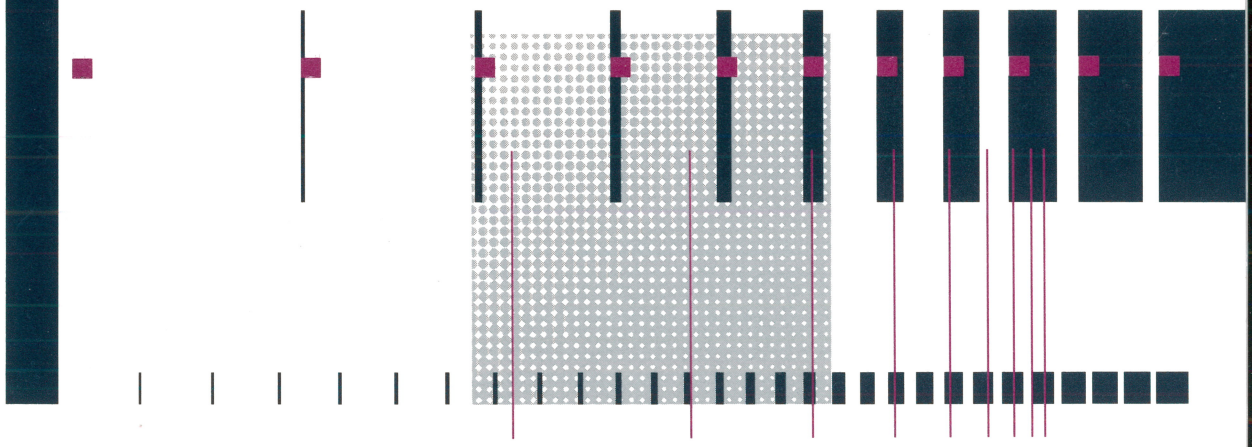
RISC/os (UMIPS)
System Administrator's Guide
Volume I

Order Number 3206DOC



mips

The power of RISC is in the system.

RISC/os (UMIPS)
System Administrator's Guide
Volume I

Order Number 3206DOC

April 1989

Your comments on our products and publications are welcome. A postage–paid form is provided for this purpose on the last page of this manual.

Mfg. Part Number 84–00039–B/02–00136

MIPS Computer Systems, Inc.
930 Arques Ave.
Sunnyvale, CA 94086

Customer Service Telephone Numbers:

California:	(800)	992–MIPS
All other states:	(800)	443–MIPS
International:	(415)	330–7966

Mfg. Part Number 84–00039–B/02–00136

Table of Contents

Chapter 5: File System Administration

Chapter 14: Timed Installation and Operations

Chapter 15: Name Server Operations for BIND

Chapter 16: Sendmail - Mail Router

Part 2: Procedures

Procedure 4: Disk Management

Procedure 5: File System Administration

Procedure 6: System Reconfiguration

Procedure 7: LP Spooler Administration

Procedure 8: TTY Management

Procedure 9: UUCP Networking

Procedure 10: TCP/IP Network

Procedure 11: NFS Network

Introduction

The UMIPS System Administrator's Guide describes the administration of the MIPS RISComputers running the RISC/os (UMIPS) operating system. It is designed to accomplish the following objectives:

- provide clear instructions on how to perform the administrative tasks of a UMIPS system
- give background information about when and why these tasks are desirable
- serve as a quick reference to administrative procedures

What Is a System Administrator?

A System Administrator performs two main tasks:

- decides what rules are needed to govern the use of the computer system
- implements those rules so as to provide the maximum amount of computing service for the system's users, consistent with the physical limitations of the machine

This manual is designed for administrators of systems ranging from a single user systems to multi-user systems networked to other systems. If you are the only user of your system, administration consists simply of those things you do to keep the machine running and your programs and data from disappearing permanently. For larger systems, administration becomes increasingly complex, as you need to anticipate the needs of multiple users and a wide variety of applications. Regardless of your situation, the appropriate information to begin system administration is covered in this guide. As you become more familiar with the system, this guide may still serve as a useful reference along with the System Administrator's Reference Manual.

How to Use This Guide

This manual is organized as follows:

Chapters 1–11	Reference material
Procedures 1–11	"Cookbook" style examples
Back Matter	Appendices, glossary and index

If you are an experienced UNIX System Administrator, you may prefer to simply consult the Procedure before performing a particular activity for the first time. (The procedures are written for less experienced administrators but they may illustrate implementation dependencies that you are not aware of.) If you are not familiar with administering a UNIX system, it is recommended that you read the appropriate chapter before performing any particular activity.

This book assumes you are familiar with the UNIX operating system at least as a user. You should know how to create files with an editor, how to move and remove files, how to make and remove directories, change directories, and login and logout.

If you are not familiar with the UNIX operating system, it is highly recommended that you become familiar with it before assuming the duties of system administrator. System administrator (also known as "root" and "superuser") privileges allow powerful manipulations of this multiuser system. It is possible, for example, to remove all of a user's files with a single command, or to mistakenly alter system files that will then keep the system from rebooting correctly. Many books are currently available that provide a good introduction to UNIX - start with them and use the system as a user as much as possible before becoming system administrator. You'll be glad you did.

A few notes on the use of this manual. Since it deals with system administration, many crucial system files are manipulated, such as **/etc/passwd**, **/etc/inittab**, **/etc/fstab**, **/etc/hosts**, and so on. It is highly recommended that you make a backup copy of those files before editing them in case anything happens. This warning will not be repeated every time you are told to edit one of these files. Also, it is advisable to operate as system administrator only when it is required. For that reason, the convention of a "#" prompt is used to represent a superuser login in the examples. Standard user logins are represented by the "$" or the "%" prompt. Login as a standard user whenever the operation permits - your chances of making catastrophic errors are greatly reduced.

Notation Conventions

Whenever the text includes examples of output from the computer and/or commands entered by you, we follow the standard notation scheme that is common throughout UNIX system documentation:

- Text that you type in from your terminal is shown in **bold** type.
- Computer response is shown in `constant width` type. In cases where the line on the computer screen is longer than can be shown in this document, the backslant ("\") is used to indicate that the next line is actually still a part of the current line.
- Comments and explanations within a display are indented and shown in *italic* type.

  Italics are also used to show substitutable values, such as *file*, when the format of a command is shown.
- There is an implied **RETURN** at the end of each command and menu response you enter. Where you may be expected to enter only a **RETURN** (as in the case where you are accepting a menu default), the symbol **<CR>** is used.

Command References

When commands are mentioned in a section of the text for the first time, a reference to the manual section where the command is described is included in parentheses: **command**(section). The numbered sections are located in the following manuals:

Section (1)	*User's Reference Manual*
Sections (1M), (7)	*System Administrator's Reference Manual*
Sections (2), (3), (4), (5)	*Programmer's Reference Manual*
Section (1SPP)	*System's Programmers Package Reference Manual*

Information in the Examples

While every effort has been made to present displays of information just as they appear on your terminal, it is possible that your system will produce slightly different output. Some displays reflect a particular machine configuration that may differ from yours. Changes between releases of the UMIPS system software may cause small differences in what appears on your terminal.

Introduction

This is Part 1, the "Chapters" section of this document. It contains reference material useful to the RISC/os system administrator and is divided into the following chapters:

Besides the reference material provided in these chapters, additional information on individual system administrator commands can be found in the *System Administrator's Reference Manual*, or by entering:

$ **man** *command_name*

to get the online manual page. In addition, a number of books on the market today provide useful tips on UNIX system administration.

While Part 2 of this book, the Procedures, provides many useful examples of particular system administration actions, it is a good idea to become familiar with the general information provided in the corresponding chapters before performing them.

Chapter 1: System Identification and Security

Introduction

This chapter deals with setting up a secure M-Series system.

- Important security guidelines

  Guidelines for setting passwords and permissions; protecting the system from unauthorized access.

- Logins and passwords

  Aging passwords to control the amount of time passwords may be kept for logins.

  Locking logins to prevent their being used.

  Protecting administrative commands and logins with passwords.

- Set-UID and Set-GID

  Preventing unauthorized use of programs conditioned to execute via an administrative login.

Important Security Guidelines

The security of the system is ultimately the responsibility of all who have access to the system. No system is totally secure. The system is not tamperproof. Some of the items to consider are as follows.

- Especially with a small computer, physical access to the machine must be considered. Anyone with physical access to the machine can literally walk off with it.
- Use the keyswitch lock (on systems that have it). After bringing up the system, lock the keyswitch and store the key in a safe place. Make sure those who need it know where it is.
- Set the access permissions to directories and files to allow only the necessary permissions for owner, group, and others.
- All logins should have passwords. Change passwords regularly. Do not pick obvious passwords. Six-to-eight character nonsense strings using letters, numbers and mixed case are recommended over standard names. Logins that are not needed should be either removed or blocked.
- Dial-up ports that do not have passwords are a security risk.
- Any system with dial-up ports is not really secure. Top secret information should not be kept on a system with dial-up ports.
- Users who make frequent use of the **su** command can compromise the security of your system. Superusers and users who know other users' passwords are able to manipulate the other user's files. The more people who know a given password, the less secure is the system.
- Login directories, **.profile**, **.login** and **.cshrc** files, and files in **/bin**, **/usr/bin**, and **/etc** that are writable by others are security give-aways, and an unknowing or malevolent user can cause problems by editing or removing files in these areas.
- Encrypt sensitive data files. The **crypt**(1) command provides some protection for sensitive information.
- Log off the system if you must be away from the data terminal. Do not leave a logged-in terminal unattended, especially if you are logged in as **root**.

Logins and Passwords

The discussion of logins and passwords covers

- password aging
- sample **/etc/passwd** entries
- locking unused logins
- special administrative logins

Password Aging

The password aging mechanism forces users to change their password on a regular basis. Provisions are also made to prevent a user from changing a new password before a specified time interval. Password aging is selectively applied to logins by editing the **/etc/passwd** file.

The password aging information is appended to the encrypted password field in the **/etc/passwd** file and consists of a comma and up to four bytes (characters) in the format:

,Mmww

The meaning of these fields is as follows:

, The delimiter between the password itself and the aging information.

M The Maximum duration of the password (in weeks, following the notation in Figure 1-1).

m The minimum time interval before the existing password can be changed by the user (in weeks, following the notation in Figure 1-1).

ww The week (counted from the beginning of 1970) when the password was last changed; two characters, *ww*, are used. You do not enter this information. The system automatically adds these characters to the password aging information.

All times are specified in weeks (0 through 63) by a 64 character alphabet. Figure 1-1 shows the relationship between the numerical values and character codes. Any of the character codes may be used in the four fields of the password aging information.

Character	Number of Weeks
. (period)	0 (zero)
/ (slash)	1
0 through 9	2 through 11
A through Z	12 through 37
a through z	38 through 63

Figure 1-1: Password Aging Character Codes

Two special cases apply for the character codes:

- If *M* and *m* are equal to zero (the code, **..**), the user is forced to change the password at the next login. No further password aging is then applied to that login.
- If *m* is greater than *M* (for example, the code, **./**), only **root** is able to change the password for that login.

Sample /etc/passwd Entries

Password administration can be set up in a variety of ways to meet the needs of different organizations. Some examples are discussed in the following sections.

Change Every Two Weeks

The following shows the password aging information required to establish a new password every 2 weeks (0) and to deny changing the new password for 1 week (/).

1. Here is a typical login/password entry in the **/etc/passwd** file for the typical user **jqu**:

   ```
   jqu:RTKESmMOE2m.E:100:1:J. Q. Username:/usr/jqu:
   ```

2. To cause **jqu** to change the password at least every 2 weeks, but keep it at least for 1 week, you should use the code **0/**. After you edit the **/etc/passwd** file, adding **,0/** to the password field, the entry looks like this:

   ```
   jqu:RTKESmMOE2m.E,0/:100:1:J. Q. Username:/usr/jqu:
   ```

   After the password entry is changed, **jqu** will have to change the password at the next login and every 2 weeks thereafter.

3. After **jqu**'s first login following the change, the system automatically adds the two-character, "last-time-changed" information to the password field.

   ```
   jqu:nCNYv9zhLTojo,0/W9:100:1:J. Q. Username:/usr/jqu:
   ```

   In this example, **jqu** changed the password in week **W9**.

Change One Time Only

The following shows the password aging information required to establish for one time only a new password when the user next logs in (using the **..** code).

1. Here is the typical login/password entry for user **jqu** again:

   ```
   jqu:RTKESmMOE2m.E:100:1:J. Q. Username:/usr/jqu:
   ```

2. To cause **jqu** to change the password at the next login (and to cause this only once), you should use the code **...** After you edit the **/etc/passwd** file, adding **,..** to the password field, the entry looks like this:

   ```
   jqu:RTKESmMOE2m.E,..:100:1:J. Q. Username:/usr/jqu:
   ```

   After the password entry is changed, **jqu** will have to change the password at the next login only.

3. After **jqu** supplies the new password, the system automatically removes the aging code (,..) from the password field:

   ```
   jqu:EFDNLqsFUj.fs:100:1:J. Q. Username:/usr/jqu:
   ```

   Note that the encrypted password information has changed.

Change by root Only

The following shows the password aging information required to establish a password for a login so that only the **root** login can change the password (using the **./** code).

1. Here is the typical login/password entry for user **jqu** again:

   ```
   jqu:RTKESmMOE2m.E:100:1:J. Q. Username:/usr/jqu:
   ```

2. To prevent **jqu** from changing the password, you should use the code **./**. After you edit the **/etc/passwd** file, adding **,./** to the password field, the entry looks like this:

   ```
   jqu:RTKESmMOE2m.E,./:100:1:J. Q. Username:/usr/jqu:
   ```

   Now only **root** can change the password for the **jqu** login. If **jqu** tries to change the password, a permission denied message is displayed.

Locking Unused Logins

If a login is not used or needed, you should do one of two things:

- remove the entry from **/etc/passwd**
- disable (lock) the login

A login is locked by editing the **/etc/passwd** file and changing the encrypted password field to contain one or more characters that are not used by the encryption process. One way to do this is to use the expression **Locked;**. In this expression, the semicolon (;) is an unused encryption character. The text, however, only serves to remind you that the login is locked. Another expression, **not valid**, could also be used to prevent the use of a login because a space is another unused encryption character. Another common technique is placing a single "*" character in the password field.

The following line entry from the **/etc/passwd** file shows the "locked" **bin** login.

```
bin:Locked;:2:2:0000-Admin(0000):/bin:
```

Special Administrative Passwords

There are two familiar ways to access the system: either via a conventional user login or the **root** login. If these were the only two ways to access the system, however, effective use of the system would have to be curtailed (because **root** would own many directories) or many users would have to know the **root** password (a bad security risk) or the system would be wide open (because **root** would own few directories). All of these conditions are undesirable.

The solution to a good mix of system use and system security is available to you through the use of special system logins and administrative commands that can be password-protected (see Procedure 1.4 for information on doing this). The commands, which are also logins, that can be password-protected perform functions that might be needed by a number of users on your M-Series system:

It is recommended that the passwords to these commands/logins be known to only a few users.

Login	Use
root	This login has no restrictions on it and it overrides all other logins, protections, and permissions. It allows the user access to the entire operating system. The password for the **root** login should be very carefully protected.
sys	This login has the power of a normal user login over the files it owns, which are in **/usr/src**.
bin	This login has the power of a normal user login over the files it owns, which are in **/bin**.
adm	This login has the power of a normal user login over the object files it owns, which are in **/usr/adm**.
uucp	This login owns the object and spooled data files in **/usr/lib/uucp**.
nuucp	This login is used by remote machines to log into the system and initiate file transfers via **/usr/lib/uucp/uucico**.
rje	This login has an entry in **/etc/passwd**. There are no files currently associated with this login.
daemon	This is the login of the system daemon, which controls background processing.
lp	This login owns the object and spooled data files in **/usr/spool/lp**.
nobody	This login is used for "weak" network services like **ftp**.

Set User and Group IDs

The set-user identification (suid) and set-group identification (sgid) bits must be used very carefully if any security is to be maintained. These bits are set through the **chmod**(1) command and can be specified for any executable file. When any user runs an executable file that has either of these bits set, the system gives the user the permissions of the owner of the executable. For example, the **/bin/passwd** command is owned by root and has the suid bit set. This allows the **passwd** command to write the root-owned **/etc/passwd** file so users can change their password.

System security can be compromised if a user copies another program onto a file with **-rwsrwxrwx** permissions. For example, if the "switch user" (**su**) command has the write access permission allowed for others, anyone can copy the shell onto it and get a password-free version of **su**. The following paragraphs provide a few examples of command lines that can be used to identify the files with a set-UID.

For more information about the set-UID and set-GID bits, see **chmod**(1) and **chmod**(2).

Check Set-UIDs Owned by root

The following command line lists all set-UID programs owned by **root**. The results are mailed to **root**. All mounted paths are checked by this command starting at **/**. Any surprises in **root**'s mail should be investigated. (Refer to the man pages for a discussion of the **find**(1) command used in the example below.)

```
# find / -user root -perm -4100 -exec ls -l {} \; | mail root
You have mail
# mail
From root Mon Aug 27 07:20 EDT 1984
-r-sr-xr-x   1 root   bin      38836 Aug 10 16:16 /usr/bin/at
-r-sr-xr-x   1 root   bin      19812 Aug 10 16:16 /usr/bin/crontab
---s--x--x   1 root   sys      46040 Aug 10 15:18 /usr/bin/ct
-r-sr-sr-x   1 root   bin      33208 Aug 10 15:55 /usr/lib/lpadmin
-r-sr-sr-x   1 root   bin      38696 Aug 10 15:55 /usr/lib/lpsched
---s--x--x   1 root   other    77824 Aug 18 15:11 /usr/rar/bin/sh
-r-sr-xr-x   1 root   sys      11416 Aug 11 01:26 /bin/mkdir
-r-sr-xr-x   1 root   sys      11804 Aug 11 01:26 /bin/rmdir
-r-sr-xr-x   1 root   bin      12524 Aug 11 01:27 /bin/df
-rwsr-xr-x   1 root   sys      21780 Aug 11 01:27 /bin/newgrp
-r-sr-sr-x   1 root   sys      23000 Aug 11 01:27 /bin/passwd
-r-sr-xr-x   1 root   sys      23824 Aug 11 01:27 /bin/su
? d
#
```

In this example, an unauthorized user ("rar" in group "other") has made a personal copy of **/bin/sh** and has made it set-UID to **root**. This means that **rar** and other members of the group **other** can now execute **/usr/rar/bin/sh** and become the super user.

Check Set-UIDs in the Root File System

The following command line reports all files with a set-UID for the root file system. The **ncheck**(1M) command, by itself, can be used on a mounted or unmounted file system. The normal output of the **ncheck −s** command includes special files. Here, the **grep** command is used to remove device files from the output. The filtering done in this example to remove the device files is applicable only for the root file system (**/dev/dsk/ips0d0s0**). The output of the modified **ncheck** is used as an argument to the **ls** command. The use of the **ls** command is possible only if the file system is mounted.

```
# ls −l 'ncheck.ffs −s /dev/dsk/ips0d0s0 | cut −f2 | grep −v dev'

-r-sr-xr-x   1 root     bin       12524 Aug 11 01:27 /bin/df
-rwxr-sr-x   1 root     sys       32272 Aug 10 15:53 /bin/ipcs
-r-xr-sr-x   2 bin      mail      32852 Aug 11 01:28 /bin/mail
-r-sr-xr-x   1 root     sys       11416 Aug 11 01:26 /bin/mkdir
-rwsr-xr-x   1 root     sys       21780 Aug 11 01:27 /bin/newgrp
-r-sr-sr-x   1 root     sys       23000 Aug 11 01:27 /bin/passwd
-r-xr-sr-x   1 bin      sys       27964 Aug 11 01:28 /bin/ps
-r-xr-sr-x   2 bin      mail      32852 Aug 11 01:28 /bin/rmail
-r-sr-xr-x   1 root     sys       11804 Aug 11 01:26 /bin/rmdir
-r-sr-xr-x   1 root     sys       23824 Aug 11 01:27 /bin/su
-r-xr-sr-x   1 bin      sys       21212 Aug 10 16:08 /etc/whodo
#
```

In this example, nothing looks suspicious. (Note that the "ips" designator for the disk device may be different on your machine. Look in the **/dev/dsk** directory for the correct disk device names for your system. Also note that the ".ffs" appended to the **ncheck** command informs **ncheck** that this is a fast file system as described in Chapter 5. Future versions of UMIPS will assume the fast file system and the .ffs will not be necessary.)

Check Set-UIDs in Other File Systems

The following command line entry shows the use of the **ncheck** command to examine the **/usr** file system (**/dev/dsk/ips0d0s6**, assuming a single-disk system with default partitioning) for files with a set-UID. In this example, the complete path names for the files start with **/usr**. **/usr** is not part of the **ncheck** output.

```
# ncheck.ffs -s /dev/dsk/ips1d0s2 | cut -f2
/dev/dsk/c1d0s2:
/bin/at
/bin/crontab
/bin/shl
/bin/sadp
/bin/timex
/bin/cancel
/bin/disable
/bin/enable
/bin/lp
/bin/lpstat
/bin/ct
/bin/cu
/bin/uucp
/bin/uuname
/bin/uustat
/bin/uux
/lib/mv_dir
/lib/expreserve
/lib/exrecover
/lib/accept
/lib/lpadmin
/lib/lpmove
/lib/lpsched
/lib/lpshut
/lib/reject
/lib/mailx/rmmail
/lib/sa/sadc
/lib/uucp/uucico
/lib/uucp/uusched
/lib/uucp/uuxqt
/usr/rar/bin/sh
#
```

In this example, the **/usr/rar/bin/sh** should be investigated.

The designation "ips" in the special file name may differ on your system. Refer to Chapter 4 and your Release Notes for details.

Chapter 2: User Services

Introduction

This chapter deals with providing a variety of services to the users of your M-Series RISComputer.

- Login administration

  Assigning user and group IDs to users of your system. Maintaining the **/etc/passwd** and **/etc/group** files.

- The user's environment

  Setting up shell environment files and helping users develop individual environment files.

- User communications services

  Establishing and maintaining such services as message-of-the-day, news and mail.

- Anticipating user requests

  Developing an organized plan for responding to user problems.

Login Administration

This section describes how to add a new user to the system.

Adding Users

Before users are permitted to log in to your system, they must be listed in the **/etc/passwd** file. You can make the necessary changes to the **passwd** file using an editor. You need to login as **root** to do this; **/etc/passwd** is generally installed as a read-only file. An entry in the **passwd** file consists of a single line with seven colon-separated fields. The fields and an example are shown below.

login_name:password:user_id:group_id:comments:home_dir:program

```
abc:gOQ3xsv05bWuM,9/TA:103:123:Allen B. Cipher:/usr/abc:/bin/csh
```

The fields are:

login name — A valid name for logging onto the system. A login name is generally from three to six characters; the first character must be alphabetic. It is usually chosen by the user.

password — The encrypted form of the password, if any, associated with the login name in the first field. All encrypted passwords occupy 13 character. The actual password chosen can be any length but is truncated to the first eight characters. At least one character must be numeric. This is to discourage users from choosing ordinary words as passwords. When you add a user to the file you may use a default password, such as **passwd9**, and instruct the user to change it at the first login. Following the encrypted password, separated by a comma, there may be a field that controls password aging. See *Password Aging* in Chapter 1, System Security.

user_id — The user-ID number (**uid**) is between 0 and 50,000. The number must not include a comma. Numbers below 100 are reserved and User-ID 0 is reserved for the super-user. The System Administration menu package does not permit you to specify a number below 100 when adding a user.

group_id — The same conditions apply to the group-ID (**gid**) number as to the **uid**, except that the group-ID 1 is reserved for the *other* group.

comments — This field contains the name of the user and any optional additional information about the user. There is no required format for this field, but avoid slashes and colons.

home_dir — The home directory is where the user is placed upon logging in. The name is usually the same as the login name, preceded by a parent directory such as **/usr**. The **modadduser** menu of the System Administration package allows you to specify the default parent directory. The home directory is the origination point of the user's directory tree.

program This is the name of a program invoked at the time the user logs in. Typical programs are **/bin/sh**, **/bin/csh** and **/bin/rsh**. If the field is empty, the default program is **/bin/sh**.

If you inspect the **/etc/passwd** file on your M-Series system you will see several commands listed among the user login names. These are commands such as **sysadm**(1) that can have passwords assigned to them.

Changing or Deleting Passwd Entries

There are **sysadm usermgmt** menu selections for changing or deleting user entries from the **/etc/passwd** file (see **sysadm**(1) and Procedures 2.2 and 2.3). You also have the option of using an editor to make the changes.

Every so often a user will forget his or her password. When that happens (let's say to user **abc**), you can login as **root** and enter the command:

```
# passwd abc          (The # prompt shows you are root.)
New password: passwd9  (The password entered is actually not echoed.)
Re-enter new password: passwd9
```

Since you did this as the super-user (**root**), you were not prompted for the old password. The command changes **abc**'s password to **passwd9**. You should make sure the user changes the password immediately.

When you delete a login from **/etc/passwd** using the **sysadm usermgmt** menu, all of the user's files and directories are removed. If you remove an entry from the **passwd** file using an editor, you have only removed the entry. The user's files remain.

Group IDs

Group IDs are a means of establishing another level of ownership of and access to files and directories. Users with some community of interest can be identified as members of the same group. Any file created by a member of the group carries the group-ID as a secondary identification. By manipulating the permissions field of the file, the owner (or someone with the effective user-ID of the owner) can grant read, write, or execute privileges to other group members.

Information about groups is kept in the **/etc/group** file. The fields and a sample entry from this file is shown and explained below:

group_name:password(empty):group_id:login_names

```
prog::123:jqp,abc
```

Each entry is one line; each line has the following fields:

group_name The group name can be from 3 to 8 characters, the first of which must be alphabetic.

password The password field should not be used. Group passwords are not generally protected as vigorously as personal passwords, and consequently create an additional security risk when they are used. You can put a "*" or enter ".Locked" in this field to make it clear that it is not to be used.

group_id	The group id is a number from 0 to 50,000. The number must not include a comma. Numbers below 100 are reserved.
login_names	The login names of group members, separated by commas.

The next section discusses how to provide an initial shell environment for each user.

The User's Environment

The user's environment is determined by the shell used (which is defined in the *program* field of the password file). The Bourne shell (**/bin/sh**) files that create the user environment are **/etc/profile** and **$HOME/.profile** ($HOME represents the user's home directory). The **/etc/profile** file is executed first, then the **$HOME/.profile** file. The C-shell (**/bin/csh**) environment files are **/etc/cshrc**, **$HOME/.login** and **$HOME/.cshrc**. The order of execution at login time is **/etc/cshrc**, **$HOME/.cshrc**, and finally **$HOME/.login**.

The User's Home Directory

Each user should be given a home directory. On a small system, you may want to add user directories to the **usr** directory, or perhaps to a subdirectory of **/usr**. If you have a large number of users, you may want to consider setting up a separate user partition or disk. (Adding new disks and partitions is discussed in Chapters 4 and 5.)

For example, if you are going to add users to a subdirectory called **/usr/people**, follow these steps:

```
# mkdir /usr/people
# mkdir /usr/people/carol
# chown carol /usr/people/carol
# chgrp software /usr/people/carol
# chmod u+rwx,g+rx,o+rx /usr/people/carol
```

where "carol" is the new user and you have already added her to the **/etc/passwd** file and assigned her to group "software" in the **/etc/group** file. Her **/etc/passwd** entry might look something like this:

```
carol:nCNYv9zhL4xUn:103:110:Carol Smith:/usr/users/carol:\
    /bin/sh
```

Environment for Bourne Shell Users

Two files determine the Bourne shell user's environment:

1. The system profile.

   The ASCII text file, **/etc/profile**, contains commands, shell procedures, and environment variables. The login process initiated when a Bourne shell user logs in causes this file to be executed. A sample **/etc/profile** is shown below. (Pound signs (#) are comment indicators, and lines beginning with "#" are not executed.)

```
#  The profile that all logins get before using their
#  own .profile.
trap "" 2 3
export LOGNAME
. /etc/TIMEZONE
#  Login and -su shells get /etc/profile services.
#  -rsh is given its environment in its .profile.
case "$0" in
-su)
      export PATH
      ;;
-sh)
      export PATH
      #  Allow the user to break the Message of the Day only.
      trap "trap '' 2" 2
      cat -s /etc/motd
      trap "" 2
      if mail -e
      then
            echo "you have mail"
      fi
      if [ ${LOGNAME} != root ]
      then
            news -n
      fi
      ;;
esac
umask 022
trap 2 3
```

Figure 2-1: A Typical /etc/profile

In addition to **/etc/profile** for all Bourne shell users, the executable file, **.profile**, may reside in a user's home directory. This individual (user-level) profile can contain additional commands and variables that further customize a user's environment. If one exists, it too is executed at login time, after the execution of **/etc/profile**.

A standard profile that can be supplied as an initial **.profile** for Bourne shell users can be found in **/etc/stdprofile**. An example of this file is shown below.

```
#ident      "@(#)sadmin:etc/stdprofile     1.2"
#ident
#     This is the default standard profile provided to a user.
#     They are expected to edit it to meet their own needs.

umask 022

stty line 1 erase '^H' kill '^U' intr '^C' echoe

# list directories in columns
ls()  { /bin/ls -C $*; }
```

Figure 2-2: A Typical **$HOME/.profile**

Several interesting things are contained in the environment files shown above:

- Some environment variables are exported (see Environment Variables below).
- A file named **/etc/motd** is **cat**-ted (see Message of the Day later in this chapter).
- If the user is not **root**, the names of news items are displayed (**news −n**; see **news** later in this chapter).
- If the user has mail (**mail −e**), a message about it is displayed.
- The **ls** command is redefined to always produce a columnar listing.

For information about the Bourne shell programming commands, see **sh**(1) in the *User's Reference Manual*.

Environment Variables

An array of strings called the environment is made available by **exec**(2) when a process begins. Since **login** is a process, the array of environment strings is made available to it. An example of a typical array of strings is shown in Figure 2-3.

```
PS1=$
LOGNAME=abc
PWD=/usr/abc
HOME=/usr/abc
PATH=:/bin:/usr/bin:/usr/ucb
SHELL=/bin/sh
MAIL=/usr/mail/abc
TERM=5420
PAGER=pg
TZ=PST8PDT
TERMINFO=/usr/lib/terminfo
EDITOR=vi
```

Figure 2-3: Environment Array for a Typical User

The environment variables shown in Figure 2-3 give values to 12 names for user **abc**. Other programs make use of the information. For example, the user's terminal is defined as a 5420 (TERM=5420). When the user invokes the editor **vi**(1), **vi** checks the file referenced by TERMINFO (**/usr/lib/terminfo**) where it learns the characteristics of a 5420 terminal (such as the 24-line screen). New variables can be defined at any time. By convention they are defined with the variable in uppercase, followed by an equal sign, followed by the value. Once defined, an environment variable can be made global for the user through the **export** statement. The individual **.profile** file can contain whatever the user wants.

umask

A system default controls the permissions mode of any files or directories created by a user. The M-Series system has default values of 666 for files and 777 for directories. That means that for files everyone automatically gets read and write permission. For directories, everyone gets read, write, and execute permission. (Execute permission on a directory means the ability to **cd** to the directory and to copy readable files from it.)

Users frequently set up a user mask in their **.profile** by means of the **umask**(1) command. **umask** alters permission levels by a specified amount. For example,

```
umask 027
```

leaves the permission level for owner unchanged, lowers the permission level for group by 2, and reduces the permissions for others to zero. The system default was 666; this user mask changes it to 640, which translates into read and write permission for the owner, read permission for the group, and no permission for others.

There may be a **umask** command in **/etc/profile**. If there is, it does not change a user's ability to put one in **.profile**. (Note that since **$HOME.profile** is read after **/etc/profile**, the umask in **$HOME/.profile** would override the **/etc/profile** umask.)

For a detailed discussion of the Bourne shell, see **sh**(1).

The Environment for C-Shell Users

The discussion of the C-Shell adds on to the previous discussion on the Bourne shell. As you will see, commands differ somewhat, and the purpose of this section is merely to introduce how to set-up users with the C-shell. For details regarding use of the C-shell, refer to **csh**(1).

Three files determine the environment for **/bin/csh** users:

- The system (machine-specific) **/etc/cshrc**.

  The file **/etc/cshrc** is executed (or "sourced") by the login process when the C-shell user logs in to the system. An example **/etc/cshrc** is shown in Figure 2-4 below.

```
# default settings for all users

umask 022
set path = ( /bin /usr/net /bin /usr/bin /usr/ucb .)

cat -s /etc/motd
if ($?LOGNAME == 0) then
      echo "$0": LOGNAME: parameter not set
      exit 1
else
      set mail=/usr/mail/$LOGNAME
endif
if ( { /bin/mail -e } ) then
      echo 'You have mail.'
endif
if ( $LOGNAME != root ) then
      news -n
endif
```

Figure 2-4: A Typical **/etc/cshrc** File

- The User's **.login**.

  The login process also executes the **.login** file that resides in the user's home directory. This file can be used, for example, to perform various user-specific actions at login time and set desired variables. Figure 2-5 shows a sample **$HOME/.login** file.

```
setenv SHELL  /bin/csh

setenv TERM vt100

stty line 1 erase '^H' kill '^U' intr '^C' echoe

date

echo "Good morning"
```

Figure 2-5: A Typical **$HOME/.login**

- The User's **.cshrc.**

  Each time a new C-shell is created for a user (for example, at login and whenever the user enters **csh** at the command line), the user's **$HOME/.cshrc** file is "sourced", initializing the new shell with whatever information the user's **$HOME/.cshrc** has specified. A starter **.cshrc** file for C-shell users is provided in **/etc/stdcshrc**. Figure 2-6 shows a sample **$HOME/.cshrc** file.

```
#ident
#       This is the default standard profile provided to a user.
#       They are expected to edit it to meet their own needs.

umask 022

stty line 1 erase '^H' kill '^U' intr '^C' echoe
eval 'tset -S -Q'

# list directories in columns and make a long listing easier
alias  ls      'ls -C'
alias  ll      "ls -al"

# make history easier to use and set it to a length of 50
alias  h       history
set history = 50

# enable command line editing for easy re-entry of commands
set lineedit
```

Figure 2-6: A Starter **$HOME/.cshrc** File

Some of the more interesting aspects of the C-shell environment files included:

- automatic sourcing of variables

  The **setenv** command is used to set variables. No specific "Export *variablename*" statement is required as it is with the Bourne shell.

- aliasing

  Names can be arbitrarily assigned (aliased) to command strings. For example, **alias h history** permits the single character "h" to be typed instead of the entire command.

- other aspects

  Similar mail, news and motd actions to those performed by the sample profile for the Bourne shell user are included in the C-shell examples.

For a detailed discussion of the C-shell environment, refer to **csh**(1).

Default Shells and Restricted Shells

Generally, when a user logs in, the default program that is started is **/bin/sh** or **/bin/csh**. There may be cases, however, where a user needs to be given a restricted shell.

With the restricted shell (**/bin/rsh**) the user is not allowed to:

- change directories

- change the value of $PATH
- redirect output - the restrictions are enforced after **.profile** has been executed.

The administrator can use a restricted shell strategy to limit certain users to the execution of a small number of commands or programs. By setting up a special directory for executables (**/usr/rbin**, for example), and controlling PATH so it only references that directory, the administrator can restrict the user's activity in whatever way is appropriate.

The restricted shell is a subset of the Bourne shell (**sh**). There is no restricted version of the C-shell (**csh**).

Be aware that **/bin/rsh** is not the same as the remote shell **/usr/ucb/rsh** discussed in Chapter 10.

User Communications Services

Several ways of communicating with and among users are available. Some of the most frequently used are described in this section.

Message of the Day

Items of broad interest that you want to make available to all users can be put in the **/etc/motd** file. The contents of **/etc/motd** are displayed on the user's terminal as part of the login process. The login process executes **/etc/profile** (for Bourne shell users) or **/etc/cshrc** (for C-shell users), which are executable shell scripts that, among other things, commonly contain the command

cat /etc/motd

Any text contained in **/etc/motd** is displayed for each user each time the user logs in. For this information to have any impact on users, you must take pains to use it sparingly but consistently and clean out outdated announcements. A typical use for the Message of the Day facility might be

```
5/30:  The system will be unavailable from 6-11pm \
       Thursday, 5/30 - preventive maintenance.
```

Part of the preventive maintenance should be to remove the notice from **/etc/motd**.

news

Another electronic bulletin board facility is in the **/usr/news** directory: the **news**(1) command. The directory is used to store announcements in text files, the names of which are normally used to provide a clue to the content of the news item. The **news** command is used to print the items on your terminal.

A typical **/etc/profile** and **/etc/cshrc** contain the line

```
news -n
```

The **-n** argument causes the names of files in the **/usr/news** directory to be printed on a user's terminal as the user logs in. Item names are displayed only for current items, that is, items added to the **/usr/news** directory since the user last looked at the news. The idea of currency is implemented like this: when you read a news item, an empty file named **.news_time** is written in your login directory. As with any other file, **.news_time** carries a time stamp indicating the date and time the file was created. When you log in, a comparison is made between the time stamp of your **.news_time** file and time stamp of items in **/usr/news**.

Unlike the Message of the Day where users have no ability to turn the message off, with **news** users have a choice of several possible actions:

read everything	If the user enters the command, **news** with no arguments, all news items posted since the last time the user typed in the command are printed on the user's terminal.
select some items	If the **news** command is entered with the names of one or more items as arguments, only those items selected are printed.

read and delete

After the **news** command has been entered, the user can stop any item from printing by sending an Interrupt. (The interrupt is often the DELETE key, but may be CTRL-C or the BREAK key depending on your terminal setup.) Sending two Interrupts in a row stops the program.

ignore everything

If the user is too busy to read announcements at the moment, they can safely be ignored. Items remain in **/usr/news** until removed. The item names will continue to be displayed each time the user logs in.

flush all items

If the user simply wants to eliminate the display of item names without looking at the items, a couple of techniques will work:

```
$ touch .news_time
```

updates the time-accessed and time-modified fields of the

```
$ news > /dev/null
```

prints the news items on the null device.

Write to All Users

The ability to write to all logged-in users, via the **wall**(1M) command, is an extension of the **write**(1) command. It is fully effective only when used by the super-user. While **wall** is a useful device for getting urgent information out quickly, some users may find it annoying to have messages print out on their terminal right in the middle of whatever else is going on. The effect is not destructive but may be somewhat irritating. Users may guard against this distraction by including the command

```
mesg n
```

in their **.profile** or **.login**. This blocks other ordinary users from interjecting a message into your **stdout**. The **wall** command, when used by the super-user, overrides the **mesg n** command. It is best to reserve this for those times when you as the system administrator need to ask users to get off the system. The use of the **wall** command is described in Procedure 2.

mail and mailx

The UMIPS system offers two electronic mail utilities through which users can communicate among themselves. If your system is connected to others by networking facilities, **mail**(1) and **mailx**(1) can be used to communicate with persons on other systems.

mail is the basic utility for sending messages. **mailx** uses **mail** to send and receive messages, but adds to it a multitude of extras that are useful for organizing messages into storage files, adding headers, and many other functions.

When **mailx** is used, a set-up file is helpful. You can find a description of how to use a **.mailrc** set-up file in the **mailx**(1) pages of the *User's Reference Manual*.

Anticipating User Requests

As the system administrator for your M-Series system you can expect users to look to you to help solve any number of problems. In addition to the system log described in Chapter 3, you will find it helpful to keep a user trouble log. Many of the problems that users encounter fall into patterns. If you keep a record of how problems were resolved, you will not have to start from scratch when a problem recurs.

Trouble Reporting

Another technique that is strongly recommended is an organized way for users to report problems. The *Release Notes* supplied with your system provide a Trouble Report Form that can be used to record and report problems to your Field Service representative. Also, a new userid that users can report problems to is often effective.

Chapter 3: Processor Operations

Introduction

This chapter deals with the day-to-day operations of your M-Series RISComputer.

- General operating policy

  Guidelines for balancing the needs of system maintenance and the interests of your user community; suggestions for record keeping; lists of important administrative directories and files.

- Operating Levels

  Definition of the run levels of the system; how they are controlled.

- Running Standalone Shell and monitor programs

  Running firmware-resident and bootable programs from the system firmware.

General Operating Policy

Many administrative tasks require the system to be shut down to a run level other than the multi-user state (see the discussion on Operating Levels below). This means that conventional users cannot access the system. When the machine is taken out of the multi-user state, the users on the machine at the time are requested to log off. You should do these types of tasks when they will interfere the least with the activities of the user community. In addition, include a message in **/etc/motd** and/or news, or mail messages to the users affected.

Sometimes situations arise that require the system to be taken down with little or no notice provided to the users. Try to provide the user community as much notice as possible about events affecting the use of the machine. When the system must be taken out of service, also tell the users when to expect the system to be available. Use the Message of the Day (**/etc/motd**) to keep users informed about changes in hardware, software, policies, and procedures.

At your discretion, the following items should be done as prerequisites for any task that requires the system to leave the multi-user state.

1. When possible, schedule service-affecting tasks to be done during periods of low system use. For scheduled actions, use the Message of the Day (**/etc/motd**) to inform users of future actions.
2. Check to see who is logged in before taking any actions that would affect a logged-in user. The **/etc/whodo** and **/bin/who** commands can be used to see who is on the system.
3. If the system is in use, provide the users advanced warning about changes in system states or pending maintenance actions. For immediate actions, use the **/etc/wall** command to send a broadcast message announcing that the system will be taken down at a given time. Give the users a reasonable amount of time to terminate their activities and log off before taking the system down.

Maintaining a System Log

In a multi-user environment it is strongly recommended that a complete set of records be maintained. A system log book can be a valuable tool when trouble-shooting transient problems or when trying to establish system operating characteristics over a period of time. Some of the things that you should consider entering into the log book are:

- maintenance records and problem information (date/time, description, name of person making entry)
- printouts of error messages and diagnostic phases
- equipment and system configuration changes (dates and actions)

The format of the system log and the types of items noted in the log should follow a logical structure. Think of the log as a diary that you update on a periodic basis. To a large measure, how you use your system will dictate the form and importance of maintaining a system log.

In addition, UMIPS provides various means to collect system data for later analysis. Refer to the **sar** command in Chapter 6, and the **crash** command example in Procedure 3 for more information.

Administrative Directories and Files

This section briefly describes the directories and files that are frequently used by a system administrator. For more detail about the purpose and contents of these directories and files, see Appendix C. For additional information on the formats of the system files, refer to Section 4 of the *UMIPS Programmer's Reference Manual*.

Root Directories

The directories of the **root** file system (**/**) are as follows.

bin	Directory that contains public commands.
boot	Directory that contains configurable object files created by the **/etc/mkboot**(1M) program.
dev	Directory containing special files that define all of the devices on the system.
etc	Directory that contains administrative programs and tables.
lib	Directory that contains public libraries.
lost+found	Directory used by **fsck**(1M) to save disconnected files.
stand	Directory containing standalone commands that reside on disk.
tmp	Directory used for temporary files.
usr	Directory used to mount the **/usr** file system. (See Chapter 5 for a description of this file system.)

Important System Files

The following files and directories are important in the administration of the M-Series RISComputer.

/etc/fstab — File used to specify the file system(s) to be mounted by **/etc/mountall** and remote file systems to be mounted by **/etc/rmountall**.

/etc/gettydefs — File containing information used by **/etc/getty** to set the speed and terminal characteristics for a line.

/etc/group — File describing each group and its members to the system.

/etc/init.d — Directory containing executable files used in upward and downward transitions to all system run levels. These files are executed in files beginning with **S** (start) or **K** (stop) in **/etc/rc***n***.d**, where *n* is replaced by the appropriate run level.

There are two good ways to make modification to these commands:

1. Copy the file in **/etc/init.d** to **/etc/rc***n***.d** and modify it there.
2. Make a copy of the file in **/etc/init.d** and change the corresponding /etc/rc*n***.d** file to point to the new file.

/etc/inittab — File containing the instructions to define the processes created or terminated by **/etc/init** for each initialization state.

/usr/reconfig/master.d — Directory containing files that define the configuration of hardware devices, software drivers, system parameters, and aliases.

/etc/motd — File containing a brief Message of the Day, called (**cat**'d) by **/etc/profile**.

/etc/passwd — File identifying each user to the system.

/etc/profile — File containing the standard (default) environment for all Bourne shell users.

/etc/cshrc — File containing the standard (default) environment for all C shell users.

/etc/rc0 — File executed by **/etc/shutdown** that executes shell scripts in **/etc/rc0.d** and **/etc/shutdown.d** directories for transitions to system run-levels 0, 5, and 6.

/etc/rc0.d — Directory containing files executed by **/etc/rc0** for transitions to system run-levels 0, 5, and 6. Files in this directory execute files in the **/etc/init.d** directory and begin with either a **K** or an **S**. **K** indicates processes that are stopped, and **S** indicates processes that are started when entering run-levels 0, 5, or 6.

/etc/rc2	File executed by **/etc/init** that executes shell scripts in **/etc/rc2.d** and **/etc/rc.d** on transitions to system run-level 2.
/etc/rc2.d	Directory containing files executed by **/etc/rc2** for transitions to system run-levels 2 and 3. Files in this directory execute files in the **/etc/init.d** directory and begin with either a **K** or an **S**. **K** indicates processes that should be stopped and **S** indicates processes that should be started when entering run-levels 2 or 3.
/etc/rc.d	Directory containing executable files that do the various functions needed to initialize the system to run-level 2; they are executed when **/etc/rc2** is run. (Files contained in this directory prior to UNIX System V Release 3.0 were moved to **/etc/rc2.d**. This directory is only maintained for compatibility.)
/etc/rc3	File executed by **/etc/init** that executes shell scripts in **/etc/rc3.d** on transitions to system run-level 3.
/etc/rc3.d	Directory containing files executed by **/etc/rc3** for transitions to system run-level 3 (Remote File Sharing mode). Files in this directory are linked from the **/etc/init.d** directory and begin with either a **K** or an **S**. **K** indicates processes that should be stopped, and **S** indicates processes that should be started when entering run-level 3.
/etc/shutdown	File containing a shell script that gracefully shuts down the system.
/etc/TIMEZONE	File used to set the time zone shell variable TZ.
/etc/utmp	File containing the information on the current run-state of the system.
/etc/wtmp	File containing a history of system logins. If it isn't there, no record is kept.
/usr/lib/spell/spellhist	File containing a history of all words that **spell** fails to match.
/usr/news	Directory containing news files. This directory should be checked periodically, and old files should be discarded.
/usr/spool/cron/crontabs	Directory containing crontab files for the **adm**, **root**, and **sys** logins and ordinary users listed in **cron.allow**.

