specific customization. Finally, ruleset four is applied to do any default conversion to external form.

The third purpose is to map addresses into the actual set of instructions necessary to get the message delivered. Ruleset zero must resolve to the internal form, which is in turn used as a pointer to a mailer descriptor. The mailer descriptor describes the interface requirements of the mailer.

Philosophy

The particular philosophy you choose will depend heavily on the size and structure of your organization. I will present a few possible philosophies here.

One general point applies to all of these philosophies: it is almost always a mistake to try to do full name resolution. For example, if you are trying to get names of the form "user@host" to the Arpanet, it does not pay to route them to "sun!snail!athena!c70:user@host" since you then depend on several links not under your control. The best approach to this problem is to simply forward to "sun!user@host" and let sun worry about it from there. In summary, just get the message closer to the destination, rather than determining the full path.

Relevant Issues

The canonical form you use should almost certainly be as specified in the Arpanet protocols RFC819 and RFC822.

RFC822 describes the format of the mail message itself. `sendmail` follows this RFC closely, to the extent that many of the standards described in this document can not be changed without changing the code. In particular, the following characters have special interpretations:

```
< > ( ) " \
```

Any attempt to use these characters for other than their RFC822 purpose in addresses is probably doomed to disaster.

RFC819 describes the specifics of the domain-based addressing. This is touched on in RFC822 as well. Essentially each host is given a name which is a right-to-left dot qualified pseudo-path from a distinguished root. The elements of the path need not be physical hosts; the domain is logical rather than physical. For example, at Sun one legal host is "a.b1.silicon.arpa". Reading from right to left, "arpa" is a top level domain (related to, but not limited to, the physical Arpanet), "silicon" is both an Arpanet host and a logical domain which is actually interpreted by a host called snail (the "major" host for this domain), "b1" represents the Building One, (in this case a strictly logical entity), and "a" is a host in Building One; this particular host happens to be connected via ethernet, but other hosts might be connected via some other network.

Be aware when reading RFC819 that there are a number of errors in it.

How to Proceed

Once you have decided on a philosophy, it is worth examining the available configuration tables to decide if any of them are close enough to steal major parts of. Even under the worst of conditions, there is a fair amount of boiler plate that can be collected safely.

The next step is to build ruleset three. This will be the hardest part of the job. Beware of doing too much to the address in this ruleset, since anything you do will reflect through to the message. In particular, stripping of local domains is best deferred, since this can leave you with addresses with no domain spec at all. Since `sendmail` likes to append the sending domain to addresses with no domain, this can change the semantics of addresses. Also try to avoid fully qualifying domains in this ruleset. Although technically legal, this can lead to unpleasantly and unnecessarily long addresses reflected into messages. The Sun configuration files define ruleset nine to qualify domain names and strip local domains. This is called from ruleset zero to get all addresses into a cleaner form.

Once you have ruleset three finished, the other rulesets should be relatively trivial. If you need hints, examine the supplied configuration tables.

Testing the Rewriting Rules — the –bt Flag

When you build a configuration table, you can do a certain amount of testing using the "test mode" of `sendmail`. For example, you could invoke `sendmail` as:

```
sendmail -bt -Ctest.cf
```

which would read the configuration file *test.cf* and enter test mode. In this mode,

D.6. Command Line Flags

Arguments must be presented with flags before addresses. The flags are:

–f *addr* The sender's machine address is *addr*. This flag is ignored unless the real user is listed as a "trusted user" or if *addr* contains an exclamation point (because of certain restrictions in UUCP).

–r *addr* An obsolete form of `-f`.

–h *cnt* Sets the "hop count" to *cnt*. This represents the number of times this message has been processed by `sendmail` (to the extent that it is supported by the underlying networks). *Cnt* is incremented during processing, and if it reaches MAXHOP (currently 30) `sendmail` throws away the message with an error.

–F*name* Sets the full name of this user to *name*.

–n Don't do aliasing or forwarding.