Each of these files is described in more detail in Appendix C.

Operating Levels

General

After you have set up your M-Series RISComputer for the first time, you and other users can use the system. Whenever you turn it on (including the first time), the system comes up in a multi-user environment in which

- The file systems in /etc/fstab are mounted.
- The **cron** daemon is started for scheduled tasks.
- Networking functions (if configured into the system) are available for use.
- The spooling and scheduling functions of the LP package (if configured into the system) are available for use.
- Users can log in. The **gettys** are spawned on all connected terminal lines listed in **/etc/inittab** to have **gettys** respawned. (With the exception of the console, **gettys** are not on when the system is installed. You have to turn them on yourself.)

This is defined as the multi-user state. It is also referred to as "init state 2" because all of the activities of initializing the system are under the control of the **init** process. The "2" refers to entries in the special table **/etc/inittab** used by **init** to initialize the system to the multi-user state.

Not all activities, however, can be performed in the multi-user state. For example, if you were able to unmount a file system while users were accessing it, you would cause a lot of data to be lost. Hence, for unmounting and other system administration tasks, there is a need for another state, the single-user state.

The single-user state is an environment in which only the console has access to the system and the root file system alone is mounted. You are free to do tasks that affect the file systems and the system configuration because you are the only one on the system.

There are other system states (see Figure 3-1), but first a note of clarification. One of the more confusing things about the discussion of system states is that there are many terms used to identify the same thing: the particular operating level of the system.

Here is a list of frequently encountered synonyms:

- run state
  run level
  run mode
  init state
  system state

Likewise, each system state may be referred to in a number of ways, for example:

- single-user
- single-user mode
- run level 1, and so on

In any case, each state or run level clearly defines the operation of the computer. Figure 3-1 defines each of them as they pertain to the M-Series system.

Run Level	Description
0	Go to Monitor mode. Ready for power-down.
1, s, or S	Single-user mode is used to install/remove software utilities, run file system backups/restores, and to check file systems. Though **s** and **1** are both used to go to single user state, **s** only kills processes spawned by **init** and does not unmount file systems. State **1** unmounts everything except root and kills all user processes, except those that relate to the console.
2	Multi-user mode is the normal operating mode for the system. The default is that the **root** (**/**) and user (**/usr**) file systems are mounted in this mode. When the system is powered up it is put in multi-user mode.
3	User defined run state
4	User defined run state.
5	Go to Monitor mode.
6	Shutdown and reboot

Figure 3-1: System States

How init Controls the System State

UNIX systems always run in one state or another. The actions that cause the various states to exist are under the control of the **init** process, which is the first general process created by the system at boot time. It reads the file **/etc/inittab**, which defines exactly which processes exist for which run level.

In the case of the multi-user state (run level 2), **init** scans the file for entries that have a run level of "2", and executes everything after the last colon (:) on the line containing the corresponding run level.

If you look at your **/etc/inittab**, you'll see something that looks like the following. (It is most unlikely that yours will look exactly like this one; **/etc/inittab** changes from one configuration to another.)

NOTE

If **/etc/inittab** was removed by mistake and is missing during shutdown, **init** will enter the single user state (**init s**). While entering single user state, **/usr** will remain mounted and processes not spawned by **init** will continue to run. You should replace **/etc/inittab** before changing states again.

```
#  Copyright (c) 1984 AT&T
#    All Rights Reserved

#  THIS IS UNPUBLISHED PROPRIETARY SOURCE CODE OF AT&T
#  The copyright notice above does not evidence any
#  actual or intended publication of such source code.

#ident   "$Header"

#
# Field #2 indicates the system's default run level.
# (X for unspecified.)
#
is:2:initdefault:

#
# Boot time system initialization.
#
fs::sysinit:/etc/bcheckrc >/dev/syscon 2>&1
#  Check (& fsck) root fs.
mt::sysinit:/etc/brc >/dev/syscon 2>&1
#  Initialize /etc/mtab.

#
# Run level changes.
#
s1:1:wait:/etc/shutdown -y -iS -g0 >/dev/syscon <&1 2>&1
s2:23:wait:/etc/rc2 >/dev/syscon <&1 2>&1
s3:3:wait:/etc/rc3 >/dev/syscon <&1 2>&1

#
# System shut down.
#
#   telinit 0 = Shutdown and halt.
#   telinit 6 = Shutdown and then reboot.
#   telinit 5 = Like 6, but asks operator which unix
#     to boot. (DOESN'T WORK.)
#

s0:056:wait:/etc/rc0 >/dev/syscon <&1 2>&1
#  Always run rc0.
of:0:wait:/etc/uadmin 2 0 >/dev/syscon <&1 2>&1
fw:5:wait:/etc/uadmin 2 2 >/dev/syscon <&1 2>&1
RB:6:wait:echo "The system is being restarted." \
      >/dev/syscon <&1 2>&1
rb:6:wait:/etc/uadmin 2 1 >/dev/syscon <&1 2>&1
#
```

```
# Gettys
#
# Enabled if field #3 is "respawn";
# Disabled if field #3 is "off".
#
# The h* entries are for M500/800/1000/2000.  The d* entries
# are for M120 with the DIGI board.
#
co:234:respawn:/etc/getty console console none \
      LDISC0  # (console == tty0)
t1:234:off:/etc/getty tty1 co_9600 none LDISC0
t2:234:off:/etc/getty tty2 co_9600 none LDISC0
t3:234:off:/etc/getty tty3 co_9600 none LDISC0
h0:234:off:/etc/getty ttyh0 dx_19200 none LDISC0
h1:234:off:/etc/getty ttyh1 dx_19200 none LDISC0
h2:234:off:/etc/getty ttyh2 dx_19200 none LDISC0
h3:234:off:/etc/getty ttyh3 dx_19200 none LDISC0
h4:234:off:/etc/getty ttyh4 dx_19200 none LDISC0
    .
    .
    .
```

The format of each line is

id:run-level:action:process

- *id* is one or two characters that uniquely identify an entry.
- *run-level* is zero or more numbers and letters (**0** through **6**, **s**, **a**, **b**, and **c**) that determines what *run-level*(s) *action* is to take place in. If *run-level* is null, the *action* is valid in all levels.
- *action* can be one of the following:

sysinit	run *process* before **init** sends anything to the system console (`Console Login:`).
bootwait	start *process* the first time **init** goes from single-user to multi-user state after the system is booted. (If **initdefault** is set to **2**, the process will run right after the boot.) *init* starts the process, waits for its termination and, when it dies, does not restart the process.
wait	when going to *level*, start *process* and wait until it's finished.
initdefault	when **init** starts, it will enter *level*; the *process* field for this *action* has no meaning.
once	run *process* once and don't start it again if it finishes.
powerfail	tells **init** to run *process* whenever a direct powerdown of the computer is requested.
respawn	if process does not exist, start it, wait for it to finish, and then start another.
ondemand	synonymous with **respawn**, but used only with *level* **a**, **b**, or **c**.

off when in *level*, kill process or ignore it.

- *process* is any executable program, including shell procedures.
- **#** can be used to add a comment to the end of a line. Everything after a **#** on a line will be ignored by **init**.

When changing levels, **init** kills all processes not specified for that level. We'll go through more manageable pieces of this table below to get a clearer idea of how the system is controlled by **init**.

A Look at Entering the Multi-User State

Powering Up

When you power up your system, it will enter multi-user state by default. (You can change the default by modifying the **initdefault** line in your **inittab** file.) In effect, going to the multi-user state follows these broad lines:

1. You turn on the computer and enter "auto" at the monitor prompt (to be discussed shortly).
2. The operating system is loaded and the early system initializations are started by **init**.
3. The run level change is prepared by the **/etc/rc2** procedure.
4. Finally the system is made public via the spawning of **gettys** along the terminal lines.

Early Initialization

Just after the operating system is first loaded into core via the specialized boot programs the **init** process is created. It immediately scans **/etc/inittab** for entries of the type **sysinit**:

```
fs::sysinit:/etc/bcheckrc </dev/console >/dev/console 2>&1
```

This entry causes the **/etc/bcheckrc** script to be executed, which performs various system checks. Note that the entry indicates a standard input/output relationship with **/dev/console**. This is the way communication is established with the system console before the system has been brought to the multi-user state.

Preparing the Run Level Change

Now the system must be placed in a particular run level. First, **init** scans the table to find an entry that specifies an *action* of the type *initdefault*. If it finds one, it uses the run level of that entry to select the next entries to be executed. In our sample **/etc/inittab**, the **initdefault** entry specifies run level 2 (the multi-user state) as the level to select and execute other entries:

```
is:2:initdefault:
s2:23:wait:/etc/rc2 >/dev/syscon <&1 2>&1
co:234:respawn:/etc/getty console console \
      none LDISC0 # (console == tty0)
t1:234:off:/etc/getty tty1 co_9600 none LDISC0
t2:234:off:/etc/getty tty2 co_9600 none LDISC0
t3:234:off:/etc/getty tty3 co_9600 none LDISC0
h0:234:off:/etc/getty ttyh0 dx_19200 none LDISC0
    .
    .
    .
```

The other entries shown above specify the actions necessary to prepare the system to change to the multi-user run level. First, **/etc/rc2** is executed. It executes all files in **/etc/rc2.d** that begin with the letter **S**. It then executes any files in the **/etc/rc.d** directory. This accomplishes (among other things) the following:

- sets up and mounts the file systems
- starts the **cron** daemon
- displays the current memory configuration
- makes **uucp** available for use, if installed
- makes line printer (lp) system available for use, if installed
- starts a **getty** for the Console
- starts the hard disk error logging daemon
- starts **getty** on the lines connected to the ports

At this time, the full multi-user environment is established, and your system is available for users to log in (see Procedures 3.1 and 3.4).

A Look at the System Life Cycle

Changing Run Levels

In effect, changing run levels follows these broad lines (see Figure 3-3):

1. The system administrator enters a command that directs **init** to execute entries in **/etc/inittab** for a new run level.
2. Key procedures, such as **/etc/shutdown**, **/etc/rc0**, **/etc/rc2**, and **/etc/rc3**, are run to initialize the new state.
3. The new state is reached, and the system administrator can proceed.

Run Level Directories

Run levels 0, 2, and 3 each have a directory of files that are executed in transitions to and from that level. These directories are **rc0.d**, **rc2.d**, and **rc3.d**, respectively. All files in these directories are associated with files in **/etc/init.d**. The run-level filenames look like this:

S*00name*

or

K*00name*

The filenames can be split into three parts:

S or **K**	The first letter defines whether the process should be started (**S**) or stopped (**K**) upon entering the new run level.
00	The next two characters are a number from 00 to 99. They indicate the order in which the files will be started (S00, S01, S02, etc.) or stopped (K00, K01, K02, etc).
name	The rest of the filename is the **/etc/init.d** filename this file is associated with.

For example, the **init.d** file **cron** is associated with the **rc2.d** file **S75cron**. When you enter **init 2** the **/etc/rc2.d/S75cron** script is executed which in turn executes the shell script **/etc/init.d/cron**. This shell script then executes **/etc/cron**.

Because these files are shell scripts, you can read them to see what they do. You can modify the files, though it is preferable to add your own since the delivered scripts may change in future releases. To create your own scripts you should follow these rules:

- Place the file in the **/etc/init.d** directory.
- Associate the file to files in appropriate run state directories using the naming convention described above.

Going to Single-User Mode

At times in a given work week, you will need to perform some administrative functions in the single-user mode, such as backing up the hard disk and running **fsck** on file systems. The normal way to go to single-user mode is through the **/etc/shutdown** command. This procedure executes all the files in **/etc/rc0.d** directory by calling the **/etc/rc0** procedure, accomplishing, among other things, the following:

- closing all open files and stopping all user processes
- stopping all daemons and services
- writing all system buffers out to the disk
- unmounting all file systems except root
- warns users at intervals

The entry for single-user processing in the sample **/etc/inittab** is:

```
s1:1:wait:/etc/shutdown -y -iS -g0 >/dev/console
   2>&1 </dev/console
```

There are two major ways to start the shutdown processing.

1. You can enter the **shutdown –iS -g0** command (recommended).

2. You can enter the **telinit 1** command (i.e., "tell init to go to state 1"), which forces the **init** process to scan the init table (**/etc/inittab**) for instructions regarding state 1. The first entry it finds is the **s1** entry, and it starts the shutdown processing. Note that when you bring the system down directly using the **init** command instead of the **shutdown** command, it is necessary to first warn users that the system is coming down, sync the memory with the disk(s) and unmount any mounted file systems. For details on these procedures, refer to the manual pages for **wall**(1), **sync**(1M), **umount**(1M) and **init**(1M).

Now the system is in the single-user environment, and you can perform the appropriate administrative tasks.

Standalone Mode

The M-Series machines provide a standalone mode to perform various functions such as formatting a disk. The standalone mode is the first operating state after power-on/system reset, and can be accessed from UMIPS single-user mode by activating **init 0**:

```
# sync;sync
# telinit 0
```

Upon entering the standalone mode, the monitor prompt "**>>**" appears. Entering "**help**" at the monitor prompt produces the following output:

```
>> help

COMMANDS:
autoboot:    auto
boot:        boot [-f FILE] [-n] [ARGS]
cat:         cat FILE_LIST
disable:     disable CONSOLE_DEVICE
dump:        dump [-(b|h|w)] [-(o|d|u|x|c|B)] RANGE
enable:      enable CONSOLE_DEVICE
fill:        fill [-(b|h|w)] [-v VAL] RANGE
get:         g [-(b|h|w)] ADDRESS
go:          go [INITIAL_PC]
help:        help [COMMAND]
help:        ? [COMMAND]
initialize:  init
load:        load CHAR_DEVICE
put:         p [-(b|h|w)] ADDRESS VALUE
printenv:    printenv [ENV_VAR_LIST]
setenv:      setenv ENV_VAR STRING
sload:       sload CHAR_DEVICE
spin:        spin [[-v VAL] [-c CNT] [-(r|w)(b|h|w) ADDR]]*
test:        test [mem(c)|cpu(c)|fpu(c)|all] [test num(s)|?(help)] \
                [l|L](oop)
unsetenv:    unsetenv ENV_VAR
warm:        warm

COMMAND FLAGS
   commands that reference memory take widths of:
      -b -- byte, -h -- halfword, -w -- word (default)
   RANGE's are specified as one of:
      BASE_ADDRESS#COUNT
      START_ADDRESS:END_ADDRESS
Erase single characters by CTRL-H or DEL
Rubout entire line by CTRL-U
>>
```

Help can also be requested for specific commands, for example:

```
>> help printenv
printenv:       printenv [ENV_VAR_LIST]
```

```
>>
```

The **printenv** command shows the setting of various environment variables, for example:

```
>> printenv
netaddr=97.1.0.97
lbaud=9600
rbaud=9600
bootfile=dkip(0,0,8)sash
bootmode=d
console=l
>>
```

Normally, you'd reboot the system by entering **auto** at the monitor prompt. You can also use the **boot** command with various arguments as demonstrated by the examples that follow:

```
>> boot dkis()unix
```

This, like **auto**, will load the sash, and start the UMIPS **init** and other programs, bringing up UNIX. (Note that the disk controller designation (dkis) may differ for your system. Refer to your Release Notes.)

```
>> boot dkis()unix showconfig
```

This boots UMIPS too, but the output to the console is verbose.

```
>> boot dkis()unix root=ips1d0s0
```

The same as above except that UMIPS will be booted from the root disk specified. (The special files designating different disks vary for different M-Series machines. Refer to your Relase Notes or observe the special file names in **/dev/dsk**.)

```
>> boot dkis()unix askme
```

The system will prompt for the device to boot from.

```
>> boot dkis()unix initfile=mv initarg=file1 initarg=file2
```

This executes the UMIPS **mv** command, renaming **file1** to **file2**.

For details regarding the various monitor commands and environment options, refer to the M-Series Technical Reference.

Turning the System Off

The final step in the life cycle of the system is turning it off. The following entries apply to powering the system down:

```
s0:056:wait:/etc/rc0 >/dev/console 2>&1 </dev/console
of:0:wait:/etc/uadmin 2 0 >/dev/console 2>&1 </dev/console
```

To bring the system down, enter the **powerdown** command or invoke the **/etc/shutdown –i0** command. In either case, the **/etc/shutdown** and **/etc/rc0** procedures are called to clean up and stop all user processes, daemons, and other services and to unmount the file systems. When you see the monitor prompt ">>", it is safe to press the Power button to power down the system.

NOTE

If you have disks that have power controls separate from the system, they should be powered–off before powering the system down, and powered–on before powering the system on.

Chapter 4: Disk/Tape Management

Introduction

This chapter covers what you need to know as system administrator about the disk device(s) on your M-Series system.

- Disk device types and sizes
- Making devices known to the operating system
- Preparing disks
- Verifying disk usability
- Handling bad blocks

This chapter does not deal with file systems or the information stored on disk devices. Those subjects are covered in Chapter 5, File System Administration.

Device Types

The M-Series system has a hard disk and a cartridge tape drive unit. Possible configurations on the different models of the M-Series system permit you to have one or more hard disk units.

All system software and user files are kept on the hard disk device(s). The tape device is used primarily as a means of getting software or user files into the system where they can be used, or out of the system for backup, storage or transfer.

Hard Disk Devices

The hard disk units come in various sizes, and the options vary for different versions of the M-Series system. Check your Release Notes for information regarding disk sizes available for your system.

The file **/etc/disktab** contains information on some common disk types for use by the system.

Identifying Devices to the Operating System

Before a disk device can be used on UMIPS, it must be made known to the system. For equipment that comes with your computer, the process of identifying devices is part of configuration and is done automatically as the system is booted.

The traditional way of handling the identification is through an entry in the **/dev** directory of the **root** file system. Of course, an entry in a directory is a file (or another directory), and conceptually a disk device is treated as if it were a file. There is a difference, however, which leads to the practice of referring to devices as "special" files. In the place where a regular file would show the character count for the file, for a special file you find two decimal numbers called the major and minor numbers. Figure 4-1 shows excerpts from the output of "**ls -l**" commands on a user's directory and the **/dev** directory.

(a regular file)

```
-rw-r----- 1 abc   dsg     1050 Apr 23 08:14 dm.ol
```

(integral hard disk device files)

```
brw------- 2 root  sys   4,   0 Apr 11 11:31 /dev/dsk/ips0d0s0
brw-r----- 2 root  sys   4,   1 Apr 11 11:52 /dev/dsk/ips0d0s1

crw------- 2 root  sys   4,   0 Sep 29  1987 /dev/rdsk/ips0d0s0
crw------- 1 root  sys   4,   1 Sep 29  1987 /dev/rdsk/ips0d0s1
```

(cartridge tape device files)

```
crw-rw-rw- 8 root  sys   5, 72 Apr  9 21:38 /dev/SA/ctape0
crw-rw-rw- 8 root  sys   5, 72 Apr  9 21:38 /dev/rSA/ctape0
```

Figure 4-1: Directory Listing Extracts: Regular and Device Files

The extracts from directory listings in Figure 4-1 show a regular file (indicated by the dash (-) in the first position.

- The owner has read/write permission, group members have read permission, other users have no permissions. It is owned by user **abc** who is a member of **dsg** group.
- It has 1050 characters.
- The filename is **dm.ol**.
- The device files are owned by **root**.
- Major and minor numbers appear in place of the character count.

  Major is the number of the device controller or driver (actually, an offset into a table of devices in the kernel); minor is the identifying number of the specific device.

- There are devices that have identical major and minor numbers, but they are designated in one entry as a block device (a `b` in the first column) and in another entry as a character device (a `c` in the first column). Notice that such pairs of files have different file names or are in different directories (for example, **/dev/SA/ctape0** and **/dev/rSA/ctape0** for the cartridge tape drive).

The particular disk device is specified as follows:

/dev/dsk/ips*C***d***D***s***P***

The particular disk device designation (in this case, "ips") varies depending on the system. The letters *C*, *D*, and *P* are replaced with the controller number, disk device number, and partition number, respectively. For example:

/dev/dsk/ips0d0s0

is the root partition on the system disk (controller 0, disk 0, partition 0) and:

/dev/dsk/ips0d1s2

is an extra disk used as more user space or whatever (controller 0, disk 1, partition 2).

NOTE

The three number fields in a disk device designation for a SCSI device are named differently. The first field is the Logical Unit Number (LUN) which is currently always "0", the second field is the "target" which is the disk number 0 to 5, and the final field is the partition field as above.

Block and Character Devices

The identification as a block device or a character device has more to do with how the device is accessed rather A block device name is used when the intent is to read from or write to the device in logical blocks. In the UNIX system, standard C language subroutines for handling file I/O work with blocks. (Blocks are discussed in detail in Chapter 5.)

A character device name is used to read from or write to the device one character at a time. A character device is also referred to as a "raw" device. This is reflected in the "r" in the device names or directory names where, for example, the character device version of the tape drive is in directory **/dev/rSA**. The character-at-a-time method is used by some file maintenance utilities.

Defining a New Special File

The need to define new special device files occurs infrequently. When the need does occur, however, there is a UMIPS command, **mknod**(1M), available to do it.

The general format of the **mknod** command is:

mknod *name* **b** | **c** *major minor*

mknod *name* **p**

The options of **mknod** are:

name	Specifies the *name* of the special file.
b	Specifies a block device.
c	Specifies a character device. The OR sign (|) indicates you must specify one or the other.
major	The *major* number is the slot number.
minor	The *minor* number is the physical device.
p	Specifies the special file as a first-in, first-out (FIFO) device. This is also known as a named pipe. (For more on pipes, see the *Programmer's Guide*.)

The MKDEV Command

UMIPS also provides the **MKDEV** command which creates a set of special files based on the contents of a database. **MKDEV** uses the database files contained in the **./DEV_DB** subdirectory. These files are subject to the rules described in **DEV_DB**(4).

For example, the entire **/dev** directory and its subdirectories were created by running **MKDEV** in the directory **/dev**. **MKDEV** uses the database files contained in **/dev/DEV_DB**.

Formatting and Partitioning

Formatting a disk means establishing addressable areas on the medium. Partitioning means assigning logical units to addressable areas.

Disk Formatting

Before a disk can be used for the storage of information, it must be formatted. Formatting maps both sides of the disk into tracks and sectors that can be addressed by the disk controller. In addition, the format procedure used on UMIPS systems can be used to check the disk for any defects. When UMIPS is shipped from the factory, it is on an already formatted hard disk. There is no need to format this disk unless something goes wrong with it later. Probably the only time you'll need to format a disk is if you get an extra disk for your system, want to reduce fragmentation, or to map out a bad block.

The first block of the disk is reserved for data having to do with the specific disk and is called the volume header. The volume header contains device parameter information, the partition table, and the volume directory. Device parameters are determined by user input or by default values for recognized devices. The partition table describes the logical device partitions. The bad sector table maps areas of the disk that are not usable. Formatting a previously used disk redefines the tracks and erases any data that may be there.

Disk Partitioning

Hard disks are shipped from the factory already formatted. Partitions on the hard disk devices on your M-Series system are allocated in a standard arrangement.

Figure 4-2 illustrates the default UMIPS system disk partitioning scheme. Partition 0 is the **root** partition, 1 is the **swap** partition, 6 is the **usr** partition and 7 is available as swap or as an extra partition. The default partitioning scheme must be preserved on the system disk to remain compatible with future installations and updates.

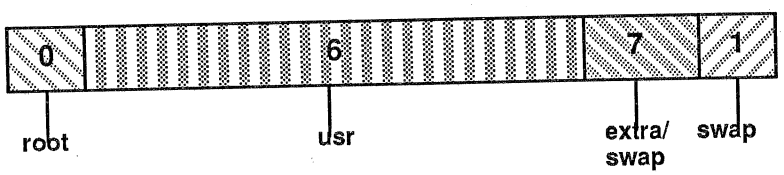


Figure 4-2 System Disk Partitions

Figure 4-3 illustrates the complete set of disk partitions assigned when a disk is formatted. These can be used in any non-overlapping combination to partition extra (non-system) disks. For example, a disk added to your system could use partitions 12, 14 and 15. (Partitions 8, 9, and 10 are reserved for the volume header, track replace, and entire disk, respectively.)

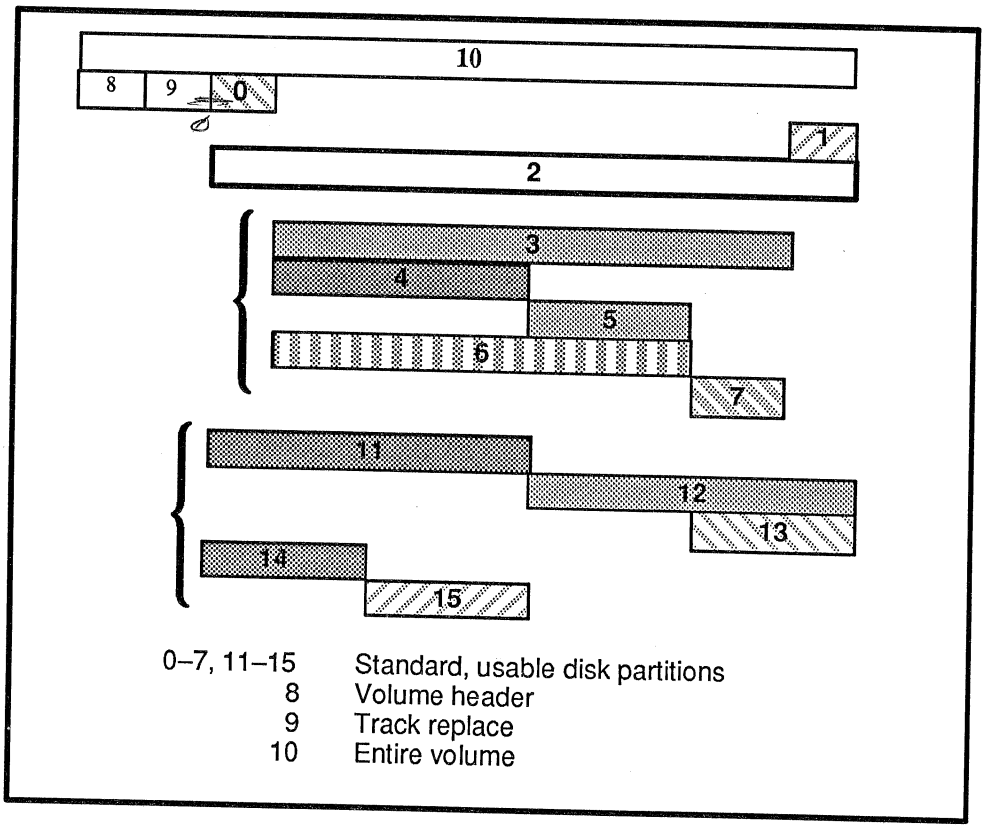


Figure 4-3 Disk Partition Options

Hard disks can be partitioned when they are formatted by use of the **format**(SPP) command, or re-partitioned through use of the **dvhtool**(1M) command. The **format** command is only available from the monitor mode (see Chapter 3), and **dvhtool** is a standard UMIPS command.

format creates a volume header for the disk based on certain default values for recognized disks and user-input values for unrecognized disks. The volume header contains the partition table for the disk. **format** lets you list the partition table, add and delete entries, and modify specific entry information. Use **prtvtoc**(1M) to get information concerning specific disk devices on your system. **dvhtool** is used to examine and modify the partition table and other parts of the disk volume header. See Procedure 4 for examples of the use of **format**, **dvhtool** and **prtvtoc**. For more information on the disk volume header, refer to **dvh**(4).

Increasing Swap Space

If you frequently get console messages warning of insufficient memory, it may mean that the system's current configuration of main memory and swap area is insufficient to support user demands. Before adding more main memory, an alternate solution is to expand the swap area (on either single or multiple hard disk systems).

The swap area can be expanded on the system disk by using partition 7, an unused partition, as an additional swap space. Swap space is added with the **swap -a** command. To add partition 7 on the system disk as swap space, enter:

```
# swap -a /dev/dsk/ips0d0s7 start length
```

where *start* is the disk sector (512-byte block) of the partition that the swap space should start at (usually 0), and *length* is the length in sectors of the swap space.

To add swap space on an extra disk, identify the SCSI disk drive number and the partition number. For example, if the SCSI disk id 1 has been added to your system and you want to use it all as swap space, enter:

```
# swap -a /dev/dsk/ips1d0s2 0 nnn
```

where *nnn* is the number of sectors contained in partition 2 for the type of disk you have.

The disk device designator "ips" may differ on your system. Refer to the special file names in **/dev/dsk**.

Automatically Adding Swap Space

You can add an entry to your **/etc/fstab** file to cause the system to automatically add swap space when the system comes up to multiuser mode.

Add a one-line entry to **/etc/fstab** of the form:

```
/dev/dsk/XXX        none   swap   rw,noauto    0 0
```

where XXX identifies the additional swap space.

Be sure to back up any data from the partition or disk being converted to swap space because it will be lost.

Bad Block Handling

It should be pointed out that new bad blocks seldom occur as long as you take reasonable precautions against movement or vibration of the computer while the disk is still spinning. But when a new bad block does occur, the data stored in the bad block is lost and the disk may be unusable in its current state. If bad blocks do occur, it is important to back-up your system and restore the usability of the disk by reformatting it.

The phrase "bad block" is an unfortunate UNIXism that should really be "bad sector". The term "block" as used here refers to the 512–byte disk sectors.

When Is a Block Bad?

A block is bad when it cannot reliably store data. This is discovered only when an attempt is made to read or write the data and the read or write fails. To make life more difficult, a read fail does not always guarantee a bad block. A read fail might also mean problems in the format of the disk or a failure in the controller or drive hardware.

A write fail generally signals a problem with the format of the disk or a more basic failure in the disk or disk controller hardware. To fix problems of these types you will need to reformat the disk or get the hardware repaired. In the latter case, you should call your service representative. Several distinct failures occurring about the same time should also prompt you to contact your service representative to check them out.

How Are Bad Blocks Fixed?

It is not really so much that a bad block is fixed, but that the system finds a way to live with it. Essentially, the bad block is "mapped-out", that is, references to the bad block are automatically referred to a good block. This mapping occurs transparently to the calling software.

Most disks come with a few manufacturing defects. Bad blocks detected in the manufacturer's quality control checks are identified on a label when the unit is delivered so that if, by some chance, the disk copy of the defect list gets overwritten or corrupted, this block information can be entered when the disk is formatted. The format command will prompt for the information. (Note that this has already been done on the system disk when it is originally formatted and loaded with the operating system at the factory).

When Are Bad Blocks Detected?

Bad blocks are detected when input/output disk operations fail for several successive attempts. This means that data being input or output is lost or corrupted.

Often Asked Questions

Why doesn't the system try to discover that a given block is bad while the system still has the data in memory?

Besides the undesirable increase in system size and complexity, severe performance degradation would result. Also, a block can become a bad block after the copy in memory no longer exists.

Why doesn't the system periodically test the disk for bad blocks?

Reading blocks with their current contents may not show a bad block to be bad. A thorough bit pattern test would take so long that you would never run it, even assuming a thorough test could be devised using ordinary write/read operations. The disk manufacturer already has tested the disk using extensive bit pattern tests and special hardware. All manufacturing defects have been dealt with already.

Why are disks with manufacturing defects used?

Allowing the disks to contain a modest number of manufacturing defects greatly increases the yield, thereby considerably reducing the cost. Many systems, including this one, take advantage of this cost reduction to provide a more powerful system at lower cost.

Scanning the Disk for Bad Blocks

If a block has become bad during the use of the system, it is necessary to scan for the bad block and then map it out. Scanning is done with the **format** command. The **format** command is accessed from the monitor as described in Procedure 4.

If you perform either the format or scan portions of the **format** command, you will lose any data contained in the formatted partition(s). Be sure you have an adequate backup before formatting.

When **format** is run, it passes through several phases:

- initializes the disk
- reads media defects supplied on hard disk list
- formats the drive/partition
- scans for bad blocks
- prompts for the media defect list
- maps-out bad sectors
- writes the disk volume header

For details regarding the **format** command, refer to the System Programmer's Package Reference, **format**(SPP). For details regarding accessing monitor commands, refer to Chapter 3, *System States*, in this document. Procedure 4 demonstrates using the **format** command to scan a partition for bad blocks.

Chapter 5: File System Administration

Introduction

A primary function of the UMIPS operating system is the support of *file systems*. The term "file system" in this case refers to a hierarchical arrangement of *directories* and *files*, as described below. The term is also used in a more global sense, referring to the method of overall management of these structures - as in "The Berkeley Fast File System." When used in this context, it refers to the actual mechanics of disk allocation, file system structure, and the manner in which the operating system accesses and manages files physically on the disk.

This chapter is divided into three parts:

File System Overview:
This section describes the principles and structure of the UMIPS file system.

Creating a File System:
This section describes the process of creating a new file system.

Maintaining a File System:
This section describes the various commands used in the administration of the file system.

File System Overview

The following is an overview of the principles and structure of the UMIPS file system. For details regarding its administration, consult *Maintaining a File System*, later in this chapter.

UMIPS uses the Berkeley Fast File System (FFS) from UNIX 4.3 BSD, as opposed to the traditional System V file system (S51K) from Bell Labs. FFS is favored for a variety of reasons, but primarily due to the significant increases in throughput rates. The new system uses a more flexible allocation policy which orders data far more efficiently on the disk. The results are file access rates of up to ten times faster than those produced by the traditional UNIX file system.

NOTE

As of this printing, S51K is still an option on UMIPS systems; however, S51K will not be supported in future releases. It is strongly recommended that you use FFS rather than S51K when setting up your file systems. For this reason, this chapter contains information pertaining to FFS only. For a comprehensive discussion of FFS vs. S51K, refer to Appendix A, *A Fast File System for UNIX* by K. McKusick, *et al.*)

The following alterations were made to the original file system to produce these results:

- increasing the minimum block size to 4096 bytes
- providing for an adjustable block size that can vary among file systems
- incorporating a block "fragment" allocation system to minimize waste
- clustering data that is sequentially accessed.

It is important to note that, while the underlying implementation has been changed by FFS, the original abstraction of the old file system has been retained. This means that massive software conversion has not been necessary. Furthermore, the increased flexibility of FFS enables it to be more readily adapted to a wide range of peripheral and processor characteristics.

In addition to the speed increase, the FFS permits:

- file names up to 255 characters in length
- symbolic links

Components of the File System

The following sections describe the physical components of the file system, and their relationship to the storage device. These are:

- partitions
- cylinder groups
- file systems
- directories
- files
- blocks

- fragments
- inodes

Partitions

Each disk drive is divided into one or more *partitions*. "Partition" refers to the logical subdivision of physical disk space. The size of partitions may vary; the amount of space to be allocated to a partition is determined when it is created. Essentially, the partition size is limited only by the total amount of available space on the disk. (Refer to Chapter 4 for a discussion of disk partitioning.)

Cylinder Groups

A disk partition is divided into one or more areas called *cylinder groups*. A disk drive consists of several plates, or disks, stacked one above the other. A *cylinder* is the vertical row of identical track positions on each of these plates, one above the other. A cylinder group consists of one or more consecutive cylinders.

Cylinder Group Bookkeeping Information

Associated with each cylinder group is some bookkeeping information that includes:

- redundant copies of the file system descriptor, called the *super block*
- space for file descriptors, called *inodes*
- a bit map describing available blocks in the cylinder group, called the *block map*
- *summary information* describing the use of data blocks within the cylinder group.

Super Block

The *super block* describes the state of the file system - its size, the number of free blocks, free inodes, whether it has been modified, etc. Note that the cylinder group bookkeeping information contains redundant copies of the super block. To protect these backup copies from simultaneous destruction, the bookkeeping information begins at a varying offset from the beginning of each cylinder group. The offset increases by about one track for each succeeding cylinder group. In this way, the redundant super block information spirals down into the pack, rather than residing entirely on the top platter. Thus, the data on any single track, cylinder, or platter can be lost without losing all copies of the super block. The space between the beginning of the cylinder group and the bookkeeping information is used for data blocks.

Summary Information

The cylinder group bookkeeping information contains data regarding the number of directories, inodes, blocks, and fragments contained in the file system. This information is referred to as the *summary information*, and is updated accordingly as the file system is modified.

File Systems

Each disk partition may contain one *file system*. "File system" in this case refers to the hierarchical structure of directories and files within a partition, as opposed to the overall method and structure of file system management.

A file system may not span multiple partitions. However, file systems are actually located in logical disk partitions that may overlap. This allows, for example, a program to copy an entire disk drive containing multiple file systems. On the other hand, it is important to be careful not to write a partition that is not being used and that overlaps a partition that is being used because you will destroy data on the used partition. For example, if you refer to the disk partitioning scheme illustrated in Chapter 4, you can see that system disk partitions 0 and 6 overlap the unused partition 11. Writing to partition 11 on the system disk will damage the information contained in partitions 0 and 6.

A file system is described by its *super block*. The super block contains file system size and status, inode information, and block information.

Files and Directories

Within the file system are files. A file consists of a one-dimensional array of bytes.

Certain files are used as directories and contain pointers to files that may themselves be directories, or regular files. A directory is a special type of file that you are permitted to use, but not to write. The operating system is responsible for writing directories.

Blocks

A file is stored in a collection of storage units called *blocks*. Block size is adjustable, and can, in fact, vary among file systems. The minimum UMIPS block size is 4096 bytes, which makes it possible to create files as large as 2^{32} bytes with only two levels of indirection (refer to *Inodes*, below). The block size can be any power of two greater than or equal to 4096, and is assigned when the file system is created. The most commonly used (and the default) block size on UMIPS systems is 8192 bytes (referred to as "8K" blocksize).

The block size is recorded in the file system's super block, and cannot be changed without rebuilding the file system.

The "8K blocksize" refers to the FFS block, a different entity than the actual disk sector which is also sometimes referred to as a block. This document attempts to make clear which type of block is being referred to whenever the term "block" is used. Be aware of the difference when reading the manual pages. There, "block" often means 512-byte sector.

Fragments

To minimize waste when storing small files, a system of *block fragment* allocation has been devised.

A block can be divided into 2, 4, or 8 portions called *fragments*. Fragment size is determined when the file system is created; the minimal size is limited by the disk sector size, which is 512 bytes on a UMIPS system. The most commonly used fragment size on a UMIPS system is 1024 bytes (referred to as 1K) in conjunction with an 8K block size.

If a block is fragmented to accommodate a small amount of data, the remaining fragments of the block can be allocated to other files. For example, an 11000 byte file, stored on a 8192-byte block/1024-byte fragment file system, would use one full block and three fragments from another block. If no block with three aligned fragments is available at the time, a full block is split to provide the necessary fragments,

leaving five unused fragments. The unused fragments can be allocated to another file as needed.

Inodes

Every file has a descriptor associated with it called an *inode*. An inode is a "reserved" block containing information describing the ownership of the file, time stamps marking last modification and access times, and an array of indices that point to the data blocks for the file.

The first 12 blocks of the file are directly referenced by values stored in the inode itself. An inode may also contain a "singly indirect" pointer to a special disk block containing pointers to additional file blocks. Some of these blocks, in turn, may contain references to another layer of file blocks. This produces a "fan" of direct and indirect block references that chain together to form the file. For example, in a file system with a 4096-byte block size, a singly indirect block contains 1024 further block addresses, a doubly indirect block contains 1024 addresses of further singly indirect blocks, and a triply indirect block contains 1024 addresses of further doubly indirect blocks. There is no quadruple indirection.

A static number of inodes are allocated to each cylinder group when the file system is created. The default policy is to allocate one inode for each 2048 bytes of space in the cylinder group.

Structure of A File System - The User's View

From the user's standpoint, a file system consists of directories, subdirectories, and files arranged in hierarchical order, like an inverted "tree" structure. The following diagram illustrates the relationship of directories to files in this tree structure; the circles represent directories:

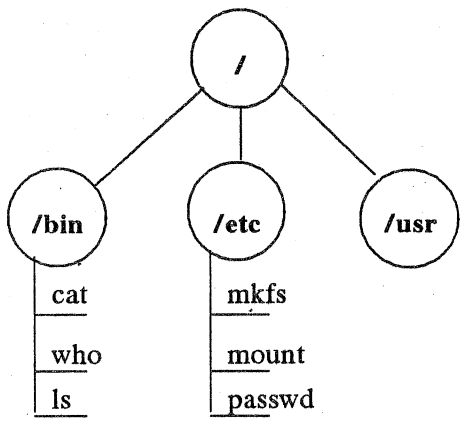


Figure 5-1: Hierarchical structure of a file system

You'll notice a circle at the top of the diagram, labeled with a single slash. This represents the "root" directory of *all* the file systems existing on your UMIPS system, and is in fact referred to by the name **root**. Traditionally, the root directory is represented by a single slash (**/**).

Attached to the root directory are other directories, one of which (**usr**) happens to be a file system that is attached or "mounted" on the **/usr** directory. The slash in front of each indicates that they are directly connected to the **root** directory. Descending from these directories are other directories, subdirectories and files, forming an increasingly intricate branching structure. Any directory can contain further subdirectories, as long as space allows. The following diagram shows the addition of directories and files to the **/usr** file system:

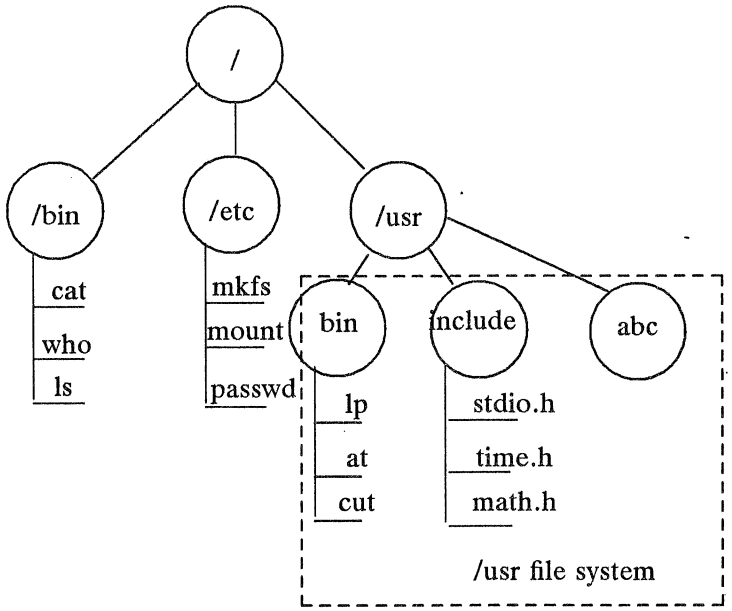


Figure 5-2: Hierarchical detail of a typical **/usr** file system

What is important to understand here is that the files and subdirectories under **/bin** and **/etc** are in the same file system (the "root" file system), while the structure under **/usr** is a separate file system, contained in a different disk partition (or possibly on a different disk altogether).

Creating a File System

This section describes how a file system is established on a disk partition, and then how that file system is mounted on the root file system.

Making and Mounting a New File System

Once disks have been formatted and partitioned, file systems can be created on them and they can then be mounted and used. On a system disk, you simply follow the operating system installation procedures, and the file systems are created and mounted automatically. On extra disks, these steps are performed manually. Once the disk is formatted and partitioned, the next step is to create the file systems, and make them available for access. The **newfs**(1FFS) and **mkfs**(1FFS) commands are used to create a new file system. The **mount**(1M) command is used to make a file system available for use.

newfs.ffs and mkfs.ffs

The **mkfs.ffs** program does the actual work of creating a new file system. The **newfs.ffs** command is a front-end to **mkfs.ffs** which calculates various parameters that are needed by **mkfs.ffs**. If you use **mkfs.ffs** directly, you must supply these parameters yourself. For this reason, **newfs.ffs** is generally used instead of **mkfs.ffs**, except under unusual circumstances when these parameters *must* for some reason be provided manually.

newfs.ffs

The **newfs.ffs** program performs the following actions:

1. determines the appropriate parameters to pass to **mkfs.ffs**
2. builds the file system by forking **mkfs.ffs**
3. creates and replicates the superblock

The syntax for **newfs.ffs** is as follows:

/etc/newfs.ffs [**-N**] [**-v**] [*mkfs.ffs-options*] **-s** *sectors special_file disk-type*

The **-N** option causes the file system parameters to be printed out without actually creating the file system. If you have never used **newfs.ffs** before, it is recommended that you try a practice run, using this option. The **-v** option turns on "verbose" mode; this prints a report of the actions taken by **newfs.ffs**, including the parameters passed to **mkfs.ffs**.

The **newfs.ffs** **-N** command output includes a list of alternate superblocks for the filesystem. It is a good idea to save this list in case the file system's superblock becomes corrupted.

The **-s** *sectors* specifies the size in 512-byte blocks of the file system. Use **prtvtoc**(1M) to get partition sizes. *special-file* is the special file name in **/dev** and *disk_type* is the type of disk as named in **/etc/disktab**.

newfs.ffs also provides options for overriding the default parameters which are passed to **mkfs.ffs**. These are the "[*mkfs.ffs-options*]" referred to in the syntax example:

-s *size* Set the number of file system sectors to *size*. (Use *sectors* field in **newfs.ffs** syntax described above.)

-b *blocksize*
Set the file system block size to *blocksize* bytes. The default is 8192.

-f *fragsize* Set the file system fragment size to *fragsize* bytes. The default is 1024.

-t *#tracks_per_cylinder*
Set the number of tracks per cylinder. The default is 16.

-c *#cylinders_per_group*
Set the number of cylinders per cylinder group. The default number is 16.

-m *free_space_%*Set the percentage of free space reserved apart from normal use. This is the minimum free space threshold; the default is 10%.

-o *optimization_preference*This parameter takes one of the two following arguments: **time** or **space**. The **time** option instructs the system to optimize by minimizing the time spent allocating blocks. The **space** option instructs it to optimize by minimizing the space fragmentation on the disk. If the free space reserve is set equal to or greater than 10%, the default is to optimize for time. If the reserve is set below 10%, the default is to optimize for space.

-r *revolutions_per_minute*
Specify the number of disk revolutions per minute. The default is 3600.

-S *sectorsize*
Set the size of a sector in bytes. The default is 512 bytes.

-i *#bytes_per_inode*
Set the density of inodes for the file system. The default is one inode per 2048 bytes of data space. If you want to increase the number of inodes, set the number of bytes to a lower number. To decrease total inodes, enter a higher number of bytes.

mkfs.ffs

The **mkfs.ffs** program performs the actual work of creating the file system. Normally, you should not have to use **mkfs.ffs** directly, since **newfs.ffs** provides options for fine-tuning file system parameters.

The syntax for **mkfs.ffs** is:

/etc/mkfs.ffs [**-N**] *special size* [*nsect* [*ntrack* [*blksize* [*fragsize* [*ncpg* [*minfree* [*rps* [*nbpi* [*opt*]]]]]]]]]

The parameters are defined as follows:

-N The **-N** option causes the file system parameters to be printed out without actually creating the file system. If you have never used **mkfs.ffs** before, it is recommended that you try a practice run, using this option.

special **mkfs.ffs** builds a file system by writing to the special device file *special* unless the **-N** option has been given.

size Specifies the number of sectors to be allotted to the file system.

nsect Specifies the number of sectors per track on the disk. The default is 32.

ntrack Specifies the number of tracks per cylinder on the disk. The default is 16.

blksize Set primary block size to *blksize* bytes. The block size must be a power of two; currently, two block sizes are allowed: 4096 or 8192 bytes. The default is 8192.

fragsize Set the fragment size to *fragsize* bytes. The fragment size must be a power of two in the range of 512 to 8192 bytes. The default is 1024.

ncpg Specifies the number of disk cylinders per cylinder group. This must be in the range of 1 to 32. The default is 16.

minfree This specifies the percentage of free space reserved from normal use. This is the minimum free space threshold; the default is 10%.

rps Specifies the number of disk revolutions per second. The default is 60.

nbpi Specifies the number of bytes per inode. The default is 2048 bytes of data space per inode. If you want to increase the number of inodes, set the number of bytes to a lower number. To decrease the number of inodes, enter a higher number of bytes.

opt This parameter takes one of the two following arguments: **t** for time or **s** for space. The **t** option instructs the system to optimize by minimizing the time spent allocating blocks. The **s** option instructs it to optimize by minimizing the space fragmentation on the disk. If the free space reserve is set equal to or greater than 10%, the default is to optimize for time. If the reserve is set below 10%, the default is to optimize for space.

Mounting and Unmounting File Systems

A file system must be **mount**ed in order for it to be accessible to the system. The **mount**(1M) command adds an entry for the file system in **/etc/mtab**, the mount table which is maintained by the kernel. This entry ties the file system from the specified disk device with the indicated directory name. For example, the mount command

```
# mount /dev/dsk/ips1d0s6 /usr
```

indicates that **/dev/dsk/ips1d0s6** contains a file system that is accessed via the directory **/usr**. The new entry in **/etc/mtab** would look like this:

```
/dev/dsk/ips1d0s6/usrffsrw1 2
```

The **mount** command has other arguments. See the *System Administrator's Reference Manual* for complete details.

The **root** file system is always mounted as part of the boot procedure. The **/usr** file system, which may be on the same disk device as **root**, is also automatically mounted as the system is being brought up to multi-user mode. The **mount** command that brings these two file systems online is issued by start-up shell procedures when the system is booted.

If you change directories (**cd**(1)) to an unmounted directory, or a directory in the process of being mounted, all you will find is the directory itself and an entry for its parent directory ("." and ".."). Until the **mount** command completes, the system is oblivious to the contents of the unmounted file system.

The command for unmounting a file system requires only the name of the special device. To unmount a file system, you issue the **umount**(1M) command:

```
# umount /dev/dsk/ips1d0s6
```

(Note that the syntax is **u**mount and not **un**mount.) Unmounting is frequently a first step before using other commands that operate on file systems (for example, **fsck**(1M), which checks and repairs a file system). Unmounting is also an important part of the process of shutting the system down.

The fstab File

The **/etc/fstab** file is used by the **mount** command when the system goes multiuser. Each line in this file identifies a file system that is to be mounted and where it is to be mounted. Additional fields describe the type of file system, permissions, and other options (see **fstab**(1M)). A sample **fstab** file is shown below.

```
/dev/root       /       ffs rw 0 0
/dev/usr        /usr    ffs rw 0 0
/dev/dsk/ips0d1s11    /work2        ffs rw 0 0
/dev/dsk/ips0d0s7     none   swap   rw,noauto    0 0
quasar:/usr     /usr/quasar nfs rw,soft,bg,timeo=20,retry=2 0 0
```

Maintaining a File System

Once a file system has been created and made available, there are several tasks routinely done to make certain that the file systems in regular use are providing the level of service and stability they should. These tasks can be grouped into the following categories:

- checking for file system consistency
- monitoring disk usage
- compressing and reorganizing file systems
- backing up and restoring file systems
- watching for errors

The Need for Policies

File system administration should be based on establishing a set of policies that are appropriate for your organization. There can be no hard-and-fast rules for such matters as the size of file systems, the number of users in a file system, the way in which backups are done, the extent to which users can be allowed to keep inactive files in the system, or the amount of disk space a single user is entitled to occupy. These questions can only be resolved within the context of the organization. There are a number of variables involved, such as the number of users, the type of work they are doing, the number of files needed, and so on. The responsible administrator must determine what best meets the needs of the organization.

Shell Scripts for File System Administration

Once policies have been designed, many of the routine tasks connected with file system administration can be automated by using shell scripts. Monitoring disk usage, for example, can be managed by a shell script that monitors the system automatically, transmitting messages to the system console when exceptions are detected. Here are a few ideas:

- Use a shell script running under **cron**(1M) control to investigate free blocks and free i-nodes, reporting on file systems that fall below a given threshold.
- Use a shell script to do automatic clean-ups of files that grow.
- Use a shell script to highlight cases of excessive use of disk space.
- Use a shell script to highlight cases of files that haven't been accessed recently.

Monitoring Disk Usage

Disk usage for the file system should be monitored for the following reasons.

- If not watched regularly, the percentage of disk space used increases until the allocated space is used up.
- When the allocated space is used up, processes run very slowly, or not at all; throughput declines radically due to allocation inefficiencies; the system also spends much of its time printing error messages concerning lack of file space.

- There is a natural tendency for users to forget about files they no longer need, unnecessarily using up storage space.
- Some files grow larger as a result of perfectly normal use of the system. It is an administrative responsibility to keep them under control.
- Some directories, notably **/tmp**, accumulate files during the day. When the system is first brought up, **/tmp** must have enough free blocks to carry it through to **shutdown**(1M). (The **crontab**(1) **periodic** file automatically cleans out **/tmp** and **/usr/tmp**.)

There are four tasks that are part of keeping disk space uncluttered:

1. monitoring percent of disk space used
2. monitoring files and directories that grow
3. identifying and removing inactive files
4. identifying large space users

Monitoring Percent of Disk Space Used

Monitoring disk space may be done at any time to see how close to capacity your system is running. Until a pattern has emerged, it is advisable to check every day. In this example, the **df**(1M) command is used.

```
$ df /user2
Filesystem         Type  kbytes   use   avail %use  Mounted on
/dev/dsk/ips0d0s7 ffs   18703    1880  16823 10%   /user2
$
```

The results of this example show a filesystem in which only 10% has been used.

Monitoring Files and Directories that Grow

Almost any system that is used daily has several files and directories that grow through normal use. Some examples are:

File	Use
/etc/wtmp	history of system logins
/usr/adm/SYSLOG	report of system activities
/usr/lib/cron/log	history of actions of **/etc/cron**
/usr/lib/spell/spellhist	words that **spell**(1) fails to match

The **periodic** crontab file handles the first three (**/etc/wtmp**, **/usr/adm/SYSLOG** and **/usr/lib/cron/log**) automatically.

Identifying and Removing Inactive Files

Part of the job of cleaning up heavily loaded file systems involves locating and removing files that have not been used recently. The commands you might use to do this work are shown below; the policy decisions involved are:

- How long should a file remain unused before it becomes a candidate for removal?
- How should users be warned that old files are about to be purged?
- Should the files be permanently removed or archived?

The **find**(1) command can be used to locate files that have not been accessed recently. **find** searches a directory tree beginning at a point named on the command line. It looks for filenames that match a given set of expressions, and when a match is found, performs a specified action on the file. This example barely begins to suggest the full power of **find**.

```
$ find /usr -type f -atime +60 -print > /tmp/deadfiles &
```

Here is what the example accomplishes:

/usr specifies the pathname where **find** is to start. Presumably, your machine is organized in such a way that inactive user files will not often be found in the **root** file system.

-type **tells find** to look only for regular files, and to ignore special files, directories, and pipes.

-atime +60 indicates you are interested only in files that have not been accessed in 60 days.

-print indicates that when a file is found that matches the **-type** and **-mtime** expressions, you want the pathname to be printed.

> /tmp/deadfiles &
directs the output to a temporary file and indicates that the process is to run in the background. This is a sensible precaution if your experience tells you to expect a substantial amount of output.

The **sysadm fileage**(1) command can be used to produce similar information (see Procedure 5.3).

Identifying Large Space Users

There are several policy decisions that need to be made concerning user disk usage for your particular installation. These include:

- What constitutes a reasonable amount of disk space for a single user?
- If a user exceeds the normal amount by 25% (for example), is it possible the user's job requires extraordinary amounts of disk space?
- Is the system as a whole running short of space? Do existing limits need to be reviewed?

Three commands produce useful information in this area: **du**(1M), **df(1M)** and **find**(1).

du produces a summary of the block counts for files or directories named in the command line. For example:

```
$ du /usr
```

displays the block count for all files and directories in the **/usr** file system. Optional arguments allow you to refine the output somewhat. For example, **du –s** may be run against each user's login directory to monitor individual users.

df reports the amount of free disk space in the specified file system. For example:

```
$ df /usr
```

displays the number of Kbytes available in the **/usr** file system.

The **find** command can be used to locate specific files that exceed a given size limit.

```
$ find /usr –size +10 –print
```

This example displays a list of the pathnames of all files (and directories) in the **/usr** file system that are larger than 10 (512-byte) blocks.

File System Backup and Restore

The importance of establishing and following a file system backup plan is too often not appreciated until data is lost and cannot be recovered. Backing-up a file system takes time. Trying to recover lost or damaged data from paper records and best-guess-work takes even more time. The value of an effective system backup plan lies in the ability to recover lost or damaged data easily and reliably.

The **dump**(1M) and **restore**(1M) commands are used to backup entire file systems in a consistent manner. **dump** supports "dump levels" which can be used to back up only those files which have changed since the previous dump.

The capability to copy selected directories and files to tape is provided by using the **find**(1) command with **cpio**(1). The directories and files are also read back to the hard disk by using the appropriate **cpio** command. Another command that can be used to archive files is **tar**(1). Both **cpio** and **tar** may be combined with the **multivol**(1) command to handle backups that exceed the capacity of a single tape. Note that backups made with the **multivol** command must be restored with the **multivol** command.

Another method of backing up data is to copy file systems to another computer system over a high-speed data link. The link between the machines must be a high-speed data link (for example, Ethernet) so that data transfers can be accomplished quickly. The target machine should be a larger system with mass storage capability. The **rdump** and **rrestore** commands can be used to perform network backups.

The backup plan can include any or all of these methods. This plan should be re-evaluated as the use of the machine changes.

Complete Backup

A complete backup of a file system is obtained by performing a level 0 dump of the file system. It is necessary to do this periodically in order to ensure that a complete backup of the file system exists. If the backup requires more than one tape, you will be prompted when it is time to put the next tape in. The following commands back up the entire **/usr** file system:

```
# umount /usr
# dump 0u /dev/dsk/ips0d0s6
```

(Note that there is no minus ("-") in front of the 0.) This causes all data in the **usr** file system (system disk partition 6) to be backed up on the default tape output device (**/dev/mt/ctape0**). If you have not inserted a tape, the system will inform you that it is unable to write to tape and will ask if you want to try again. Dumps are best performed on unmounted file systems.

The disk controller designation "ips" may differ on your system. Refer to the special device names in **/dev/dsk** on your system for the correct prefix.

The 0 in the example above specifies the "level 0" or complete dump of the file system. The **u** causes the **/etc/dumpdates** file to be updated with the date and level of the dump. This information is used in incremental backups as described in the next section.

Incremental Backup

An incremental dump backs-up those files that have changed since the last lower-level dump. When **dump** is run with the **u** (update) and dump-level keys specified, the file **/etc/dumpdates** is checked to determine when the last lower-level dump occurred. All files with a later modification time in the specified file system are then backed-up.

Below is an example of an **/etc/dumpdates** file.

```
/dev/dsk/ips0d1s5          0 Sat Jun  4 08:38:53 1988
/dev/dsk/ips0d1s5          1 Sat Jun 18 06:44:59 1988
/dev/dsk/ips0d1s11         0 Sat Jun  4 14:46:03 1988
/dev/dsk/ips1d0s2          0 Sat Jun  4 11:42:26 1988
/dev/dsk/ips1d1s2          0 Sat Jun  4 09:10:27 1988
/dev/usr                   0 Sat Jun  4 07:46:11 1988
/dev/dsk/ips1d0s2          1 Sat Jun 18 07:24:45 1988
/dev/dsk/ips1d1s2          1 Sat Jun 18 06:47:10 1988
/dev/dsk/ips0d1s11         1 Sat Jun 18 07:14:00 1988
/dev/usr                   1 Sat Jun 18 06:42:38 1988
/dev/dsk/ips0d1s5          1 Sat Jan 23 15:57:38 1988
/dev/root                  1 Sat Jun 18 06:41:55 1988
/dev/dsk/ips1d0s2          2 Tue May  3 18:01:54 1988
/dev/dsk/ips1d1s2          2 Tue May  3 18:29:50 1988
/dev/root                  0 Sat Jun  4 07:23:06 1988
```

If a level 1 dump of **/usr** is now made:

 # dump 1u /dev/dsk/ips0d0s6

all files modified since June 4th are written to disk.

Note that files backed-up with the **dump** command can only be recovered by use of the **restore** command described below.

The restore Command

The **restore**(1M) command is used to read files from a tape created by **dump** and write them to disk. Caution must be exercised when using this command. A complete restore of a file system could be made by entering this command in the appropriate directory:

 # restore r

The result of this is that the files on the dump tape are written to the current directory (and any subdirectories contained on the tape). These files overwrite any existing files with the same name! Therefore, this is usually done on a newly-created file system. A complete multilevel series of incremental dumps can restore the most current backups of the complete file system by beginning with a level 0 dump and then inserting progressively higher numbered dump tapes until the most recent dump is reached.

The **restore** command also permits selective backups by use of the **x** key. The name of the desired file(s) is supplied on the command line. For example:

 # restore x /user/johnr/.cshrc

would restore the file **/user/johnr/.cshrc** from the dump tape to the current directory.

A useful interface to the **restore** command is provided by the "-i", or interactive, option. Used with "-i", **restore** reads in the directory information from tape and then provides a standard shell-like interface permitting you to move around the directory tree to extract files. Commands supported include **ls**, **cd**, **pwd**, and **add** (to add files to the extraction list).

The tar and cpio Commands

tar and **cpio** are two additional commands that can be used to backup files. They are most useful when backing up directories, single files or files with matching patterns rather than entire file systems. In addition, these commands usually retain execute permissions accessible by everyone so users can make personal backups, exchange file tapes, etc. Both commands have options for saving files as well as restoring them.

If you are going to be using **tar** or **cpio** to make backups that extend over more than one tape, use the **multivol** command described below.

The **tar** command (Tape ARchiver) takes a filename pattern and writes it or reads it as requested. For example:

```
$ tar c /user/jar/tester.c
```

creates a tape containing the file /user/jar/tester.c. The command:

```
$ tar x /user/jar/tester.c
```

extracts that file from tape. (Note that the **tar** command does not use a preceding "-" before arguments.)

The **cpio** command takes standard input as the filename(s), so a common way to use **cpio** is in conjunction with **ls** or **find**. The current directory can be written to tape like this:

```
$ env - ls | cpio -o > /dev/mt/ctape0
```

(The "env -" removes any environment-specifics from inclusion in the output from the current command line. It must precede the **ls** command if the **ls** command has been aliased to output listings in columns or add any characters to the file names.)

To restore the same directory using **cpio**, read in the tape from the directory in which you want the files to be written:

```
cpio -i < /dev/mt/ctape0
```

The main advantage of **cpio** is the ease with which it can be combined with the **find** command as standard input. The great flexibility of the **find** command makes it easy to backup files of a certain length, say, or age. For example:

```
$ find . -type f -mtime -2days -print | cpio -o >/dev/mt/ctape0
```

backs up all files (**-type f**) in the current directory (**.**) and its subdirectories that have been modified in the last two days (**-mtime -2days**) and writes (**cpio -o**) them to tape (**/dev/mt/ctape0**).

Making Multi-tape Backups With tar and cpio

If you think your backup is going to require more than a single tape, it is a good idea to use the **multivol** command. This way, you will not have to figure out how to divide the backup into tape-sized chunks, but can let **multivol** do that for you. **multivol** prompts for each tape as it is required.

When using **multivol**, you must either specify the backup device each time or associate the default backup device **/dev/multivol** with an actual device. If, for example, you want your default backup device to be your tape drive, you can link **/dev/multivol** to the tape drive:

 # ln /dev/rmt/ctape0 /dev/multivol

and you will not have to specify the tape device each time you use **multivol**. To backup using **multivol**, pipe to it using this syntax:

backup command | **multivol -o** *[device_name]*

where *backup command* is a **tar** or **cpio** command and the rest of the command line uses **multivol -o** to output to the default backup device (the tape, if you made the link described above, or the device name supplied).

To restore a **multivol** backup, begin the command line with **multivol**:

multivol -i *[device_name]* | *backup command*

where the **-i** option is used and *backup command* is either a **tar** or **cpio** command, depending on which was used to make the backup. Note also that backups made using **multivol** must be restored using **multivol**.

Network Backups

In addition to using the tape drive or a locally attached disk as a backup device, it is also possible to perform backups to a remote host if you are attached to a network. For example, if you have a TCP/IP connection to another machine that is running the **/etc/rmt** program, you can use its resources for making backups (assuming you have so arranged with the system administrator). The commands you would use are **rdump**(1M) and **rrestore**(1M).

rdump uses the same syntax as **dump** with the exception that you always use the **f** option to specify the remote dump site. The syntax is:

rdump f *hostname:device filesystem*

where *hostname* is the remote host, *device* is the disk (or tape) device on the remote system being used for the backup, and *filesystem* is the special file name of the file system being dumped. For example:

```
# /etc/rdump.ffs f diamond:/dev/disk07 /dev/dsk/ips0d0s7
DUMP: Date of this level 9 dump: Tue Jun 14 15:22:59 1988
DUMP: Date of last level 0 dump: Thu Dec 31 17:11:40 1987
DUMP: Dumping /dev/dsk/ips0d0s7 to /dev/disk07 on host diamond
DUMP: mapping (Pass I) [regular files]
DUMP: mapping (Pass II) [directories]
DUMP: estimated 26 tape blocks on 0.00 tape(s).
DUMP: dumping (Pass III) [directories]
DUMP: dumping (Pass IV) [regular files]
DUMP: DUMP: 18 tape blocks on 1 tape(s)
DUMP: DUMP IS DONE
DUMP: Tape rewinding
#
```

(Note that the example uses the **dump.ffs** version of the **dump** command to specify the type of filesystem. This will not be necessary on later versions of UMIPS.)

What Can Go Wrong With a File System

Most of the things that can corrupt a file system have to do with the failure of the correct pointer and count information to make it out to the storage medium. This can be caused by

- hardware failure
- program interrupts
- human error

or a combination of hardware/program failures and incorrect procedures.

Hardware Failure

There is no very effective way of predicting when hardware failure will occur. The best way of dealing with it is to be sure that recommended diagnostic and maintenance procedures are followed conscientiously. For information regarding proper care of your M-Series system, refer to the M-Series Technical Manual.

Program Interrupts

It is possible that errors that cause a program to fail might result in the loss of some data. It is not easy to generalize about this because the range of possibilities is so large. Perhaps the best thing to be said is that programs should be exhaustively tested before they are put into production with valuable data.

Human Error

While it may be painful to admit it, probably the greatest cause of file system corruption falls under this heading. We are going to recommend three rules that should be followed by anyone who manages file systems.

1. ALWAYS check a file system before mounting it. Nothing complicates the problem of cleaning up a corrupt file system so much as allowing it to be used when it is bad.
2. NEVER remove a file system physically without first unmounting it.
3. ALWAYS use the **sync** command before shutting the system down and before unmounting a file system.
4. Protect your users with regular backups. Accidents can, and do, happen! Let your users know your backup policies.

The random nature of all these mishaps simply underscores the importance of establishing and observing good backup practices. It is the most effective form of insurance against data loss.

How to Check a File System for Consistency

When the UMIPS operating system is brought up, a consistency check of the file systems should always be done. On the M-Series system this check is automatically done as part of the power-up process. Included as part of that process is the command **fsstat**(1M). **fsstat** returns a code for each file system on the hard disk indicating whether the consistency checking and repair program, **fsck**(1M), should be run.

These same commands should be used to check file systems not mounted routinely as part of the power-up process. If inconsistencies are discovered, corrective action must be taken before the file systems are mounted. The remainder of this section is designed to acquaint you with the **fsck** utility.

It should be said at the outset that file system corruption, while serious, is not all that common. Most of the time a check of the file systems finds everything all right. The reason we put so much emphasis on file system checking is that if errors are allowed to go undetected, the ultimate loss can be substantial. After a system crash, however, file system corruption is the norm. On reboot, the system will automatically perform an fsck and reboot the repaired file system unless this is manually overridden. Using a file system that has not been fsck'ed and has damage can cause extensive files system damage.

The fsck Utility

The file system check (**fsck**) utility is an interactive file system check and repair program. **fsck** uses the information carried in the file system itself to perform consistency checks. If an inconsistency is detected, a message describing the inconsistency is displayed. You may elect to have **fsck** either make the repair or not. The reason you might choose to have **fsck** ignore an inconsistency is that you judge the problem to be so severe that you want either to fix it yourself using the **fsdb**(1M) utility, or you plan to go back to an earlier version of the file system. The decision to have **fsck** ignore inconsistencies and then do nothing about them yourself is not a viable one. File system inconsistencies do not repair themselves. If they are ignored, they only get worse.

The fsck Command

The **fsck** command is used to check and repair inconsistencies in a file system. With the exception of the **root** file system, a file system should be unmounted while it is being checked. The **root** file system should be checked only when the computer is in run level S and no other activity is taking place in the machine.

The following is the general format of the **fsck** command:

fsck [**−y**][**−n**][**−b** *block#*][*filesystem*]

The options of the **fsck** command are as follows:

−y Specifies a "yes" response for all questions. This is the normal choice when the command is being run as part of a shell procedure. It generally causes **fsck** to correct all errors. The "-y" option is not recommended when running **fsck** manually.

-n Specifies a "no" response for all questions. **fsck** will not write the file system.

-b *block#* Use the block number as the superblock for the file system. When you create a file system with **newfs** (or **mkfs**) the numbers of alternate superblocks are echoed to the screen. If you have not recorded these, you can always try block 32, which is always a backup superblock.

filesystem The file system to be checked. If no file system is supplied, the list of file systems in /etc/fstab is used.

Sample Command Use

The command line below shows **fsck** being entered to check the **/usr** file system. No options are specified. The system response means that no inconsistencies were detected. The command operates in phases, some of which are run only if required or in response to a command line option. As each phase is completed, a message is displayed. At the end of the program a summary message is displayed showing the number of files (i-nodes), blocks, and free blocks.

```
# fsck /dev/dsk/ips0d0s6
** /dev/usr
** Last Mounted on
** Phase 1 - Check Blocks and Sizes
** Phase 2 - Check Pathnames
** Phase 3 - Check Connectivity
** Phase 4 - Check Reference Counts
** Phase 5 - Check Cyl groups
4627 files, 103421 used, 84944 free (208 frags, \
                10592 blocks, 0.1% fragmentation)

***** FILE SYSTEM WAS MODIFIED *****

#
```

File System Components Checked by fsck

The following section reviews the components of a UMIPS file system and describes the kinds of consistency checks that are applied to them.

Super-Block

Summary information associated with the file system's superblock is the most likely part of a filesystem to become corrupted. The cause of this is usually shutting down the machine or unmounting a file system before it has been **sync**ed. **fsck** checks the superblock information for file system size, number of inodes, free block count, and free inode count. If problems result from the check of the superblock, **fsck** prompts the operator for an alternate superblock number.

Free Blocks

fsck verifies the status of the file system's free blocks by checking that the free blocks are not claimed by any file and that the number of free blocks plus the number of claimed blocks equals the total number of blocks in the file system.

If the check indicates the cylinder group's block map is incorrect, **fsck** will rebuild the maps based on the information it has determined.

Inodes

Each inode is checked to determine if it is in the state known as "neither allocated nor unallocated". The only repair **fsck** can then make is to clear the inode. **fsck** also checks to see that the inode link count is accurate, that the number of data blocks the inode references is valid, and that the type of data referenced by the inode is consistent with the actual data.

Connectivity

If **fsck** finds files or directories that are not linked into the file system, it places them into the **lost+found** directory in that file system's root directory (i.e., the directory the file system is mounted on). The **lost+found** directory is created automatically when a new file system is made (see **newfs**(1M)and **mkfs**(1M)).

Chapter 6: Performance Management

Introduction

This chapter describes ways to monitor and enhance the performance of your M-Series system.

- General approach to performance management

  Finding and fixing performance problems

- Improving performance
  - Tuning the kernel for minimum overhead and tuning the disk subsystem for maximum throughput
  - Workload analysis and housekeeping techniques for reducing peak load
  - Estimating capacity
  - Samples of typical procedures
- Performance tools

  Description of the performance tools.

- Tunable parameters

  Extensive definition of the tunable parameters.

General Approach to Performance Management

Performance management is not an activity that may need your attention when you first set up your M-Series system. When you bring the computer up for the first time, the system is automatically set to a basic configuration that is satisfactory for most applications. This configuration, however, cannot take into account the usage patterns and the behavior of your particular applications. For this reason, the structure of the system allows you to reconfigure it to enhance the performance for your particular environment over that of the standard configuration.

It is very possible that you may never have to do any special fine tuning of your system, and that your only experience with reconfiguration is when you add new memory and peripherals.

Finding Problems

The system is automatically configured to its default configuration the first time it is booted. After the system has been running a day or so, you may receive signals from a variety of sources that the system needs tuning. In particular, you may see that the response time is frequently slow. Your job of performance management really begins then. To proceed, you will use the tools described later in this chapter (most notably the **sar** command) to pinpoint the problem.

Fixing Problems

At this point, you need to take some corrective action. Some of the major areas for action are:

- Modifying the tunable configuration parameters.

  This is usually referred to as tuning the kernel, since you are adjusting the essential control structures at the heart of the system (the kernel). Many of these parameters are described in detail later in this chapter.

  When you change any of these parameters you must then reconfigure the system, which is the way to recreate a new bootable version of the operating system that incorporates the new parameter definitions.

- Uninstalling optional packages not needed by your applications.

  This procedure makes disk and memory space available for user programs and can benefit performance.

- Improving disk utilization.

  In addition to the allocation of system memory space defined by the tunable parameters, you have some control over how your file systems are organized on the disk and policies to cache frequently used programs in memory.

- Defining best system usage patterns.

  Finally, you can establish the model for best system usage, such as encouraging users to run large non-interactive programs at night.

These areas are discussed in more detail in the remaining sections of the chapter.

Improving Performance

This section discusses various ways in which the overall performance of your system can be improved.

Modifying the Tunable Configuration Parameters

The setting of the core system tunable parameters is accomplished by editing the parameter entries in the **/usr/reconfig/master.d** kernel files. For full definitions of the individual tunable parameters, and suggestions about setting them, refer to the section "Tunable Parameters" near the end of this chapter. (See Figure 6-5 for the recommended initial values for the tunable parameters relative to a particular memory size.)

Generally, the default parameters for your configuration will result in acceptable performance. If, however, you are running an application that has special performance needs, you can use the tools described in the section "Performance Tools" below to measure system load and determine which parameters might be changed to improve performance.

See Procedure 6.1, Reconfiguring the System, for examples of editing the tunables and reconfiguring the system. Also, look at the end of this section for a sample of a typical reconfiguration.

Improving Disk Utilization

Disk input/output may cause a bottleneck in system performance. Two steps in tuning the disk subsystem for better utilization are:

- Setting text-bit (sticky-bit).
- Organizing the file systems to minimize disk activity.

Setting the Text-Bit (Sticky-Bit)

Setting the text-bit can reduce the disk traffic of a select group of commands. Text pages of sticky commands are kept resident in memory, even when the process terminates. Once loaded into memory, such pages will usually remain. One exception is if the pages are reclaimed by the paging daemon in a tight memory situation (that is, when the number of available pages falls below the low-water mark as defined by the tunable parameter GPGSLO). Finding sticky pages already in memory can reduce considerably the loading time for the text pages of a process.

On systems that usually run a light to medium workload and that are seldom in a tight-memory situation, setting the text-bit can cause a significant improvement in performance. On systems with limited memory or a heavy workload, however, the text-bit should not be used.

The average amount of free memory should be determined by setting the text-bit on some test commands with the **chmod**(1) command, and by using the **-r** option of **sar**(1) to determine the average FREEMEM count over a typical interval of system activity. If the **sar** report shows that it is safe to set the text-bit on for some commands, logical candidates would be frequently used, (preferably) small commands. If the average free memory for the interval is less than the high-water mark plus 100, then performance is not likely to be significantly improved and may be hurt by setting the text-bit.

Defining Best System Usage Patterns

After the kernel and the system activities are tuned, and the file systems organized, the next step for improving system performance is to perform some housekeeping activities and to check whether prime time load can be reduced. The person responsible for administering the system should check for:

- less important jobs interfering with more important jobs
- unnecessary activities being carried out
- scheduling selected jobs when the system is not so busy
- the efficiency of user-defined features, such as **.profile** and $PATH

The ps Command

The **ps**(1) command is used to obtain information about active processes. The command gives a "snapshot" picture of what is going on, which is useful when you are trying to identify what processes are loading the system. Things will probably change by the time the output appears; however, the entries that you should be interested in are TIME (minutes and seconds of CPU time used by processes) and STIME (time when process first started).

When you spot a "runaway" process (one that uses progressively more system resources over a period of time while you are monitoring it), you should check with the owner. It is possible that such a process should be stopped immediately via the **kill**(1) **−9** command. When you have a real runaway, it continues to eat up system resources until everything grinds to a halt.

When you spot processes that take a lot of time to execute you should consider using **cron**(1M) to execute the job during off-hours.

Samples of General Procedures

This section depicts typical approaches to performance management. It describes a general procedure for troubleshooting performance problems.

Sample Procedure for Investigating Performance Problems

Locating the source of the problem can require some careful detective work. Hence, what follows is not a canned procedure, but rather a sample of a typical approach, covering basic areas where problems usually surface, and suggesting some of the actions to take that will alleviate the problem. The most common symptom that a problem exists is consistently poor response time.

Check for Excess Swapping

The first thing to look at is swapping activity, since the swapping of pages is costly in both disk and CPU overhead. Get the **sar**(1) **−qw** report. Look at the percentage that the swap queue is occupied (%swpocc) for values greater than 5. Then look at the swap-out rate (swpot/s) for values greater than 1.00.

Check whether the "freemem" count (number of pages available to user programs), shown by **sar −r**, is consistently less than the value of the tunable parameter GPGSHI (high-water mark).

If any or all of these are happening frequently, increase your system memory.

Check for Disk Bottleneck

If the value of %wio (from the **sar −u** report above) is greater than 10%, or if the %busy for a disk drive (obtained by **sar −d**) is greater than 50%, then the system has a disk bottleneck. Some ways to alleviate a disk bottleneck are:

1. Organize the file system to minimize disk activity. If you have two disks, distribute the file systems for a more balanced load.
2. Consider adding more memory if the situation persists. Additional memory reduces swapping/paging traffic and allows an expanded buffer pool (reducing the number of user-level reads and writes that need to go out to disk).
3. Setting the text-bit on for frequently used files may help, too. See the earlier discussion of this subject under "Setting Text-Bit (Sticky-Bit)."
4. Consider adding an additional disk and balancing the most active file systems across the two disks.

Check for Potential Table Overflows

To check for potential table overflows, get the **sar −v** report. This report will let you know if overflows have occurred in the process, file, or inode tables. Overflows in these tables are avoided by increasing NPROC, NFILE, and NINODE kernel parameters.

Shift Workload to Off-Peak Hours

Examine **/usr/spool/crontab** to see if jobs are queued up for peak periods that might better be run at times when the system is idle. Use the **ps** command to determine what processes are heavily loading the system. Encourage users to run large, non-interactive commands (such as **nroff**(1) or **troff**(1)) at off-peak hours. You may also want to run such commands with a low priority by using the **nice**(1) command.

Performance Tools

Internal activity is measured by a number of counters contained in the UMIPS system kernel. Each time an operation is performed, an associated counter is incremented. **sar** and the other performance tools allow you to monitor the values of these counters. The functions monitored by **sar** are discussed in the following sections. The performance tools for the UMIPS system internal activities described in this section are:

timex	Reports both system-wide and per-process activity during the execution of a command or program.
vsar	Visual display of **sar** information. Output is displayed using the **curses** database.
sar	Samples cumulative activity counters internal to the UMIPS system and provides reports on various system-wide activities.
sa1 and sa2	Shell scripts used to sample, process and record system activity data.
sadc	system activity data collector samples and records system activity data. Produces profiles of disk access location and seek distance.

Examples for these tools are provided in the following sections. Command outputs are typical values observed for user workloads on the UMIPS operating system. Values you receive may be quite different from values in the examples, depending on your environment. When tuning your system, it is recommended that you use a benchmark or have the system under a normal load.

timex

The **timex** command times a command and reports the system activities that occurred during the time the command was executing. If no other programs are running, then **timex** can give you a good idea of which resources a specific command uses during its execution. System consumption can be collected for each application program and used for tuning the heavily loaded resources. For our example, the **date** command is used. Enter the following:

```
$ timex -s date
Wed Jul  6 16:10:07 PDT 1988

real         0.09
user         0.02
sys          0.05


dunkshot dunkshot 3_0 UMIPS mips     07/06/88

16:10:07     %usr     %sys     %sys     %wio    %idle
                     local   remote
16:10:07       23       77        0        0        0

16:10:07  device    %busy    avque    r+w/s   blks/s   avwait   avserv

16:10:07
16:10:07 rawch/s canch/s outch/s rcvin/s xmtin/s mdmin/s
```

```
16:10:07       0       0       0       0       0       0

16:10:07 scall/s sread/s swrit/s  fork/s  exec/s rchar/s wchar/s
16:10:07
   in          0       0       0              0.00       0       0
   out         0       0       0              0.00       0       0
   local     455      42       6    9.68    54.84  695887    1771

16:10:07 swpin/s bswin/s swpot/s bswot/s pswch/s
16:10:07    0.00     0.0    0.00     0.0      39

16:10:07  iget/s namei/s dirbk/s
16:10:07     174       0     277

16:10:07 runq-sz %runocc swpq-sz %swpocc
16:10:07

16:10:07 proc-sz ov inod-sz ov file-sz ov lock-sz    fhdr-sz
16:10:07  39/300  0 228/1500  0  66/1000  0   0/100       0/  0

16:10:07   msg/s  sema/s
16:10:07    0.00    0.00

16:10:07  vflt/s  pflt/s pgfil/s  rclm/s
16:10:07  158.06  174.19    0.00    0.00

16:10:07 freemem freeswp
16:10:07    3178   83840

16:10:07 serv/lo-hi   request   request    server    server
               0 - 0     %busy  avg lgth    %avail avg avail
16:10:07          0       0.0         0       0.0         0
```

While **date**, for its simplicity, was used for the preceding demonstration, it is not the best example since it is not a major user of system resources.

timex can be used in the following way:

$ **timex -s** *application program*

Your application program will operate normally. When you finish and exit, the **timex** result will be printed on your screen. This can be extremely interesting: you get a clear picture of system resources used by your program.

The vsar Command

The **vsar**(1) command provides a visual, more easily accessible display of the information otherwise provided by the **sar** command. The options available to **vsar** are the same as those supplied to **sar** described in detail below. Of particular interest with vsar are the "-S" option which displays all information in a single page display. When using other options that create outputs greater than a single screen display, the j, k, n, and p keys can be used for pagination. The letter q or an Interrupt will exit the program.

The sar Command

Throughout this section, **sar** options are described with an analysis of sample outputs of the options. **sar** can be used either to gather system activity data or to extract what has been collected in data files created by **sa1** and **sa2**. **sa1** and **sa2** are initiated by entries put in the **/usr/spool/cron/crontabs/sys** file.

sar −a

The **sar −a** option reports the use of file access operations. The UMIPS operating system routines reported are as follows:

iget/s Number of files located by i-node entry per second.

namei/s Number of file system path searches per second. **namei** calls **iget**, so **iget/s** is always larger than **namei/s**.

dirbk/s Number of directory block reads issued per second.

An example of **sar −a** output, with a 30-second sampling interval, follows:

```
dunkshot dunkshot 3_0 UMIPS mips    07/06/88

00:00:03  iget/s namei/s dirbk/s
01:00:02        2       0       7
02:00:02        1       0       3
03:00:01        1       0       2
    .
    .
    .
  (etc.)
    .
    .
    .
    .
14:40:00        2       0       5
15:00:01        3       0       6
15:20:00        4       0       9
15:40:00        4       0       9
16:00:01        2       0       4
16:20:00        4       0       7

Average        23       0      32
```

The larger the values reported, the more time the UMIPS kernel is spending to access user files. This indicates how heavily programs and applications are using the file system(s). The **−a** option is helpful for understanding how disk-dependent the application system is; it is not used for any specific tuning step.

sar −b

The **−b** option reports the following buffer activity.

bread/s Average number of physical blocks read into the system buffers from the disk (or other block devices) per second.

lread/s Average number of logical blocks read from system buffers per second.

%rcache Fraction of logical reads found in buffer cache (100% minus the ratio of **breads** to **lreads**).

bwrit/s Average number of physical blocks written from the system buffers to disk (or other block devices) per second.

lwrit/s Average number of logical blocks written to system buffers per second.

%wcache Fraction of logical writes found in buffer cache (100% minus the ratio of **bwrit/s** to **lwrit/s**).

pread/s Average number of physical (raw) read requests per second.

pwrit/s Average number of physical (raw) write requests per second.

The entries that you should be most interested in are the cache hit ratios **%rcache** and **%wcache** which measure the effectiveness of system buffering.

An example of **sar −b** output follows:

```
dunkshot dunkshot 3_0 UMIPS mips     07/06/88

00:00:03 bread/s lread/s %rcache  ... %wcache pread/s pwrit/s
01:00:02       0      12     100  ...      69       0       0
02:00:02       0       5      99  ...      64       0       0
03:00:01       0       0     100  ...      63       0       0
   .
   .
   .
(etc.)
Average        0      42      99  ...      85       0       0
```

sar −c

The **−c** option reports system calls in the following categories:

scall/s All types of system calls per second, generally about 30 per second on a busy 4 to 6 user system.

sread/s Read system calls per second.

swrit/s Write system calls per second.

fork/s Fork system calls per second, about 0.5 per second on a 4 to 6 user system. This number will increase if shell scripts are running.

exec/s Exec system calls per second. (If (**exec/s**) / (**fork/s**) is greater than 3, look for inefficient $PATHs.)

rchar/s Characters (bytes) transferred by read system calls per second.

wchar/s Characters (bytes) transferred by write system calls per second.

Typically, reads plus writes account for about half of the total system calls, although this varies greatly with the activities that are being performed by the system.

An example of **sar −c** output follows:

```
dunkshot dunkshot 3_0 UMIPS mips    07/06/88

00:00:03 scall/s sread/s swrit/s  fork/s  exec/s rchar/s wchar/s
01:00:02      20       9       0    0.30    0.28    2805      90
02:00:02      17       8       1    0.10    0.08    2757     197
15:20:00      45      15       5    0.18    0.35    3689     259
15:40:00      40      20       4    0.14    0.18    5345     923
    .
    .
    .
  (etc)
Average       41      13       5    0.22    0.24    6689    3362
```

sar −d

The **sar −d** option reports the activity of block devices.

device Name of the block device(s) that **sar** is monitoring.

%busy Percent of time the device was servicing a transfer request.

avque The average number of requests outstanding during the period of time (measured only when the queue is occupied).

r+w/s Number of read and write transfers to the device per second.

blks/s Number of 512 byte blocks transferred to the device per second.

avwait Average time in milliseconds that transfer requests wait idly in the queue (measured only when the queue is occupied).

avserv Average time in milli-seconds for a transfer request to be completed by the device (for disks this includes seek, rotational latency, and data transfer times).

An example of **sar −d** is as follows:

```
dunkshot dunkshot 3_0 UMIPS mips    07/06/88

00:00:03  device  %busy  avque  r+w/s  blks/s  avwait avserv

01:00:02 dkip-0      1    2.4      0       8    20.5    14.6

02:00:02 dkip-0      0    2.5      0       2     0.0    14.5

03:00:01 dkip-0      0    2.5      0       2     0.0    14.4
    .
    .
    .
  (etc.)
Average  dkip-0      1    3.3      0       7    34.4    14.7
         dkip-1      0    2.2      0       0     0.0    17.5
         dkip-2      0    1.6      0       2     0.0     8.7
         dkip-3      0    4.3      0       5     0.0    11.4
```

Note that queue lengths and wait times are measured while the queue had something on it. If **busy** is small, large queues and service times probably represent the periodic **sync** efforts by the system to ensure that altered blocks are written to the disk in a timely fashion.

sar −m

The **sar −m** option reports on inter-process communication activities. Message and semaphore calls are reported as follows:

msg/s Number of message operations (sends and receives) per second.

sema/s Number of semaphore operations per second.

An example of **sar −m** output follows:

```
dunkshot dunkshot 3_0 UMIPS mips     07/06/88

00:00:03   msg/s  sema/s
01:00:02     0.00    0.00
02:00:02     0.00    0.00
03:00:01     0.00    0.00
    .
    .
    .
  (etc.)
    .
    .
15:40:00     0.00    0.00
16:00:01     0.00    0.00
16:20:00     0.00    0.00

Average      0.00    0.00
```

These figures will usually be zero (0.00) unless you are running applications that use the message or semaphore features.

sar −q

The **sar −q** option reports the average queue length while the queue is occupied and percent of time occupied.

runq-sz Run queue of processes in memory; typically, this should be less than 2. Consistently higher values mean you are CPU-bound.

%runocc The percentage of time the run queue is occupied; the larger this value is, the better.

swpq-sz Swap queue of processes to be swapped out; the smaller this number is, the better.

%swpocc The percentage of time the swap queue is occupied; the smaller this value is, the better.

An example of **sar −q** follows:

```
dunkshot dunkshot 3_0 UMIPS mips     07/06/88
```

```
00:00:03 runq-sz %runocc swpq-sz %swpocc
01:00:02     1.2       3
02:00:02     1.6       1
03:00:01     1.4       1
04:00:01     1.5       1
   .
   .
   .
  (etc.)
   .
   .
   .
15:00:01     1.5       5
15:20:00     1.5       4
15:40:00     1.7       5
16:00:01     1.7       4
16:20:00     1.5       5

Average      1.4       7
```

If **%swpocc** is greater than 20, more memory would help reduce swapping/paging activity.

sar −u

The CPU utilization is listed by **sar −u** (default). At any given moment the processor will be either busy or idle. When busy, the processor will be in either user or system mode. When idle, the processor is either waiting for input/output completion or has no work to do. The **−u** option of **sar** lists the percent of time that the processor is in system mode (%sys), user mode (%user), waiting for input/output completion (%wio), and idle time (%idle).

In typical timesharing use, %sys and %usr are about the same value. In special applications, either of these may be larger than the other without anything being abnormal. A high %wio generally means a disk bottleneck. A high %idle, with degraded response time, may mean memory constraints; time spent waiting for memory is attributed to %idle.

An example of **sar −u** follows:

```
dunkshot dunkshot 3_0 UMIPS mips    07/06/88

00:00:03 swpin/s bswin/s swpot/s bswot/s pswch/s
01:00:02    0.00     0.0    0.00     0.0       3
02:00:02    0.00     0.0    0.00     0.0       2
03:00:01    0.00     0.0    0.00     0.0       2
04:00:01    0.00     0.0    0.00     0.0       2
   .
   .
   .
  (etc.)
   .
   .
   .
16:20:00    0.00     0.0    0.00     0.0       5
```

```
Average      0.00      0.0      0.00      0.0        6
```

sar –v

The **–v** option reports the status of process, i-node, file, shared memory record, and shared memory file tables. From this report you know when the system tables need to be modified.

proc-sz Number of process table entries presently being-used/allocated in the kernel.

inod-sz Number of i-node table entries presently being-used/allocated in the kernel.

file-sz Number of file table entries presently being-used/allocated in the kernel.

ov Number of times an overflow occurred. (One column for each of the above three items.)

lock-sz The number of shared memory record table entries presently being-used/allocated in the kernel.

fhdr-sz No longer applicable.

The values are given as *level/table size*. An example of **sar –v** follows:

```
dunkshot dunkshot 3_0 UMIPS mips     07/06/88

00:00:03 proc-sz ov inod-sz  ov file-sz  ov lock-sz fhdr-sz
01:00:02  25/300  0 207/1500  0  49/1000  0   0/100    0/  0
02:00:02  25/300  0 207/1500  0  49/1000  0   0/100    0/  0
03:00:01  25/300  0 207/1500  0  49/1000  0   0/100    0/  0
   .
   .
   .
  (etc.)
   .
   .
   .
14:40:00  38/300  0 230/1500  0  65/1000  0   0/100    0/  0
15:00:01  42/300  0 234/1500  0  68/1000  0   0/100    0/  0
15:20:00  39/300  0 232/1500  0  66/1000  0   0/100    0/  0
15:40:00  36/300  0 228/1500  0  64/1000  0   0/100    0/  0
16:00:01  36/300  0 228/1500  0  64/1000  0   0/100    0/  0
16:20:00  36/300  0 228/1500  0  64/1000  0   0/100    0/  0
```

This example shows that all tables are large enough to have no overflows. Sizes could be reduced to save main memory space if these are the highest values ever recorded.

sar −w

The **−w** option reports swapping and switching activity. The following are some target values and observations.

swpin/s Number of transfers into memory per second.

bswin/s Number of 512-byte-block units (blocks) transferred for swap-ins (including initial loading of some programs) per second.

swpot/s Number of transfers from memory to the disk swap area per second. If greater than 1, memory may need to be increased or buffers decreased.

bswot/s Number of blocks transferred for swap-outs per second.

pswch/s Process switches per second. This should be 30 to 50 on a busy 4 to 6 user system.