–t Read the header for "To:," "Cc:," and "Bcc:" lines, and send to everyone listed in those lists. The "Bcc:" line will be deleted before sending. Any addresses in the argument vector will be deleted from the send list.

–b*x* Set operation mode to *x*. Operation modes are:

```
m  Deliver mail (default)
a  Run in arpanet mode (see below)
s  Speak SMTP on input side
d  Run as a daemon
t  Run in test mode
v  Just verify addresses, don't collect or deliv
i  Initialize the alias database
p  Print the mail queue
z  Freeze the configuration file
```

The special processing for the ARPANET includes reading the "From:" line from the header to find the sender, printing ARPANET style messages (preceded by three digit reply codes for compatibility with the FTP protocol [Neigus73, Postel74, Postel77]), and ending lines of error messages with <CRLF>.

–q*time* Try to process the queued up mail. If the time is given, a `sendmail` will run through the queue at the specified interval to deliver queued mail; otherwise, it only runs once.

–C*file* Use a different configuration file.

–d*level* Set debugging level.

–o*xvalue* Set option *x* to the specified *value*. These options are described in D.7.

There are a number of options that may be specified as primitive flags (provided for compatibility with *delivermail*). These are the e, i, m, and v options. Also,

D.7. Configuration Options

The following options may be set using the `-o` flag on the command line or the **O** line in the configuration file:

A*file* — Use the named *file* as the alias file instead of */usr/lib/aliases*. If no file is specified, use *aliases* in the current directory.

a*time* — If set, time to wait for an "@:@" entry to exist in the alias database before starting up. If it does not appear after that time, rebuild the database.

B*value* — Blank substitute. Default is "."

C*n* — Checkpoint after *n* recipients.

c — If an outgoing mailer is marked as being expensive, don't connect immediately. This requires that queueing be compiled in, since it will depend on a queue run process to actually send the mail.

d*x* — Deliver in mode *x*. Legal modes are:

```
i   Deliver interactively (synchronously)
b   Deliver in background (asynchronously)
q   Just queue the message (deliver during queue run)
```

D — If set, rebuild the alias database if necessary and possible. If this option is not set, `sendmail` will never rebuild the alias database unless explicitly requested using `-bi`.

e*x* — Dispose of errors using mode *x*. The values for *x* are:

```
p   Print error messages (default)
q   No messages, just give exit status
m   Mail back errors
w   Write back errors (mail if user not logged in)
e   Mail back errors and give zero exit stat always
```

F*n* — The temporary queue file mode, in octal. 644 and 600 are good choices.

f — Save UNIX-style "From" lines at the front of headers. Normally they are assumed redundant and discarded.

g*n* — Set the default group id for mailers to run in to *n*.

H*file* — Specify the help file for SMTP.

i — Ignore dots in incoming messages.

L*n* — Set the default log level to *n*.

M*xvalue* — Set the macro *x* to *value*. This is intended only for use from the command line.

m — Send to me too, even if I am in an alias expansion.

o — Assume that the headers may be in old format, that is, spaces delimit names. This actually turns on an adaptive algorithm: if any recipient address contains a comma, parenthesis, or angle bracket, it will be

D.8. Mailer Flags

The following flags may be set in the mailer description.

C If mail is *received* from a mailer with this flag set, any addresses in the header that do not have an at sign ("@") after being rewritten by ruleset three will have the "@domain" clause from the sender tacked on. This allows mail with headers of the form:

```
From: usera@hosta
To: userb@hostb, userc
```

to be rewritten as:

```
From: usera@hosta
To: userb@hostb, userc@hosta
```

automatically.

D This mailer wants a "Date:" header line.

E Escape "From" lines to be ">From" (usually specified with `U`.

e This mailer is expensive to connect to, so try to avoid connecting normally; any necessary connection will occur during a queue run.

F This mailer wants a "From:" header line.

f The mailer wants a `-f` *from* flag, but only if this is a network forward operation (that is, the mailer will give an error if the executing user does not have special permissions).

h Upper case should be preserved in host names for this mailer.

L Limit the line lengths as specified in RFC821.

l This mailer is local (that is, final delivery will be performed).

M This mailer wants a "Message-Id:" header line.

m This mailer can send to multiple users on the same host in one transaction. When a **$u** macro occurs in the *argv* part of the mailer definition, that field will be repeated as necessary for all qualifying users. The L= field of the mailer description can be used to limit the total length of the **$u** expansion.

n Do not insert a UNIX-style "From" line on the front of the message.

P This mailer wants a "Return-Path:" line.

p Always add local host name to the "MAIL From:" line of *SMTP*, even if there already is one.

r Same as f, but sends a `-r` flag.

S Don't reset the userid before calling the mailer. This would be used in a secure environment where `sendmail` ran as root. This could be used to avoid forged addresses.

s Strip quote characters off of the address before calling the mailer.

U This mailer wants UNIX-style "From" lines with the ugly UUCP-style "remote from <host>" on the end.

D.9. Summary of Support Files

This is a summary of the support files that `sendmail` creates, uses, or generates.

/usr/lib/sendmail	The binary of `sendmail`.
/usr/ucb/newaliases	A link to /usr/lib/sendmail; causes the alias database to be rebuilt. Running this program is completely equivalent to giving `sendmail` the `-bi` flag.
/usr/lib/sendmail.cf	The configuration file, in textual form.
/usr/lib/sendmail.fc	The frozen configuration file.
/usr/lib/sendmail.hf	The SMTP help file.
/usr/lib/aliases	The textual version of the alias file.
/usr/lib/aliases.{pag,dir}	The alias file in *dbm* (3) format.
/usr/etc/in.syslog	The program to do logging.
/etc/syslog.conf	The configuration file for syslog.
/etc/syslog.pid	Contains the process id of the currently running syslog.
/usr/spool/mqueue	The directory in which the mail queue and temporary files reside.
/usr/spool/mqueue/qf*	Control (queue) files for messages.
/usr/spool/mqueue/df*	Data files.
/usr/spool/mqueue/lf*	Lock files
/usr/spool/mqueue/tf*	Temporary versions of the qf files, used during queue file rebuild.
/usr/spool/mqueue/nf*	A file used when creating a unique id.
/usr/spool/mqueue/xf*	A transcript of the current session.

E

sendmail — Internetwork Mail Router

E

sendmail — Internetwork Mail Router

Routing mail through a heterogenous internet presents many new problems. Among the worst of these is that of address mapping. Historically, this has been handled on an *ad hoc* basis. However, this approach has become unmanageable as internets grow.

`sendmail` acts a unified "post office" to which all mail can be submitted. Address interpretation is controlled by a production system, which can parse both domain-based addressing and old-style *ad hoc* addresses. The production system is powerful enough to rewrite addresses in the message header to conform to the standards of a number of common target networks, (TCP/RFC822) Arpa Internet, UUCP, and Phonenet. Sendmail also implements an SMTP server, message queueing, and aliasing.

`sendmail` implements a general internetwork mail routing facility, featuring aliasing and forwarding, automatic routing to network gateways, and flexible configuration.

In a simple network, each node has an address, and resources can be identified with a host-resource pair; in particular, the mail system can refer to users using a host-username pair. Host names and numbers have to be administered by a central authority, but usernames can be assigned locally to each host.

In an internet, multiple networks with different characterstics and managements must communicate. In particular, the syntax and semantics of resource identification change. Certain special cases can be handled trivially by *ad hoc* techniques, such as providing network names that appear local to hosts on other networks, as with the Ethernet at Xerox PARC. However, the general case is extremely complex. For example, some networks require point-to-point routing, which simplifies the database update problem since only adjacent hosts must be entered into the system tables, while others use end-to-end addressing. Some networks use a left-associative syntax and others use a right-associative syntax, causing ambiguity in mixed addresses.