An example of **sar −w** output follows:

```
dunkshot dunkshot 3_0 UMIPS mips     07/06/88

00:00:03 swpin/s bswin/s swpot/s bswot/s pswch/s
01:00:02    0.00     0.0    0.00     0.0       3
02:00:02    0.00     0.0    0.00     0.0       2
03:00:01    0.00     0.0    0.00     0.0       2
04:00:01    0.00     0.0    0.00     0.0       2
05:00:01    0.00     0.0    0.00     0.0       2
   .
   .
   .
  (etc.)
   .
   .
   .
15:20:00    0.00     0.0    0.00     0.0       6
15:40:00    0.00     0.0    0.00     0.0       6
16:00:01    0.00     0.0    0.00     0.0       6
16:20:00    0.00     0.0    0.00     0.0       5

Average     0.00     0.0    0.00     0.0       6
```

This example shows that there is sufficient memory for the currently active users, since no swapping is occurring.

sar −p

The **−p** option reports paging activity. The following page rates are recorded.

vflt/s Number of address translation page faults per second (valid page not present in memory).

pflt/s Number of page faults from protection errors per second (illegal access to page) or "copy-on-writes". **pflt/s** generally consists entirely of "copy-on-writes."

pgfil/s Number of **vflt/s** per second satisfied by a page-in from the file system.

rclm/s Number of valid pages per second that the system has reclaimed (added to list of free pages).

An example of **sar −p** output follows:

```
dunkshot dunkshot 3_0 UMIPS mips    07/06/88

00:00:03  vflt/s  pflt/s pgfil/s  rclm/s
01:00:02    3.18    5.14    0.02    0.00
02:00:02    1.03    1.69    0.04    0.00
03:00:01    0.85    1.50    0.00    0.00
    .
    .
    .
  (etc.)
    .
    .
    .
15:20:00    2.35    3.71    0.04    0.00
15:40:00    1.87    2.77    0.00    0.00
16:00:01    1.01    1.66    0.00    0.00
16:20:00    1.40    2.37    0.03    0.00

Average     2.29    4.11    0.07    0.00
```

sar −r

The **−r** option records the number of memory pages and swap file disk blocks that are currently unused. The following are recorded.

freemem Average number of 4K pages of memory available to user processes over the intervals sampled by the command.

freeswap Number of 512-byte disk blocks available for process swapping.

An example of **sar −r** output follows:

```
dunkshot dunkshot 3_0 UMIPS mips    07/06/88

00:00:03 freemem freeswp
01:00:02    3715   83840
02:00:02    3716   83840
03:00:01    3707   83840
    .
    .
    .
  (etc.)
    .
    .
    .
```

```
14:40:00     3159    83840
15:00:01     3209    83840
15:20:00     3207    83840
15:40:00     3222    83840
16:00:01     3257    83840
16:20:00     3244    83840
16:40:00     3240    83840

Average      3535   83840
```

sar −y

The **−y** option monitors terminal device activities. If you have a lot of terminal I/O, you can use this report to determine if there are any bad lines. Activities recorded are defined as follows:

rawch/s	Input characters (raw queue) per second.
canch/s	Input characters processed by canon (canonical queue) per second.
outch/s	Output characters (output queue) per second.
rcvin/s	Receiver hardware interrupts per second.
xmtin/s	Transmitter hardware interrupts per second.
mdmin/s	Modem interrupts per second.

The number of modem interrupts per second (**mdmin/s**) should be close to 0, and the receive and transmit interrupts per second (**xmtin/s** and **rcvin/s**) should be less than or equal to the number of incoming or outgoing characters, respectively. If this is not the case, check for bad lines.

An example of **sar −y** output follows:

```
dunkshot dunkshot 3_0 UMIPS mips     07/06/88

00:00:03 rawch/s canch/s outch/s rcvin/s xmtin/s mdmin/s
01:00:02       0       0       0       0       0       0
02:00:02       0       0       0       0       0       0
03:00:01       0       0       0       0       0       0
    .
    .
    .
  (etc.)
    .
    .
    .
14:40:00       0       0       0       0       0       0
15:00:01       0       0       0       0       0       0
15:20:00       0       0       0       0       0       0
15:40:00       0       0       0       0       0       0
16:00:01       0       0       0       0       0       0
16:20:00       0       0       0       0       0       0
16:40:00       0       0       0       0       0       0

Average        0       0       0       0       0       0
```

sar −A

The **sar −A** option is equivalent to **sar −udqbwcayvmprS**. The **−A** option provides a view of overall system performance. Use it to get a more global perspective. If data from more than one time slice is shown, the report includes averages.

Tunable Parameters

Tunable system parameters are used to set various table sizes and system thresholds to handle the expected system load. Caution should be used when changing these variables since such changes can directly affect system performance. For the most part, the initial tunable parameter values for a new M-Series system are acceptable for most configurations and applications. If your application has special performance needs, you may have to experiment with different combinations of parameter values to find an optimal set.

Table 6-1 shows the recommended tunable parameter values for a Release 3.0 system. The parameters shown in the table are defined in the appropriate **/usr/reconfig/master.d/kernel.r***XXXX***-std** file (where XXXX is the CPU designation for your system such as 2300, 2400 or 3200). The meaning of each parameter is described below.

Kernel Parameters

The following parameters are defined in the **/usr/reconfig/master.d** kernel file.

NCALL Specifies how many call-out table entries to allocate. Each entry represents a function to be invoked at a later time by the clock handler portion of the kernel. The default value is 700. If more processes are allowed, or if drivers that use timeouts are added, this value may need to be increased.

Software drivers may also use call entries to check hardware device status. When the call-out table overflows, the system crashes and outputs the following message on the system console:

```
PANIC: Timeout table overflow
```

NINODE Specifies how many i-node table entries to allocate. Each table entry represents an in-core i-node that is an active file. For example, an active file might be a current directory, an open file, or a mount point. The file control structure is modified when changing this variable. The number of entries used depends on the number of opened files. The default value is set at 1500. The value for NINODE pertains directly to the NFILE value. (NINODE is equal to or greater than NFILE). When the i–node table overflows, the following is output on the system console:

```
WARNING: i–node table overflow
```

NS5INODE Not required unless S51K file system is used (not recommended).

NFILE Specifies how many open file table entries to allocate. The default is 1000. Each entry contains 12 bytes. The NFILE entry relates directly to the NINODE entry. (NFILE is less than or equal to NINODE). The NFILE control structure operates in the same manner as the NINODE structure. When the file table overflows, the following warning message is output on the system console.

```
NOTICE: file table overflow
```

As a reminder, this parameter does not affect the number of open

files per process (see the NOFILES parameter).

NMOUNT Specifies how many mount table entries to allocate. Each entry represents a mounted file system. The root (/) file system is always the first entry. When full, the **mount**(2) system call returns the error EBUSY. Since the mount table is searched linearly, this value should be as low as possible.

NPROC Specifies how many process table entries to allocate. Each table entry represents an active process. The swapper is always the first entry and **/etc/init** is always the second entry. The number of entries depends on the number of terminal lines available and the number of processes spawned by each user. The average number of processes per user is in the range of 2 to 5 (also see MAXUP). When full, the **fork**(2) system call returns the error EAGAIN.

NREGION Specifies how many region table entries to allocate. Most processes have 3 regions: text, data, and stack. Additional regions are needed for each shared memory segment and shared library (text and data) attached. However, the region table entry for the text of a "shared text" program will be shared by all processes executing that program. Each shared memory segment attached to one or more processes uses another region table entry. A good starting value for this parameter is about 3.5 times NPROC. If the system runs out of region table entries, the following message is output on the system console.

```
Region table overflow
```

NCLIST Specifies how many character list buffers to allocate. Since this system is a streams-based tty IO system, the character list buffers are not used.

MAXUP Specifies how many concurrent processes a non-superuser is allowed to run. This value should not exceed the value of NPROC (NPROC should be at least 10% more than MAXUP). This value is per user identification number, not per terminal.

NOFILES Specifies the maximum number of open files per process. Default is 100. Values higher than 100 are accessible only to processes using system calls (**open**(2), **creat**(2); for example). Processes using standard I/O subroutines are limited to 100, independent of the value of NOFILES. Unless an application package recommends that NOFILES be changed, the default setting of 40 should be left as is.

NHBUF Specifies how many "hash buckets" to allocate. These are used to search for a buffer given a device number and block number rather than a linear search through the entire list of buffers. **This value must be a power of 2.** Each entry contains 12 bytes. The NHBUF value must be chosen so that the value NBUF divided by NHBUF is approximately equal to 4. While the value of NBUF is normally calculated by the system, that is not true for NHBUF. NHBUF must be specified in the **/usr/reconfig/master.d** kernel file.

NPBUF Specifies how many physical input/output buffers to allocate. One input/output buffer is needed for each physical read or write active. The default value is 64.

NAUTOUP The NAUTOUP entry specifies the buffer age in seconds for automatic file system updates. A system buffer is written to the hard disk when it has been memory-resident for the interval specified by the NAUTOUP parameter. Specifying a smaller limit increases system reliability by writing the buffers to disk more frequently and decreases system performance. Specifying a larger limit increases system performance at the expense of reliability.

BDFLUSHR Specifies the rate in seconds for checking the need to write the file system buffers to disk. The default is 1 second.

MAXPMEM Specifies the maximum amount of physical memory to use in pages. The default value of 0 specifies that all available physical memory be used.

SHLBMAX Specifies the maximum number of shared libraries that can be attached to a process at one time.

FLCKREC Specifies the number of records that can be locked by the system. The default value is 100.

PUTBUFSZ Specifies the size of a circular buffer, **putbuf**, that is used to contain a copy of the last PUTBUFSZ characters written to the console by the operating system. The contents of **putbuf** can be viewed using **crash**(1M).

MAXSLICE Specifies in clock ticks the maximum time slice for user processes. After a process executes for its allocated time slice, that process is suspended. The operating system then dispatches the highest priority process and allocates to it MAXSLICE clock ticks. MAXSLICE, defined in **/usr/reconfig/master.d/disp**, is normally one tenth of a second.

Paging Parameters

There exists in the system a paging daemon, **vhand**, whose sole responsibility is to free up memory as the need arises. It uses a "least recently used" algorithm to approximate process working sets, and it writes those pages out to disk that have not been touched during some period of time. The page size is 4K bytes. When memory is exceptionally tight, the working sets of entire processes may be swapped out.

The following tunable parameters determine how often **vhand** runs and under what conditions. The default values in **/etc/master.d/kernel** should be adequate for most applications.

VHNDFRC and **VHANDL**
Used to determine the initial value for the system variable VHANDL. VHANDL is set to the maximum user-available memory divided by VHNDFRC or the value of GPGSHI, whichever is larger. The value of VHANDL determines when the paging daemon **vhand** runs. The amount of available free memory is compared with the value of VHANDL every VHANDR seconds. If free memory is less than VHANDL, then the paging daemon

vhand is awakened.

The default for VHNDFRC is 8. Decrease the value to make the daemon more active; increase the value to make the daemon less active (must be > 0 and < 25 percent of available memory).

VHANDR Specifies in seconds the maximum rate at which **vhand** can run. **vhand** will only run at this rate if free memory is less than VHANDL, as explained above for VHNDFRC. The default is 1. Increase the value to make the daemon less active (must be an integer > 0 and ≤ 300). If you have set the value higher, decreasing it makes the daemon more active.

GPGSLO Specifies the low water mark of free memory in pages for **vhand** to start stealing pages from processes. The default is (4*MAXFC). Increase the value to make the daemon more active; decrease the value to make the daemon less active (must be an integer ≥ 0 and < GPGSHI).

GPGSHI Specifies the high water mark of free memory in pages for **vhand** to stop stealing pages from processes. The default is (GPGSLO+3*MAXFC). Increase the value to make the daemon more active; decrease the value to make the daemon less active (The value must be an integer > 0, > GPGSLO and < 25 percent of the number of pages of available memory).

MAXSC Specifies the maximum number of pages which will be swapped out in a single operation. The default value is 32.

MAXFC Specifies the maximum number of pages that will be added to the freelist in a single operation. The default value is 32.

MAXUMEM Specifies the maximum size of a user's virtual address space in pages. The default is 8192.

MINARMEM Specifies the minimum number of memory pages reserved for the text and data segments of user processes. The default is 25.

MINASMEM Threshold value that specifies the number of memory and swap pages reserved for system purposes (unavailable for the text and data segments of user processes). The default is 25.

Streams Parameters

The following tunable parameters are associated with Streams processing. These parameters are defined in the **/usr/reconfig/master.d** kernel file.

NQUEUE The number of STREAMS queues to be configured. Queues are always allocated in pairs, so this number should be even. A minimal Stream contains four queues (two for the Stream head, two for the driver). Each module pushed on a Stream requires an additional two queues. The default is 150.

NSTREAM The number of "Stream-head" (stdata) structures to be configured. One is needed for each Stream opened, including both Streams currently open from user processes and Streams linked under multiplexers. The recommended configuration value is highly application-dependent. The default is 50.

NSTRPUSH The maximum number of modules that may be pushed onto a Stream. This is used to prevent an errant user process from consuming all of the available queues on a single Stream. By default this value is 9, but in practice, existing applications have pushed at most four modules on a Stream.

NSTREVENT The initial number of Stream event cells to be configured. Stream event cells are used for recording process-specific information in the **poll**(2) system call. They are also used in the implementation of the STREAMS I_SETSIG **ioctl** and in the kernel **bufcall**() mechanism. A rough minimum value to configure would be the expected number of processes to be simultaneously using **poll**(2) times the expected number of Streams being polled per process, plus the expected number of processes expected to be using STREAMS concurrently. The default is 25. Note that this number is not necessarily a hard upper limit on the number of event cells that will be available on the system (see MAXSEPGCNT).

MAXSEPGCNT The number of additional pages of memory that can be dynamically allocated for event cells. If this value is 0, only the allocation defined by NSTREVENT is available for use. If the value is not 0 and if the kernel runs out of event cells, it will under some circumstances attempt to allocate another batch of event cells. Currently, event cells are allocated in batches of 20. MAXSEPGCNT places a limit on the number of new batches that can be allocated in this way.

NMUXLINK The maximum number of multiplexer links to be configured. One link structure is required for each active multiplexor link (STREAMS I_LINK **ioctl**). The default is 1.

STRMSGSZ The maximum allowable size of the data portion of any STREAMS message. This should usually be set just large enough to accommodate the maximum packet size restrictions of the configured STREAMS modules. If it is larger than necessary, a single **write**(2) or **putmsg**(2) can consume an inordinate number of message blocks. The recommend value of 4096 is sufficient for existing applications.

STRCTLSZ The maximum allowable size of the control portion of any STREAMS message. The control portion of a **putmsg**(2) message is not subject to the constraints of the min/max packet size, so the value entered here is the only way of providing a limit for the control part of a message. The recommended value of 1024 is more than sufficient for existing applications.

NBLK*n* The number of STREAMS data blocks and buffers to be allocated for each size class. Message block headers are also allocated based on these numbers: the number of message blocks is 1.25 times the total of all data block allocations. This provides a message block for each data block, plus some extras for duplicating messages (kernel functions **dupb**(), **dupmsg**()). The optimal configuration depends on both the amount of primary memory available and the intended application.

STRLOFRAC The percentage of data blocks of a given class at which low-priority block allocation requests are automatically failed. For example, if STRLOFRAC is 80 and there are 48 256-byte blocks, a low-priority allocation request will fail when more than 38 256-byte blocks are already allocated. The parameter is used to help prevent deadlock situations by starving out low-priority activity. The recommended value of 80 works well for current applications. STRLOFRAC must always be in the range 0 <= STRLOFRAC <= STRMEDFRAC.

STRMEDFRAC

The percentage cutoff at which medium priority block allocations are failed (see STRLOFRAC discussion above). The recommended value of 90 works well for current applications. STRMEDFRAC must always be in the range STRLOFRAC <= STRMEDFRAC <= 100.

There is no cutoff fraction for high-priority allocation requests; it is effectively 100.

Buffer Cache Size

It is not necessary to reconfigure the UMIPS kernel to change the buffer cache size. (In other systems this is done by modifying the NBUFS parameter.)

The system determines the size of physical memory at boot time and adjusts the size of the buffer cache accordingly. In addition, the size of the buffer cache is adjusted dynamically while the system is running, based on the memory requirements of user processes as well as kernel needs. The size of the buffer cache can vary from 10% to 50% of physical memory.

Chapter 7: LP Spooler Administration

Introduction

This chapter tells you about

- How the LP Spooling system works
- How to find instructions for installing the LP Spooler
- The commands used to administer the system
- Printer interface programs
- LP Spooler files and directories

Error messages issued by the LP Spooler are listed in Appendix C.

How the LP Spooling System Works

The term **spool** is an acronym for "simultaneous peripheral output on-line." LP spooling means that you can send a file to be printed (LP originally stood for Line Printer, but has come to include many other types of printing devices), while you continue with other work. The LP Spooling system is software that

- handles the task of receiving files users want printed
- schedules the work of one or more printers
- starts programs that interface with the printer(s)
- keeps track of the status of jobs
- issues error messages when problems arise

The LP Spooler has five user commands. These are shown in Figure 7-1.

Command	Description
enable(1)	Activate the named printer(s).
cancel(1)	Cancel a request for a file to be printed.
disable(1)	Deactivate the named printer(s).
lp(1)	Send a file or files to a printer.
lpstat(1)	Print the status of the LP system.

Figure 7-1: User Commands for the LP Spooling System

In addition to being able to send requests to the LP Spooling system, check the status of requests, and cancel requests, users are given the ability to disable and enable a printer. The idea is that if a user finds a printer is malfunctioning in some way, it should not be necessary to call the administrator to turn the printer off.

Administrative Commands

A separate set of commands available for the LP administrator is shown in Figure 7-2. These commands are found in the **/usr/lib** directory. If you expect to use them frequently, you might find it convenient to include that directory in your PATH variable. To use the administrative commands, you must be logged in either as **root** or as **lp**. **lp** is a system login (see Chapter 1, System Security, or Procedure 1.4 for a description of how to set up a password for a system login).

Command	Description
/usr/lib/accept(1M)	Permit job requests to be queued for a specified destination.
/usr/lib/reject(1M)	Prevent jobs from being queued for a specified destination. Described on the same manual page as **accept**(1M).
/usr/lib/lpadmin(1M)	Set up or change the LP configuration.
/usr/lib/lpmove(1M)	Move output requests from one destination to another. Described on the same manual page as **lpsched**(1M).
/usr/lib/lpsched(1M)	Start the LP scheduler.
/usr/lib/lpshut(1M)	Stop the LP scheduler. Described on the same manual page as **lpsched**(1M).

Figure 7-2: Administrative Commands for the LP Spooling System

In Figure 7-2 the administrative commands are listed in the order in which they occur in the *System Administrator's Reference Manual*. In the section that follows, we describe the commands in the order in which they are usually used.

Command Descriptions and Examples

/usr/lib/lpadmin

The **lpadmin**(1M) command is used to add a new printer to the system, assign classes of printers, name or remove a default destination, and specify interface programs to be used. **lpadmin** may not be used when the LP scheduler, **lpsched**(1M), is running, except when the **−d** option is specified.

One (and only one) of the following three options must always be included on the command line when you execute **lpadmin**:

−d[*dest*]

−x*dest*

−p*printer*

No other options are allowed with **−d** and **−x**. However, many arguments are allowed with the **−p***printer* option, and at least one argument must always be present. The **−p** option names a printer to which the other argument(s) applies. If *printer* does not exist, it is created. The arguments that can be used with **−p** are as follows:

−c*class*	This argument assigns the printer specified in the **−p** option to the specified *class*.
−e*printer*	This argument allows you to use an existing interface program for a new printer that you are adding to the LP system. When you select this argument, the interface program for the printer specified in this argument is copied for the printer specified in the **−p** option.
−h	When adding a new printer, this argument means that the printer is hard-wired to the M series system.
−i*interface*	Use this argument to specify a new interface program for the printer specified in the **−p** option. *interface* is the path name of the new program.
−l	When adding a new printer, this argument means that the device associated with the printer is a login terminal.
−m*model*	Several "model" interface programs are supplied with the LP Spooling Utilities. These model interface programs support some of the common printers that may be used with the M series system. Use this argument to select the model interface program that you want to use with the printer you are adding to the LP system.
−r*class*	Use this argument to remove a printer from a class.
−v*device*	This argument must be used when you add a new printer to the LP system. It associates the printer with the UNIX system file specified by *device*. The complete path name must be given for the file.

The **−d**[*dest*] option is used to define an existing system destination as the new system default destination. If *dest* is not specified, there is no default destination. The LP scheduler may be running when you use this option. The default destination is used to determine where a file named in a user's **lp** command is sent (assuming the user does not specify a destination on the command line and the environment variable LPDEST is null). The destination (*dest*) must already exist.

To remove a destination (*dest*), the **−x***dest* option is used with **lpadmin**. This option cannot be invoked when the scheduler is running. If it is running, you must issue the **lpshut** command before **lpadmin**.

Command Examples

NOTE

In examples 2 through 7, it is assumed that the LP scheduler has already been stopped. This is done through the **lpshut**(1M) command. Example 1 does not require the scheduler to be stopped since only the **−d** option to **lpadmin** is used.

Example 1

Make printer lp0 the system default destination.

```
# lpadmin −dlp0
#
```

Example 2

Add a new printer called printer2 and associate it with device **/dev/tty11**. Use the pprx model interface program.

```
# lpadmin −pprinter2 −v/dev/tty11 −mpprx
#
```

When you add a new printer, it is left in a disabled state and does not accept requests.

Example 3

Create a hardwired printer called lp1 on device **/dev/tty13**. Add lp1 to a new class called cl1, and use the same interface program that is used with printer printer3.

```
# lpadmin −plp1 −v/dev/tty13 −eprinter3 −ccl1
#
```

Example 4

Change the interface program for printer lp1 to model interface program pprx.

```
# lpadmin −plp1 −mpprx
#
```

Example 5

Add printer printer2 to class cl1:

```
# lpadmin −pprinter2 −ccl1
#
```

Printers that are added to a class are ordered according to the sequence in which they are added. For example, assume that class cl1, in the example above, already had printers lp1 and lp2 as members. After adding printer printer2, the order of the printers would be lp1, lp2, and printer2. If all three printers are available, and a request is routed to class cl1, the request will be serviced by lp1. If all three printers are busy, the request will be serviced by the first available printer.

Example 6

Remove printers lp1 and lp2 from class cl1:

```
# lpadmin −plp1 −rcl1
# lpadmin −plp2 −rcl1
#
```

Example 7

No destination (class or printer) may be removed if it has pending requests. The pending requests must either be cancelled using the **cancel**(1) command or moved to other destinations using the **lpmove**(1M) command before the destination can be removed.

Removing the last remaining member of a class causes the class to be deleted. If the destination removed is the system default destination, the system will no longer have a default destination. However, the removal of a class does not imply the removal of printers that were assigned to that class.

```
# lpadmin -xlp3
#
```

/usr/lib/lpsched

The **lpsched**(1M) command starts the LP scheduler. The LP scheduler takes the top job request off the queue and "hands" it to the appropriate interface program to be printed on a printer. The LP scheduler keeps track of the job progress, and as soon as the job is completed, it takes the next job request off the queue and repeats the process. As long as the LP scheduler is running, jobs requested by **lp** will be printed. If the scheduler is not running, jobs will not be printed.

Every time the scheduler is started, **lpsched** creates a file called SCHEDLOCK in the **/usr/spool/lp** directory. As long as the SCHEDLOCK file is present, the system will not allow another scheduler to run. When the scheduler is stopped under normal conditions, either with **lpshut**(1M) or as part of the normal shutdown procedure, the SCHEDLOCK file is removed. However, if the system comes down abnormally, there is a possibility that the SCHEDLOCK file may not get removed. To ensure that the SCHEDLOCK file does not exist, **/etc/rc.d/lp** contains a command line to remove SCHEDLOCK first before it attempts to start the scheduler.

The command is entered without arguments.

```
# lpsched
#
```

Notice in the example that the command shows no response to let you know that the scheduler is running. To verify that the scheduler is running, use the **lpstat**(1M) command with the **-r** option.

```
# lpstat -r
scheduler is running
#
```

If many job requests are queued, there may be a delay before the **lpstat** command reports that the scheduler is running.

/usr/lib/lpshut

Two of the three **lpadmin** command options (**-x** and **-p**) cannot be executed unless the LP scheduler is stopped. The **lpshut** command stops the LP scheduler and terminates all printing activity. All requests that were in the middle of printing will be reprinted in their entirety when the scheduler is restarted. The command is entered without arguments.

```
# lpshut
scheduler stopped
#
```

/usr/lib/lpmove

Occasionally, you may find it necessary to move output requests from one destination to another. For example, if you have a printer that was removed for repairs, you will want to move all the pending job requests to a destination with a working printer. This is done using the **lpmove** command. Be aware that job requests routed to a destination without a printer are automatically rejected.

Another use of the **lpmove** command is to move specific requests from one destination to another. When this is done, **lp** will no longer accept requests for the original destination (this is the same effect as a **reject** command). **lpmove** refuses, however, to move requests while the LP scheduler is running. The general format of the **lpmove** command is as follows:

lpmove *requests dest*

requests are the request identification numbers (request IDs) of jobs waiting to be printed, and *dest* is the destination to which the requests are to be moved. The destination can be a printer or a class of printers.

Command Examples

Example 1

Move all the requests for printer lp1 to printer lp2. Moving the requests renames the request IDs from lp1-nnn to lp2-nnn. After the requests are moved, **lp** will no longer accept requests for lp1 (this is the same effect as a **reject lp1** command issued after the **lpmove**).

```
# lpmove lp1 lp2
#
```

Example 2

Move requests lp1-54 and lp2-55 to printer lp0:

```
# lpmove lp1-54 lp2-55 lp0
total of 2 requests moved to lp0
#
```

NOTE

The two requests are now renamed lp0–54 and lp0–55.

/usr/lib/accept

The **accept**(1M) command allows job requests to be placed in a queue at the named destination(s), destination being the name of a printer or class of printers. The general format of the **accept** command is as follows:

accept *destination(s)*

Command Example

The sample command line allows printer lp0 to start receiving requests.

```
# /usr/lib/accept lp0
destination "lp0" now accepting requests
#
```

/usr/lib/reject

Sometimes it is necessary to stop **lp** from routing requests to a destination. For example, if a printer has been removed for repairs, or if too many requests are building at a destination, you may want to prevent new jobs from being queued at this destination. The **reject**(1M) command performs this function.

Requests in the queue when the **reject** command is invoked will be printed as long as the printer is enabled. After the condition that led to denying requests has been corrected, use the **accept** command to allow requests to be received again. The general format of the **reject** command is as follows:

reject [**−r**[*reason*]] *destinations*

The **−r** option enables you to let users know why requests are being rejected by the specified destination. *reason* is a brief explanation of the purpose for rejecting requests. If the reason consists of more than one word, enclose it in double quotes ("). The *destinations* are the printers that are not to accept requests any longer.

Command Example

The example given here is for a printer, printer3, that is being repaired. While printer3 is out of service you want to prevent **lp** from routing requests to it.

```
# reject −r"printer printer3 under repair" printer3
destination "printer3" is no longer accepting requests
#
```

Users who try to route a job to printer3 will receive the following message:

```
$ lp −dprinter3 filename
lp: can't accept requests for destination "printer3" -
        printer printer3 under repair
$
```

Printer Interface Programs

Printers that are used as LP Spooling printers must have a printer interface program. Every print request made with the **lp** command is routed through the appropriate printer interface program before the request is printed on a line printer. The printer interface program to use is specified by the **lpadmin**(1M) command.

Model Interface Programs

Each type of printer requires its own interface program. Several, referred to as "model" interface programs, are furnished with the LP Spooling Utilities. The model interface programs support the DQP-10 printer, the LQP-40 printer, and several other popular printers. The model interface programs are written as shell procedures, but they can be written as C programs or any other executable program. They are located in the **/usr/spool/lp/model** directory.

Writing Interface Programs

If you have a printer that is not supported by one of the model programs, you will have to furnish an interface program for it. The shell script for a "dumb" printer interface program (a model program) is shown in Figure 7-3. This program may be used as a guide if you have to provide one of your own.

When the LP scheduler routes an output request to a printer, the interface program for the printer is invoked in the directory **/usr/spool/lp** as follows:

interface/*P id user title copies options file ...*

Arguments for the interface program are:

P	printer name
id	request id returned by lp
user	logname of user who made the request
title	optional title specified by the user
copies	number of copies requested by user
options	blank-separated list of class or printer-dependent options specified by user
file	full path name of a file to be printed

When the interface program is invoked, its standard input comes from **/dev/null** and both the standard output and standard error output are directed to the printing device. Interface programs format their output based on the command line arguments. You want to make sure that the interface program has the proper stty modes (terminal characteristics such as baud rate, output options). You can do this by adding **stty**(1) command lines of the form:

stty *mode options <&1*

This command line takes the standard input for the **stty** command from the device. An example of an **stty** command line that sets the baud rate at 1200 and sets some of the option modes is shown below.

stty −parenb −parodd 1200 cs8 cread clocal ixon 0<&1

Because different printers have different numbers of columns, make sure the header and trailer for your interface program correspond to your printer. When printing is complete, your interface program should exit with a code that tells the status of the print job. Exit codes are interpreted by **lpsched** as follows:

Code	Meaning to lpsched
0	The print job has completed successfully.
1 to 127	A problem was encountered in printing this particular request (for example, too many nonprintable characters). This problem will not affect future print jobs. The **lpsched** command notifies users by **mail**(1) that there was an error in printing the request.
greater than 127	These codes are reserved for internal use by **lpsched**. Interface programs must not exit with codes in this range.

When problems occur that may affect future print jobs—for example, a device filter program is missing—it is wise to have your interface program disable printers so that print requests are not lost. When an active printer is disabled, the interface program can be halted with signal 15 (see **kill**(1) and **signal**(2)).

Below is an example of a dumb line printer interface program.

```
# lp interface for dumb line printer
#
x="XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
echo "\014\c"
echo "$x\n$x\n$x\n$x\n"
banner "$2"
echo "\n"
user='grep "^$2:" /etc/passwd | line | cut -d: -f5'
if [ -n "$user" ]
then
        echo "User: $user\n"
else
        echo "\n"
fi
echo "Request id; $1    Printer: 'basename $0'\n"
date
echo "\n"
if [ -n "$3" ]
then
       banner $3
fi
copies=$4
echo "\014\c"
shift; shift; shift; shift; shift
files="$x"
i=1
while [ $i -le $copies ]
do
        for file in $files
        do
                cat "$file" 2>&1
                echo "\014\c"
        done

        i='expr $i + 1'
done
echo "$x\n$x\n$x\n$x\n$x\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n"
echo "\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n$x\n$x\n$x\n$x\n$x\n$x"
echo "$x\n$x\n$x\n$x\n$x\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n"
echo "\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n"
echo "$x\n$x\n$x\n$x\n$x\n$x\n$x\n$x\n$x\n"
exit 0
```

Figure 7-3: Dumb Line Printer Interface Program

Files and Directories

This section describes the files and directories in the LP Spooling structure.

/usr/spool/lp/FIF0

FIFO is a special file that all the commands use to send messages to **lpsched**. Any of the LP commands may write to **FIFO**, but only **lpsched** may read it.

/usr/spool/lp/default

This file contains the name of the system default destination. If this file does not exist or if it is empty, the LP system has no default destination.

/usr/spool/lp/log

The purpose of the **log** file is to keep a record of all the printing activity that has taken place since the LP scheduler was last started. This file contains the **logname** of the user who made the request, the request id, the name of the printer that the request was printed on, and the date and time that printing started. Any **lpsched** error messages that occur are also recorded. The first line of the log file shows the time that the LP scheduler was started.

/usr/spool/lp/oldlog

The **oldlog** file contains a record of what was in the **log** file. When the scheduler is stopped, the **log** file is closed. When the scheduler is restarted, all the information that had accumulated in the **log** file is copied to the **oldlog** file, and a new **log** file is started. Any information that had been in the **oldlog** file is overwritten. The first line of the file tells the time that the scheduler was turned on, and the last line tells the time the scheduler was turned off.

/usr/spool/lp/outputq

When an output request is made by the **lp** command, an entry is made in this binary file. The LP scheduler takes the job request and hands it to the appropriate interface program to be printed. After the job is completed, the job request is removed, and the scheduler takes the next job request from this file and has it printed. Only those requests made since the last time the LP scheduler was started are contained in this file (that is, starting the LP scheduler clears this file).

Entries in **outputq** may be modified by the **lpmove**, **disable**, and **lpsched** command. The **cancel**, **disable**, and **lpsched** commands can mark entries in this file "deleted." If a job request is deleted before the job is completed, the entry will remain in the file.

/usr/spool/lp/pstatus

The binary file **pstatus** contains status information for each printer. Entries are added and removed from this file by the **lpadmin** command, and they are modified by the **cancel**, **enable**, **disable**, and **lpsched** commands. When the **lpstat** command is invoked with the **–p** option, printer status information is obtained from this file.

/usr/spool/lp/qstatus

This binary file keeps track of whether a destination is accepting or rejecting requests. Entries are added or removed from this file by the **lpadmin** command and modified by the **accept** and **reject** commands. When the **lpstat** command is invoked with the **–o** option, the request status is obtained from this file.

/usr/spool/lp/seqfile

The **seqfile** file contains the sequence number of the last request id that was assigned by the **lp** command. The sequence number is incremented by **lp** for each request. When the number 9999 is reached, the sequence number is reset to 1.

/usr/spool/lp/class

This is a directory that contains one file for each LP class that has been identified. (The name of the file is the same as the name of the class.) The file identifies each member, in this case an LP printer, that is assigned to the class. Class files are created, modified, and deleted by the **lpadmin** command. Every class file must always have at least one member.

/usr/spool/lp/interface

The interface directory contains one executable interface program for each printer that is in the LP system. The filename of the interface program is the same as the printer name. The interface program is invoked with its standard error and standard output directed to the printer. Interface programs may be shell procedures or compiled C programs.

/usr/spool/lp/member

The member directory contains one file for each LP printer. The filename is the same as the printer name. Each file contains the pathname of the device to which the member is connected.

/usr/spool/lp/model

This is a directory that contains the printer interface programs that are distributed with the LP Spooling Utilities.

/usr/spool/lp/request

This directory contains a subdirectory for each destination in the LP system. The name of the subdirectory is the same as the name of the destination. When an **lp** request is made, a **request** file (or "r" file) and, in most cases, a **data** file (or "d" file) are created in the subdirectory of the destination to which the request is going. The **data** file stores the file to be printed until the scheduler is ready to print it. A **data** file is not created if the file to be printed cannot be linked to the request subdirectory.

The name of the request file is derived from the request identification number and is of the form **r-seqno**. The name of the data file is of the form **dn-seqno**, where **n** is a non-negative integer.

The request and data files are deleted by the **cancel** and **lpsched** commands. They may be moved from one subdirectory to another by the **lpmove** command.

Lock Files

To guarantee LP commands exclusive access to data files, several "lock" files are maintained in the LP system. They are binary files that contain the process id of the locking process. The lock files and their associated data files are:

Lock File	Data File
OUTQLOCK	outputq
PSTATLOCK	pstatus
QSTATLOCK	qstatus
SEQLOCK	seqfile

Lock files "expire" after a given time and may be unlinked by any LP process. Thus, commands that lock a data file for longer than this interval must update the modification time on the lock file. The creation, updating, and unlinking of lock files is handled automatically by the LP low-level file access routines.

Another lock file, SCHEDLOCK, is present while the LP scheduler is running to ensure that only one invocation of **lpsched** is active. Unlike other lock files, SCHEDLOCK has no expiration time.

Cleaning Out Log Files

As described above, when the scheduler is stopped, the **log** file is closed. When the scheduler is restarted, the **log** file is copied to **/usr/spool/lp/oldlog**, and a new **log** file is started.

If the scheduler is not stopped for long periods of time and if you have a large number of LP requests, the **log** file can grow to be a large file. You can manually remove the contents of this file, or you can let the system do it for you on a scheduled basis (see "Monitoring Files and Directories that Grow" in Chapter 5, File System Administration).

To have the system clean out the log file, put an entry in a file in the **/usr/spool/cron/crontabs** directory. One way to do this is to log in as **root** and use the **crontab**(1) command. The other way is to edit a **crontabs** directory file.

The example below shows some typical **crontab** command lines. **crontab** adds these command lines to the **root** file in the **/usr/spool/cron/crontabs** directory. Every Friday at 11:00 PM **cron**(1M) executes the commands. First, the contents of the **log** file are copied to the **oldlog** file, and then the **log** file is cleaned out.

```
# crontab -l

0 23 * * 5 /bin/su lp -c "cp /usr/spool/lp/log
/usr/spool/lp/oldlog" 1 23 * * 5 /bin/su lp -c
">/usr/spool/lp/log" #
```

Lock File	Data File
OUTQLOCK	outputq
PSTATLOCK	pstatus
QSTATLOCK	qstatus
SEQLOCK	seqfile

Chapter 8: TTY Management

Introduction

This chapter covers the following topics:

- The terms used in discussing TTY management
- How the TTY system works
- How to tell what line settings are defined
- How to create new line settings and hunt sequences
- How to modify TTY line characteristics
- How to set terminal options

Definition of Terms

The following terms are used in this chapter:

TTY	Derived from the near-classic abbreviation for teletypewriter, the term covers the whole area of access between the UNIX system and peripheral devices, including the system console. It shows up in commands such as **getty**(1M) and **stty**(1), in the names of device special files such as **/dev/tty1**, and in the names of files such as **/etc/gettydefs**, which is used by **getty**.
TTY line	The physical equipment through which access to the computer is made.
port	A synonym for TTY line.
line settings	A set of line characteristics.
baud rate	The speed at which data is transmitted and received over the line. A part of line settings.
mode	The characteristics of the terminal interface. A part of line settings. The TTY line and the terminal must be working in the same mode before communication can take place. Described in **termio**(7).
hunt sequence	A circular series of line settings such as different baud rates. During the login sequence, a user looking for a compatible connection to the computer can go from one setting to the next by sending a BREAK signal.
terminal options	Selectable settings that define the way a given terminal operates. Described in **termio**(7).

The TTY System

This section describes how the TTY system operates, and how you can administer it.

How the TTY System Works

A series of four processes (**init**(1M), **getty**(1M), **login**(1), **sh**(1) or **csh**(1)) connects a user to the UMIPS system. **init** is a general process spawner that is invoked as the last step in the boot procedure. It spawns a **getty** process for each line that a user may log in on, guided by instructions in **/etc/inittab**. An argument required by the **getty** command is **line**. The TTY line argument is the name of a special file in the **/dev** directory. For a description of other arguments that may be used with **getty** see the *System Administrator's Reference Manual*.

A user attempting to make a connection generates a request-to-send signal that is routed by the hardware to the **getty** process for one of the TTY line files in **/dev**. (We're omitting how the signal gets from the user's terminal to the M series system.) **getty** responds by sending an entry from file **/etc/gettydefs** down the line. The **gettydefs** entry used depends on the **speed** argument used with the **getty** command. (In the SYNOPSIS of the **getty**(1M) command the argument name is **speed**, but it is really a pointer to the **label** field of a **gettydefs** entry.) If no **speed** argument is provided, **getty** uses the first entry in **gettydefs**. Among the fields in the **gettydefs** entry (described later in this chapter) is the login prompt.

On receiving the login prompt, the user enters a login name. **getty** starts **login**, using the login name as an argument. **login** issues the prompt for a password, evaluates the user's response, and assuming the password is acceptable, calls in the user's shell as listed in the last field of the **/etc/passwd** entry for the login name. If no shell is named, **/bin/sh** is furnished by default. **login** also executes **/etc/profile** for the **/bin/sh** user and **/etc/cshrc** for the **/bin/csh user.**

/bin/sh executes the user's **.profile**, if it exists. **/bin/csh** executes the user's **.cshrc and .login**, if they exist. These files often contain **stty** commands that reset terminal options that differ from the defaults. The connection between the user and the UNIX system has now been made.

How to Tell What Line Settings Are Defined

You have three ways to check line settings:

1. Through the System Administration Menus, specifically the **sysadm**(1) **lineset** subcommand. **sysadm lineset** first shows the full range of line settings, then gives you the chance to examine a line in detail (see Procedure 8.1).
2. By looking directly in **/etc/gettydefs**.
3. By entering "**stty -a**" at the appropriate terminal.

The **/etc/gettydefs** file contains information used by the **getty**(1M) command to establish the speed and terminal settings for a line. The general format of the **gettydefs** file is:

```
label# initial-flags # final-flags #login-prompt #next-label
```

Figure 8-1 shows a few lines from a **gettydefs** file.

```
du_9600# B9600 # B9600 SANE TAB3 HUPCL #$HOSTNAME login: #du_4800

du_4800# B4800 # B4800 SANE TAB3 HUPCL #$HOSTNAME login: #du_2400

du_2400# B2400 # B2400 SANE TAB3 HUPCL #$HOSTNAME login: #du_1200

du_1200# B1200 # B1200 SANE TAB3 HUPCL #$HOSTNAME login: #du_300

du_300# B300 # B300 SANE TAB3 HUPCL #$HOSTNAME login: #du_9600
```

Figure 8-1: **gettydefs** Entries

The entries shown in Figure 8-1 form a single, circular hunt sequence; the last field on each line is the label of the next line. The next-label field for the last line shown points back to the first line in the sequence. The object of the hunt sequence is to link a range of line speeds. If you see garbage characters instead of a clear login prompt, entering a BREAK causes **getty** to step to the next entry in the sequence. The hunt continues until the baud rate of the line matches the speed of the user's terminal. The flag fields shown have the following meanings:

B300-B19200	The baud rate of the line.
HUPCL	Hang up on close.
SANE	A composite flag that stands for a set of normal line characteristics.
TAB3	Send tabs to the terminal as spaces.

For a description of all **getty** flags, see **termio**(7).

How to Create New Line Settings and Hunt Sequences

You have two ways to do this.

1. Use the System Administration Menus, specifically the **sysadm mklineset**(1) subcommand. **sysadm mklineset** leads you through a series of prompts. Your responses make up the information for a new **gettydefs** entry (see Procedure 8.2).
2. By using **ed**(1) or **vi**(1) to edit **/etc/gettydefs**.

Create new lines for the **gettydefs** file by following the example shown above. Each entry in the file is followed by a blank line. After editing the file run the command:

 # /etc/getty −c /etc/gettydefs

This causes **getty** to scan the file and print the results on your terminal. If there are any unrecognized modes or improperly constructed entries, they are reported.

How to Modify TTY Line Characteristics

You have three ways to modify TTY line characteristics.

1. Use the System Administration Menus, specifically the **sysadm modtty**(1) sub-command. **sysadm modtty** leads you through a series of prompts. Your responses edit a "getty" entry in **/etc/inittab** (see Procedure 8.2).
2. By using **ed**(1) or **vi**(1) to edit **/etc/inittab**.
3. By using **stty**(1) to make a temporary change.

The **/etc/inittab** file contains instructions for the **/etc/init**(1M) command. The general format of a line entry in the **/etc/inittab** file is as follows.

id:run-level:action:process

The four colon-separated fields are as follows.

id	A unique one- or two-character identifier for the line entry.
run-level	The run-level (init state) in which the entry is to be performed.
action	How **/etc/init** treats the process field (refer to the **inittab**(4) manual page for complete information).
process	The shell command to be executed.

/etc/inittab contains several entries that spawn **getty** processes. Figure 8-2 is a selection of such entries **grep**'ped from an **/etc/inittab** file on an M-Series system.

```
co:234:respawn:/etc/getty console console none LDISC0
t1:234:respawn:/etc/getty tty1 co_9600 none LDISC0
h0:234:off:/etc/getty ttyh0 dx_9600 none LDISC0
h1:234:off:/etc/getty ttyh1 dx_19200 none LDISC0
h2:234:off:/etc/getty ttyh2 dx_19200 none LDISC0
      .
      .
      .
```

Figure 8-2: **getty** Entries from **/etc/inittab**

For example, the entry with the *id* of **t1** runs in multiuser mode (run-level=234), performs the respawn action, and executes the "getty" command:

for the **/dev/tty1** file

uses the speed associated with the c0_9600 label in **/etc/gettydefs**

with a type of "none"

with the default line discipline of 0 (LDISC0).

Refer to **getty**(1M) for details regarding the arguments supplied to **getty** in the **inittab** *process* field.

There are at least three things you might want to do to an **inittab** entry for a TTY line:

1. Change the action. Two actions that apply to TTY lines are "respawn" and "off". Respawn turns the line on for use as a terminal line and off is used for an unused line or, for example, a printer (see the **inittab**(4) manual page for complete information on this field).
2. Add or change arguments to **/etc/getty** in the process field. A frequently used argument is **–t***nn*. This tells **getty** to hang up if nothing is received within *nn* seconds. It's good practice to use the **–t** argument on dial-up lines.
3. Add or change comments. Comments can be inserted after a semi-colon (;) to end the command, and a pound sign (**#**) to start the comments.

Identifying a Terminal to the System

/usr/lib/terminfo is a compiled database containing descriptions of the many terminals known to the system. To tell the system what kind of terminal it is communicating with, it is necessary to know the name by which the terminal is identified in this database.

The **/usr/lib/terminfo** directory contains the subdirectories 1 though 9, and a through z. Each subdirectory contains the descriptions of the terminals beginning with the associated number or letter. For example, descriptions of Wyse terminals are found in **/usr/lib/terminfo/w**, and Hewlett-Packard terminals are described in **/usr/lib/terminfo/h.**

To find the name of a particular terminal description, use the **infocmp**(1M) command. Change directory to the **/usr/lib/terminfo** subdirectory beginning with the first character of the terminal and supply **infocmp -I** the name of your terminal:

```
% cd /usr/lib/terminfo/w
% infocmp -I wyse75
wyse75|wyse_75|wyse 75,
      mir, msgr, xon,
      cols#80, lines#24,
      bel=
      .
      .
      .
%
```

The first field is the name of the terminal, in the case of the example, wyse75. By placing the appropriate statement in the user's **.login** or **.profile**, the system will associate that login with the correct terminal. For example, for a Bourne shell user using a wyse75 terminal, add this line to the **$HOME/.profile** file:

```
TERM=wyse75;       export TERM
```

For a C shell user, add this line to the **$HOME/.login** file:

```
setenv TERM wyse75
```

For information on writing new terminal descriptions, see **terminfo**(4).

How to Set Terminal Options

The TTY system described thus far establishes a basic style of communication between the user's terminal and the UMIPS operating system. Once the user has successfully logged in, there may be terminal options that would be preferable to ones in the default set.

The command that is used to control terminal options is **stty**(1). Many users add an **stty** command to their **.profile** or **.login** so the options they want are automatically set as part of the **login** process. Here is an example of a simple **stty** command.

```
$ stty cr0 nl0 echoe -tabs erase ^H
```

The options in the example mean:

cr0 nl0	No delay for carriage return or new line. Delays are not used on a video display terminal, but are necessary on some printing terminals to allow time for the mechanical parts of the equipment to move.
echoe	Erases characters as you backspace.
-tabs	Expand tabs to spaces when printing.
erase ^H	Change the character-delete character to a ^H. The default character-delete character is the pound sign (#). Most terminals transmit a ^H when the **backspace** key is pressed. Specifying this option makes the **backspace** key useful.

The **stty -a** command displays all the current stty settings for your terminal. For an explanation of these settings, refer to **stty**(1).

Chapter 9: Basic Networking

Introduction

The UUCP utilities let computers using the UNIX operating system communicate with each other. These utilities range from those used to copy files between computers (**uucp** and **uuto**) to those used for remote login and command execution (**cu**, **ct**, and **uux**).

As an administrator, you need to be familiar with the administrative tools, logs, and data base files used by UUCP. Procedure 9, UUCP, presents instructions to install and maintain these utilities; this chapter goes into greater detail about the UUCP files, directories, daemons, and commands. A section at the end of this chapter discusses using UUCP over the ethernet connection.

A Note on Terminology

"UUCP" is the name generally used to apply to the group of utilities which includes the **uucp** (unix-to-unix copy) utility. In this document the capitalized UUCP refers to the group of utilities while the lower-case (and bolded) **uucp** refers to the specific copy command.

Networking Hardware

Before your computer can communicate with other computers, you must set up the hardware to complete the communications link. The cables and other hardware you will need depend on how you want to connect the computers: direct links, telephone lines, or local area networks.

Direct Links

You can create a direct link to another computer by running cables between serial ports on the two computers. Direct links are useful where two computers communicate regularly and are physically close—within 50 feet of each other. You can use a limited distance modem to increase this distance somewhat. Transfer rates of up to 19200 bits per second (bps) are possible when computers are directly linked.

Telephone Lines

Using an Automatic Call Unit (ACU), your computer can communicate with other computers over standard phone lines. The ACU dials the telephone number requested by the networking utilities. The computer it is trying to contact must have a telephone modem capable of answering incoming calls.

Local Area Network

A Local Area Network (LAN) can be the communication medium for basic networking. Once your computer is established as a node on a LAN, it will be able to contact any other computer connected to the LAN.

Networking Commands

UUCP programs can be divided into two categories: user programs and administrative programs. The following paragraphs describe the programs in each category.

User Programs

The user programs for UUCP are in **/usr/bin**. No special permission is needed to use these programs. These commands are all described in the *User's Reference Manual*.

cu(1C) Connects your computer to a remote computer so you can be logged in on both at the same time, allowing you to transfer files or execute commands on either computer without dropping the initial link.

ct(1C) Connects your computer to a remote terminal so the user of the remote terminal can log in. The user of a remote terminal can call the computer and request that the computer call it back. In this case, the computer drops the initial link so that the remote terminal's modem will be available when it is called back.

uucp(1C) Lets a user copy a file from one computer to another. It creates work files and data files, queues the job for transfer, and calls the **uucico** daemon, which in turn attempts to contact the remote computer.

uuto(1C) Copies files from one computer to a public spool directory on another computer (**/usr/spool/uucppublic/receive**). Unlike **uucp**, which lets you copy a file to any accessible directory on the remote computer, **uuto** places the file in an appropriate spool directory and tells the remote user to pick it up with **uupick**.

uupick(1C) Retrieves the files placed under **/usr/spool/uucppublic/receive** when files are transferred to a computer using **uuto**.

uux(1C) Creates the work, data, and execute files needed to execute commands on a remote computer. The work file contains the same information as work files created by **uucp** and **uuto**. The execute files contain the command string to be executed on the remote computer and a list of the data files. The data files are those files required for the command execution.

uustat(1C) Displays the status of requested transfers (**uucp**, **uuto**, or **uux**). It also provides you with a means of controlling queued transfers.

Administrative Programs

Most of the administrative programs are in **/usr/lib/uucp**, along with UUCP data base files and shell scripts. The only exception is **uulog**, which is in **/usr/bin**. These commands are described in the *System Administrator's Reference Manual*.

You should use the **uucp** login ID when you administer UUCP. The home directory of the **uucp** login ID is **/usr/lib/uucp**. (The other UUCP login ID is **nuucp**, used by remote computers to access your computer. Calls from **nuucp** are answered by **uucico**.)

uulog(1C) Displays the contents of a specified computer's log files. Log files are created for each remote computer your computer communicates with. The log files contain records of each use of **uucp**, **uuto**, and **uux**.

uucleanup(1M) Cleans up the spool directory. It is normally executed from a shell script called **uudemon.cleanup**, which is started by **cron**.

Uutry(1M) Tests call processing capabilities and does a moderate amount of debugging. It invokes the **uucico** daemon to establish a communication link between your computer and the remote computer you specify.

uucheck(1M) Checks for the presence of uucp directories, programs, and support files. It can also check certain parts of the **Permissions** file for obvious syntactic errors.

Daemons

There are three daemons in the UUCP utilities. A daemon is a routine that runs as a background process and performs a system-wide, public function. The following daemons handle file transfers and command executions. They can also be run manually from the shell.

uucico Selects the device used for the link, establishes the link to the remote computer, performs the required login sequence and permission checks, transfers data and execute files, logs results, and notifies the user by **mail** of transfer completions. When the local **uucico** daemon calls a remote computer, it "talks" to the **uucico** daemon on the remote computer during the session.

The **uucico** daemon is executed by **uucp**, **uuto**, and **uux** programs, after all the required files have been created, to contact the remote computer. It is also executed by the **uusched** and **Uutry** programs.

uuxqt Executes remote execution requests. It searches the spool directory for execute files (always named X.*file*) that have been sent from a remote computer. When an X.*file* file is found, **uuxqt** opens it to get the list of data files that are required for the execution. It then checks to see if the required data files are available and accessible. If the files are present and can be accessed, **uuxqt** checks the **Permissions** file to verify that it has permission to execute the requested command. The **uuxqt** daemon is executed by the **uudemon.hour** shell script, which is started by **cron**.

uusched Schedules the queued work in the spool directory. Before starting the **uucico** daemon, **uusched** randomizes the order in which remote computers will be called. **uusched** is executed by a shell script called **uudemon.hour**, which is started by **cron**.

Internal Programs

uugetty This program is very similar to the **getty** program except it permits a line (port) to be used in both directions. **uugetty** can be executed as a function of the **init** program and is described in the *System Administrator's Reference Manual*.

Supporting Data Base

The uucp support files are in the **/usr/lib/uucp** directory. The descriptions below provide details on the structure of these files so you can edit them manually.

Devices Contains information concerning the location and line speed of the automatic call unit, direct links, and network devices.

Dialers Contains character strings required to negotiate with network devices (automatic calling devices) in the establishment of connections to remote computers (non 801-type dialers).

Systems Contains information needed by the **uucico** daemon and the **cu** program to establish a link to a remote computer. It contains information such as the name of the remote computer, the name of the connecting device associated with the remote computer, when the computer can be reached, telephone number, login ID, and password.

Dialcodes This file contains dial-code abbreviations that may be used in the phone number field of **Systems** file entries.

Permissions This file defines the level of access that is granted to computers when they attempt to transfer files or remotely execute commands on your computer.

Poll This file defines computers that are to be polled by your system and when they are polled.

Devconfig This file is used to configure utilities for uucp on a transport provider that conforms to the AT&T Transport Interface.

Sysfiles This file is used to assign different or multiple files to be used by **uucico** and **cu** as **Systems**, **Devices**, and **Dialers** files.

There are several other files that may be considered part of the supporting data base, but are not directly related to the process of establishing a link and transferring files. These files—**Maxuuxqts**, **Maxuuscheds**, and **remote.unknown**—are described briefly in Procedure 9.

Devices File

The **Devices** file (**/usr/lib/uucp/Devices**) contains information for all the devices that may be used to establish a link to a remote computer, devices such as automatic call units, direct links, and network connections.

NOTE

This file works closely with the **Dialers**, **Systems**, and **Dialcodes** files. Before you make changes in any of these files, you should be familiar with them all. A change to an entry in one file may require a change to the related entry in another file.

Each entry in the **Devices** file has the following format:

Type Line Line2 Class Dialer-Token-Pairs

Each of these fields is defined in the following section.

Type This field may contain one of two keywords (**Direct** or **ACU**), the name of a Local Area Network switch, or a system name.

Direct This keyword indicates a Direct Link to another computer or a switch (for **cu** connections only).

ACU This keyword indicates that the link to a remote computer is made through an automatic call unit (Automatic Dial Modem). This modem may be connected either directly to your computer or indirectly through a Local Area Network (LAN) switch.

LAN_Switch
This value can be replaced by the name of a LAN switch. The **Dialers** file comes configured for TCP/IP and also contains many examples so you can add your own LAN switch entries.

Sys-Name
This value indicates a direct link to a particular computer. (*Sys-Name* is replaced by the name of the computer.) This naming scheme is used to convey the fact that the line associated with this **Devices** entry is for a particular computer in the **Systems** file.

The keyword used in the *Type* field is matched against the third field of **Systems** file entries as shown below:

```
Devices: ACU tty11 - 1200 penril

Systems: eagle Any ACU 1200 3251 ogin: nuucp \
                ssword: Oakgrass
```

You can designate a protocol to use for a device within this field. See the "Protocols" section at the end of the description of this file.

Line This field contains the device name of the line (port) associated with the **Devices** entry. For instance, if the Automatic Dial Modem for a particular entry was attached to the **/dev/tty11** line, the name entered in this field would be **tty11**.

Line2 If the keyword **ACU** was used in the *Type* field and the ACU is an 801 type dialer, *Line2* would contain the device name of the 801 dialer. (801 type ACUs do not contain a modem. Therefore, a separate modem is required and would be connected to a different line, defined in the *Line* field.) This means that one line would be allocated to the modem and another to the dialer. Since non-801 dialers will not normally use this configuration, the *Line2* field will be ignored by them, but it must still contain a hyphen (-) as a placeholder.

Class If the keyword **ACU** or **Direct** is used in the *Type* field, *Class* may be just the speed of the device. However, it may contain a letter and a speed (for example, C1200, D1200) to differentiate between classes of dialers (Centrex or Dimension PBX). This is necessary because many larger offices may have more than one type of telephone network: one network may be dedicated to serving only internal office communications while another handles the external communications. In such a case, it becomes necessary to distinguish which line(s) should be used for internal communications and which should be used for external communications. The keyword used in the *Class* field of the **Devices** file is matched against the fourth field of **Systems** file entries as shown

below:

Devices: ACU tty11 - **D1200** penril

Systems: eagle Any ACU **D1200** 3251 ogin: nuucp \
ssword: Oakgrass

Some devices can be used at any speed, so the keyword **Any** may be used in the *Class* field. If **Any** is used, the line will match any speed requested in a **Systems** file entry. If this field is **Any** and the **Systems** file *Class* field is **Any**, the speed defaults to 1200 bps.

Dialer-Token-Pairs:
This field contains pairs of dialers and tokens. The *dialer* portion may be the name of an automatic dial modem, a LAN switch, or it may be **direct** for a Direct Link device. You can have any number of Dialer-Token-Pairs. The *token* portion may be supplied immediately following the *dialer* portion or if not present, it will be taken from a related entry in the **Systems** file.

This field has the format:

dialer token dialer token

where the last pair may or may not be present, depending on the associated device (dialer). In most cases, the last pair contains only a *dialer* portion and the *token* portion is retrieved from the *Phone* field of the **Systems** file entry.

A valid entry in the *dialer* portion may be defined in the **Dialers** file or may be one of several special dialer types. These special dialer types are compiled into the software and are therefore available without having entries in the **Dialers** file.

801 - Bell 801 auto dialer
TLI - Transport Level Interface Network (without STREAMS)
TLIS - Transport Level Interface Network (with STREAMS)

The *Dialer-Token-Pairs* (*DTP*) field may be structured four different ways, depending on the device associated with the entry:

1. If an automatic dialing modem is connected directly to a port on your computer, the *DTP* field of the associated **Devices** file entry will only have one pair. This pair would normally be the name of the modem. This name is used to match the particular **Devices** file entry with an entry in the **Dialers** file. Therefore, the *dialer* field must match the first field of a **Dialers** file entry as shown below:

   Devices: ACU tty11 - 1200 **ventel**

   Dialers: **ventel** =&-% "" \r\p\r\c $ <K\T%%\r>\c ONLINE!

   Notice that only the *dialer* portion (**ventel**) is present in the *DTP* field of the **Devices** file entry. This means that the *token* to be passed on to the dialer (in this case the phone number) is taken from the *Phone* field of a **Systems** file entry. (\T is implied, see below.) Backslash sequences are described below.

2. If a direct link is established to a particular computer, the *DTP* field of the associated entry would contain the keyword **direct**. This is true for both types of direct link entries, **Direct** and *System-Name* (refer to discussion on the *Type* field).

3. If a computer with which you wish to communicate is on the same local network switch as your computer, your computer must first access the switch and the switch can make the connection to the other computer. In this type of entry, there is only one pair. The *dialer* portion is used to match a **Dialers** file entry as shown below:

   Devices: `develcon tty13 - 1200` **develcon** `\D`

   Dialers: **develcon** `"" "" \pr\ps\c est:\007 \E\D\e \007`

   As shown, the *token* portion is left blank, which indicates that it is retrieved from the **Systems** file. The **Systems** file entry for this particular computer will contain the token in the *Phone* field, which is normally reserved for the phone number of the computer (refer to **Systems** file, *Phone* field). This type of *DTP* contains an escape character (**\D**), which ensures that the contents of the *Phone* field will not be interpreted as a valid entry in the **Dialcodes** file.

4. If an automatic dialing modem is connected to a switch, your computer must first access the switch and the switch will make the connection to the automatic dialing modem. This type of entry requires two *dialer-token-pairs*. The *dialer* portion of each pair (fifth and seventh fields of entry) will be used to match entries in the **Dialers** file as shown below:

   Devices: `ACU tty14 - 1200` **develcon** `vent` **ventel**

   Dialers: **develcon** `"" "" \pr\ps\c est:\007 \E\D\e \007`
   Dialers: **ventel** `=&-% "" \r\p\r\c $ <K\T%%\r>\c ONLINE!`

   In the first pair, **develcon** is the dialer and **vent** is the token that is passed to the Develcon switch to tell it which device (Ventel modem) to connect to your computer. This token would be unique for each LAN switch since each switch may be set up differently. Once the ventel modem has been connected, the second pair is accessed, where ventel is the dialer and the token is retrieved from the **Systems** file.

There are two escape characters that may appear in a *DTP* field:

\T Indicates that the *Phone* (*token*) field should be translated using the **Dialcodes** file. This escape character is normally placed in the **Dialers** file for each caller script associated with an automatic dial modem (penril, ventel, etc.). Therefore, the translation will not take place until the caller script is accessed.

\D Indicates that the *Phone* (*token*) field should not be translated using the **Dialcodes** file. If no escape character is specified at the end of a **Devices** entry, the **\D** is assumed (default). A **\D** is also used in the **Dialers** file with entries associated with network switches (develcon and micom).

Protocols

You can define the protocol to use with each device. In most cases it is not needed since you can use the default or define the protocol with the particular System you are calling (see **Systems** file, type field). If you do specify the protocol, you must do so in the form *Type,Protocol* (for example, **STARLAN,e**). Available protocols are:

g This protocol is slower and more reliable than **e**. It is good for transmission over noisy telephone lines.

e This protocol is faster than **g**, but it assumes error-free transmission.

For reliable local area networks, you should use the **e** protocol.

Dialers File

The **Dialers** file (**/usr/lib/uucp/Dialers**) specifies the initial conversation that must take place on a line before it can be made available for transferring data. This conversation is usually a sequence of ASCII strings that is transmitted and expected, and it is often used to dial a phone number using an ASCII dialer (such as the Automatic Dial Modem).

As shown in the above examples, the fifth field in a **Devices** file entry is an index into the **Dialers** file or a special dialer type (801, TLI, or TLIS). Here an attempt is made to match the fifth field in the **Devices** file with the first field of each **Dialers** file entry. In addition, each odd numbered **Devices** field starting with the seventh position is used as an index into the **Dialers** file. If the match succeeds, the **Dialers** entry is interpreted to perform the dialer negotiations. Each entry in the **Dialers** file has the following format:

dialer substitutions expect-send ...

The *dialer* field matches the fifth and additional odd numbered fields in the **Devices** file. The *substitutions* field is a translate string: the first of each pair of characters is mapped to the second character in the pair. This is usually used to translate = and - into whatever the dialer requires for "wait for dialtone" and "pause."

The remaining *expect-send* fields are character strings. The meaning of some of the escape characters (those beginning with "\") used in the **Dialers** file are listed below:

`\p` pause (approximately ¼ to ½ second)

`\d` delay (approximately 2 seconds)

`\D` phone number or token without **Dialcodes** translation

`\T` phone number or token with **Dialcodes** translation

`\K` insert a BREAK

`\E` enable echo checking (for slow devices)

`\e` disable echo checking

`\r` carriage return

`\c` no new-line or carriage return

\n	send new-line
\nnn	send octal number.

Additional escape characters that may be used are listed in the section discussing the **Systems** file.

The penril entry in the **Dialers** file is executed as follows. First, the phone number argument is translated, replacing any **=** with a **W** (wait for dialtone) and replacing any **-** with a **P** (pause). The handshake given by the remainder of the line works as follows:

""	Wait for nothing. (In other words, proceed to the next thing.)
\d	Delay for 2 seconds.
>	Wait for a **>**.
s\p9\c	Send an **s**, pause for ½ second, send a **9**, send no terminating new-line
)-W\p\r\ds\p9\c-)	Wait for a **)**. If it is not received, process the string between the **-** characters as follows. Send a **W**, pause, send a carriage-return, delay, send an **s**, pause, send a **9**, without a new-line, and then wait for the **)**.
y\c	Send a **y**.
:	Wait for a **:**.
\E\TP	Enable echo checking. (From this point on, whenever a character is transmitted, it will wait for the character to be received before doing anything else.) Then, send the phone number. The **\T** means take the phone number passed as an argument and apply the **Dialcodes** translation and the modem function translation specified by field 2 of this entry. Then send a **P**.
>	Wait for a **>**.
9\c	Send a **9** without a new-line.
OK	Waiting for the string **OK**.

Systems File

The **Systems** file (**/usr/lib/uucp/Systems**) contains the information needed by the **uucico** daemon to establish a communication link to a remote computer. Each entry in the file represents a computer that can be called by your computer. In addition, the uucp software can be configured to prevent any computer that does not appear in this file from logging in on your computer (refer to the section "Other Networking Files" in this chapter for a description of the **remote.unknown** file). More than one entry may be present for a particular computer. The additional entries represent alternative communication paths that will be tried in sequential order. The management of this file is supported by the System Administration Menu subcommand **systemmgmt**.

Using **Sysfiles**, you can define several files to be used as "Systems" files. See the description of the **Sysfiles** file for details. Each entry in a **Systems** file has the following format:

System-Name Time Type Class Phone Login

Each of these fields is defined in the following section.

System-name
This field contains the node name of the remote computer.

Time This field is a string that indicates the day-of-week and time-of-day when the remote computer can be called. The format of the *Time* field is:

daytime[*;retry*]

The day portion may be a list containing some of the following:

Su Mo Tu We Th Fr Sa
for individual days

Wk for any week-day (Mo Tu We Th Fr)

Any for any day

Never for a passive arrangement with the remote computer. If the *Time* field is **Never**, your computer will never initiate a call to the remote computer. The call must be initiated by the remote computer. In other words, your computer is in a passive mode in respect to the remote computer (see discussion of **Permissions** file).

Here is an example:

```
Wk 1700-0800, Sa, Su
```

This example allows calls from 5:00 p.m. to 8:00 am, Monday through Thursday, and calls any time Saturday and Sunday. The example would be an effective way to call only when phone rates are low, if immediate transfer is not critical.

The *time* portion should be a range of times such as 0800-1230. If no *time* portion is specified, any time of day is assumed to be allowed for the call. A time range that spans 0000 is permitted. For example, **0800-0600** means all times are allowed other than times between 6 a.m. and 8 a.m. An optional subfield, *retry*, is available to specify the minimum time (in minutes) before a retry, following a failed attempt. The default wait is 60 minutes. The subfield separator is a semicolon (;). For example, **Any;9** is interpreted as call any time, but wait at least 9 minutes before retrying after a failure occurs.

Type This field contains the device type that should be used to establish the communication link to the remote computer. The keyword used in this field is matched against the first field of **Devices** file entries as shown below:

```
Systems:  eagle Any ACU,g D1200 3251 ogin: nuucp \
            ssword: Oakgrass

Devices:  ACU tty11 - D1200 penril
```

You can define the protocol used to contact the system by adding it on to the *Type* field. The example above shows how to attach the protocol **g** to the device type **ACU**. See the information under the "Protocols" section in the description of the **Devices** file for details.

Class This field is used to indicate the transfer speed of the device used in establishing the communication link. It may contain a letter and speed (for example, C1200, D1200) to differentiate between classes of dialers (refer to the discussion on the **Devices** file, *Class* field). Some devices can be used at any speed, so the keyword **Any** may be used. This field must match the *Class* field in the associated **Devices** file entry as shown below:

```
Systems:  eagle Any ACU D1200 NY3251 ogin: nuucp \
            ssword: Oakgrass

Devices:  ACU tty11 - D1200 penril
```

If information is not required for this field, use a **-** as a place holder for the field.

Phone This field is used to provide the phone number (token) of the remote computer for automatic dialers (LAN switches). The phone number is made up of an optional alphabetic abbreviation and a numeric part. If an abbreviation is used, it must be one that is listed in the **Dialcodes** file. For example:

```
Systems:  eagle Any ACU D1200 NY3251 ogin: nuucp \
            ssword: Oakgrass

Dialcodes:  NY 9=1212555
```

In this string, an equal sign (=) tells the ACU to wait for a secondary dial tone before dialing the remaining digits. A dash in the string (-) instructs the ACU to pause 4 seconds before dialing the next digit.

If your computer is connected to a LAN switch, you may access other computers that are connected to that switch. The **Systems** file entries for these computers will not have a phone number in the *Phone* field. Instead, this field will contain the token that must be passed on to the switch so it will know which computer your computer wishes to communicate with. (This is usually just the system name.) The associated **Devices** file entry should have a **\D** at the end of the entry to ensure that this field is not translated using the **Dialcodes** file.

Login This field contains login information given as a series of fields and subfields of the format:

expect send

where *expect* is the string that is received and *send* is the string that is 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, with **login--login**, UUCP will expect **login**. If UUCP gets **login,** it will go on to the next field. If it does not get **login**, it will send nothing followed by a new line, then look for **login** again. If no characters are initially expected from the remote computer, the characters "" (null string) should be used in the first *expect* field. Note that all *send* fields will be sent followed by a new-line unless the *send* string is terminated with a **\c**.

Here is an example of a **Systems** file entry that uses an expect-send string:

```
owl Any ACU 1200 Chicago6013 "" \r ogin:-BREAK-ogin: \
uucpx word: xyzzy
```

This example says send a carriage return and wait for `ogin:` (for `Login:`). If you don't get `ogin`, send a **BREAK**. When you do get `ogin:` send the login name **uucpx**, then when you get `word:` (for Password:), send the password **xyzzy**.

There are several escape characters that cause specific actions when they are a part of a string sent during the login sequence. The following escape characters are useful in UUCP communications:

`\N`	Send or expect a null character (ASCII NUL).
`\b`	Send or expect a backspace character.
`\c`	If at the end of a string, suppress the new-line that is normally sent. Ignored otherwise.
`\d`	Delay two seconds before sending or reading more characters.
`\p`	Pause for approximately ¼ to ½ second.
`\E`	Start echo checking. (From this point on, whenever a character is transmitted, it will wait for the character to be received before doing anything else.)
`\e`	Echo check off.
`\n`	Send a new-line character.
`\r`	Send or expect a carriage-return.
`\s`	Send or expect a space character.
`\t`	Send or expect a tab character.
`\\`	Send or expect a \ character.
`EOT`	Send or expect EOT new-line twice.
`BREAK`	Send or expect a break character.
`\K`	Same as BREAK.
`\ddd`	Collapse the octal digits (ddd) into a single character.

Dialcodes File

The **Dialcodes** file (**/usr/lib/uucp/Dialcodes**) contains the dial-code abbreviations that can be used in the *Phone* field of the **Systems** file. Each entry has the format:

abb dial-seq

where *abb* is the abbreviation used in the **Systems** file *Phone* field and *dial-seq* is the dial sequence that is passed to the dialer when that particular **Systems** file entry is accessed.

The entry

```
jt 9=847-
```

would be set up to work with a *Phone* field in the **Systems** file such as `jt7867`. When the entry containing `jt7867` is encountered, the sequence 9=847-7867 would be sent to the dialer if the token in the dialer-token-pair is `\T`.

Permissions File

The **Permissions** file (**/usr/lib/uucp/Permissions**) specifies the permissions that remote computers have with respect to login, file access, and command execution. There are options that restrict the remote computer's ability to request files and its ability to receive files queued by the local site. Another option is available that specifies the commands that a remote site can execute on the local computer.

How Entries are Structured

Each entry is a logical line with physical lines terminated by a \ to indicate continuation. Entries are made up of options delimited by white space. Each option is a name/value pair in the following format:

name=value

Note that no white space is allowed within an option assignment.

Comment lines begin with a "#" and they occupy the entire line up to a newline character. Blank lines are ignored (even within multi-line entries).

There are two types of **Permissions** file entries:

LOGNAME Specifies the permissions that take effect when a remote computer logs in on (calls) your computer.

MACHINE Specifies permissions that take effect when your computer logs in on (calls) a remote computer.

LOGNAME entries will contain a LOGNAME option and MACHINE entries will contain a MACHINE option.

Considerations

The following items should be considered when using the **Permissions** file to restrict the level of access granted to remote computers:

- All login IDs used by remote computers to login for UUCP communications must appear in one and only one LOGNAME entry.
- Any site that is called whose name does not appear in a MACHINE entry, will have the following default permissions/restrictions:
  - Local send and receive requests will be executed.
  - The remote computer can send files to your computer's **/usr/spool/uucppublic** directory.
  - The commands sent by the remote computer for execution on your computer must be one of the default commands; usually **rmail**.

Options

This section describes each option, specifies how they are used, and lists their default values.

REQUEST When a remote computer calls your computer and requests to receive a file, this request can be granted or denied. The REQUEST option specifies whether the remote computer can request to set up file transfers from your computer. The string

```
REQUEST=yes
```

specifies that the remote computer can request to transfer files from your computer. The string

```
REQUEST=no
```

specifies that the remote computer cannot request to receive files from your computer. This is the default value. It will be used if the REQUEST option is not specified. The REQUEST option can appear in either a LOGNAME (remote calls you) entry or a MACHINE (you call remote) entry. A note on security: When a remote machine calls you, unless you have a unique login and password for that machine you don't know if the machine is who it says it is.

SENDFILES When a remote computer calls your computer and completes its work, it may attempt to take work your computer has queued for it. The SENDFILES option specifies whether your computer can send the work queued for the remote computer.

The string

```
SENDFILES=yes
```

specifies that your computer may send the work that is queued for the remote computer as long as it logged in as one of the names in the LOGNAME option. This string is mandatory if your computer is in a "passive mode" with respect to the remote computer.

The string

```
SENDFILES=call
```

specifies that files queued in your computer will be sent only when your computer calls the remote computer. The call value is the default for the SENDFILE option. This option is only significant in LOGNAME entries since MACHINE entries apply when calls are made out to remote computers. If the option is used with a MACHINE entry, it will be ignored.

READ and WRITE

These options specify the various parts of the file system that **uucico** can read from or write to. The READ and WRITE options can be used with either MACHINE or LOGNAME entries.

The default for both the READ and WRITE options is the **uucppublic** directory as shown in the following strings:

```
READ=/usr/spool/uucppublic
WRITE=/usr/spool/uucppublic
```

The strings

```
READ=/ WRITE=/
```

specify permission to access any file that can be accessed by a local user with "other" permissions.

The value of these entries is a colon separated list of path names. The READ option is for requesting files, and the WRITE option for depositing files. One of the values must be the prefix of any full path name of a file coming in or going out. To grant permission to deposit files in **/usr/news** as well as the public directory, the following values would be used with the WRITE option:

```
WRITE=/usr/spool/uucppublic:/usr/news
```

It should be pointed out that if the READ and WRITE options are used, all path names must be specified because the path names are not added to the default list. For instance, if the **/usr/news** path name was the only one specified in a WRITE option, permission to deposit files in the public directory would be denied.

You should be careful what directories you make accessible for reading and writing by remote systems. For example, you probably wouldn't want remote computers to be able to write over your **/etc/passwd** file so **/etc** shouldn't be open to writes.

NOREAD and NOWRITE

The NOREAD and NOWRITE options specify exceptions to the READ and WRITE options or defaults. The strings

```
READ=/ NOREAD=/etc WRITE=/usr/spool/uucppublic
```

would permit reading any file except those in the **/etc** directory (and its subdirectories—remember, these are prefixes) and writing only to the default **/usr/spool/uucppublic** directory. NOWRITE works in the same manner as the NOREAD option. The NOREAD and NOWRITE can be used in both LOGNAME and MACHINE entries.

CALLBACK The CALLBACK option is used in LOGNAME entries to specify that no transaction will take place until the calling system is called back. There are two examples of when you would use CALLBACK. From a security standpoint, if you call back a machine you can be sure it is the machine it says it is. If you are doing long data transmissions, you can choose the machine that will be billed for the longer call.

The string

```
CALLBACK=yes
```

specifies that your computer must call the remote computer back before any file transfers will take place.

The default for the COMMAND option is

```
CALLBACK=no
```

The CALLBACK option is very rarely used. Note that if two sites have this option set for each other, a conversation will never get started.

COMMANDS The COMMANDS option can be hazardous to the security of your system. Use it with extreme care.

The **uux** program will generate remote execution requests and queue them to be transferred to the remote computer. Files and a command are sent to the target computer for remote execution. The COMMANDS option can be used in MACHINE entries to specify the commands that a remote computer can execute on your computer. Note that COMMANDS is not used in a LOGNAME entry; COMMANDS in MACHINE entries define command permissions whether we call the remote system or it calls us.

The string

```
COMMANDS=rmail
```

indicates the default commands that a remote computer can execute on your computer. If a command string is used in a MACHINE entry, the default commands are overridden. For instance, the entry

```
MACHINE=owl:raven:hawk:dove \
COMMANDS=rmail:rnews:lp
```

overrides the COMMAND default so that the computers owl, raven, hawk, and dove can now execute **rmail**, **rnews**, and **lp** on your computer.

In addition to the names as specified above, there can be full path names of commands. For example,

```
COMMANDS=rmail:/usr/lbin/rnews:/usr/local/lp
```

specifies that command **rmail** uses the default path. The default paths for your computer are **/bin**, **/usr/bin**, and **/usr/lbin**. When the remote computer specifies **rnews** or **/usr/lbin/rnews** for the command to be executed, **/usr/lbin/rnews** will be executed regardless of the default path. Likewise, **/usr/local/lp** is the **lp** command that will be executed.

Including the ALL value in the list means that any command from the remote computer(s) specified in the entry will be executed. If you use this value, you give the remote computer full access to your computer. BE CAREFUL. This allows far more access than normal users have.

The string

```
COMMANDS=/usr/lbin/rnews:ALL:/usr/local/lp
```

illustrates two points: The ALL value can appear anywhere in the string, and the path names specified for **rnews** and **lp** will be used (instead of the default) if the requested command does not contain the full path names for **rnews** or **lp**.

The VALIDATE option should be used with the COMMANDS option whenever potentially dangerous commands like **cat** and **uucp** are specified with the COMMANDS option. Any command that reads or writes files is potentially dangerous to local security when executed by the UUCP remote execution daemon (**uuxqt**).

VALIDATE The VALIDATE option is used in conjunction with the COMMANDS option when specifying commands that are potentially dangerous to your computer's security. It is used to provide a certain degree of verification of the caller's identity. The use of the VALIDATE option requires that privileged computers have a unique login/password for UUCP transactions. An important aspect of this validation is that the login/password associated with this entry be protected. If an outsider gets that information, that particular VALIDATE option can no longer be considered secure. (VALIDATE is merely an added level of security on top of the COMMANDS option, though it is a more secure way to open command access than ALL.)

Careful consideration should be given to providing a remote computer with a privileged login and password for UUCP transactions. Giving a remote computer a special login and password with file access and remote execution capability is like giving anyone on that computer a normal login and password on your computer. Therefore, if you cannot trust someone on the remote computer, do not provide that computer with a privileged login and password.

The LOGNAME entry

```
LOGNAME=uucpfriend VALIDATE=eagle:owl:hawk
```

specifies that if one of the remote computers that claims to be eagle, owl, or hawk logs in on your computer, it must have used the login **uucpfriend**. As can be seen, if an outsider gets the **uucpfriend** login/password, masquerading is trivial.

But what does this have to do with the COMMANDS option, which only appears in MACHINE entries? It links the MACHINE entry (and COMMANDS option) with a LOGNAME entry associated with a privileged login. This link is needed because the execution daemon is not running while the remote computer is logged in. In fact, it is an asynchronous process with no knowledge of what computer sent the execution request. Therefore, the real question is how does your computer know where the execution files came from?

Each remote computer has its own "spool" directory on your computer. These spool directories have write permission given only to the UUCP programs. The execution files from the remote computer are put in its spool directory after being transferred to your computer. When the **uuxqt** daemon runs, it can use the spool directory name to find the MACHINE entry in the **Permissions** file and get the COMMANDS list, or if the computer name does not appear in the **Permissions** file, the default list will be used. The following example shows the relationship between the MACHINE and LOGNAME entries:

```
MACHINE=eagle:owl:hawk REQUEST=yes \
COMMANDS=rmail:/usr/lbin/rnews \
READ=/  WRITE=/

LOGNAME=uucpz VALIDATE=eagle:owl:hawk \
REQUEST=yes SENDFILES=yes \
READ=/  WRITE=/
```

The value in the COMMANDS option means that remote mail and **/usr/lbin/rnews** can be executed by remote users.

In the first entry, you must make the assumption that when you want to call one of the computers listed, you are really calling either **eagle**, **owl**, or **hawk**. Therefore, any files put into one of the **eagle**, **owl**, or **hawk** spool directories is put there by one of those computers. If a remote computer logs in and says that it is one of these three computers, its execution files will also be put in the privileged spool directory. You therefore have to validate that the computer has the privileged login **uucpz**.

You may want to specify different option values for the computers your computer calls that are not mentioned in specific MACHINE entries. This may occur when there are many computers calling in, and the command set changes from time to time. The name "OTHER" for the computer name is used for this entry as shown below:

```
MACHINE=OTHER \
COMMANDS=rmail:rnews:/usr/lbin/Photo:/usr/lbin/xp
```

All other options available for the MACHINE entry may also be set for the computers that are not mentioned in other MACHINE entries.

Combining MACHINE and LOGNAME Entries

It is possible to combine MACHINE and LOGNAME entries into a single entry where the common options are the same. For example, the two entries

```
MACHINE=eagle:owl:hawk REQUEST=yes \
  READ=/  WRITE=/

LOGNAME=uucpz REQUEST=yes SENDFILES=yes \
  READ=/  WRITE=/
```

share the same REQUEST, READ, and WRITE options. These two entries can be merged as shown below:

```
MACHINE=eagle:owl:hawk REQUEST=yes \
LOGNAME=uucpz SENDFILES=yes \
  READ=/  WRITE=/
```

Poll File

The **Poll** file (**/usr/lib/uucp/Poll**) contains information for polling remote computers. Each entry in the **Poll** file contains the name of a remote computer to call, followed by a <TAB> character (a space won't work), and finally the hours the computer should be called. The format of entries in the **Poll** file are:

sys-name hour ...

For example the entry:

```
eagle 0 4 8 12 16 20
```

will provide polling of computer **eagle** every four hours.

The **uudemon.poll** script does not actually perform the poll. It merely sets up a polling work file (always named C.*file*), in the spool directory that will be seen by the scheduler, which is started by **uudemon.hour**.

Devconfig File

The **/usr/lib/uucp/Devconfig** file is used when your computer communicates over a STARLAN network or some other Streams-based transport provider that conforms to the AT&T Transport Interface (TI).

Devconfig entries define the STREAMS modules that are used for a particular TI device. Entries in the **Devconfig** file have the format:

service=*x* **device=***y* **push=***z*[:*z* ...]

where *x* can be **cu**, **uucico**, or both separated by a colon; *y* is the name of a TI network and must match an entry in the **Devices** file; and *z* is replaced by the names of STREAMS modules in the order that they are to be pushed onto the Stream. Different modules and devices can be defined for **cu** and **uucp** services.

Sysfiles File

The **/usr/lib/uucp/Sysfiles** file lets you assign different files to be used by **uucp** and **cu** as **Systems**, **Devices**, and **Dialers** files. Here are some cases where this optional file may be useful.

- You may want different **Systems** files so requests for login services can be made to different addresses than **uucp** services.
- You may want different **Dialers** files to use different handshaking for **cu** and **uucp**.
- You may want to have multiple **Systems**, **Dialers**, and **Devices** files. The **Systems** file in particular may become large, making it more convenient to split it into several smaller files.

The format of the **Sysfiles** file is

service=*w* **systems=***x:x* **dialers=***y:y* **devices=***z:z*

where *w* is replaced by **uucico**, **cu**, or both separated by a colon; *x* is one or more files to be used as the **Systems** file, with each file name separated by a colon and read in the order presented; *y* is one or more files to be used as the **Dialers** file; and *z* is one or more files to be used as the **Devices** file. Each file is assumed to be relative to the **/usr/lib/uucp** directory, unless a full path is given. A backslash-carriage return (**\<CR>**) can be used to continue an entry on to the next line.

Here's an example of using a local **Systems** file in addition to the usual **Systems** file:

```
service=uucico:cu       systems=Systems:Local_Systems
```

If this is in **/usr/lib/uucp/Sysfiles**, then both **uucico** and **cu** will first look in **/usr/lib/uucp/Systems**. If the system they're trying to call doesn't have an entry in that file, or if the entries in the file fail, then they'll look in **/usr/lib/uucp/Local_Systems**.

When different **Systems** files are defined for **uucico** and **cu** services, your machine will store two different lists of Systems. You can print the **uucico** list using the **uuname** command or the **cu** list using the **uuname −c** command.

Other Networking Files

There are three other files that impact the use of uucp. In most cases, the default values are fine and no changes are needed. If you want to change them, however, use any standard UNIX system text editor (**ed** or **vi**).

Maxuuxqts — This file defines the maximum number of **uuxqt** programs that can run at once.

Maxuuscheds — This file defines the maximum number of **uusched** programs that can run at once.

remote.unknown — This file is a shell script that executes when a machine that is not in any of the **Systems** starts a conversation. It will log the conversation attempt and fail to make a connection. If you change the permissions of this file so it cannot execute

(**chmod 000 remote.unknown**), your system will accept any conversation requests.

Administrative Files

The uucp administrative files are described below. These files are created in spool directories to lock devices, hold temporary data, or keep information about remote transfers or executions.

TM (temporary data file)

These data files are created by Basic Networking processes under the spool directory (i.e., **/usr/spool/uucp/***X*) when a file is received from another computer. The directory *X* has the same name as the remote computer that is sending the file. The names of the temporary data files have the format:

TM.*pid.ddd*

where *pid* is a process-ID and *ddd* is a sequential three digit number starting at 0.

When the entire file is received, the **TM.***pid.ddd* file is moved to the path name specified in the **C.***sysnxxxx* file (discussed below) that caused the transmission. If processing is abnormally terminated, the **TM.***pid.ddd* file may remain in the *X* directory. These files should be automatically removed by **uucleanup**.

LCK (lock file)

Lock files are created in the **/usr/spool/locks** directory for each device in use. Lock files prevent duplicate conversations and multiple attempts to use the same calling device. The names of lock files have the format:

LCK..*str*

where *str* is either a device or computer name. These files may remain in the spool directory if the communications link is unexpectedly dropped (usually on computer crashes). The lock files will be ignored (removed) after the parent process is no longer active. The lock file contains the process ID of the process that created the lock.

C. (work file)

Work files are created in a spool directory when work (file transfers or remote command executions) has been queued for a remote computer. The names of work files have the format:

C.*sysnxxxx*

where *sys* is the name of the remote computer, *n* is the ASCII character representing the grade (priority) of the work, and *xxxx* is the four digit job sequence number assigned by UUCP. Work files contain the following information:

Full pathname of the file to be sent or requested

Full pathname of the destination or user/file name

User login name

List of options

Name of associated data file in the spool directory. If the **uucp −c** or **uuto −p** option was specified, a dummy name (**D.0**) is used

Mode bits of the source file

Remote user's login name to be notified upon completion of the transfer

D. (Data file) Data files are created when it is specified in the command line to copy the source file to the spool directory. The names of data files have the following format:

D.*systmxxxxyyy*

where *systm* is the first five characters in the name of the remote computer, *xxxx* is a four-digit job sequence number assigned by **uucp**. The four digit job sequence number may be followed by a sub-sequence number, *yyy* that is used when there are several **D.** files created for a work (**C.**) file.

X. (Execute file) Execute files are created in the spool directory prior to remote command executions. The names of execute files have the following format:

X.*sysnxxxx*

where *sys* is the name of the remote computer, *n* is the character representing the grade (priority) of the work, and *xxxx* is a four digit sequence number assigned by UUCP. Execute files contain the following information:

- ☐ Requester's login and computer name.
- ☐ Name of file(s) required for execution.
- ☐ Input to be used as the standard input to the command string.
- ☐ Computer and file name to receive standard output from the command execution.
- ☐ Command string.
- ☐ Option lines for return status requests.

Direct Links

Direct links are beneficial only when:

- It is not possible to link the computers together through a Local Area Network (LAN).
- Two computers transfer large amounts of data on a regular basis.
- Two computers are located no more than several hundred cable feet apart.

The distance between two directly linked computers is dependent on the environment in which the cable is run. The standard for RS-232 connections is 50 feet or less with transmission rates as high as 19200 bits per second (bps). As the cable length is increased, noise on the lines may become a problem, which means that the transmission rate must be decreased or limited distance modems be placed on each end of the line.

UUCP and the Ethernet

UMIPS UUCP can be easily configured to use ethernet instead of phone lines or a direct connection. This is especially convenient if you have ethernet connections anyway, as it requires no additional lines.

With ethernet, UUCP connects to a remote machine via a TCP socket (the next chapter discusses the TCP/IP network in detail). A daemon must monitor this socket to verify the login and execute **/usr/lib/uucp/uucico** to accept the connection.

File Changes

The following files require modifications or additions to be made to enable the UUCP connection over ethernet. Note that these changes need to be made in both hosts connected by the ethernet.

/etc/services

Verify that the following line exists (and is not commented out with a preceding "#") in **/etc/services**:

```
uucp        540/tcp             uucpd        # uucp daemon
```

/etc/inetd.conf

Verify that the following line exists (and is not commented out with a preceding "#") in **/etc/inetd.conf**:

```
uucpd stream      tcp   nowait      root  /usr/etc/uucpd    uucpd
```

/usr/lib/uucp/Devices

Verify that the following line exists (and is not commented out with a preceding "#") in **/usr/lib/uucp/Devices**:

```
TCP - 0 Any TCP ""
```

/usr/lib/uucp/Systems

Verify that the following line exists (and is not commented out with a preceding "#") in **/usr/lib/uucp/Systems:**

hostname `Any TCP - - ogin:--ogin` *login* `word:` *password*

where *hostname* is the name of the machine to be contacted and *login* and *password* contain the appropriate account information.

Chapter 10: Remote File Sharing

Introduction

This chapter explains how to configure and use Transmission Control Protocol/Internet Protocol (TCP/IP) communications software on an M-Series system. An M-Series system can communicate across an Ethernet local area network with other hosts and terminals using TCP/IP communications software. TCP/IP offers file transfer and remote login services. To connect an M-Series system to the Ethernet, see "Ethernet Interface" in the M-Series Technical Manual.

This is the 4.3 BSD UNIX version of TCP/IP (except that UMIPS does not support the program **talk** and the UNIX domain socket, UNIX_AF.)

Relevant Documentation

You may find useful information to help you plan and set up your network in this document:

Defense Data Network Protocol Handbook, which is available from the Network Information Center, Defense Data Network, SRI International, Room EJ-291, 333 Ravenswood Ave., Menlo Park, California 94025, telephone: (415) 326-6200

And the following manual pages:

Manual Page	Description:
arp(1M)	Address resolution display and control
ftp(1C)	File transfer program
ftpd(1M)	DARPA Internet File Transfer Protocol server
ifconfig(1M)	Configure network interface parameters
inetd(1M)	Internet "super-server"
hosts(4)	The hosts database
hosts.equiv(4)	Network hosts permissions
netstat(1)	Show network status
rcp(1C)	Copy file to or from a remote system
rhosts(4)	List of trusted users
rlogin(1C)	Login to a remote system
rlogind(1M)	Remote login server
route(1M)	Manually manipulate the routing tables
routed(1M)	Network routing daemon

Software Administration

This section describes the procedures for configuring TCP/IP communications software. It includes these basic configuration procedures:

- Choosing an address class
- Naming your system
- Configuring TCP/IP software

Connecting Networks

The following checklist provides instructions on the necessary steps for configuring the TCP/IP software. Each step is explained in detail below. You must follow the steps listed below for all machines.

- Choose an address class and assign an address for your system. If you are connecting your system to an existing network, ask your network administrator for the address you should assign to your system.
- Select your host number.
- Assign a unique name to your system. (The default name is `no_hostname_set`.)
- Create the host database for your system by editing the file **/etc/hosts**. Edit the file **/etc/hosts.equiv** for controlling host access.
- Update the databases on other systems on the network with your new Internet address and system name. For systems with the UNIX operating system, enter the changes in the **/etc/hosts** file and possibly **/etc/hosts.equiv** files.
- Read about and try the network application programs described in Procedure 10. Among other functions, these programs send and receive files, and log in to remote systems.
- And possibly, create gateway mappings for routing table initialization by adding **route** entries to the file **/etc/init.d/tcp**. If you are connecting your system to an existing network in which gateways have already been established, obtain the gateway addresses from your network administrator.

Choosing an Address Class

TCP/IP uses two addresses to uniquely identify each node in a network: a physical Ethernet address and a logical Internet address. The Ethernet address is a 6-byte physical address that identifies the Ethernet board. Because it is burned into the hardware, it never changes.