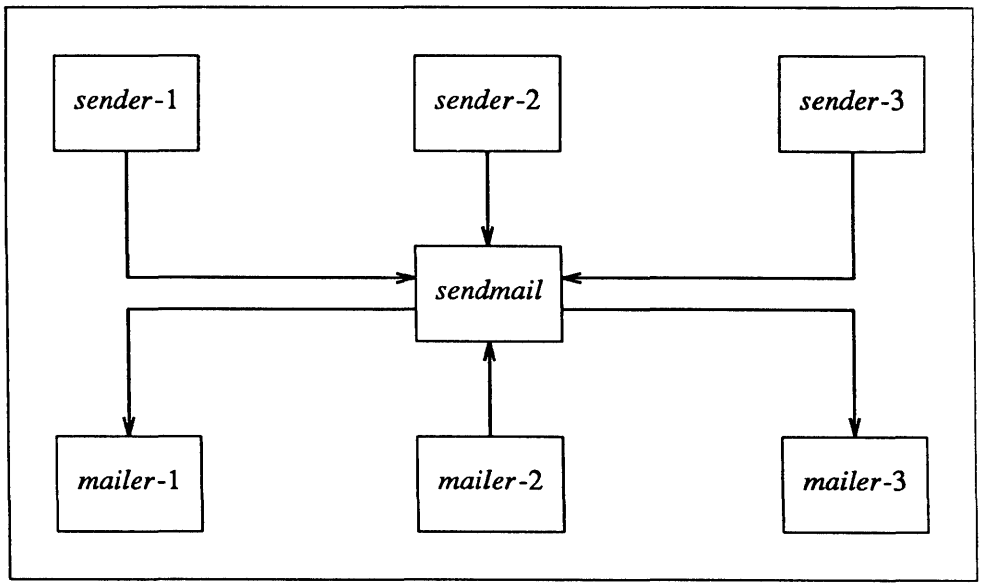
Internet standards seek to eliminate these problems. Initially, the standards proposed expanding the address pairs to address triples, consisting of {network, host, resource} triples. Network numbers must be universally agreed upon, and hosts can be assigned locally on each network. The user-level presentation was quickly expanded to address domains, comprised of a local resource identification and a hierarchical domain specification with a common static root.

the CPU type or the operating system). We have found this seemingly unimportant goal to be critical in real life. Besides the simple problems that occur when any program gets recompiled in a different environment, many sites like to "fiddle" with anything that they will be recompiling anyway.

6. `sendmail` must be able to let various groups maintain their own mailing lists, and let individuals specify their own forwarding, without modifying the system alias file.
7. Each user should be able to specify which mailer to execute to process mail being delivered for him. This feature allows users who are using specialized mailers that use a different format to build their environment without changing the system, and facilitates specialized functions (such as returning an "I am on vacation" message).
8. Network traffic should be minimized by batching addresses to a single host where possible, without assistance from the user.

These goals motivated the architecture illustrated in Figure 1.

Figure E-1 *Sendmail System Structure*



The user interacts with a mail generating and sending program. When the mail is created, the generator calls `sendmail`, which routes the message to the correct mailer(s). Since some of the senders may be network servers and some of the mailers may be network clients, `sendmail` may be used as an internet mail gateway.

done at this step: syntax is checked, and local addresses are verified, but detailed checking of host names and addresses is deferred until delivery. Forwarding is also performed as the local addresses are verified.

`sendmail` appends each address to the recipient list after parsing. When a name is aliased or forwarded, the old name is retained in the list, and a flag is set that tells the delivery phase to ignore this recipient. This list is kept free from duplicates, preventing alias loops and duplicate messages deliverd to the same recipient, as might occur if a person is in two groups.

Message Collection

`sendmail` then collects the message. The message should have a header at the beginning. No formatting requirements are imposed on the message except that they must be lines of text (in other words, binary data is not allowed). The header is parsed and stored in memory, and the body of the message is saved in a temporary file.

To simplify the program interface, the message is collected even if no addresses were valid. The message will be returned with an error.

Message Delivery

For each unique mailer and host in the recipient list, `sendmail` calls the appropriate mailer. Each mailer invocation sends to all users receiving the message on one host. Mailers that only accept one recipient at a time are handled properly.