The Internet address is a 4-byte logical address that identifies the host on the network. The logical address is in two parts: the network number followed by the host number. You can specify addresses in decimal, octal, or hexadecimal. The default is decimal. (Use a leading x or X for hexadecimal notation, and use a leading 0 for octal notation.) See **inet**(3N) for more information on the dot (.) notation employed below.

There are three Internet address classes: Class A, Class B, and Class C. The most significant bits of the address determine the address class. Each address has the form:

#.#.#.#

where "#" is either a host or network designation. All Internet addresses on a particular network must be of the same class.

The number of bytes comprising the *net* and *host* portions of address differs according to the address class. These differences are illustrated in Figure 10-1.

Class A addresses:

net.host.host.host

Class B addresses:

net.net.host.host

Class C addresses:

net.net.net.host

Figure 10-1: Net and Host Divisions of Addresses

To specify an Internet address, you must list all four bytes of the address. If any of the bytes is zero (0), you must use a 0 to specify that byte (see below for examples).

NOTE: If you are connecting your system to an existing network, ask your network administrator for the appropriate network address.

In any network, all Internet addresses must be of the same class. If you are connecting your system to an existing network, ask your network administrator for the address class and network number. If you are building a new network, see the information on assigning network classes below.

The M-Series system is shipped with only the local host address in the **/etc/hosts** file:

```
127.1 localhost
```

If you do not plan to connect your network to a larger network and you do not already have a network number, 192.0.0.1 is a generic network number that you can use. For example, to start a two-machine network, add the following to your **/etc/hosts** file:

```
192.0.0.1   onemips
192.0.0.2   twomips
```

If you plan to connect your local network to a larger network, you must apply for an address through the agency that administers the network to which you wish to connect. For example, the Network Information Center (NIC) at Stanford Research Institute (SRI) is responsible for all network numbers assignments for the Internet. If you have questions about Internet network numbers or want further network information, call 1-800-235-3155 or contact the Hostmaster at the NIC. The network address is:

HOSTMASTER@SRI-NIC.ARPA

Each of the three classes of network is described below. (Network numbers are shown in boldface type and host numbers are shown in italics.)

Class A Internet Addresses

As Figure 10-1 illustrates, in a class A address, the *net* portion of the address consists of one byte, and the *host* portion consists of three bytes.

In a class A address, the most significant bit of the *net* portion of the address must be 0. Because the most significant bit is 0, the possible range for the network identification number is from 1 to 127 (decimal), instead of 1 to 254. The *host* portion of the address consists of three bytes (24 bits). You must specify each byte of the 24-bit host address separately using dot notation. The possible range of values for each byte of the host address is 1 to 254. The possible range of values (in decimal) for each byte of the address is shown below:

[0..127].*[1..254].[1..254].[1..254]*

This is an example of a class A address:

42.*9.7.1*

In the above address, the network identification number is 42. It uses one byte of the specification. The host address is 9.7.1.

Class B Internet Addresses

As Figure 10-1 illustrates, in a Class B address the *net* and *host* portions each consist of two bytes. In a Class B address, the two most significant bits of the first byte of the *net* portion of the address must be 10 (in binary). If the most significant bits are 10, then you must use only the remaining six bits of the first byte and all 64 bits of the second byte to specify the rest of network identification number. This means the possible range (in decimal) for the first byte of the *net* portion of the address is 128 to 191 and the range for the second byte is 1 to 254. Because the *host* portion of the address also consists of two bytes, there are 216 possible host identification numbers. The possible range (in decimal) for the two bytes of the *host* portion of the address is 1 to 254. The possible range of values (in decimal) for each byte is shown below:

[128..191].[0..254].*[1..254].[1..254]*

This is an example of a class B address:

128.0.1.1

In the above address, the network identification number is 128.0. It uses two bytes of the specification. The host number is 1.1.

Class C Internet Addresses

As shown in Figure 10-1, in a Class C address the *net* portion of the address consists of three bytes and the *host* portion of the address consists of one byte. In a Class C address, the three most significant bits of the *net* portion of the address must be 110 (in binary). If the most significant bits are 110, then you use the remaining five bits of the first byte and the 16 bits of the second and third bytes to specify the the rest of network identification number. This means there are 221 network identification numbers. For the *net* portion of the address, the possible range of values (in decimal) is 192 to 223 for the first byte of the address and 1 to 254 for the second and third bytes. Since the *host* portion of the address consists of one byte, there are 28 host identification numbers. This means the possible range of values for the host identification number is 1 to 254. The possible range of values (in decimal) for each byte is shown below:

[192..223].[0..254].[0..254].*[1..254]*

This is an example of a class C address:

192.0.0.*1*

In the above address, the network identification number is 192.0.0. It uses three bytes of the specification. The host number is 1.

Selecting a Host Number

After you select an address class, you need to select a specific Internet host number for your system. As previously mentioned, the TCP/IP protocols use logical Internet addresses while the Ethernet hardware uses physical Ethernet addresses. Therefore, in order to send messages between two systems connected by the Ethernet, the systems must be able to translate the logical Internet addresses into the physical Ethernet addresses. This translation is done with the Address Resolution Protocol (ARP).

The sending system broadcasts a message containing the destination system's Internet address. The sending system then waits for the destination system to send back its physical Ethernet address. After the sending system receives the Ethernet address, it associates this Ethernet address with the destination system's Internet address. The sending system then stores this translation and the Ethernet address in a table for later use. Each system on the network maintains a table of recently used address translations.

Thus, the ARP protocol allows you to choose any host number as long as it is unique and is the same class as the other systems on the network. The only number that you cannot use is 127. This is the localhost entry number, which is used for testing. **/etc/hosts** lists all system names and addresses that your system can access.

Normally, hosts using the ARP method of address translation can communicate only with other hosts that support the ARP method. The TCP/IP software, however, supports communication with hosts that do not support ARP. To establish communication with a host that does not support ARP, add the mapping for its Ethernet address to the Internet-to-Ethernet address translation table on your system. To do this, use the **arp**(1M) utility.

To use the **arp** utility, follow these steps:

1. On the system that supports ARP, log in as **root**.

   login: **root**

2. Add information about the system that does not support ARP (host-name) to the translation table. Type:

   # **arp -s** *host_name Ethernet_address* **pub**

   The **−s** option tells **arp** to add the entry that follows to the translation table. The **pub** option publishes the entry. That is, it enables a host to respond to an ARP broadcast even if the specified Internet address is not its own address. (See **arp**(1M) for more information.)

   Once you add an entry to the table, it remains in the translation table until you reboot. If you want to make the entry you add permanent, you can do this two ways:

   - Edit **/etc/init.d/tcp**. Find this line:

     # Add routing commands here

     Add this line:

     arp -s *host_name Ethernet_address* **pub**

   - Use the **arp −d** option to delete an entry.
   - To insert multiple entries in the translation table by reading them from a text file, type:

     arp -f *filename*

   - To display all the current entries in the table, type:

     # **arp -a**

     The system responds with a list, similar to this:

```
opus      (192.60.0.1)    at  8:0:14:60:0:1
george    (192.10.1.81)   at  8:14:10:10:1:81
fred      (192.0.0.65)    at  8:0:14:0:0:65
```

Naming the System

The default name of a new M-Series system is `no_hostname_set`. If you have more than one system on a network, you must assign each system a unique name. The name can be up to eight characters long and cannot contain blank spaces.

The name must be in lowercase letters. To change the name of a system, edit the files **/etc/hosts** and **/etc/local_hostname** and reboot the system. See the manual pages for **hosts**(4) and **/etc/local_hostname**(4).

All the systems on your network must add the name of your system to their **/etc/hosts** files. If you are adding a new machine to an existing network, edit the **/etc/hosts** file on your new system to include the name of another system on your network. Then, log in to the other system and copy the **/etc/hosts** file from the old system to the new one. The network administrator should add the new system name to their **/etc/hosts** file and update the **/etc/hosts** file on the other systems.

The /etc/local_hostname File

The **/etc/local_hostname** file contains your hostname as well as your netmask and broadcast address. If you are connecting to an existing network, use the netmask and broadcast address used by the other machines on the network. If you are building a new network, substitute the network portion of your address for the "255"s in the network portion of the broacast address. For example, if you are using a Class C network address of 192.0.0, your **/etc/local_hostname** file would look something like this:

```
newmips
netmask 0xff000000 broadcast 192.255.255.255
```

(Note that networks conforming to the BSD 4.2 protocol use "0"s instead of "255"s when specifying the broadcast address.)

Configuring TCP/IP Software

After you establish an address for your network and your system, you need to configure your TCP/IP software to fit your needs. Specifically, you must perform the following tasks:

- Add your new host name to the host data base
- Set remote access privileges

Adding a New Host

The file **/etc/hosts** is the host names database. It contains mappings between the Internet addresses and the names and aliases for the systems on the network. This file exists on each UNIX system on a network. When you reference a host by name in an application program, the application program uses this file to determine the Internet address of the host to which you are referring.

The **/etc/hosts** file can contain two elements:

- lines of text with two or more fields
- optional comments that begin with a pound sign (#) and continue to the end of the line

The first field of the line is the Internet address, which consists of a network number and a host number. In the entry for your machine, the Internet address contains the network number that you obtained or selected according to the instructions above.

NOTE: The first field must be followed by one or more spaces or tabs.

The second field contains the name of the system that is associated with the Internet address in the first field. Figure 10-2 shows an example of an **/etc/hosts** file.

```
#
# Internet Host Database
# test entry
127.0.0.1 localhost
# my machine
31.0.3.13  mips
# active entries
31.0.0.48  cyrano
31.0.0.57  percival
31.0.0.13  arthur
31.0.3.59  zurich
31.0.0.66  opus
31.0.0.81  dagwood
31.0.0.10  mac
31.0.0.11  george
31.0.0.26  blondie
```

Figure 10-2: Sample /etc/hosts File

If you are connecting to an existing network, obtain a copy of the **/etc/hosts** file from your network administrator. If you are building a new network, add the name and address for your system as well as for any other systems on the network to **/etc/hosts**. If you are going to use your system as a gateway, it requires two host names for your system in your **/etc/hosts**.

Figure 10-2 shows an entry called *localhost*. This entry is a special address that you use to test the TCP/IP software. When you reference this address, the message is looped back internally; it is never physically transmitted across the network. This localhost entry is a Class A address with network number 127 and host number 1 (127.0.0.1).

Setting Remote Access Privileges

The **/etc/hosts.equiv** file contains a list of remote systems with which the local system is equivalent, that is, with which it shares account names. This file is used by the application programs **rlogin**, **rsh**, and **rcp** to allow remote access to accounts on the local system.

There are two different ways to set up remote access privileges:

- Create a list of systems that can access your system. To do this, you edit **/etc/hosts.equiv**.
- Create a list of other users that can access your system. To do this, you edit **$HOME/.rhosts**.

/etc/hosts.equiv contains lines of text with one or more fields that are separated by exactly one blank space. A line consisting of a simple host name means that anyone may log in from that host.

Figure 10-3 shows an example **/etc/hosts.equiv** file that corresponds to the **/etc/hosts** file shown in Figure 10-2.

NOTE

The **root** login is not equivalent across systems listed in **/etc/hosts.equiv**. If you want to enable root permissions across machines (are you sure?), the machine(s) must be listed in **/.rhosts**.

```
localhost
mips
cyrano
percival
arthur
zurich
opus
dagwood
mac
george
blondie
```

Figure 10-3: Sample /etc/hosts.equiv File

Each line in **/etc/hosts.equiv** is the name of a host that can access the local system. If you are connecting your system to an existing network, ask your network or system administrator for the host names that you should include in this file. If you are building a new network, edit the **/etc/hosts.equiv** so that it contains the names of hosts that should have access to the files on your system. Include only those hosts in **hosts.equiv** that are equivalent to yours in security and administration.

Customizing .rhosts

You can use the **.rhosts** file to control remote access to systems. This file serves the same purpose as **/etc/hosts.equiv**, but gives access to individual users. The **.rhosts** file is located in a user's home directory and specifies the remote systems and users that can access the local system under the login name of that user's directory.

If you are listed in another user's **.rhosts** file on a remote machine, you can log in to that machine with your login id and you have all the same privileges as the user who listed you in their **.rhosts** file. For example, user **sam**'s **rhosts** file is shown in Figure 10-4. In this example, *donna*, *fred*, and *rich* use the machine *mips* and have all

of *sam*'s privileges on his local machine. The **rhosts** file is used to validate a user when the name of the remote system does not exist in the **/etc/hosts.equiv** file.

.rhosts contains lines of text with fields for host and user separated by exactly one space. Tab characters are not allowed. Each line contains the name of the remote systems and the users on that remote system who can access the local system.

```
mips fred
mips rich
mips donna
zurich edwardo
opus bloom
opus pickles
dagwood daisy
mac henry
blondie marilyn
```

Figure 10-4: Example of a **.rhosts** File

If the example **.rhosts** file above, was in user *joe*'s directory on host *penguin*, for example, user *fred* on *mips* could "rlogin penguin -l joe" and gain access to *penguin* as *joe*. (See the discussion on using **rlogin** in this chapter.)

Network Security

This section contains information on two network security issues: network access control and the TCP/IP security algorithm.

Controlling Network Access

The configuration files **/etc/hosts.equiv** and **rhosts** maintain network security by controlling user access from remote systems to the local system.

Users on remote systems whose systems are listed in the local system's **/etc/hosts.equiv** file can gain access to all files on the local system without using a password. **hosts.equiv** makes users on one machine equivalent to users on another machine. For example, user *paul* is a local user on system *chess*. If system *chess* is listed in system *spider*'s **host.equiv** file, then *paul* can use the **rsh, rlogin**, and **rcp** commands without supplying a password. The **host.equiv** makes the remote user equivalent to the local user with the same user name. Therefore, *paul* is *paul* on both *chess* and *spider*. See **hosts.equiv**(4).

.rhosts is in **root**'s home directory. This file allows the superuser or permitted users on a remote host to log in as the superuser on the local host without specifying a password. **hosts.equiv** does not permit root access. See **rhosts**(4).

Another network security feature is configuration file ownership. You must be logged in as **root** in order to edit most of the configuration files.

Warnings

Use **/.rhosts** only if all systems and their consoles are physically secure. Given access to a console with **/.rhosts** privileges, anyone can log in as any user, including the superuser, and become **root** on any system that has your system's name and *root* in its **.rhosts** file.

Be very selective about the systems you add to **hosts.equiv** because adding a system to **hosts.equiv** makes all users on the remote system equivalent to users on the local system. Making users on systems equivalent can be dangerous. For example, if **host_A** lists **host_B** in its **/etc/hosts.equiv** file, then a user with superuser privileges on **host_B** can create an account with any name that matches a user on **host_A**, thus gaining access to that user's files on **host_A**, if **hosts.equiv** is not restricted to specific users.

Securing the Network

You can maintain network security and allow users to have access to several machines by using three strategies:

- When more than one machine has the same protection domain (i.e., they are equivalent) , all the machines within that protection domain have identical **hosts.equiv** files. All users, except for **root** in this case, can use the **rlogin**, **rcp**, and **rsh** commands without supplying a password.
- When the user **root** on all machines within the same protection domain is put in each machine's **.rhosts** file, any user logged in as **root** on one machine can use the **rlogin**, **rcp**, and **rsh** commands without supplying a password.
- When one user has the same name on several machines, this user is listed in the **.rhosts** file in his home directory on each machine. This user can use the **rlogin**, **rcp**, and **rsh** commands without supplying a password.

Initializing and Checking Your Network Setup

To initialize your system with the new contents of the various network files, reboot it. When you are back to multiuser mode, you can use **ping**(1M) and **ifconfig**(1M) to verify your setup.

/usr/etc/ping

"Ping" a machine to see if the basic communication protocols are working. First, ping your own machine:

```
$ /usr/etc/ping  localhost
```

After a few seconds, enter an interrupt (usually CONTROL-C or the DELETE key, enter "stty -a" on your system if you don't know what your interrupt character is). You will see some transmission statistics including the message "0% packet loss". If you do not get 0% but instead get some percentage of lost packets, re-check all the steps described in this chapter. If everything seems allright, ping another machine on the network that is contained in your **/etc/hosts** and is cuurently in multiuser mode:

```
$ /usr/etc/ping  <hostname>
```

Again, you should get "0% packet loss".

/usr/etc/ifconfig

ifconfig can be used to modify various network parameters that are normally set at boottime, and it can also be used to report the current network settings. For example, enter:

```
$ /usr/etc/ifconfig enp0
```

and it will echo back your internet address, netmask and broadcast address (as well as other things). For more information, refer to **ifconfig**(1M).

Connecting Networks

After you set up your local network, you can connect it to other networks. These networks must be connected physically through a **gateway**. A gateway is an intermediate system that controls the flow of information between two (or more) systems. A **route** is a sequence of host names through which information is sent to its destination host. A route may go through one or more gateways. When you send information to a host on a remote network, it looks like the information is sent directly to the host; however, the information is actually sent to a device that channels it to a remote host. In order to set up your system as a gateway, it must have two ethernet connections. If your system cannot support two ethernet connections, another host must serve as the gateway.

Linking to Another Network

To link to another network, you must include details about the network in the file **/etc/networks**, which is the network host database. It contains mappings between networks and their Internet addresses. You must be the superuser to edit this file. See **networks**(4).

Large networks, particularly those connected to the Internet, are administered by the Network Information Center (NIC), operated by Stanford Research Institute (SRI), in Palo Alto, CA. Contact the NIC for information on becoming a part of a large network.

You must create a map of the gateways that your system will use. You can create these gateways by using the **route** utility or by editing **/etc/init.d/tcp**. All mappings between hosts and gateways appear in the routing table. To access or change the mappings in the routing table use the **route**(1M) utility.

When the network is up and running, the routing table resides in the TCP kernel. The initial entries in the table are based on the **route** commands in the **/etc/init.d/tcp** file.

Use the **netstat −r** command to display gateway entries in the routing table. Usually, the route daemon **routed** maintains the routing tables automatically.

route Utility

The **route**(1M) utility lets you add or delete gateway addresses for inter-network communication. To add an address, type:

```
# route add destination gateway metric
```

To delete an address, type:

```
# route delete destination gateway metric
```

destination is either the name of a host on the remote network or it is the entire remote network. *gateway* is the gateway that connects the two networks. *metric* refers to the number of gateways information must go through to get to its destination. If you do not specify the metric value, *route* assumes a value of 0, which implies no gateways.

You can specify both *destination* and *gateway* as either names or Internet addresses. Internet addresses must be in the same class as the network class. You must list both *destination* and *gateway* in your **/etc/hosts** file; *gateway* must be listed in **/etc/networks**.

Below are examples of commands for adding and deleting a new route using the destination network. In each example, the system's response to the command contains the Internet addresses (in 4-part dot notation) that corresponds to the destination and gateway given in the command. See **inet**(7) for information on dot notation.

NOTE: You must be logged in as **root** to add or delete gateways from the routing table.

Displaying Routes

To display a route, type:

 # netstat −r

The system displays this information:

```
Routing tables

Destination     Gateway      Flags Refcnt   Use      Interface
localhost       localhost    UH    1        0        lo0
44              norman       UG    5        42151    ex0
194.26.51       alfred       U     1        18231    ex0
194.26.52       gate-TBsrc   U     1        65534    ex1
194.26.53       gate-owls    UG    5        3853     ex0
194.26.54       gate-goose   UG    2        28145    ex0
194.26.55       194.26.51.7  UG    1        741      ex0
```

Adding a New Route

The following example adds a new route and gateway. The new route goes to the network **sanjose**. This route is through the gateway **sj**. The metric number indicates the number of gateways in this route. The metric number is one (1) in this example. To add the route, type:

 # route add sanjose sj 1

After you enter this command, the system responds:

```
add  network  21.0.0.0  gateway  90.1.32.131
```

Deleting a Route

The following example deletes a route and a gateway. To remove **sj** as a route through which the network **sanjose** can be reached, delete the entry from the routing table. To delete the route in this example, type:

 # route delete sanjose sj 1

After you enter this command, the system responds:

```
delete  network  21.0.0.0  gateway  90.1.32.131
```

Setting Up a Gateway

You must have two Ethernet boards in your M-Series system to use it as a gateway. Each Ethernet board has a unique Ethernet and Internet address and a unique host name. The entry in **/etc/hosts** for an system that is a gateway would look like this:

```
191.25.50.10      gate-marketing
191.25.51.10      penguin
```

To use your M-Series system as a gateway, follow these steps:

1. Become the superuser.

   $ **su**

2. Use the file **/etc/init.d/tcp** to configure gateways, modify the routing table, and activate the **rwho** utility. **tcp** is an initialization script that is invoked by **/etc/init**, which is executed when you put the system into multi-user mode. **tcp** performs TCP/IP-specific initializations on gateways. **/usr/etc/ifconfig** initializes the Ethernet interface and causes **tcp** to start the TCP/IP daemon **inetd**.

   Edit **/etc/init.d/tcp** to make your system function as a gateway. Find these lines:

   ```
   #  change and install the following line for gatewaying
   #  ifconfig ex1 inet other-hostname
      ifconfig ex1 inet penguin
   ```

   You must give the system an alternative host name when you make it a gateway. You must put both names for your system in **/etc/hosts**. To enable the sysem as a gateway, change the entry for **other-hostname** to your machine's alternative host name and remove the comment mark (#) from that line. Below is an example of these lines in **/etc/init.d/tcp** changed to use the host *penguin*.

   ```
   #  change and install the following line(s) for gatewaying
     /usr/etc/ifconfig en1 inet penguin
   #  /usr/etc/ifconfig enp0 inet other-hostname2
   #  /usr/etc/ifconfig enp1 inet other-hostname3
   ```

3. Edit **tcp** to modify the script if you need to change the automatically inferred routing tables. Find this line and substitute either the host name or the Internet address for either the destination or gateway machine.

   ```
   #  Example: "route add destination gateway 1"
   ```

   Here is an example of the above line with the destination and gateway machines specified.

```
route  add  walrus  northpole  2
```

You must use static routing to work with hosts that do not run **routed**(1M). The easiest way to specify static routing is in **/etc/gateways**. To use static routing with **route**, follow these steps:

1. Log in as **root**.

   ```
   login: root
   ```

2. Edit the file **/etc/init.d/tcp**. Search for the following comment:

   ```
   # Add routing commands here
   ```

   Below this comment is another comment that shows how to specify the gateway mappings:

   ```
   # Example: "route add destination gateway 1"
   ```

   Add the gateway mappings for your system to the file. If you are connecting to a preexisting network, get the gateway addresses from your network administrator. For example, to map the gateway *sj* to the host *sanjose*, add this line:

   ```
   /usr/etc/net/route  add  sanjose  sj  2
   ```

3. To effect the changes you specify in gateway mapping, reboot the system.

Maintaining the Network

This section provides information on network statistics.

Network Statistics

The **netstat** command provides statistical displays of network performance. **netstat** has the following options:

−A shows the address of any associated protocol control blocks. Use this for debugging.

−a shows the state of all sockets (normally, sockets used by server processes are not shown)

−i shows the state of interfaces that have been automatically configured. Interfaces that are statically configured into a system, but are not located at boot time, are not shown.

−m shows statistics recorded by the memory management routines (the network manages a private share of memory).

−n shows network addresses as numbers (normally *netstat* interprets addresses and attempts to display them symbolically).

−s shows statistics for each protocol.

−r shows the routing tables.

Network Application Utilities

This section describes the network application utilities included in the TCP/IP communications software:

rcp	copies a file from one UNIX system to another.
rsh	executes a command on a remote host running UNIX.
rlogin	initiates a login on a remote host running UNIX.
rwho	displays a list of the current users on remote UNIX hosts.
ruptime	displays the status of remote UMIPS systems.

Using rcp, the Remote Copy Program

rcp copies a file from one system to another. You specify the source machine, user, and the pathname for the file, followed by the destination machine, user, and the destination pathname for the file:

rcp *[[user@]host:]pathname [user@][destination:]pathname*

The square brackets indicate that the information contained within them is optional. If you do not specify a name for either the source or destination machines, the system assumes the local machine.

Below are some examples for using **rcp**. In these examples, you must either have accounts on both hosts with the same user name or else specify the user's and host's name as shown below.

user@host

or

host.user:pathname

The user names on each machine must be equivalent to each other, through either **/etc/hosts.equiv** or the user's **.rhosts** file.

Copying Files from Local to Remote Machines

This example copies the file **sqiral.c** in the current directory on the local machine to the file **sqiral.c** in the directory **/oh4/doc/install** on a destination machine named *opus*. The system assumes that the local machine is the source of the file, because no machine is specified for the file **sqiral.c**.

$ **rcp sqiral.c opus:/oh4/doc/install/sqiral.c**

A **\&.rhosts** file can help if you have difficulty transferring files to a specific directory in your home directory. Follow this format to transfer files to a specific directory.

$ **rcp** *filename user host:~user/pathname*

Another way to solve the file transfer problem is to copy the files to **/tmp**.

$ **rcp sqiral.c opus:/tmp**

Copying Files from Remote to Local Machines

This example copies the file **/usr/include/stdio.h** from the remote machine **sting** to the file **test** on the local machine. The system assumes that the local machine is the source of the file, because no machine is specified for the file **stdio.h**.

```
$ rcp  sting:/usr/include/stdio.h  test
```

Copying Files between Remote Machines

This example copies the file **temp_vi** from a remote machine named *puppy* to a file with the same name on a remote machine named **sting.** The system assumes that the user's directory exists on both machines since no home directory is specified.

```
$ rcp  puppy:temp_vi  sting:temp_vi
```

Copying Directory Trees

You can also use **rcp** to copy the entire directory tree from a remote machine to the local machine. The **−r** option causes recursive copying that copies the directory tree. In this example, all files and directories from **/usr/include** on the remote machine **sting** are copied to the directory **localinclude** on the local host.

```
$  rcp  -r  sting:/usr/include  localinclude
```

Using rsh, the Remote Shell Program

rsh connects your system to a remote host and executes the commands you specify.

Note that this is not the same as **/bin/rsh**, the restricted shell. The remote shell is **/usr/net/rsh** or **/usr/net/remsh**, both of which are linked to **/usr/ucb/rsh**. If **/usr/ucb** or**/usr/net** are not in your $PATH before **/usr/bin**, you must specify the full path name or you will get the restricted shell instead of the remote shell.

Like **rcp**, this network utility assumes that you have accounts under the same user name on both the remote and local hosts. This is the **rsh** syntax:

```
$ rsh  hostname  [-l  username]  [-n]  command
```

For example, to find the load average as user "pat" on another machine, type:

```
$ rsh  mips  −l  pat  uptime
```

The **-n** option redirects the input of the **rsh** to **/dev/null**.

If you do not specify a command for **rsh** to execute, **rsh** initiates an **rlogin** (remote login) on the remote machine. (The **rlogin** command is described below.)

Interactive commands such as **vi**(1) do not run correctly when executed using **rsh**.

Using rlogin, the Remote Login Program

rlogin initiates a login on a remote host across the network. The program takes the remote system name as an argument. This is the **rlogin** syntax:

$ **rlogin** *hostname [−l username]*

For example, to log in remotely to a system named *opus*, type:

$ **rlogin opus**

The specified remote system must be listed in the file **/etc/hosts**.

If your system is listed in the remote host's **/etc/hosts.equiv** file, you can **rlogin** to the remote host without a user name or password. Alternatively, you can provide an **.rhosts** file in your home directory on a remote machine. A **.rhosts** file specifies an account on a host that is considered to be equivalent to your local account. The **.rhost** file allows you to log in to a remote system as another user. In this example, the user logs in to the system *opus* as a user named *rick*.

$ **rlogin opus −l rick**

This example assumes that the user, *rick*, has an account on both the local and remote systems.

Using rwho, the Remote Who Program

The **rwho**(1C) command generates a list of users on all UMIPS systems on the local network. In addition to the user's login name, **rwho** lists the terminal name, the system name, and the login time for each user currently logged in. The list of hosts used to generate the names is obtained from the file **/usr/spool/rwho**.

To display a list of all users logged in to UMIPS hosts on the network, enter:

$ **rwho**

A list of users similar to the example below appears on the screen:

```
percy   opus:tty15     Mar 14 08:17
arnold  opus:ttyp4     Mar 14 08:35:25
root    knot:console   Mar 12 15:42
mary    knot:tty03     Mar 14 09:38
```

In this display, the first column lists the name of the user; the second column lists the name of the host followed by the terminal name; the third column lists the login date and time as follows:

```
month day hour:minute:second
```

If a user has not typed to the system for a minute or more, **rwho** reports this idle time. If the user has not typed to the system for one hour or more, **rwho** does not display that user at all. To display a list of users that includes those who have not

used their system for more than one hour, use the **–a** argument to **rwho**.

Using ruptime, the Remote Uptime Program

The **ruptime**(1C) utility displays the status of all UMIPS hosts on the local network. Along with the name of the host, **ruptime** displays the current time, how long the system has been up, and the average number of jobs in the run queue for each system.

To display the status information for all UMIPS systems on the network, type:

$ **ruptime**

The status list displayed is similar to this example:

```
opus        up       12+18:544 users, load 1.10, 0.83, 0.75
knot        down     18+13:25
percival    up 5+02:4510 users, load 4.51, 3.87, 3.51
```

In the status list the first column displays the system name, and the second column indicates whether the system is up or down and the amount of time it has been in this condition. This column uses this format:

```
days + hours:minutes
```

The third column lists the number of users logged in to the system and the average number of jobs in the event queue in the last 5, 10, and 15 minutes.

ruptime does not report users who are idle for more than one hour unless you use the **–a** flag with **ruptime**.

Using ARPANET Utilities

The utilities **telnet** and **ftp** are documented in this section. **telnet** and **ftp** are ARPANET-derived. This means they are commonly used in the ARPANET community. **rlogin** and **rcp** perform similar functions to **telnet** and **ftp**, and are faster and more reliable.

Using telnet, a Remote Login Program

Like **rlogin, telnet** initiates a login on a remote host across the network. **telnet** uses the TELNET protocol for remote logins.

To use the **telnet** program, follow these steps:

1. To start the **telnet** program, use the name of the remote machine as an argument. For example, to log in remotely to a machine named *puppy*, enter:

   $ **telnet puppy**

   The screen shows:

   ```
   Trying...
   Connected to puppy.
   Escape character is ^].

   login:
   ```

2. Execute commands on the remote machine.
3. To disconnect from the remote machine, type **logout**. If you do not get the local machine prompt, exit the **telnet** program by pressing <**CTRL-**]>. This gets the attention of the **telnet** program. At the "telnet>" prompt, enter "quit":

   telnet> **quit**

   The **telnet** program has these options:

? *[command]*	Get help on commands.
status	Show status of **telnet**.
escape *[esc-char]*	Change character for exiting **telnet**.
close	Close a TELNET session.
open *host_[port]*	Open connection on host.
sendcrlf	Send a carriage return/line feed.

See **telnet**(1C) for details.

Using ftp, the File Transfer Program

ftp is a program for transferring files using the ARPANET File Transfer Protocol. Below is a brief description of some of the *ftp* commands. For more information, see **ftp**(1C). The procedure below describes how to enter and exit **ftp**.

1. To start the **ftp** program, type:

   $ **ftp** *hostname*

   hostname is the name of the machine with which you wish to communicate. For example, to connect to the host named **abbott**, type:

   $ **ftp abbott**

   After you execute the above command, the screen shows:

```
Connected to abbott.
220 FTP server (Version 4.81 Mon Sep 26 08:36:28 PDT 1983)
ready.
```

   The host then requests your name and password. This example assumes the user's name is **peter**.

```
Remote User Name: peter
331 Password required for peter.
```

   Enter your password and the host replies:

```
230 User peter logged in.
```

   After you have logged in, the **ftp** prompts appears:

```
ftp>
```

2. Now you are ready to send and receive files. (File transfer commands are described below.) To see a list of the commands for **ftp**, enter "help":

   ftp> **help**

3. To exit from **ftp** after you finish transferring files, enter "quit":

   ftp> **quit**

ftp Commands

Each section below describes some of the commands that you can use with **ftp**. Other commands are available; see **ftp**(1C) for more information. Both the command syntax and an example for that command are given.

Transferring Files from Local to Remote Machines

To send one file to the remote host, use this syntax:

```
ftp> send localfile [remotefile]
```

For example, the following command sends the local file **myfile** to the remote machine using the same file name as the file on the local machine:

```
ftp> send myfile
```

ftp displays messages to indicate that the file has been sent to the remote machine. An **ftp** message generated by the above command looks like:

```
1200  PORT command okay.
150  Opening Data connection for myfile (47.0.0.131,1601)
226  Transfer complete.
21 bytes send in 0.031 seconds (0.62 Kbytes/s)
```

To append a file to another file on the remote machine, use this syntax:

```
ftp> append localfile remotefile
```

For example, to append the file **dictionary.a** to **dictionary.b**, type:

```
ftp> append dictionary.a dictionary.b
```

The current working directory on the remote machine is the assumed destination for the file. If you do not specify a remote file name, the local file name is used as the file name on the remote machine.

To transfer multiple files from the local machine to a directory on the remote machine, type:

```
ftp> cd remote-directory
ftp> mput localfile1 localfile2 ...
```

For example, to transfer the files **thisfile** and **thatfile** to the directory **/usr/john/filexfer**, type:

```
ftp> cd /usr/john/filexfer
ftp> mput thisfile thatfile
```

ftp displays messages to indicate that the files have been sent to the remote machine.

Transferring Files from Remote to Local Machines

To bring a file from the remote machine to the local machine, type:

```
ftp> get remotefile localfile
```

For example, this command gets the file **yourfile** from the remote host and stores it on the local machine in the current directory as **myfile**:

```
ftp> get yourfile myfile
```

ftp prints messages to indicate that the file has been received by the local machine.

Another command for obtaining a file from a remote machine has this syntax:

```
ftp> recv remotefile [localfile]
```

For example, to get the file **sqiral.c** from the remote machine and give it the same name on the local machine, type:

```
ftp> recv sqiral.c
```

To transfer multiple files from the remote machine to the local machine into the current directory, type:

```
ftp> mget remotefile1 remotefile2 ...
```

For example, to transfer the files **localinclude** and **graph.c**, type:

```
ftp> mget localinclude graph.c
```

The screen indicates that the files have been received.

Executing Local Commands

To invoke commands on the local machine, precede the command with an exclamation point (!). For example, to see if the file that you sent to the local machine has been received, type:

```
ftp> !ls
```

This command displays a listing of the current directory on the local machine.

To put a listing of remote files in a local file, type:

```
ftp> mls remotefiles localfile
```

For example, to make a listing of the files **local.h**, **float.h**, and **graphics.h** in the file **includefiles** on the local machine, type:

```
ftp> mls local.h float.h graphics.h includefiles
```

To change directories on the local machine, type:

```
ftp> lcd directory
```

For example, to change your local directory to **fish** on the local machine, type:

```
ftp> lcd fish
```

Executing Remote Commands

Commands on the remote machine do not need to be preceded by an exclamation point. The commands explained below fall into this category. To change your working directory on the remote machine, use the command:

```
ftp> cd remote_directory
```

For example, to change your working directory to **localinclude**, type:

```
ftp> cd localinclude
```

To print the current working directory on the remote machine, type:

```
ftp> pwd
```

To print the contents of a remote directory, type:

```
ftp> dir [remote_directory] [localfile]
```

If you specify a local file name, the contents of the directory are placed in this local file. If you do not specify a local file name, the list is displayed on the screen. If you do not specify a remote directory, the current working directory is assumed.

Transferring Binary or ASCII Files

You can transfer either binary or ASCII files with **ftp**. By default, **ftp** transfers ASCII files. To transfer binary files, type:

```
ftp> binary
```

After issuing this command, any subsequent files are transferred in binary mode. To return to the default of transferring ASCII files, type:

```
ftp> ascii
```

Chapter 11: NFS User's Guide

Introduction

Network File System (NFS) allows you to mount file systems from other machines, and use them as though they actually reside on your own machine.

MIPS Computer Systems, Inc. has adapted NFS 3.0 from Sun Microsystems, Inc. for use on the M-Series systems.

Getting Started

This chapter makes the following assumptions:

- Your system is up and running and you know how to boot it.
- You are familiar with the UMIPS operating system and can use a text editor operating under that system.
- You are familiar with TCP/IP commands. If not, read the **TCP/IP User's Guide** chapter in this document.

The sections below summarize the terms, key files, and commands used by NFS.

Terminology

NFS uses a number of terms which have specific meaning in the NFS environment. These terms are defined below.

mount When you use the **mount** command, it announces to your machine that there is a file system that is to be attached to the file tree at a specified point in your directory. An NFS **mount** allows you to put a file system from another machine on your machine. Mounting a file system is analogous to grafting a branch on a tree.

export To **export** a file system means to allow other machines to mount your file system on their machines. You can specify which file systems you want to export in the **/etc/exports** file. A machine can export only its own file systems.

server A *server* is any machine that provides the NFS file systems for export. In NFS, servers are entirely passive. The servers wait for clients to call them; they never call the clients.

client A *client* is any entity that accesses a network service. A client can be anything from an actual machine to a UMIPS process generated by a piece of software.

A machine can be both a server and a client. An NFS client may choose to mount file systems from any number of servers at any time. The client initiates the binding and the server completes the binding. Since most network administration problems occur at bind time, a system administrator should know how a client binds to a server and how the server controls access.

NFS Permissions

The configuration files listed below are used by NFS to allow access by remote machines to local file systems.

/etc/hosts This file contains a list of Internet address and machine name pairs (for example, 42.0.0.5 **elvis**) for use with TCP/IP. This file must be kept current and must include all the machines with which you want to communicate.

/etc/exports This file is used by NFS to grant mount permissions to other machines. Each line of the file contains a directory plus an optional list of machines. For example:

```
/          abbott  costello
/usr
```

This example accomplishes two things: 1) it allows users on *abbott* and *costello* to mount any directory on the local machine's root file system, and 2) allows users on any machine to mount any directory on the local machine's **/usr** file system.

A shortcoming in this mechanism is that you cannot export a single directory like **/usr/people/demos**, but instead must export the entire file system root directory, **/usr**.

NFS Commands

NFS uses two basic commands:

mount If all of the permissions are correct on the machine exporting the file systems, this command mounts a directory from another machine. Once mounted, this directory **appears** as though it is resident on your machine. After you run the **mount** command, type **df** to see a report of the remote file systems as well as the local ones. Below is an example of the syntax for **mount**:

/etc/mount *host:/pathname /pathname*

umount This command unmounts the remote directory. You can use either the remote name of the directory (**host:pathname**) or the local name (**pathname**) as the argument. Below is an example of the syntax for **umount**:

/etc/umount */pathname*

You must be the superuser to run both commands.

Relevant Documentation

You may find this documentation useful in planning and setting up your network.

- **TCP/IP User's Guide**, chapter 10 in this manual.
- **Defense Data Network Protocol Handbook**, available from the Network Information Center, Defense Data Network, SRI International, Room EJ-291, 333 Ravenswood Ave., Menlo Park, California 94025; telephone: (415) 326-6200.

This three-volume set contains information on TCP/IP and UDP/IP.

- The manuals on networking and NFS from Sun Microsystems, Inc.

Networking Models

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

A distributed operating system allows the network software designer to make assumptions about the other machines on the network. Usually, two of these assumptions are that the remote and local hardware is identical, and that the remote and local software is identical. 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. A closed environment is one in which it is very difficult to integrate new hardware or software, 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.

On the other hand, the network services approach is not closed. Network services are made up of remote programs composed of remote procedures called from the network. Optimally, a remote procedure computes results based entirely on its own parameters. This procedure and the network service are not tied to any particular operating system or hardware. The design of the network service makes it possible for a variety of machines and software to talk to the network services. This enables M Series systems to talk to various types of computers.

The network services approach is more complex in design and implementation than a closed distributed operating system. Since remote procedures are independent of operating systems and hardware, multiple remote procedures must sometimes be called in the NFS environment, where a single transaction might suffice in a closed system.

NFS Service

This section begins with an explanation of some NFS terms and concepts. It then describes how to create an NFS server that exports file systems, how to mount and use remote file systems, how to debug NFS when problems occur, and how to work with incompatibilities between NFS files and normal UMIPS files.

NFS Service Overview

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**(1M) command. Typically, a client mounts one or more remote file systems at startup by placing lines like those shown in the example below 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
```

For example, the first line means "mount the file system **/usr2** from the server **titan** on **/usr2** on the local machine; it is an NFS file sytem with read-write permission and it must be mounted (it is a "hard" mount) or the system keeps trying before continuing on to multiuser mode. See **fstab**(4) for a full description of the format.

Clients initiate all remote mounts and servers control who may mount a file system. Servers do this by limiting named file systems to desired clients with an entry in the **/etc/exports** file. In the example below, the file system **/usr** is exported to the world, while the file system **/usr2** is limited to specific machines.

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

Note that pathnames given in **/etc/exports** must be the mount point of a local file system. See **exports**(4) for a full description of the format.

How NFS Works

Two remote programs implement NFS service: **mountd**(1M) and **nfsd**(1M). A client's **mount** request calls **mountd**, which checks the access permission of the client and returns a pointer to a file system. After the **mount** completes, a pointer is established at that mount point; access to that mount point and to the directories below it goes through the pointer to the server's **nfsd** daemon using the Remote Procedure Call, **rpc**. For more information on Remote Procedure Calls, see **rpc**(4). Client kernel file access requests (delayed-write and read-ahead) are handled by the **biod**(1M) daemons on the client.

How to Become an NFS Server

An NFS server is a machine that exports one or more file systems. To enable any machine to export a file system, follow these steps:

1. Become the superuser and add the pathname from the mount point of the file system you want to export in the file **/etc/exports**. See **exports**(4) for file format details. For example, to export **/usr**, the export file would look like this:

```
/usr
```

   An NFS server can export only its own file systems.

2. Make sure **mountd** is available for an **rpc** call by checking **/usr/etc/inetd.conf** on the NFS server for these lines:

```
rpc  100005    1     1
     dgram     udp   wait    root  /usr/etc/rpc.mountd  mountd
```

   If the lines above are not present, add them. For details, see **servers**(4) and **inetd**(1M).

3. Make sure that the system initialization script, **/etc/init.d/nfs**, on the server machine starts some NFS server daemons. The number of daemons is typically 4. Check **/etc/init.d/nfs** for lines like these:

```
#  NFS  server  daemons
if  test  -x  $nfs/nfsd;  then
        $nfs/nfsd  4;echo    "  nfsd"
```

   Check to see if the **nfsd** daemons are running:

```
$ ps -d | grep nfsd
```

   and you should see something like this:

```
  172 ?        1:54 nfsd
  171 ?        2:30 nfsd
  173 ?        0:09 nfsd
  174 ?        0:07 nfsd
```

   Also, be sure that the portmapper is running:

```
# ps -e | grep portmap
  262 ?        0:32 portmap
```

   If the NFS daemons are not running, you can enable them manually:

```
# /etc/rcd.2/S40nfs
```

After these steps, the NFS server should be able to export the file system named in **/etc/exports**. The next section describes how to mount a remote file system.

How to Become an NFS Client

This section contains information on mounting a remote file system. You can mount any exported file system onto your machine as long as you can reach the server over the network and your machine is included in its **/etc/exports** list for that file system. The terms **hard** and **soft mount** are defined as follows:

hard mount A **hard mount** is an attempt to mount a remote file system on a local machine. A hard mount causes the client to continue to call the server until the server responds. A hard mount will always wait until it gets a response. If the server is down or slow, the hard mount causes the client to wait indefinitely for any operation on that file system (unless that calling process is terminated by the **kill** command or the local system is rebooted). A hard mount is the default mount.

soft mount A **soft mount** is also an attempt to mount a remote file system on a local machine. However, the client will not continue to call indefinitely. Rather, it will call a number of times and then give up. If a client is not able to execute any command on a soft-mounted file system, it will print an error on the console. A soft mount is an option that you must specify when using the **mount** command.

On the machine on which you want to mount the file system, become the superuser and type:

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

For example, to soft mount the **/usr** file system from the remote machine **elvis** onto the local empty directory **/usr/elvis**, type:

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

This file system is mounted with the **soft** option, so that if **elvis** goes down, this command fails, rather than waiting for the server to come back up. This keeps the local machine from continually looking for the down machine, draining system resources. To make sure you have mounted a file system where you expected, use either **df**(1) or **mount**(1M), without an argument. Each of these commands displays the currently mounted file systems.

Typically, you mount frequently used file systems at startup by placing entries for them in the file **/etc/fstab**. See **fstab**(4).

Refer to Procedure 11 for example NFS procedures.

UNIX and NFS

Unlike many recently marketed distributed operating systems, UNIX was originally designed without the knowledge that networks existed. This networking ignorance presents three impediments to linking UNIX with currently available high-performance networks.

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 across a network. For example, trusting remote users to log in as **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 down all of its bound neighbors. The treatment of node failure on a network raises difficulties in any system and is especially difficult in the UNIX environment.

   NFS has implemented a system of **stateless** protocols to circumvent the problem of a crashing server dragging down its bound clients. Stateless 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 from a very slow server.

In implementing UNIX over the network, NFS attempted to remain compatible with UNIX whenever possible. However, two kinds of incompatibilities have been introduced. First, there are issues that would make a networked UNIX evolve into a distributed operating system, rather than a collection of network services. Second, there are 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 the MIPS documentation.

Hints about Debugging UMIPS in the Network Environment

When you cannot perform a task involving NFS network services, the problem probably lies in one of the following five areas, listed below in descending order of frequency.

1. You did not set something up properly. Always check your work first.
2. The NFS network access control policies do not allow the operation, or architectural constraints prevent the operation.
3. The client software or environment is broken.
4. The server software or environment is broken.
5. The network is broken.

The following sections present specific instructions on how to check for these causes of failure in the NFS environment.

Debugging NFS

If you experience difficulties with NFS, read the manual pages for **mount**(1M), **nfsd**(1M), **showmount**(1M), **rpcinfo**(1M), **mountd**(1M), **inetd**(1M), **fstab**(4), **mtab**(4), and **exports**(4), before trying to debug NFS. You do not have to understand them fully, but be familiar with the names and functions of the various daemons and database files.

When analyzing an NFS problem, keep in mind that, like all network services, there are three main points of failure: the server, the client, and the network itself. The debugging strategy outlined below tries to isolate each individual component to find the one that is not working.

For example, here is a sample mount request made from an NFS client machine:

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

Try to understand how it works and how it can fail. The example asks the server machine *krypton* to return a file handle (a unique identifier) for the file system **/usr**. This file handle is then passed to the kernel in the **nfsmount**(2) system call. The kernel looks up the directory **/krypton.src** and, if everything is working, it ties the file handle to the file system in a mount record. From now on, all file system requests to that file system and below will go through the file handle to the server *krypton*.

The example above shows how a remote mount should work. The next section contains some general information and lists the possible errors and their causes.

When there are network or server problems, programs that access hard-mounted remote files fail differently from those that access soft-mounted remote files. Hard-mounted remote file systems cause programs to continue to try until the server responds again. Soft-mounted remote file systems return an error message after trying for a specified number of intervals. See **mount**(1) for more information.

When using **mount**, if the server for a remote file system fails to respond, it will try the mount request until it succeeds. However, if you specify a soft mount, it will try once in the foreground, then put itself in the background and try periodically.

Programs that access hard-mounted file systems will not respond until the server responds. In this case, NFS displays the message:

```
server  not  responding
```

On a soft-mounted file system, programs that access a file whose server is inactive get the message:

```
Connection  timed  out  (ETIMEDOUT)
```

Unfortunately, many programs do not check return conditions on file system operations, so this error message may not be displayed when accessing soft-mounted files. Nevertheless, an NFS error message is displayed on the console.

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

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

This checks whether the server is running. If the server is running, this command displays a list of program, version, protocol, and port numbers similar to the following:

```
program vers proto   port
100005    1   udp   1025  mountd
100001    1   udp   1027  rstatd
100001    2   udp   1027  rstatd
100008    1   udp   1030  walld
100002    1   udp   1032  rusersd
100012    1   udp   1034  sprayd
100003    2   udp   2049  nfs
```

If the server you want to use is running, also use **rpcinfo** to check if the **mountd** server is running by using its program number as an argument followed by the version number. Type:

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

The system responds:

```
program 100005 version 1 ready and waiting
```

If these fail, log in to the server's console to see if it is working.

If the server is operative but your machine cannot reach it, check the Ethernet connections between your machine and the server. You can also check other systems on your network to see if they can reach the server. Also, you can use **ping**(1M) to see if other systems are responding on the network.

If the server and the network are working, type **ps -de** to check your client daemons. **portmap** and several **biod** daemons should be running. For example, typing **ps -de** produces output similar to the following:

```
PID   TTY  TIME   COMMAND
 97   ?    0:00   routed
102   ?    0:00   portmap
103   ?    0:00   inetd
113   ?    0:00   nfsd
116   ?    0:00   nfsd
117   ?    0:00   nfsd
118   ?    0:00   nfsd
119   ?    0:00   biod
121   ?    0:00   biod
123   ?    0:00   biod
125   ?    0:00   biod
```

If you get the message:

```
Network Unavailable
```

First, check the physical Ethernet connection. If the same message occurs, try turning the Ethernet interface back on and off with **ifconfig**(1M).

You may have a **route**(1M) problem or a host address may be wrong if you get:

```
network unreachable
```

If you get NFS timeout messages, take one system off the net at a time until the problem clears in order to isolate the culprit. Re-examine the hardware and software setup on the offending machine.

Types of Failures

The four sections below describe the most common types of failure. The first section tells what to do if your remote mount fails; the next three sections discuss servers that do not respond once you have mounted file systems.

Remote Mount Failed

This section describes problems related to mounting. If for any reason **mount** fails, check the sections below for specific details about what to do. The sections are arranged according to where the problems occur in the mounting sequence. Each section is labeled with the error message you are likely to see.

mount can get its parameters either from the command line or from the file **/etc/fstab**. See **mount**(1M). The example below assumes command line arguments, but the same debugging techniques described below also work if the **mount −a** command uses **/etc/fstab**.

This section explains the interaction of the various players in the **mount** request. If you understand this interaction, the problem descriptions below will make more sense. Here is the example **mount** request previously given.

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

Below are the steps **mount** goes through to mount a remote file system.

1. **mount** opens **/etc/mtab** and checks that this mount has not already been done.
2. **mount** parses the first argument into host *krypton* and remote directory **/usr**.
3. **mount** uses **/etc/hosts** to translate the host name into its Internet Protocol (IP) address for *krypton*.
4. **mount** calls *krypton*'s **portmap** daemon to get the port number of **mountd**. See **portmap**(1M).
5. **mount** calls *krypton*'s **mountd** and passes it to **/usr**.
6. *krypton*'s **mountd** reads **/etc/exports** and looks for the exported file system that contains **/usr**.
7. *krypton*'s **mountd** calls **/etc/hosts** to expand the host names and network groups in the export list for **/usr**.
8. *krypton*'s **mountd** performs a system call on **/usr** to get the file handle.
9. *krypton*'s **mountd** returns the file handle.
10. **mount** does an **nfsmount**(2) system call with the file handle and *krypton.src*.
11. **nfsmount** checks to see if the caller is the 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 of these steps can fail, some of them in more than one way. The section below gives 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**(4). The entry referenced must exist before **mount** can succeed. Use **mkdir**(1) to create it.

mount: ... already mounted
The file system that you are trying to mount is already mounted or there is an incorrect entry for it in **/etc/mtab**.

mount: ... Block device required
You probably left off the *krypton:* part of:

```
#   mount krypton:/usr   /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**. See **fstab**(1).

mount: ... not found in /etc/fstab
If you use **mount** with only a directory or file system name, but not both, it looks in **/etc/fstab** for an entry whose file system or directory field matched the argument. For example,

```
mount /krypton.src
```

searches **/etc/fstab** for a line that has a directory name field of **/krypton.src**. If it finds an entry, such as:

```
krypton:/usr   /krypton.src   nfs   rw,hard 0 0
```

it mounts as if you had typed:

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

If you see this message, it means the argument you gave **mount** is not in any of the entries in **/etc/fstab**.