The message is sent to the mailer using one of the same three interfaces used to submit a message to sendmail. Each copy of the message is prepended by a customized header. The mailer status code is caught and checked, and a suitable error message given as appropriate. The exit code must conform to a system standard or a generic message (''Service unavailable'') is given.

Queueing for Retransmission

If the mailer returned an status that indicated that it might be able to handle the mail later, `sendmail` will queue the mail and try again later.

Return to Sender

If errors occur during processing, `sendmail` returns the message to the sender for retransmission. The letter can be mailed back or written in the *dead.letter* file in the sender's home directory[12].

Message Header Editing

Certain editing of the message header occurs automatically. Header lines can be inserted under control of the configuration file. Some lines can be merged; for example, a ''From:'' line and a ''Full-name:'' line can be merged under certain circumstances.

[12] Obviously, if the site giving the error is not the originating site, the only reasonable option is to mail back to the sender. Also, there are many more error disposition options, but they only effect the error message — the ''return to sender'' function is always handled in one of these two ways.

Aliasing, Forwarding, Inclusion

`sendmail` reroutes mail three ways. Aliasing applies system wide. Forwarding allows each user to reroute incoming mail destined for that account. Inclusion directs `sendmail` to read a file for a list of addresses, and is normally used in conjunction with aliasing.

Aliasing

Aliasing maps names to address lists using a system-wide file. This file is indexed to speed access. Only names that parse as local are allowed as aliases; this guarantees a unique key (since there are no nicknames for the local host).

Forwarding

After aliasing, recipients that are local and valid are checked for the existence of a ".forward" file in their home directory. If it exists, the message is *not* sent to that user, but rather to the list of users in that file. Often this list will contain only one address, and the feature will be used for network mail forwarding.

Forwarding also permits a user to specify a private incoming mailer. For example, forwarding to:

```
"|/usr/local/newmail myname"
```

will use a different incoming mailer.

Inclusion

Inclusion is specified in RFC 733 [Crocker77a] syntax:

```
:Include: pathname
```

An address of this form reads the file specified by *pathname* and sends to all users listed in that file.

The intent is *not* to support direct use of this feature, but rather to use this as a subset of aliasing. For example, an alias of the form:

```
project: :include:/usr/project/userlist
```

is a method of letting a project maintain a mailing list without interaction with the system administration, even if the alias file is protected.

It is not necessary to rebuild the index on the alias database when a :include: list is changed.

Message Collection

Once all recipient addresses are parsed and verified, the message is collected. The message comes in two parts: a message header and a message body, separated by a blank line.

The header is formatted as a series of lines of the form

```
field-name: field-value
```

Field-value can be split across lines by starting the following lines with a space or a tab. Some header fields have special internal meaning, and have appropriate special processing. Other headers are simply passed through. Some header fields may be added automatically, such as time stamps.

The body is a series of text lines. It is completely uninterpreted and untouched, except that lines beginning with a dot have the dot doubled when transmitted over an SMTP channel. This extra dot is stripped by the receiver.

Header Declarations

Header declarations inform `sendmail` of the format of known header lines. Knowledge of a few header lines is built into `sendmail`, such as the "From:" and "Date:" lines.

Most configured headers will be automatically inserted in the outgoing message if they don't exist in the incoming message. Certain headers are suppressed by some mailers.

Mailer Declarations

Mailer declarations tell `sendmail` of the various mailers available to it. The definition specifies the internal name of the mailer, the pathname of the program to call, some flags associated with the mailer, and an argument vector to be used on the call; this vector is macro-expanded before use.

Address Rewriting Rules

The heart of address parsing in `sendmail` is a set of rewriting rules. These are an ordered list of pattern-replacement rules, (somewhat like a production system, except that order is critical), which are applied to each address. The address is rewritten textually until it is either rewritten into a special canonical form — for example, a (mailer, host, user) 3-tuple, such as {arpanet, usc-isif, postel} representing the address "postel@usc-isif" — or it falls off the end. When a pattern matches, the rule is reapplied until it fails.