/etc/fstab: No such file or directory
mount tried to look up the name in **/etc/fstab** but there was no **/etc/fstab**.

... not in hosts database
This means the host name you gave is not in the **/etc/hosts** database. First, check the spelling and the placement of the colon in your **mount** call. Try to **rlogin** or **rcp** to some other machine.

mount: directory path must begin with a slash (/).
The second argument to **mount** is the path of the directory to be covered. This must be an absolute path starting at **/**.

mount: ... server not responding: RPC_PMAP_FAILURE - RPC_TIMED_OUT
Either the server from which you are trying to mount is inactive, or its **portmap** daemon is inactive or hung. Try logging in to that machine. If you can log in, type:

```
/usr/etc/rpcinfo  -p  hostname
```

This should produce a list of registered program numbers. If it does not, start the **portmap** daemon again. Note that starting the **portmap** daemon again requires that you kill and restart **inetd**.

There are two methods for dealing with a server that is inactive or whose **portmap** daemon is not responding. You could reboot the server or you could do the following:

1. Display the process identification numbers (pid) of **portmap** and **inetd**. Type:

   $ **ps -de**

2. Become the superuser and kill the daemons. Type:

   $ **su**
   # kill -9 portmap_pid inetd_pid

3. Start new daemons. Type:

   # **/usr/etc/portmap**
   # **/usr/etc/inetd**

If you cannot **rlogin** to the server, but the server is operational, check your Ethernet connection by trying **rlogin** to some other machine. Also check the server's Ethernet connection.

mount: ... server not responding: RPC_PROG_NOT_REGISTERED
This means **mount** reached the **portmap** daemon 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 **/usr/etc/inetd.conf** exactly like the following:

```
rpc  100005 1
     dgram  udp  wait  root  /etc/nfs/rpc.mountd  mountd
```

Type **ps -de** to be sure that the internet daemon (**inetd**) is running. If you had to change **/usr/etc/inetd.conf**, kill **inetd** and restart it. Look in **/etc/init.d/tcp** to see how **init.d** is started at boot time and enter the same command arguments.

mount: ... No such file or directory
Either the remote directory or the local directory does not exist. Check your spelling. Use the **ls** command for the local and remote directories.

mount: not in export list for ...
Your machine name is not in the export list for the file system you want to mount from the server. You can get a list of the server's exported file systems by running:

```
showmount -e hostname
```

If the file system you want is not in the list, or your machine name or network group name is not in the user list for the file system, log in to the server and check the **/etc/exports** file for the correct file system entry. A file system

name that appears in the **/etc/exports** file but not in the output from **showmount**, indicates a failure in **mountd**. Either it could not parse that line in the file, it could not find the file system, or the file system name was not a local mounted file system. See **exports**(4) for more information.

mount: ... Permission denied
This message is a generic indication that some authentication failed on the server. It could simply be that you are not in the export list (see above), the server could not figure out who you are, or the server does not believe you are who you say you are. Check the server's **/etc/exports**. In the last case, just change your hostname with **hostname**(1) and retry the **mount**.

mount: ... Not a directory
Either the remote path or the local path is not a directory. Check your spelling and use the **ls** command for both the local and remote directories.

mount: ... Not owner
You must do the mount as **root** on your machine because it affects the file system for the whole machine, not just your directories.

df: ... cannot open... No such file or directory
If the system date is set to a date before January 1 00:00:00 GMT 1970 NFS will not work. After a directory is mounted, any operation including **df** gives the above error message.

The solution to this problem is to make sure the date is current. Use the **date** command. See **date**(1).

Programs Do not Respond

If programs stop responding while doing file related work, your NFS server may be inactive. You may see the message:

```
NFS server host_name not responding, still trying
```

The message includes the host name of the NFS server that is down. This is probably a problem either with one of your NFS servers or with the Ethernet hardware. Attempt to **rlogin** to the server to determine whether the server is down. If you can successfully **rlogin** to it, its server function is probably disabled. If **rlogin** fails, you may need to reboot the server.

Another problem may occur if the remote machine has rebuilt a file system (**newfs**(1M)) while that file system has been mounted on the local machine. The file system will give messages about a "stale NFS handle" until the file system is unmounted and remounted on the local machine.

If your machine hangs completely, check the servers from which you have mounted. If one or more of them is down, do not worry. When the server comes back up, your programs will continue automatically, as if the server had not become inactive. No files will be destroyed. This procedure assumes a hard mount.

If a soft-mounted server is inactive, other work should not be affected. Programs that timeout trying to access soft-mounted remote files will fail, but you should still be able to use your other file systems.

If all of the servers are running, ask someone else who is using the same NFS server or servers if they are having trouble. If more than one machine is having difficulty getting service, then it is probably a problem with the server's NFS daemon **nfsd**(1M). Log in to the server and type `ps -de` to see if **nfsd** is running and accumulating CPU time. If not, you may be able to kill, and then restart **nfsd**. If this does not work, reboot the server.

If other people seem to be able to use the server, 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, one or more servers are probably down or your network connection may be bad. This applies only to hard mounts.

Everything Works Slowly

If access to remote files seems unusually slow, go to the server and type:

```
$ ps -de
```

Check whether the server is being slowed by a runaway daemon, bad **tty** line, etc. If the server seems to be working and other people are getting good response, make sure your block I/O daemons are running; type `ps -de` on your client machine and look for **biod**. To determine whether the processes are hung, type `ps -de` as shown below, then copy a large remote file and type `ps -de` again. If the **biod**s do not accumulate CPU time, they are probably hung. If they are not running or are hung, find the process identification numbers (pid) by typing:

```
$ ps -de | grep biod
```

An example of the output from this command looks like:

```
PID   TT   STAT  TIME   COMMAND
119   ?    IW    0:00   /usr/etc/biod
121   ?    IW    0:00   /usr/etc/biod
123   ?    IW    0:00   /usr/etc/biod
125   ?    IW    0:00   /usr/etc/biod
```

Kill the processes by using the **kill** command with their pid. For the above example, type:

```
# kill -9 119 121 123 125
```

Restart the processes by typing:

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

If **biod** is working, check your Ethernet connection. The command **netstat -i** tells you if packets are being dropped. (A packet is a unit of transmission sent across the Ethernet.) Also, you can use **/usr/etc/nfsstat -c** and **/usr/etc/nfsstat -s** to tell if the client or server is retransmitting a lot. A retransmission rate of 5% is considered high. Excessive retransmission usually indicates a bad Ethernet board, a bad Ethernet tap, a mismatch between board and tap, or a mismatch between your Ethernet board and the server's board. Finally, a good, quick way to check on network

connections is by using **ping**(1M).

Incompatibilities with NFS

On remote NFS file systems, a few things work differently, or do not work at all. This section discusses the incompatibilities and suggests how to work around them.

File Operations not Supported

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

Cannot Access Remote Devices

In NFS, you cannot access a remote mounted device or any other character or block special file.

Setting the Time in User Programs

Since NFS architecture differs in some minor ways from earlier versions of UNIX, be aware of those places where your own programs could run up against these incompatibilities.

Because each workstation keeps its own time, the clocks will be out of sync between the NFS server and client. Obviously, this might introduce a problem in certain situations. Here is an example of a problem and how it was fixed.

Many programs make the assumption that an existing file could not have been created in the future. For example, **ls** does this. The command **ls −l** has two basic forms of output, depending upon how old the file is:

```
date
Jan 22 15:27:01 PST 1985
touch file2
ls -l file*
-rw-r--r--  1 root          0 Dec 27  1983 file
-rw-r--r--  1 root          0 Jan 22 15:27 file2
```

The first form of **ls** prints the year, month, and day of last file modification if the file is more than six months old. The second form prints the month, day, and minute of last file modification if the file is less than six months old.

ls calculates the age of a file by simply subtracting the modification time of the file from the current time. If the results are greater than six months worth of seconds, the file is old.

Now assume that the time on the server is Jan 22 15:30:31 (three minutes ahead of our local machine's time):

```
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 between the time on the server and the time on the client is huge. The formula is:

```
(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. ls has been modified to deal with files that are created a short time in the future.

Chapter 12: Printer Spooler

Introduction

This chaper describes the structure and installation procedure for the line printer spooling system developed for the 4.3BSD version of the UNIX* operating system.

The line printer system supports:

- multiple printers,
- multiple spooling queues,
- both local and remote printers, and
- printers attached via serial lines that require line initialization such as the baud rate.

Raster output devices such as a Varian or Versatec, and laser printers such as an Imagen, are also supported by the line printer system.

The line printer system consists mainly of the following files and commands:

/etc/printcap	printer configuration and capability data base
/usr/lib/lpd	line printer daemon, does all the real work
/usr/ucb/lpr	program to enter a job in a printer queue
/usr/ucb/lpq	spooling queue examination program
/usr/ucb/lprm	program to delete jobs from a queue
/etc/lpc	program to administer printers and spooling queues
/dev/printer	socket on which lpd listens

The file /etc/printcap is a master data base describing line printers directly attached to a machine and, also, printers accessible across a network. The manual page entry provides the authoritative definition of the format of this data base, as well as specifying default values for important items such as the directory in which spooling is performed. This document introduces some of the information that may be placed

Commands

lpd – line printer daemon

The program usually invoked at boot time from the /etc/rc file, acts as a master server for coordinating and controlling the spooling queues configured in the printcap file. When *lpd* is started it makes a single pass through the *printcap* database restarting any printers that have jobs. In normal operation *lpd* listens for service requests on multiple sockets, one in the UNIX domain (named "/dev/printer") for local requests, and one in the Internet domain (under the "printer" service specification) for requests for printer access from off machine; see *socket* (2) and *services* (5) for more information on sockets and service specifications, respectively. *lpd* spawns a copy of itself to process the request; the master daemon continues to listen for new requests.

Clients communicate with *lpd* using a simple transaction oriented protocol. Authentication of remote clients is done based on the "privilege port" scheme employed by *rshd* (8C) and *rcmd* (3X). The following table shows the requests understood by In each request the first byte indicates the "meaning" of the request, followed by the name of the printer to which it should be applied. Additional qualifiers may follow, depending on the request.

Request	Interpretation
^Aprinter\n	check the queue for jobs and print any found
^Bprinter\n	receive and queue a job from another machine
^Cprinter [users ...] [jobs ...]\n	return short list of current queue state
^Dprinter [users ...] [jobs ...]\n	return long list of current queue state
^Eprinter person [users ...] [jobs ...]\n	remove jobs from a queue

The *lpr* (1) command is used by users to enter a print job in a local queue and to notify the local *lpd* that there are new jobs in the spooling area. *Lpd* either schedules the job to be printed locally, or if printing remotely, attempts to forward the job to the appropriate machine. If the printer cannot be opened or the destination machine is unreachable, the job will remain queued until it is possible to complete the work.

lpq – show line printer queue

The *lpq* (1) program works recursively backwards displaying the queue of the machine with the printer and then the queue(s) of the machine(s) that lead to it. *Lpq* has two forms of output: in the default, short, format it gives a single line of output per queued job; in the long format it shows the list of files, and their sizes, that comprise a job.

lprm – remove jobs from a queue

The *lprm* (1) command deletes jobs from a spooling queue. If necessary, *lprm* will first kill off a running daemon that is servicing the queue and restart it after the required files are removed. When removing jobs destined for a remote printer, *lprm* acts similarly to *lpq* except it first checks locally for jobs to remove and then tries to remove files in queues off-machine.

lpc – line printer control program

The 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 daemons.

Access control

The printer system maintains protected spooling areas so that users cannot circumvent printer accounting or remove files other than their own. The strategy used to maintain protected spooling areas is as follows:

- The spooling area is writable only by a *daemon* user and *daemon* group.
- The *lpr* program runs set-user-id to *root* and set-group-id to group *daemon*. The *root* access permits reading any file required. Accessibility is verified with an *access*(2) call. The group ID is used in setting up proper ownership of files in the spooling area for *lprm*.
- Control files in a spooling area are made with *daemon* ownership and group ownership *daemon*. Their mode is 0660. This insures control files are not modified by a user and that no user can remove files except through *lprm*.
- The spooling programs, *lpd*, *lpq*, and *lprm* run set-user-id to *root* and set-group-id to group *daemon* to access spool files and printers.
- 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 or /etc/hosts.lpd and the request message must come from a reserved port number.

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 set-user-id to *daemon*, set-group-id to group *spooling*, and *lpq* and *lprm* ran set-group-id to group *spooling*.

Setting up

The 4.3BSD release comes with the necessary programs installed and with the default line printer queue created. If the system must be modified, the makefile in the directory /usr/src/usr.lib/lpr should be used in recompiling and reinstalling the necessary programs.

The real work in setting up is to create the *printcap* file and any printer filters for printers not supported in the distribution system.

Creating a printcap file

The *printcap* database contains one or more entries per printer. A printer should have a separate spooling directory; otherwise, jobs will be printed on different printers depending on which printer daemon starts first. This section describes how to create entries for printers that do not conform to the default printer description (an LP-11 style interface to a standard, band printer).

Printers on serial lines

When a printer is connected via a serial communication line it must have the proper baud rate and terminal modes set. The following example is for a DecWriter III printer connected locally via a 1200 baud serial line.

```
lp|LA-180 DecWriter III:\
    :lp=/dev/lp:br#1200:fs#06320:\
    :tr=\f:of=/usr/lib/lpf:lf=/usr/adm/lpd-errs:
```

The **lp** entry specifies the file name to open for output. Here it could be left out since "/dev/lp" is the default. The **br** entry sets the baud rate for the tty line and the **fs** entry sets CRMOD, no parity, and XTABS (see *tty* (4)). The **tr** entry indicates that a form-feed should be printed when the queue empties so the paper can be torn off without turning the printer off-line and pressing form feed. The **of** entry specifies the filter program *lpf* should be used for printing the files; more will be said about filters later. The last entry causes errors to be written to the file "/usr/adm/lpd-errs" instead of the console. Most errors from *lpd* are logged using *syslogd* (8) and will not be logged in the specified file. The filters should use *syslogd* to report errors; only those that write to standard error output will end up with errors in the **lf** file. (Occasionally errors sent to standard error output have not appeared in the log file; the use of *syslogd* is highly recommended.)

Remote printers

Printers that reside on remote hosts should have an empty **lp** entry. For example, the following printcap entry would send output to the printer named "lp" on the machine "ucbvax".

```
lp|default line printer:\
    :lp=:rm=ucbvax:rp=lp:sd=/usr/spool/vaxlpd:
```

The **rm** entry is the name of the remote machine to connect to; this name must be a known host name for a machine on the network. The **rp** capability indicates the name of the printer on the remote machine is "lp"; here it could be left out since this is the default value. The **sd** entry specifies "/usr/spool/vaxlpd" as the spooling

directory instead of the default value of "/usr/spool/lpd".

Output filters

Filters are used to handle device dependencies and to do accounting functions. The output filtering of **of** is used when accounting is not being done or when all text data must be passed through a filter. It is not intended to do 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 (if specified) are started for each file printed and do accounting if there is an **af** entry. If entries for both **of** and 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 that 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#58:px=2112:py=1700:tr=\f:
```

The **tf** entry specifies "/usr/lib/rvcat" as the filter to be used in printing *troff*(1) output. This filter is needed to set the device into print mode for text, and plot mode for printing *troff* files and raster images (see *va*(4V)). Note that the page length is set to 58 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** filter 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:px=2112:py=1700:tr=\f:
```

Access Control

Local access to printer queues is controlled with the **rg** printcap entry.

```
:rg=lprgroup:
```

Users must be in the group *lprgroup* to submit jobs to the specified printer. The default is to allow all users access. Note that once the files are in the local queue, they can be printed locally or forwarded to another host depending on the configuration.

Remote access is controlled by listing the hosts in either the file /etc/hosts.equiv or /etc/hosts.lpd, one host per line. Note that and use /etc/hosts.equiv to determine which hosts are equivalent for allowing logins without passwords. The file /etc/hosts.lpd is only used to control which hosts have line printer access. Remote access can be further restricted to only allow remote users with accounts on the local host to print jobs by using the **rs** printcap entry.

```
:rs:
```

Output filter specifications

The filters supplied with 4.3BSD handle printing and accounting for most common line printers, the Benson-Varian, the wide (36") and narrow (11") Versatec printer/plotters. For other devices or accounting methods, it may be 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 or *syslogd* may be used 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 that need to do 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.

filter **−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.

filter [**−c**] **−w**width **−l**length **−i**indent **−n** login **−h** host accounting_file

The **−c** flag is optional, and only supplied when control characters are to be passed uninterpreted to the printer (when using the **−l** option of *lpr* to print the file). The **−w** and **−l** parameters are the same as for the **of** filter. The **−n** and **−h** parameters specify the login name and host name of the job owner. The last argument is the name of the accounting file from

All other filters are called with the following arguments:

filter **−x**width **−y**length **−n** login **−h** host accounting_file

The **−x** and **−y** options specify the horizontal and vertical page size in pixels (from the **px** and **py** entries in the printcap file). The rest of the arguments are the same as for the **if** filter.

Line printer Administration

The *lpc* program provides local control over line printer activity. The major commands and their intended use will be described. The command format and remaining commands are described in *lpc*

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 This is normally used to forcibly restart a hung line printer daemon (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 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 halts a spooling daemon after the current job completes; this also disables printing. This is a clean way to shutdown a printer to do 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.

Troubleshooting

There are several messages that may be generated by the the line printer system. This section categorizes the most common and explains the cause for their generation. Where the message implies a failure, directions are given to remedy the problem.

In the examples below, the name *printer* is the name of the printer from the *printcap* database.

LPR

lpr: *printer*: unknown printer

The *printer* 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). Usually it is enough to get a super-user to type the following to restart

% /usr/lib/lpd

You can also check the state of the master printer daemon with the following.

% ps l'cat /usr/spool/lpd.lock'

Another possibility is that the *lpr* program is not set-user-id to *root*, set-group-id to group *daemon*. This can be checked with

% ls -lg /usr/ucb/lpr

lpr: *printer*: printer queue is disabled

This means the queue was turned off with

% lpc disable *printer*

to prevent *lpr* from putting files in the queue. This is normally done by the system manager when a printer is going to be down for a long time. The printer can be turned back on by a super-user with

LPQ

waiting for *printer* to become ready (offline ?)

The printer device could not be opened by the daemon. This can happen for several reasons, the most common is that the printer is turned off-line. This message can also be generated if the printer is out of paper, the paper is jammed, etc. The actual reason is dependent on the meaning of error codes returned by system device driver. Not all printers supply enough information to distinguish when a printer is off-line or having trouble (e.g. a printer connected through a serial line). Another possible cause of this message is some other process, such as an output filter, has an exclusive open on the device. Your only recourse here is to kill off the offending program(s) and restart the printer with

printer is ready and printing

The *lpq* program checks to see if a daemon process exists for *printer* and prints the file *status* located in the spooling directory. If the daemon is hung, a super user can use *lpc* to abort the current daemon and start a new one.

waiting for *host* to come up

This implies there is a daemon trying to connect to the remote machine named *host* to send the files in the local queue. If the remote machine is up, *lpd* on the remote machine is probably dead or hung and should be restarted as mentioned for

sending to *host*

The files should be in the process of being transferred to the remote If not, the local daemon should be aborted and started with

Warning: *printer* is down

The printer has been marked as being unavailable with

Warning: no daemon present

The *lpd* process overseeing the spooling queue, as specified in the "lock" file in that directory, does not exist. This normally occurs only when the daemon has unexpectedly died. The error log file for the printer and the *syslogd* logs should be checked for a diagnostic from the deceased process. To restart an *lpd*, use

% lpc restart *printer*

no space on remote; waiting for queue to drain

This implies that there is insufficient disk space on the remote. If the file is large enough, there will never be enough space on the remote (even after the queue on the remote is empty). The solution here is to move the spooling queue or make more free space on the remote.

LPRM

lprm: *printer*: cannot restart printer daemon

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

LPD

The *lpd* program can log many different messages using *syslogd* (8). Most of these messages are about files that can not be opened and usually imply that the *printcap* file or the protection modes of the files are incorrect. 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 any of the filters that *lpd* spawns may log messages using *syslogd* or to the error log file (the file specified in the **lf** entry in *printcap*).

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 because of incorrect ownership or protection mode of the lock file, spooling directory or the *lpc* program.

Chapter 13: SendMail Installation and Operations

Introduction

Sendmail implements a general purpose internetwork mail routing facility under the UNIX (UNIX is a trademark of Bell Laboratories) 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 files 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 one describes how to do a basic *sendmail* installation. Section two 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 three describes some parameters that may be safely tweaked. Section four has information regarding the command line arguments. Section five contains the nitty-gritty information about the configuration file. This section is for masochists and people who must write their own configuration file. The appendixes 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.* This other paper should be read before this manual to gain a basic understanding of how the pieces fit together.

BASIC INSTALLATION

There are two basic steps to installing sendmail. The hard part is to build the configuration table. This is a file that sendmail reads when it starts up that describes the mailers it knows about, how to parse addresses, how to rewrite the message header, and the settings of various options. Although the configuration table is quite complex, a configuration can usually be built by adjusting an existing off-the-shelf configuration. The second part is actually doing the installation, i.e., creating the necessary files, etc.

The remainder of this section will describe the installation of sendmail assuming you can use one of the existing configurations and that the standard installation parameters are acceptable. All pathnames and examples are given from the root of the *sendmail* subtree, normally */usr/src/usr.lib/sendmail* on 4.3BSD.

Off-The-Shelf Configurations

The configuration files are all in the subdirectories *cf.named* and *cf.hosttable* of the sendmail directory. The directory *cf.named* contains configuration files that have been tailored for the name server *named* (8). These are the configuration files currently being used at Berkeley. The configuration files in *cf.hosttable* are some typical ones and the old Berkeley versions from before the name server was being used. You should create a symbolic link from *cf* to the directory that you are going to use. For example, to use the name server:

```
ln -s cf.named cf
```

The ones used at Berkeley are in *m4* (1) format; files with names ending ".m4" are *m4* include files, while files with names ending ".mc" are the master files. Files with names ending ".cf" are the *m4* processed versions of the corresponding ".mc" file.

Three off the shelf configurations are supplied to handle the basic cases:

- Arpanet (TCP) sites not running the name server can use *cf.hosttable/arpaproto.cf*. For simple sites, you should be able to use this file without modification. This file is not in *m4* format.
- UUCP sites can use *cf.hosttable/uucpproto.cf*. If your UUCP node name is not the same as your system name (as printed by the *hostname*(1) command) you may have to modify the **U** macro. This file is not in *m4* format.
- A group of machines at a single site connected by an ethernet with (only) one host connected to the outside world via UUCP is represented by two configuration files: *cf.hosttable/lanroot.mc* should be installed on the host with outside connections and *cf.hosttable/lanleaf.mc* should be installed on all other hosts. These will require slightly more configuration. First, in both files the **D** macro and **D** class must be adjusted to indicate your local domain. For example, if your company is known as "Muse" you will want to change both of those accordingly. (As distributed, they are called XXX.) Second, in *lanleaf.mc* you will have to change the **R** macro to the name of the root host, that is, the host that runs *lanroot.mc*. For example, they might appear as:

```
DDMuse
CDLOCAL Muse
DRErato
```

Internally, the root host will be known as "Erato.Muse" and other hosts will be known as "Thalia.Muse", "Clio.Muse", etc.

The file you need should be copied to a file with the same name as your system, e.g., cp uucpproto.cf ucsfcgl.cf

This file is now ready for installation as */usr/lib/sendmail.cf*.

Installation Using the Makefile

A makefile exists in the root of the *sendmail* directory that will do all of these steps for a 4.3BSD system. It may have to be slightly tailored for use on other systems.

Before using this makefile, you should create a symbolic link from *cf* to the directory containing your configuration files. hYou should also have created your configuration file nd left it in the file "cf/*system*.cf" where *system* is the name of your system (i.e., what is returned by *hostname* (1)). If you do not have *hostname* you can use the declaration HOST=*system* on the *make* (1) command line. You should also examine the file *md/config.m4* and change the *m4* macros there to reflect any libraries and compilation flags you may need.

The basic installation procedure is to type:

```
make
make install
make installcf
```

in the root directory of the *sendmail* distribution. This will make all binaries and install them in the standard places. The second and third *make* commands must be executed as the superuser (root).

Installation by Hand

Along with building a configuration file, you will have to install the *sendmail* startup into your UNIX system. If you are doing this installation in conjunction with a regular Berkeley UNIX install, these steps will already be complete. Many of these steps will have to be executed as the superuser (root).

lib/libsys.a

The library in lib/libsys.a contains some routines that should in some sense be part of the system library. These are the system logging routines and the new directory access routines (if required). If you are not running the 4.3BSD directory code and do not have the compatibility routines installed in your system library, you should execute the command:

```
(cd lib; make ndir)
```

This will compile and install the 4.3 compatibility routines in the library. You should then type:

```
(cd lib; make)
```

This will recompile and fill the library.

/usr/lib/sendmail

The binary for sendmail is located in /usr/lib. There is a version available in the source directory that is probably inadequate for your system. You should plan on recompiling and installing the entire system:

```
cd src
make clean
make
cp sendmail /usr/lib
chgrp kmem /usr/lib/sendmail
```

/usr/lib/sendmail.cf

The configuration file that you created earlier should be installed in /usr/lib/sendmail.cf:

```
cp cf/system.cf /usr/lib/sendmail.cf
```

/usr/ucb/newaliases

If you are running delivermail, it is critical that the *newaliases* command be replaced. This can just be a link to *sendmail*:

```
rm -f /usr/ucb/newaliases
ln /usr/lib/sendmail /usr/ucb/newaliases
```

/usr/spool/mqueue

The directory */usr/spool/mqueue* should be created to hold the mail queue. This directory should be mode 777 unless *sendmail* is run setuid, when *mqueue* should be owned by the sendmail owner and mode 755.

/usr/lib/aliases*

The system aliases are held in three files. The file "/usr/lib/aliases" is the master copy. A sample is given in "lib/aliases" which includes some aliases which *must* be defined:

```
cp lib/aliases /usr/lib/aliases
```

You should extend this file with any aliases that are apropos to your system.

Normally *sendmail* looks at a version of these files maintained by the *dbm* (3) routines. These are stored in "/usr/lib/aliases.dir" and "/usr/lib/aliases.pag." These can initially be created as empty files, but they will have to be initialized promptly. These should be mode 666 if you are running a reasonably relaxed system:

```
cp /dev/null /usr/lib/aliases.dir
cp /dev/null /usr/lib/aliases.pag
chmod 666 /usr/lib/aliases.*
```

newaliases

/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
```

/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.

Add the following lines to "/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.

If you are not running a version of UNIX that supports Berkeley TCP/IP, do not include the **-bd** flag.

/usr/lib/sendmail.hf

This is the help file used by the SMTP **HELP** command. It should be copied from "lib/sendmail.hf:"

```
cp lib/sendmail.hf /usr/lib
```

/usr/lib/sendmail.st

If you wish to collect statistics about your mail traffic, you should 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/ucb/newaliases

If *sendmail* is invoked as "newaliases", it will simulate the **−bi** flag (i.e., will rebuild the alias database; see below). This should be a link to /usr/lib/sendmail.

/usr/ucb/mailq

If *sendmail* is invoked as "mailq", it will simulate the **−bp** flag (i.e., *sendmail* will print the contents of the mail queue; see below). This should be a link to /usr/lib/sendmail.

NORMAL OPERATIONS

Quick Configuration Startup

A fast version of the configuration file may be set up by using the **-bz** flag:

```
/usr/lib/sendmail -bz
```

This creates the file */usr/lib/sendmail.fc* ("frozen configuration"). This file is an image of *sendmail*'s data space after reading in the configuration file. If this file exists, it is used instead of */usr/lib/sendmail.cf sendmail.fc* must be rebuilt manually every time *sendmail.cf* is changed.

The frozen configuration file will be ignored if a **-C** flag is specified or if sendmail detects that it is out of date. However, the heuristics are not strong so this should not be trusted.

The System Log

The system log is supported by the *syslogd* (8) program.

Format

Each line in the system log consists of a timestamp, the name of the machine that generated it (for logging from several machines over the ethernet), the word "sendmail", and a message.

Levels

If you have *syslogd* (8) or an equivalent installed, you will be able to do logging. There is a large amount of information that can be logged. The log is arranged as a succession of levels. At the lowest level only extremely strange situations are logged. At the highest level, even the most mundane and uninteresting events are recorded for posterity. As a convention, log levels under ten are considered "useful"; log levels above ten are usually for debugging purposes.

A complete description of the log levels is given in section 4.6.

The Mail Queue

The mail queue should be processed transparently. However, you may find that manual intervention is sometimes necessary. For example, if a major host is down for a period of time the queue may become clogged. Although sendmail ought to recover gracefully when the host comes up, you may find performance unacceptably bad in the meantime.

Printing the queue

The contents of the queue can be printed using the *mailq* command (or by specifying the **-bp** flag to sendmail):

```
mailq
```

This will produce a listing of the queue id's, the size of the message, the date the message entered the queue, and the sender and recipients.

Format of queue files

All queue files have the form *x***f***AA99999* 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.

E An error address. If any such lines exist, they represent the addresses that should receive error messages.

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 changes 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 the *mailq* command, 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" :

```
DdfA13557
Seric
T404261372
P132
Rmckusick@calder
Rwnj
H?D?date: 23-Oct-82 15:49:32-PDT (Sat)
H?F?from: eric (Eric Allman)
H?x?full-name: Eric Allman
Hsubject: this is an example message
Hmessage-id: <8209232249.13557@UCBARPA.BERKELEY.ARPA>
Hreceived: by UCBARPA.BERKELEY.ARPA (3.227 [10/22/82])
    id A13557; 23-Oct-82 15:49:32-PDT (Sat)
HTo: mckusick@calder, wnj
```

This shows the name of the data file, the person who sent the message, the submission time (in seconds since January 1, 1970), the message priority, the message class, the recipients, and the headers for the message.

Forcing the queue

Sendmail should run the queue automatically at intervals. The algorithm is to read and sort the queue, and then to attempt to process all jobs in order. When it attempts to run the job, *sendmail* first checks to see if the job is locked. If so, it ignores the job.

There is no attempt to insure that only one queue processor exists at any time, since there is no guarantee that a job cannot take forever to process. Due to the locking algorithm, it is impossible for one job to freeze the queue. However, an uncooperative recipient host or a program recipient that never returns can accumulate many processes in your system. Unfortunately, there is no way to resolve this without

violating the protocol.

In some cases, you may find that a major host going down for a couple of days may create a prohibitively large queue. This will result in *sendmail* spending an inordinate amount of time sorting the queue. This situation can be fixed by moving the queue to a temporary place and creating a new queue. The old queue can be run later when the offending host returns to service.

To do this, it is acceptable to move the entire queue directory:

```
cd /usr/spool
mv mqueue omqueue; mkdir mqueue; chmod 777 mqueue
```

You should then kill the existing daemon (since it will still be processing in the old queue directory) and create a new daemon.

To run the old mail queue, run the following command:

```
/usr/lib/sendmail -oQ/usr/spool/omqueue -q
```

The **-oQ** flag specifies an alternate queue directory and the **-q** flag says to just run every job in the queue. If you have a tendency toward voyeurism, you can use the **-v** flag to watch what is going on.

When the queue is finally emptied, you can remove the directory:

```
rmdir /usr/spool/omqueue
```

The Alias Database

The alias database exists in two forms. 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; e.g.,

```
eric@mit-xx: eric@berkeley.EDU
```

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:

- The DBM version of the database is mode 666. -or-
- *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 aborting the process leaving a partially rebuilt database. Second, at the end of the rebuild it adds an alias of the form

```
@: @
```

(which is not normally legal). Before sendmail will access the database, it checks to insure that this entry exists (The "a" option is required in the configuration for this action to occur. This should normally be specified unless you are running *delivermail* in parallel with *sendmail.)* *Sendmail* will wait for this entry to appear, at which point it will force a rebuild itself (Note: the "D" option must be specified in the configuration file for this operation to occur. If the "D" option is not specified, a warning message is generated and *sendmail* continues).

List owners

If an error occurs on sending to a certain address, say "x" , *sendmail* will look for an alias of the form "owner-*x*" to receive the errors. This is typically useful for a mailing list where the submitter of the list has no control over the maintenance of the list itself; in this case the list maintainer would be the owner of the list. For example:

```
unix-wizards: eric@ucbarpa, wnj@monet, nosuchuser,
  sam@matisse
owner-unix-wizards: eric@ucbarpa
```

would cause "eric@ucbarpa" to get the error that will occur when someone sends to unix-wizards due to the inclusion of "nosuchuser" on the list.

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, that is, when successfully delivered to a mailer with the l 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.

Apparently-To:

If a message comes in with no recipients listed in the message (in a To:, Cc:, or Bcc: line) then *sendmail* will add an "Apparently-To:" header line for any recipients it is aware of. This is not put in as a standard recipient line to warn any recipients that the list is not complete.

At least one recipient line is required under RFC 822.

ARGUMENTS

The complete list of arguments to *sendmail* is described in detail in Appendix A. Some important arguments are described here.

Queue Interval

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
```

Forcing the Queue

In some cases you may find that the queue has gotten clogged for some reason. You can force a queue run using the **-q** flag (with no value). It is entertaining to use the **-v** flag (verbose) when this is done to watch what happens:

```
/usr/lib/sendmail -q -v
```

Debugging

There are a fairly large number of debug flags built into *sendmail*. Each debug flag has a number and a level, where higher levels means to print out more information. The convention is that levels greater than nine are "absurd", i.e., they print out so much information that you wouldn't normally want to see them except for debugging that particular piece of code. Debug flags are set using the **-d** option; the syntax is:

```
debug-flag:     -d debug-list
debug-list:     debug-option [ , debug-option ]
debug-option:   debug-range [ . debug-level ]
debug-range:    integer | integer - integer
debug-level:    integer
```

where spaces are for reading ease only. For example,

-d12	Set flag 12 to level 1
-d12.3	Set flag 12 to level 3
-d3-17	Set flags 3 through 17 to level 1
-d3-17.4	Set flags 3 through 17 to level 4

For a complete list of the available debug flags you will have to look at the code (they are too dynamic to keep this documentation up to date).

Trying a Different Configuration File

An alternative configuration file can be specified using the **–C** flag; for example,

```
/usr/lib/sendmail –Ctest.cf
```

uses the configuration file *test.cf* instead of the default */usr/lib/sendmail.cf*. If the **–C** flag has no value it defaults to *sendmail.cf* in the current directory.

Changing the Values of Options

Options can be overridden using the **–o** flag. For example,

```
/usr/lib/sendmail –oT2m
```

sets the **T** (timeout) option to two minutes for this run only.

TUNING

There are a number of configuration parameters you may want to change, depending on the requirements of your site. Most of these are set using an option in the configuration file. For example, the line "OT3d" sets option "T" to the value "3d" (three days).

Most of these options default appropriately for most sites. However, sites having very high mail loads may find they need to tune them as appropriate for their mail load. In particular, sites experiencing a large number of small messages, many of which are delivered to many recipients, may find that they need to adjust the parameters dealing with queue priorities.

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.

Forking During Queue Runs

By setting the **Y** option, *sendmail* will fork before each individual message while running the queue. This will prevent *sendmail* from consuming large amounts of memory, so it may be useful in memory-poor environments. However, if the **Y** option is not set, *sendmail* will keep track of hosts that are down during a queue run, which can improve performance dramatically.

Queue Priorities

Every message is assigned a priority when it is first instantiated, consisting of the message size (in bytes) offset by the message class times the "work class factor" and the number of recipients times the "work recipient factor." The priority plus the creation time of the message (in seconds since January 1, 1970) are used to order the queue. Higher numbers for the priority mean that the message will be processed later when running the queue.

The message size is included so that large messages are penalized relative to small messages. The message class allows users to send "high priority" messages by including a "Precedence:" field in their message; the value of this field is looked up in the **P** lines of the configuration file. Since the number of recipients affects the amount of load a message presents to the system, this is also included into the priority.

The recipient and class factors can be set in the configuration file using the **y** and **z** options respectively. They default to 1000 (for the recipient factor) and 1800 (for the class factor). The initial priority is:

pri = size − (class * z) + (nrcpt * y)

(Remember, higher values for this parameter actually mean that the job will be treated with lower priority.)

The priority of a job can also be adjusted each time it is processed (that is, each time an attempt is made to deliver it) using the "work time factor," set by the **Z** option. This is added to the priority, so it normally decreases the precedence of the job, on the grounds that jobs that have failed many times will tend to fail again in the future.

Load Limiting

Sendmail can be asked to queue (but not deliver) mail if the system load average gets too high using the **x** option. When the load average exceeds the value of the **x** option, the delivery mode is set to **q** (queue only) if the *Queue Factor* (**q** option) divided by the difference in the current load average and the **x** option plus one exceeds the priority of the message — that is, the message is queued iff:

$$pri > \frac{QF}{LA - x + 1}$$

The **q** option defaults to 10000, so each point of load average is worth 10000 priority points (as described above, that is, bytes + seconds + offsets).

For drastic cases, the **X** option defines a load average at which sendmail will refuse to accept network connections. Locally generated mail (including incoming UUCP mail) is still accepted.

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.

Log Level

The level of logging can be set for sendmail. The default using a standard configuration table is level 9. The levels are as follows:

- 0 No logging.
- 1 Major problems only.
- 2 Message collections and failed deliveries.
- 3 Successful deliveries.
- 4 Messages being deferred (due to a host being down, etc.).
- 5 Normal message queueups.
- 6 Unusual but benign incidents, e.g., trying to process a locked queue file.
- 9 Log internal queue id to external message id mappings.
  This can be useful for tracing a message as it travels between several hosts.
- 12 Several messages that are basically only of interest when debugging.
- 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).

Should my alias database be writable?

At Berkeley we have the alias database (/usr/lib/aliases*) mode 666. There are some dangers inherent in this approach: any user can add him-/her-self to any list, or can "steal" any other user's mail. 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.

The database that *sendmail* actually used is represented by the two files *aliases.dir* and *aliases.pag* (both in /usr/lib). The mode on these files should match the mode on /usr/lib/aliases. If *aliases* is writable and the DBM files (*aliases.dir* and *aliases.pag*) are not, users will be unable to reflect their desired changes through to the actual database. However, if *aliases* is read-only and the DBM files are writable, a slightly sophisticated user can arrange to steal mail anyway.

If your DBM files are not writable by the world or you do not have auto-rebuild enabled (with the "D" option), then you must be careful to reconstruct the alias database each time you change the text version:

```
newaliases
```

If this step is ignored or forgotten any intended changes will also be ignored or forgotten.

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 this is done every time *sendmail* starts up, rather than easy for a human to read or write. On the "future project" list is a configuration-file compiler.

An overview of the configuration file is given first, followed by details of the semantics.

The Syntax

The configuration file is organized as a series of lines, each of which begins with a single character defining the semantics for the rest of the line. Lines beginning with a space or a tab are continuation lines (although the semantics are not well defined in many places). Blank lines and lines beginning with a sharp symbol ('#') are comments.

R and S – rewriting rules

The core of address parsing are the rewriting rules. These are an ordered production system. *Sendmail* scans through the set of rewriting rules looking for a match on the left hand side (LHS) of the rule. When a rule matches, the address is replaced by the right hand side (RHS) of the rule.

There are several sets of rewriting rules. Some of the rewriting sets are used internally and must have specific semantics. Other rewriting sets do not have specifically assigned semantics, and may be referenced by the mailer definitions or by other rewriting sets.

The syntax of these two commands are:

S*n*

Sets the current ruleset being collected to *n*. If you begin a ruleset more than once it deletes the old definition.

R*lhs rhs comments*

The fields must be separated by at least one tab character; there may be embedded spaces in the fields. The *lhs* is a pattern that is applied to the input. If it matches, the input is rewritten to the *rhs*. The *comments* are ignored.

D – define macro

Macros are named with a single character. These may be selected from the entire ASCII set, but user-defined macros should be selected from the set of upper case letters only. Lower case letters and special symbols are used internally.

The syntax for macro definitions is:

D*x val*

where *x* is the name of the macro and *val* is the value it should have. Macros can be interpolated in most places using the escape sequence $*x*.

C and F – define classes

Classes of words may be defined to match on the left hand side of rewriting rules. For example a class of all local names for this site might be created so that attempts to send to oneself can be eliminated. These can either be defined directly in the configuration file or read in from another file. Classes may be given names from the set of upper case letters. Lower case letters and special characters are reserved for system use.

The syntax is:

C*c word1 word2...*
F*c file*

The first form defines the class *c* to match any of the named words. It is permissible to split them among multiple lines; for example, the two forms:

CHmonet ucbmonet

and

CHmonet
CHucbmonet

are equivalent. The second form reads the elements of the class *c* from the named *file*.

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

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*o value*

This sets option *o* to be *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.

T – define trusted users

Trusted users are those users who are permitted to override the sender address using the **–f** flag. These typically are "root", "uucp", and "network", but on some users it may be convenient to extend this list to include other users, perhaps to support a separate UUCP login for each host. The syntax of this line is:

T*user1 user2...*

There may be more than one of these lines.