The configuration file also supports the editing of addresses into different formats. For example, an address of the form:

```
ucsfcgl!tef
```

might be mapped into:

```
tef@ucsfcgl.UUCP
```

to conform to the domain syntax. Translations can also be done in the other direction.

Option Setting

There are several options that can be set from the configuration file. These include the pathnames of various support files, timeouts, default modes, etc.

E.4. Evaluations and Future Plans

`sendmail` is designed to work in a nonhomogeneous environment. Every attempt is made to avoid imposing unnecessary constraints on the underlying mailers. This goal has driven much of the design. One of the major problems has been the lack of a uniform address space, as postulated in [Postel79a] and [Postel79b].

A nonuniform address space implies that a path will be specified in all addresses, either explicitly (as part of the address) or implicitly (as with implied forwarding to gateways). This restriction has the unpleasant effect of making replying to messages exceedingly difficult, since there is no one "address" for any person, but only a way to get there from wherever you are.

Interfacing to mail programs that were not initially intended to be applied in an internet environment has been amazingly successful, and has reduced the job to a manageable task.

References

[Birrell82] Birrell, A. D., Levin, R., Needham, R. M., and Schroeder, M. D., "Grapevine: An Exercise in Distributed Computing." In *Comm. A.C.M. 25*, 4, April 82.

[Borden79] Borden, S., Gaines, R. S., and Shapiro, N. Z., *The MH Message Handling System: Users' Manual.* R-2367-PAF. Rand Corporation. October 1979.

[Crocker77a] Crocker, D. H., Vittal, J. J., Pogran, K. T., and Henderson, D. A. Jr., *Standard for the Format of ARPA Network Text Messages.* RFC 733, NIC 41952. In [Feinler78]. November 1977.

[Crocker77b] Crocker, D. H., *Framework and Functions of the MS Personal Message System.* R-2134-ARPA, Rand Corporation, Santa Monica, California. 1977.

[Crocker79] Crocker, D. H., Szurkowski, E. S., and Farber, D. J., *An Internetwork Memo Distribution Facility — MMDF.* 6th Data Communication Symposium, Asilomar. November 1979.

[Crocker82] Crocker, D. H., *Standard for the Format of Arpa Internet Text Messages.* RFC 822. Network Information Center, SRI International, Menlo Park, California. August 1982.

[Metcalfe76] Metcalfe, R., and Boggs, D., "Ethernet: Distributed Packet Switching for Local Computer Networks ," *Communications of the ACM 19*, 7. July 1976.

[Feinler78] Feinler, E., and Postel, J. (eds.), *ARPANET Protocol Handbook.* NIC 7104, Network Information Center, SRI International, Menlo Park, California. 1978.

[NBS80] National Bureau of Standards, *Specification of a Draft Message Format Standard.* Report No. ICST/CBOS 80-2. October 1980.

[Neigus73] Neigus, N., *File Transfer Protocol for the ARPA Network.* RFC 542, NIC 17759. In [Feinler78]. August, 1973.

[Nowitz78a] Nowitz, D. A., and Lesk, M. E., *A Dial-Up Network of* UNIX Systems. Bell Laboratories. In UNIX Programmer's Manual,

Index

N

O

P

Q

R

Revision History

Revision	Date	Comments
1	19 November 1984	First α release of this System Administration Manual.
2	14 February 1985	Release α.1 incorporating review comments, but not reflecting effects of NFS.
3	16 March 1985	Release α.2 incorporating NFS and *yp* information.
50	29 March 1985	First β release of this System Administration Manual.
A	15 April 1985	First customer release of this System Administration Manual.
4	15 August 1985	Second α release of this System Administration Manual, for 3.0.
51	15 October 1985	Second β release of this System Administration Manual, for 3.0.
B	15 February 1986	Second customer release of this System Administration Manual, for 3.0.

Notes

Notes

Notes