P – precedence definitions

Values for the "Precedence:" field may be defined using the **P** control line. The syntax of this field is:

P*name***=***num*

When the *name* is found in a "Precedence:" field, the message class is set to *num*. Higher numbers mean higher precedence. Numbers less than zero have the special property that error messages will not be returned. The default precedence is zero. For example, our list of precedences is:

```
Pfirst-class=0
Pspecial-delivery=100
Pjunk=-100
```

The Semantics

This section describes the semantics of the configuration file.

Special macros, conditionals

Macros are interpolated using the construct $*x* , where *x* is the name of the macro to be interpolated. In particular, lower case letters are reserved to have special semantics, used to pass information in or out of sendmail, and some special characters are reserved to provide conditionals, etc.

Conditionals can be specified using the syntax:

```
$?x text1 $| text2 $.
```

This interpolates *text1* if the macro **$x** is set, and *text2* otherwise. The "else" (**$|**) clause may be omitted.

The following macros *must* be defined to transmit information into *sendmail:*

e	The SMTP entry message
j	The official domain name for this site
l	The format of the UNIX from line
n	The name of the daemon (for error messages)
o	The set of "operators" in addresses
q	default format of sender address

The **$e** macro is printed out when SMTP starts up. The first word must be the **$j** macro. The **$j** macro should be in RFC821 format. The **$l** and **$n** macros can be considered constants except under terribly unusual circumstances. The **$o** macro consists of a list of characters which will be considered tokens and which will separate tokens when doing parsing. For example, if "@" were in the **$o** macro, then the input "a@b" would be scanned as three tokens: "a", "@", and "b". Finally, the **$q** macro specifies how an address should appear in a message when it is defaulted. For example, on our system these definitions are:

```
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.

When sending, the **$h**, **$u**, and **$z** macros get set to the host, user, and home directory (if local) of the recipient. The first two are set from the **$@** and **$:** part of the rewriting rules, respectively.

The **$p** and **$t** macros are used to create unique strings (e.g., for the "Message-Id:" field). The **$i** macro is set to the queue id on this host; if put into the timestamp line it can be extremely useful for tracking messages. The **$v** macro is set to be the version number of *sendmail*; this is normally put in timestamps and has been proven extremely useful for debugging. The **$w** macro is set to the name of this host if it can be determined. The **$c** field is set to the "hop count," i.e., the number of times this message has been processed. This can be determined by the **−h** flag on the command line or by counting the timestamps in the message.

The **$r** and **$s** fields are set to the protocol used to communicate with sendmail and the sending hostname; these are not supported in the current version.

Special classes

The class $=w is set to be the set of all names this host is known by. This can be used to delete local hostnames.

The left hand side

The left hand side of rewriting rules contains a pattern. Normal words are simply matched directly. Metasyntax is introduced using a dollar sign. The metasymbols are:

$*	Match zero or more tokens
$+	Match one or more tokens
$−	Match exactly one token
$=*x*	Match any token in class *x*
$~*x*	Match any token not in class *x*

If any of these match, they are assigned to the symbol $*n* for replacement on the right hand side, where *n* is the index in the LHS. For example, if the LHS:

```
$−:$+
```

is applied to the input:

```
UCBARPA:eric
```

the rule will match, and the values passed to the RHS will be:

$1	UCBARPA
$2	eric

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
$[*name*$]	Canonicalize *name*
$>*n*	Call ruleset *n*
$#*mailer*	Resolve to *mailer*
$@*host*	Specify *host*
$:*user*	Specify *user*

The $*n* syntax substitutes the corresponding value from a $+, $−, $*, $=, or $~ match on the LHS. It may be used anywhere.

A host name enclosed between $[and $] is looked up using the *gethostent* (3) routines and replaced by the canonical name. For example, "$[csam$]" would become "lbl-csam.arpa" and "$[[128.32.130.2]$]" would become "vangogh.Berkeley.edu."

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 preceded 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 precede a **$>** spec; for example:

```
R$+     $:$>7$1
```

matches anything, passes that to ruleset seven, and continues; the **$:** is necessary to avoid an infinite loop.

Substitution occurs in the order described, that is, parameters from the LHS are substituted, hostnames are canonicalized, "subroutines" are called, and finally **$#**, **$@**, and **$:** are processed.

Semantics of rewriting rule sets

There are five rewriting sets that have specific semantics. These are related as depicted by figure 2.

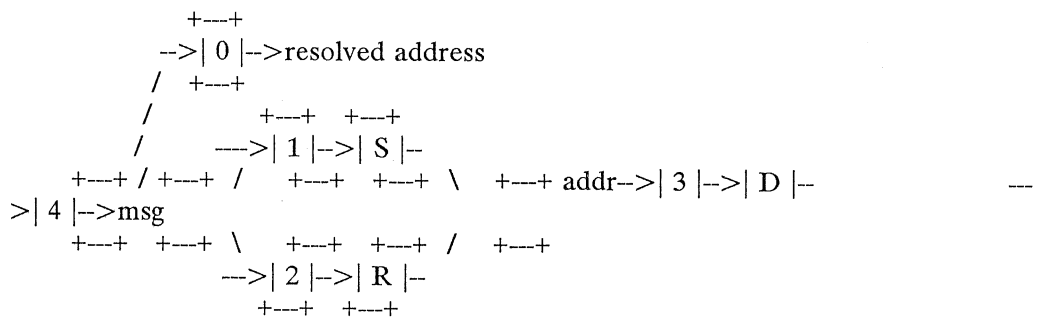


Figure 2 - Rewriting set semantics

D sender domain addition
S mailer-specific sender rewriting
R mailer-specific recipient rewriting

Ruleset three should turn the address into "canonical form." This form should have the basic syntax:

local-part@host-domain-spec

If no "@" sign is specified, then the host-domain-spec *may* be appended from the sender address (if the **C** flag is set in the mailer definition corresponding to the *sending* mailer). Ruleset three is applied by sendmail before doing anything with any address.

Ruleset zero is applied after ruleset three to addresses that are going to actually specify recipients. It must resolve to a *{mailer, host, user}* triple. The *mailer* must be defined in the mailer definitions from the configuration file. The *host* is defined into the **$h** macro for use in the argv expansion of the specified mailer.

Rulesets one and two are applied to all sender and recipient addresses respectively. They are applied before any specification in the mailer definition. They must never resolve.

Ruleset four is applied to all addresses in the message. It is typically used to translate internal to external form.

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 Appendix C. 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.

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 "xyzvax!decvax!ucbvax!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 "xyzvax!user@host" and let xyzvax worry about it from there. In summary, just get the message closer to the destination, rather than determining the full path.

Large site, many hosts - minimum information

Berkeley is an example of a large site, i.e., more than two or three hosts and multiple mail connections. We have decided that the only reasonable philosophy in our environment is to designate one host as the guru for our site. It must be able to resolve any piece of mail it receives. The other sites should have the minimum amount of information they can get away with. In addition, any information they do have should be hints rather than solid information.

For example, a typical site on our local ether network is "monet". When monet receives mail for delivery, it checks whether it knows that the destination host is directly reachable; if so, mail is sent to that host. If it receives mail for any unknown host, it just passes it directly to "ucbvax", our master host. Ucbvax may determine that the host name is illegal and reject the message, or may be able to do delivery. However, it is important to note that when a new mail connection is added, the only host that *must* have its tables updated is ucbvax; the others *may* be updated if convenient, but this is not critical.

This picture is slightly muddied due to network connections that are not actually located on ucbvax. For example, some UUCP connections are currently on "ucbarpa". However, monet *does not* know about this; the information is hidden totally between ucbvax and ucbarpa. Mail going from monet to a UUCP host is transferred via the ethernet from monet to ucbvax, then via the ethernet from ucbvax to ucbarpa, and then is submitted to UUCP. Although this involves some extra hops, we feel this is an acceptable tradeoff.

An interesting point is that it would be possible to update monet to send appropriate UUCP mail directly to ucbarpa if the load got too high; if monet failed to note a host as connected to ucbarpa it would go via ucbvax as before, and if monet incorrectly sent a message to ucbarpa it would still be sent by ucbarpa to ucbvax as before. The only problem that can occur is loops, for example, if ucbarpa thought that ucbvax had

the UUCP connection and vice versa. For this reason, updates should *always* happen to the master host first.

This philosophy results as much from the need to have a single source for the configuration files (typically built using *m4*(1) or some similar tool) as any logical need. Maintaining more than three separate tables by hand is essentially an impossible job.

Small site complete information

A small site (two or three hosts and few external connections) may find it more reasonable to have complete information at each host. This would require that each host know exactly where each network connection is, possibly including the names of each host on that network. As long as the site remains small and the the configuration remains relatively static, the update problem will probably not be too great.

Single host

This is in some sense the trivial case. The only major issue is trying to insure that you don't have to know too much about your environment. For example, if you have a UUCP connection you might find it useful to know about the names of hosts connected directly to you, but this is really not necessary since this may be determined from the syntax.

Relevant issues

The canonical form you use should almost certainly be as specified in the Arpanet protocols RFC819 and RFC822. Copies of these RFC's are included on the *sendmail* tape as *doc/rfc819.lpr* and *doc/rfc822.lpr*.

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 Berkeley one legal host might be "a.CC.Berkeley.EDU"; reading from right to left, "EDU" is a top level domain comprising educational institutions, "Berkeley" is a logical domain name, "CC" represents the Computer Center, (in this case a strictly logical entity), and "a" is a host in the Computer Center.

Beware 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 Berkeley 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, you enter lines of the form:

```
rwset address
```

where *rwset* is the rewriting set you want to use and *address* is an address to apply the set to. Test mode shows you the steps it takes as it proceeds, finally showing you the address it ends up with. You may use a comma separated list of rwsets for sequential application of rules to an input; ruleset three is always applied first. For example:

```
1,21,4 monet:bollard
```

first applies ruleset three to the input "monet:bollard". Ruleset one is then applied to the output of ruleset three, followed similarly by rulesets twenty-one and four.

If you need more detail, you can also use the "-d21" flag to turn on more debugging. For example,

```
sendmail –bt –d21.99
```

turns on an incredible amount of information; a single word address is probably going to print out several pages worth of information.

Building mailer descriptions

To add an outgoing mailer to your mail system, you will have to define the characteristics of the mailer.

Each mailer must have an internal name. This can be arbitrary, except that the names "local" and "prog" must be defined.

The pathname of the mailer must be given in the P field. If this mailer should be accessed via an IPC 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 $g" in the argv template. If the mailer must be called as **root** the "S" flag should be given; this will not reset the userid before calling the mailer (*Sendmail* must be running setuid to root for this to work). If this mailer is local (i.e., will perform final delivery rather than another network hop) the "l" flag should be given. Quote characters (backslashes and " marks) can be stripped from addresses if the "s" flag is specified; if this is not given they are passed through. If the mailer is capable of sending to more than one user on the same host in a single transaction the "m" flag should be stated. If this flag is on, then the argv template containing **$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 (The "c" configuration option must be given for this to be effective).

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 (i.e., 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:

```
From: eric@ucbarpa
To: wnj@monet, mckusick@ucbarpa
```

if and only if the "C" flag is defined in the mailer corresponding to "eric@ucbarpa".

Other flags are described in Appendix C.

The S and R fields in the mailer description are per-mailer rewriting sets to be applied to sender and recipient addresses respectively. These are applied after the sending domain is appended and the general rewriting sets (numbers one and two) are applied, but before the output rewrite (ruleset four) is applied. A typical use is to append the current domain to addresses that do not already have a domain. For example, a header of the form:

```
From: eric
```

might be changed to be:

```
From: eric@ucbarpa
```

or

```
From: ucbvax!eric
```

depending on the domain it is being shipped into. These sets can also be used to do special purpose output rewriting in cooperation with ruleset four.

The E field defines the string to use as an end-of-line indication. A string containing only newline is the default. The usual backslash escapes (\r, \n, \f, \b) may be used.

Finally, an argv template is given as the E field. It may have embedded spaces. If there is no argv with a **$u** macro in it, *sendmail* will speak SMTP to the mailer. If the pathname for this mailer is "[IPC]", the argv should be

IPC $h [*port*]

where *port* is the optional port number to connect to.

For example, the specifications:

```
Mlocal,   P=/bin/mail,  F=rlsm    S=10,  R=20,  A=mail -d $u
Mether,   P=[IPC],      F=meC,    S=11,  R=21,  A=IPC $h, M=100000
```

specifies a mailer to do local delivery and a mailer for ethernet delivery. The first is called "local", is located in the file "/bin/mail", takes a picky **-r** flag, does local delivery, quotes should be stripped from addresses, and multiple users can be delivered at once; ruleset ten should be applied to sender addresses in the 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.

APPENDIX A

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 deliver
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. *Sendmail* runs as the invoking user (rather than root) when this flag is specified.

-d*level* Set debugging level.

-o*x value* Set option *x* to the specified *value*. These options are described in Appendix B.

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, the f option may be specified as the **-s** flag.

APPENDIX B

CONFIGURATION OPTIONS

The following options may be set using the **-o** flag on the command line or the **O** line in the configuration file. Many of them cannot be specified unless the invoking user is trusted.

A*file* — Use the named *file* as the alias file. If no file is specified, use *aliases* in the current directory.

a*N* — If set, wait up to *N* minutes for an "@:@" entry to exist in the alias database before starting up. If it does not appear in *N* minutes, rebuild the database (if the **D** option is also set) or issue a warning.

B*c* — Set the blank substitution character to *c*. Unquoted spaces in addresses are replaced by this character.

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 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*x value*	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.
N*netname*	The name of the home network; "ARPA" by default. The the argument of an SMTP "HELO" command is checked against "hostname.netname" where *hostname* is requested from the kernel for the current connection. If they do not match, "Received:" lines are augmented by the name that is determined in this manner so that messages can be traced accurately.
o	Assume that the headers may be in old format, i.e., spaces delimit names. This actually turns on an adaptive algorithm: if any recipient address contains a comma, parenthesis, or angle bracket, it will be assumed that commas already exist. If this flag is not on, only commas delimit names. Headers are always output with commas between the names.
Q*dir*	Use the named *dir* as the queue directory.
q*factor*	Use *factor* as the multiplier in the map function to decide when to just queue up jobs rather than run them. This value is divided by the difference between the current load average and the load average limit (**x** flag) to determine the maximum message priority that will be sent. Defaults to 10000.
r*time*	Timeout reads after *time* interval.
S*file*	Log statistics in the named *file*.
s	Be super-safe when running things, i.e., always instantiate the queue file, even if you are going to attempt immediate delivery. *Sendmail* always instantiates the queue file before returning control the the client under any circumstances.
T*time*	Set the queue timeout to *time*. After this interval, messages that have not been successfully sent will be returned to the sender.
t*S,D*	Set the local time zone name to *S* for standard time and *D* for daylight time; this is only used under version six.
u*n*	Set the default userid for mailers to *n*. Mailers without the *S* flag in the mailer definition will run as this user.
v	Run in verbose mode.
x*LA*	When the system load average exceeds *LA*, just queue messages (i.e., don't try to send them).
X*LA*	When the system load average exceeds *LA*, refuse incoming SMTP connections.
y*fact*	The indicated *fact*or is added to the priority (thus *lowering* the priority of the job) for each recipient, i.e., this value penalizes jobs with large numbers of recipients.

Y	If set, deliver each job that is run from the queue in a separate process. Use this option if you are short of memory, since the default tends to consume considerable amounts of memory while the queue is being processed.
z*fact*	The indicated *fact*or is multiplied by the message class (determined by the Precedence: field in the user header and the **P** lines in the configuration file) and subtracted from the priority. Thus, messages with a higher Priority: will be favored.
Z*fact*	The *fact*or is added to the priority every time a job is processed. Thus, each time a job is processed, its priority will be decreased by the indicated value. In most environments this should be positive, since hosts that are down are all too often down for a long time.

APPENDIX C

MAILER FLAGS

The following flags may be set in the mailer description.

f	The mailer wants a **-f** *from* flag, but only if this is a network forward operation (i.e., the mailer will give an error if the executing user does not have special permissions).
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. This flag is suppressed if given from an "unsafe" environment (e.g, a user's mail.cf file).
n	Do not insert a UNIX-style "From" line on the front of the message.
l	This mailer is local (i.e., final delivery will be performed).
s	Strip quote characters off of the address before calling the mailer.
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.
F	This mailer wants a "From:" header line.
D	This mailer wants a "Date:" header line.
M	This mailer wants a "Message-Id:" header line.
x	This mailer wants a "Full-Name:" header line.
P	This mailer wants a "Return-Path:" line.
u	Upper case should be preserved in user names for this mailer.
h	Upper case should be preserved in host names for this mailer.
A	This is an Arpanet-compatible mailer, and all appropriate modes should be set.
U	This mailer wants Unix-style "From" lines with the ugly UUCP-style ""remote from <host>" on the end.
e	This mailer is expensive to connect to, so try to avoid connecting normally; any necessary connection will occur during a queue run.
X	This mailer want to use the hidden dot algorithm as specified in RFC821; basically, any line beginning with a dot will have an extra dot prepended (to be stripped at the other end). This insures that lines in the message containing a dot will not terminate the message prematurely.
L	Limit the line lengths as specified in RFC821.
P	Use the return-path in the SMTP "MAIL FROM:" command rather than just the return address; although this is required in RFC821, many hosts do not process return paths properly.

I — This mailer will be speaking SMTP to another *sendmail*- as such it can use special protocol features. This option is not required (i.e., if this option is omitted the transmission will still operate successfully, although perhaps not as efficiently as possible).

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.

E — Escape lines beginning with "From" in the message with a '>' sign.

APPENDIX D

OTHER CONFIGURATION

There are some configuration changes that can be made by recompiling *sendmail*. These are located in three places:

md/config.m4	These contain operating-system dependent descriptions. They are interpolated into the Makefiles in the *src* and *aux* directories. This includes information about what version of UNIX you are running, what libraries you have to include, etc.
src/conf.h	Configuration parameters that may be tweaked by the installer are included in conf.h.
src/conf.c	Some special routines and a few variables may be defined in conf.c. For the most part these are selected from the settings in conf.h.

Parameters in md/config.m4

The following compilation flags may be defined in the *m4CONFIG* macro in *md/config.m4* to define the environment in which you are operating.

V6	If set, this will compile a version 6 system, with 8-bit user id's, single character tty id's, etc.
VMUNIX	If set, you will be assumed to have a Berkeley 4BSD or 4.1BSD, including the *vfork* (2) system call, special types defined in <sys/types.h> (e.g, u_char), etc.

If none of these flags are set, a version 7 system is assumed.

You will also have to specify what libraries to link with *sendmail* in the *m4LIBS* macro. Most notably, you will have to include **-ljobs** if you are running a 4.1BSD system.

Parameters in src/conf.h

Parameters and compilation options are defined in conf.h. Most of these need not normally be tweaked; common parameters are all in sendmail.cf. However, the sizes of certain primitive vectors, etc., are included in this file. The numbers following the parameters are their default value.

MAXLINE[1024]	The maximum line length of any input line. If message lines exceed this length they will still be processed correctly; however, header lines, configuration file lines, alias lines, etc., must fit within this limit.
MAXNAME[256]	The maximum length of any name, such as a host or a user name.

MAXFIELD[2500]	The maximum total length of any header field, including continuation lines.
MAXPV[40]	The maximum number of parameters to any mailer. This limits the number of recipients that may be passed in one transaction.
MAXHOP[17]	When a message has been processed more than this number of times, sendmail rejects the message on the assumption that there has been an aliasing loop. This can be determined from the **-h** flag or by counting the number of trace fields (i.e, "Received:" lines) in the message header.
MAXATOM[100]	The maximum number of atoms (tokens) in a single address. For example, the address "eric@Berkeley" is three atoms.
MAXMAILERS[25]	The maximum number of mailers that may be defined in the configuration file.
MAXRWSETS[30]	The maximum number of rewriting sets that may be defined.
MAXPRIORITIES[25]	The maximum number of values for the "Precedence:" field that may be defined (using the **P** line in sendmail.cf).
MAXTRUST[30]	The maximum number of trusted users that may be defined (using the **T** line in sendmail.cf).
MAXUSERENVIRON[40]	The maximum number of items in the user environment that will be passed to subordinate mailers.
QUEUESIZE[600]	The maximum number of entries that will be processed in a single queue run.

A number of other compilation options exist. These specify whether or not specific code should be compiled in.

DBM	If set, the "DBM" package in UNIX is used (see *dbm(3X)* in [UNIX80]). If not set, a much less efficient algorithm for processing aliases is used.
NDBM	If set, the new version of the DBM library that allows multiple databases will be used. "DBM" must also be set.
DEBUG	If set, debugging information is compiled in. To actually get the debugging output, the **-d** flag must be used.
LOG	If set, the *syslog* routine in use at some sites is used. This makes an informational log record for each message processed, and makes a higher priority log record for internal system errors.
QUEUE	This flag should be set to compile in the queueing code. If this is not set, mailers must accept the mail immediately or it will be returned to the sender.

SMTP	If set, the code to handle user and server SMTP will be compiled in. This is only necessary if your machine has some mailer that speaks SMTP.
DAEMON	If set, code to run a daemon is compiled in. This code is for 4.2 or 4.3BSD.
UGLYUUCP	If you have a UUCP host adjacent to you which is not running a reasonable version of *rmail*, you will have to set this flag to include the "remote from sysname" info on the from line. Otherwise, UUCP gets confused about where the mail came from.
NOTUNIX	If you are using a non-UNIX mail format, you can set this flag to turn off special processing of UNIX-style "From " lines.

Configuration in src/conf.c

Not all header semantics are defined in the configuration file. Header lines that should only be included by certain mailers (as well as other more obscure semantics) must be specified in the *HdrInfo* table in *conf.c*. This table contains the header name (which should be in all lower case) and a set of header control flags (described below), The flags are:

H_ACHECK	Normally when the check is made to see if a header line is compatible with a mailer, *sendmail* will not delete an existing line. If this flag is set, *sendmail* will delete even existing header lines. That is, if this bit is set and the mailer does not have flag bits set that intersect with the required mailer flags in the header definition in sendmail.cf, the header line is *always* deleted.
H_EOH	If this header field is set, treat it like a blank line, i.e., it will signal the end of the header and the beginning of the message text.
H_FORCE	Add this header entry even if one existed in the message before. If a header entry does not have this bit set, *sendmail* will not add another header line if a header line of this name already existed. This would normally be used to stamp the message by everyone who handled it.
H_TRACE	If set, this is a timestamp (trace) field. If the number of trace fields in a message exceeds a preset amount the message is returned on the assumption that it has an aliasing loop.
H_RCPT	If set, this field contains recipient addresses. This is used by the **-t** flag to determine who to send to when it is collecting recipients from the message.
H_FROM	This flag indicates that this field specifies a sender. The order of these fields in the *HdrInfo* table specifies *sendmail's* preference for which field to return error messages to.

Let's look at a sample *HdrInfo* specification:

```
struct hdrinfo          HdrInfo[] =
{
        /* originator fields, most to least significant */
    "resent-sender",H_FROM,
    "resent-from",  H_FROM,
    "sender",       H_FROM,
    "from",         H_FROM,
    "full-name",    H_ACHECK,
        /* destination fields */
    "to",           H_RCPT,
    "resent-to",    H_RCPT,
    "cc",           H_RCPT,
        /* message identification and control */
    "message",      H_EOH,
    "text",         H_EOH,
        /* trace fields */
    "received",     H_TRACE|H_FORCE,

    NULL,           0,
};
```

This structure indicates that the "To:", "Resent-To:", and "Cc:" fields all specify recipient addresses. Any "Full-Name:" field will be deleted unless the required mailer flag (indicated in the configuration file) is specified. The "Message:" and "Text:" fields will terminate the header; these are specified in new protocols [NBS80] or used by random dissenters around the network world. The "Received:" field will always be added, and can be used to trace messages.

There are a number of important points here. First, header fields are not added automatically just because they are in the *HdrInfo* structure; they must be specified in the configuration file in order to be added to the message. Any header fields mentioned in the configuration file but not mentioned in the *HdrInfo* structure have default processing performed; that is, they are added unless they were in the message already. Second, the *HdrInfo* structure only specifies cliched processing; certain headers are processed specially by ad hoc code regardless of the status specified in *HdrInfo*. For example, the "Sender:" and "From:" fields are always scanned on ARPANET mail to determine the sender; this is used to perform the "return to sender" function. The "From:" and "Full-Name:" fields are used to determine the full name of the sender if possible; this is stored in the macro **$x** and used in a number of ways.

The file *conf.c* also contains the specification of ARPANET reply codes. There are four classifications these fall into:

```
char    Arpa_Info[] =   "050";      /* arbitrary info */
char    Arpa_TSyserr[] =            "455";  /* some (transient) system error */
char    Arpa_PSyserr[] =            "554";  /* some (permanent) system error */
char    Arpa_Usrerr[] =             "554";  /* some (fatal) user error */
```

The class *Arpa_Info* is for any information that is not required by the protocol, such as forwarding information. *Arpa_TSyserr* and *Arpa_PSyserr* is printed by the *syserr* routine. TSyserr is printed out for transient errors, that is, errors that are likely to go away without explicit action on the part of a systems administrator. PSyserr is printed for permanent errors. The distinction is made based on the value of *errno*. Finally, *Arpa_Usrerr* is the result of a user error and is generated by the *usrerr* routine; these are generated when the user has specified something wrong, and hence the error is

permanent, i.e., it will not work simply by resubmitting the request.

If it is necessary to restrict mail through a relay, the *checkcompat* routine can be modified. This routine is called for every recipient address. It can return **TRUE** to indicate that the address is acceptable and mail processing will continue, or it can return **FALSE** to reject the recipient. If it returns false, it is up to *checkcompat* to print an error message (using *usrerr*) saying why the message is rejected. For example, *checkcompat* could read:

```
bool
checkcompat(to)
        register ADDRESS *to;
{
        if (MsgSize > 50000 && to->q_mailer != LocalMailer)
        {
                usrerr("Message too large for non-local delivery");
                NoReturn = TRUE;
                return (FALSE);
        }
        return (TRUE);
}
```

This would reject messages greater than 50000 bytes unless they were local. The *NoReturn* flag can be sent to suppress the return of the actual body of the message in the error return. The actual use of this routine is highly dependent on the implementation, and use should be limited.

Configuration in src/daemon.c

The file *src/daemon.c* contains a number of routines that are dependent on the local networking environment. The version supplied is specific to 4.3 BSD.

The routine *maphostname* is called to convert strings within $[... $] symbols. It can be modified if you wish to provide a more sophisticated service, e.g., mapping UUCP host names to full paths.

APPENDIX E

SUMMARY OF SUPPORT FILES

This is a summary of the support files that *sendmail* creates or generates.

/usr/lib/sendmail	The binary of *sendmail*.
/usr/bin/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/bin/mailq	Prints a listing of the mail queue. This program is equivalent to using the **−bp** flag to *sendmail*.
/usr/lib/sendmail.cf	The configuration file, in textual form.
/usr/lib/sendmail.fc	The configuration file represented as a memory image.
/usr/lib/sendmail.hf	The SMTP help file.
/usr/lib/sendmail.st	A statistics file; need not be present.
/usr/lib/aliases	The textual version of the alias file.
/usr/lib/aliases.{pag,dir}	The alias file in *dbm* (3) format.
/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.

Chapter 14: Timed Installation and Operations

Introduction

The clock synchronization service for the UNIX 4.3BSD operating system is composed of a collection of time daemons (*timed*) running on the machines in a local area network. The algorithms implemented by the service is based on a master-slave scheme. The time daemons communicate with each other using the *Time Synchronization Protocol* (TSP) which is built on the DARPA UDP protocol and described in detail in [4].

A time daemon has a twofold function. First, it supports the synchronization of the clocks of the various hosts in a local area network. Second, it starts (or takes part in) the election that occurs among slave time daemons when, for any reason, the master disappears. The synchronization mechanism and the election procedure employed by the program *timed* are described in other documents [1,2,3]. The next paragraphs are a brief overview of how the time daemon works. This document is mainly concerned with the administrative and technical issues of running *timed* at a particular site.

A *master time daemon* measures the time differences between the clock of the machine on which it is running and those of all other machines. The master computes the *network time* as the average of the times provided by nonfaulty clocks. (A clock is considered to be faulty when its value is more than a small specified interval apart from the majority of the clocks of the other machines [1,2].)

It then sends to each *slave time daemon* the correction that should be performed on the clock of its machine. This process is repeated periodically. Since the correction is expressed as a time difference rather than an absolute time, transmission delays do not interfere with the accuracy of the synchronization. When a machine comes up and joins the network, it starts a slave time daemon which will ask the master for the correct time and will reset the machine's clock before any user activity can begin. The time daemons are able to maintain a single network time in spite of the drift of clocks away from each other. The present implementation keeps processor clocks synchronized within 20 milliseconds.

To ensure that the service provided is continuous and reliable, it is necessary to implement an election algorithm to elect a new master should the machine running the current master crash, the master terminate (for example, because of a run-time error), or the network be partitioned. Under our algorithm, slaves are able to realize when the master has stopped functioning and to elect a new master from among themselves. It is important to note that, since the failure of the master results only in a gradual divergence of clock values, the election need not occur immediately.

The machines that are gateways between distinct local area networks require particular care. A time daemon on such machines may act as a *submaster*. This artifact depends on the current inability of transmission protocols to broadcast a message on a network other than the one to which the broadcasting machine is connected. The submaster appears as a slave on one network, and as a master on one or more of the other networks to which it is connected.

A submaster classifies each network as one of three types. A *slave network* is a network on which the submaster acts as a slave. There can only be one slave network. A *master network* is a network on which the submaster acts as a master. An *ignored network* is any other network which already has a valid master. The submaster tries periodically to become master on an ignored network, but gives up immediately if a master already exists.

Guidelines

While the synchronization algorithm is quite general, the election one, requiring a broadcast mechanism, puts constraints on the kind of network on which time daemons can run. The time daemon will only work on networks with broadcast capability augmented with point-to-point links. Machines that are only connected to point-to-point, non-broadcast networks may not use the time daemon.

If we exclude submasters, there will normally be, at most, one master time daemon in a local area internetwork. During an election, only one of the slave time daemons will become the new master. However, because of the characteristics of its machine, a slave can be prevented from becoming the master. Therefore, a subset of machines must be designated as potential master time daemons. A master time daemon will require CPU resources proportional to the number of slaves, in general, more than a slave time daemon, so it may be advisable to limit master time daemons to machines with more powerful processors or lighter loads. Also, machines with inaccurate clocks should not be used as masters. This is a purely administrative decision: an organization may well allow all of its machines to run master time daemons.

At the administrative level, a time daemon on a machine with multiple network interfaces, may be told to ignore all but one network or to ignore one network. This is done with the *−n network* and *−i network* options respectively at start-up time. Typically, the time daemon would be instructed to ignore all but the networks belonging to the local administrative control.

There are some limitations to the current implementation of the time daemon. It is expected that these limitations will be removed in future releases. The constant NHOSTS in /usr/src/etc/timed/globals.h limits the maximum number of machines that may be directly controlled by one master time daemon. The current maximum is 29 (NHOSTS - 1). The constant must be changed and the program recompiled if a site wishes to run *timed* on a larger (inter)network.

In addition, there is a *pathological situation* to be avoided at all costs, that might occur when time daemons run on multiply-connected local area networks. In this case, as we have seen, time daemons running on gateway machines will be submasters and they will act on some of those networks as master time daemons. Consider machines A and B that are both gateways between networks X and Y. If time daemons were started on both A and B without constraints, it would be possible for submaster time daemon A to be a slave on network X and the master on network Y, while submaster time daemon B is a slave on network Y and the master on network X. This *loop* of master time daemons will not function properly or guarantee a unique time on both networks, and will cause the submasters to use large amounts of system resources in the form of network bandwidth and CPU time. In fact, this kind of *loop* can also be generated with more than two master time daemons, when several local area networks are interconnected.

Installation

In order to start the time daemon on a given machine, the following lines should be added to the *local daemons* section in the file */etc/rc.local*:

```
if [ -f /etc/timed ]; then
        /etc/timed flags & echo -n ' timed' >/dev/console
fi
```

In any case, they must appear after the network is configured via ifconfig(8).

Also, the file */etc/services* should contain the following line:

```
timed           525/udp                 timeserver
```

The *flags* are: to consider the named network. to ignore the named network. to place tracing information in */usr/adm/timed.log*. to allow this time daemon to become a master. A time daemon run without this option will be forced in the state of slave during an election.

Daily Operation

Timedc(8) is used to control the operation of the time daemon. It may be used to:

- measure the differences between machines' clocks,
- find the location where the master *timed* is running,
- cause election timers on several machines to expire at the same time,
- enable or disable tracing of messages received by *timed*.

See the manual page on *timed* (8) and *timedc* (8) for more detailed information.

The *date(1)* command can be used to set the network date. In order to set the time on a single machine, the *-n* flag can be given to date(1).

References

R. Gusella and S. Zatti, *TEMPO: A Network Time Controller for Distributed Berkeley UNIX System*, USENIX Summer Conference Proceedings, Salt Lake City, June 1984.

R. Gusella and S. Zatti, *Clock Synchronization in a Local Area Network*, University of California, Berkeley, Technical Report, *to appear*.

R. Gusella and S. Zatti, *An Election Algorithm for a Distributed Clock Synchronization Program*, University of California, Berkeley, CS Technical Report #275, Dec. 1985.

R. Gusella and S. Zatti, *The Berkeley UNIX 4.3BSD Time Synchronization Protocol*, UNIX Programmer's Manual, 4.3 Berkeley Software Distribution, Volume 2c.

Chapter 15: Name Server Operations for BIND

Introduction

The Berkeley Internet Name Domain (BIND) Server implements the DARPA Internet name server for the UNIX† operating system. †UNIX is a Trademark of AT&T Bell Laboratories A name server is a network service that enables clients to name resources or objects and share this information with other objects in the network. This in effect is a distributed data base system for objects in a computer network. BIND is fully intergrated into 4.3BSD network programs for use in storing and retrieving host names and address. The system administrator can configure the system to use BIND as a replacement to the original host table lookup of information in the network hosts file */etc/hosts*. The default configuration for 4.3BSD uses BIND.

Building A System with a Name Server

BIND is comprised of two parts. One is the user interface called the *resolver* which consists of a group of routines that reside in the C library */lib/libc.a*. Second is the actual server called *named*. This is a daemon that runs in the background and services queries on a given network port. The standard port for UDP and TCP is specified in */etc/services*.

Resolver Routines in libc

When building your 4.3BSD system you may either build the C library to use the name server resolver routines or use the host table lookup routines to do host name and address resolution. The default resolver for 4.3BSD uses the name server.

Building the C library to use the name server changes the way *gethostbyname* (3N), *gethostbyaddr* (3N), and *sethostent* (3N) do their functions. The name server renders *gethostent* (3N) obsolete, since it has no concept of a next line in the database. These library calls are built with the resolver routines needed to query the name server.

The *resolver* is comprised of a few routines that build query packets and exchange them with the name server.

Before building the C library, set the variable *HOSTLOOKUP* equal to *named* in */usr/src/lib/libc/Makefile*. You then make and install the C library and compiler and then compile the rest of the 4.3BSD system. For more information see section 6.6 of "Installing and Operating 4.3BSD on the VAX‡". ‡VAX is a Trademark of Digital Equipment Corporation

The Name Service

The basic function of the name server is to provide information about network objects by answering queries. The specifications for this name server are defined in RFC882, RFC883, RFC973 and RFC974. These documents can be found in */usr/src/etc/named/doc* in 4.3BSD or *ftp*ed from sri-nic.arpa. It is also recommeded that you read the related manual pages, *named* (8), *resolver* (3), and *resolver* (5).

The advantage of using a name server over the host table lookup for host name resolution is to avoid the need for a single centralized clearinghouse for all names. The authority for this information can be delegated to the different organizations on the network responsible for it.

The host table lookup routines require that the master file for the entire network be maintained at a central location by a few people. This works fine for small networks where there are only a few machines and the different organizations responsible for them cooperate. But this does not work well for large networks where machines cross organizational boundaries.

With the name server, the network can be broken into a hierarchy of domains. The name space is organized as a tree according to organizational or administrative boundaries. Each node, called a *domain*, is given a label, and the name of the domain is the concatenation of all the labels of the domains from the root to the current domain, listed from right to left separated by dots. A label need only be unique within its domain. The whole space is partitioned into several areas called *zones*,

each starting at a domain and extending down to the leaf domains or to domains where other zones start. Zones usually represent administrative boundaries. An example of a host address for a host at the University of California, Berkeley would look as follows:

monet . Berkeley . EDU

The top level domain for educational organizations is EDU; Berkeley is a subdomain of EDU and monet is the name of the host.

Types of Servers

There are three types of servers, Master, Caching and Remote.

Master Servers

A Master Server for a domain is the authority for that domain. This server maintains all the data corresponding to its domain. Each domain should have at least two master servers, a primary master and some secondary masters to provide backup service if the primary is unavailable or overloaded. A server may be a master for multiple domains, being primary for some domains and secondary for others.

Primary

A Primary Master Server is a server that loads its data from a file on disk. This server may also delegate authority to other servers in its domain.

Secondary

A Secondary Master Server is a server that is delegated authority and receives its data for a domain from a primary master server. At boot time, the secondary server requests all the data for the given zone from the primary master server. This server then periodically checks with the primary server to see if it needs to update its data.

Caching Only Server

All servers are caching servers. This means that the server caches the information that it receives for use until the data expires. A *Caching Only Server* is a server that is not authoritative for any domain. This server services queries and asks other servers, who have the authority, for the information needed. All servers keep data in their cache until the data expires, based on a time to live field attached to the data when it is received from another server.

Remote Server

A Remote Server is an option given to people who would like to use a name server on their workstation or on a machine that has a limited amount of memory and CPU cycles. With this option you can run all of the networking programs that use the name server without the name server running on the local machine. All of the queries are serviced by a name server that is running on another machine on the network.

Setting up Your Own Domain

When setting up a domain that is going to be on a public network the site administrator should contact the organization in charge of the network and request the appropriate domain registration form. An organization that belongs to multiple networks (such as *CSNET*, *DARPA Internet* and *BITNET*) should register with only one network.

The contacts are as follows:

DARPA Internet

Sites that are already on the DARPA Internet and need information on setting up a domain should contact *HOSTMASTER@SRI-NIC.ARPA*. You may also want to be placed on the BIND mailing list, which is a mail group for people on the DARPA Internet running BIND. The group discusses future design decisions, operational problems, and other related topic. The address to request being placed on this mailing list is:

bind-request @ ucbarpa . Berkeley . EDU.

CSNET

A *CSNET* member organization that has not registered its domain name should contact the *CSNET* Coordination and Information Center (*CIC*) for an application and information about setting up a domain.

An organization that already has a registered domain name should keep the *CIC* informed about how it would like its mail routed. In general, the *CSNET* relay will prefer to send mail via *CSNET* (as opposed to *BITNET* or the *Internet*) if possible. For an organization on multiple networks, this may not always be the preferred behavior. The *CIC* can be reached via electronic mail at *cic @ sh . cs . net*, or by phone at (617) 497-2777.

BITNET

If you are on the BITNET and need to set up a domain, contact INFO@BITNIC.

Files

The name server uses several files to load its data base. This section covers the files and their formats needed for *named*.

Boot File

This is the file that is first read when *named* starts up. This tells the server what type of server it is, which zones it has authority over and where to get its initial data. The default location for this file is */etc/named.boot*. However this can be changed by setting the *BOOTFILE* variable when you compile *named* or by specifying the location on the command line when *named* is started up.

Domain

The line in the boot file that designates the default domain for the server looks as follows:

domain Berkeley.Edu

The name server uses this information when it receives a query for a name without a ".". When it receives one of these queries, it appends the name in the second field to the query name.

Primary Master

The line in the boot file that designates the server as a primary server for a zone looks as follows:

primary Berkeley.Edu /etc/ucbhosts

The first field specifies that the server is a primary one for the zone stated in the second field. The third field is the name of the file from which the data is read.

Secondary Master

The line for a secondary server is similar to the primary except for the word secondary and the third field.

secondary Berkeley.Edu 128.32.0.10 128.32.0.4

The first field specifies that the server is a secondary master server for the zone stated in the second field. The rest of the line, lists the network addresses for the name servers that are primary for the zone. The secondary server gets its data across the network from the listed servers. Each server is tried in the order listed until it successfully receives the data from a listed server.

Caching Only Server

You do not need a special line to designate that a server is a caching server. What denotes a caching only server is the absence of authority lines, such as *secondary* or *primary* in the boot file.

All servers should have a line as follows in the boot file to prime the name servers cache:

cache . /etc/named.ca

For information on cache file see section on *Cache Initialization*.

Remote Server

To set up a host that will use a remote server instead of a local server to answer queries, the file */etc/resolv.conf* needs to be created. This file designates the name servers on the network that should be sent queries. It is not advisable to create this file if you have a local server running. If this file exists it is read almost every time *gethostbyname()* or *gethostbyaddr()* is called.

Cache Initialization

named.ca

The name server needs to know the server that is the authoritative name server for the network. To do this we have to prime the name server's cache with the address of these higher authorities. The location of this file is specified in the boot file. This file uses the Standard Resource Record Format covered further on in this paper.

Domain Data Files

There are three standard files for specifying the data for a domain. These are *named.local*, *hosts* and *host.rev*. These files use the Standard Resource Record Format covered later in this paper.

named.local

This file specifies the address for the local loopback interface, better known as *localhost* with the network address 127.0.0.1. The location of this file is specified in the boot file.

hosts

This file contains all the data about the machines in this zone. The location of this file is specified in the boot file.

hosts.rev

This file specifies the IN-ADDR.ARPA domain. This is a special domain for allowing address to name mapping. As internet host addresses do not fall within domain boundaries, this special domain was formed to allow inverse mapping. The IN-ADDR.ARPA domain has four labels preceding it. These labels correspond to the 4 octets of an Internet address. All four octets must be specified even if an octets is zero. The Internet address 128.32.0.4 is located in the domain 4.0.32.128.IN-ADDR.ARPA. This reversal of the address is awkward to read but allows for the natural grouping of hosts in a network.

Standard Resource Record Format

The records in the name server data files are called resource records. The Standard Resource Record Format (RR) is specified in RFC882 and RFC973. The following is a general description of these records:

{name} {ttl} addr-class Record Type Record Specific data

Resource records have a standard format shown above. The first field is always the name of the domain record. For some RR's the name may be left blank; in that case it takes on the name of the previous RR. The second field is an optional time to live field. This specifies how long this data will be stored in the data base. By leaving this field blank the default time to live is specified in the *Start Of Authority* resource record (see below). The third field is the address class; there are currently two classes: *IN* for internet addresses and *ANY* for all address classes. The fourth field states the type of the resource record. The fields after that are dependent on the type of the RR. Case is preserved in names and data fields when loaded into the name server. All comparisons and lookups in the name server data base are case insensitive.

The following characters have special meanings:

.	A free standing dot in the name field refers to the current domain.
@	A free standing @ in the name field denotes the current origin.
..	Two free standing dots represent the null domain name of the root when used in the name field.
X	Where X is any character other than a digit (0-9), quotes that character so that its special meaning does not apply. For example, "\." can be used to place a dot character in a label.
DDD	Where each D is a digit, is the octet corresponding to the decimal number described by DDD. The resulting octet is assumed to be text and is not checked for special meaning.
()	Parentheses are used to group data that crosses a line. In effect, line terminations are not recognized within parentheses.
;	Semicolon starts a comment; the remainder of the line is ignored.
*	An asterisk signifies wildcarding.

Most resource records will have the current origin appended to names if they are not terminated by a ".". This is useful for appending the current domain name to the data, such as machine names, but may cause problems where you do not want this to happen. A good rule of thumb is that, if the name is not in of the domain for which you are creating the data file, end the name with a ".".

$INCLUDE

An include line begins with $INCLUDE, starting in column 1, and is followed by a file name. This feature is particularly useful for separating different types of data into multiple files. An example would be:

```
$INCLUDE /usr/named/data/mailboxs
```

The line would be interpreted as a request to load the file */usr/named/data/mailboxes*. The $INCLUDE command does not cause data to be loaded into a different zone or tree. This is simply a way to allow data for a given zone to be organized in separate files. For example, mailbox data might be kept separately from host data using this mechanism.

$ORIGIN

The origin is a way of changing the origin in a data file. The line starts in column 1, and is followed by a domain origin. This is useful for putting more then one domain in a data file.

SOA - Start Of Authority

name	*{ttl}*	*addr-class*	*SOA*	*Origin*	*Person in charge*
@		IN	SOA	ucbvax.Berkeley.Edu.	kjd.ucbvax.Berkeley.Edu. (
			1.1	; Serial	
			3600	; Refresh	
			300	; Retry	
			3600000	; Expire	
			3600)	; Minimum	

The *Start of Authority, SOA,* record designates the start of a zone. The name is the name of the zone. Origin is the name of the host on which this data file resides. Person in charge is the mailing address for the person responsible for the name server. The serial number is the version number of this data file, this number should be incremented whenever a change is made to the data. The name server cannot handle numbers over 9999 after the decimal point. The refresh indicates how often, in seconds, a secondary name servers is to check with the primary name server to see if an update is needed. The retry indicates how long, in seconds, a secondary server is to retry after a failure to check for a refresh. Expire is the upper limit, in seconds, that a secondary name server is to use the data before it expires for lack of getting a refresh. Minimum is the default number of seconds to be used for the time to live field on resource records. There should only be one *SOA* record per zone.

NS - Name Server

{name}	*{ttl}*	*addr-class*	*NS*	*Name servers name*
		IN	NS	ucbarpa.Berkeley.Edu.

The *Name Server* record, *NS*, lists a name server responsible for a given domain. The first name field lists the domain that is serviced by the listed name server. There should be one *NS* record for each Primary Master server for the domain.

A - Address

{name}	*{ttl}*	*addr-class*	*A*	*address*
ucbarpa		IN	A	128.32.0.4
		IN	A	10.0.0.78

The *Address* record, *A*, lists the address for a given machine. The name field is the machine name and the address is the network address. There should be one *A* record for each address of the machine.

HINFO - Host Information

{name}	*{ttl}*	*addr-class*	*HINFO*	*Hardware*	*OS*
		ANY	HINFO	VAX-11/780	UNIX

Host Information resource record, *HINFO*, is for host specific data. This lists the hardware and operating system that are running at the listed host. It should be noted that only a single space separates the hardware info and the operating system info. If you want to include a space in the machine name you must quote the name. Host information is not specific to any address class, so *ANY* may be used for the address class. There should be one *HINFO* record for each host.

WKS - Well Known Services

{name}	*{ttl}*	*addr-class*	*WKS*	*address*	*protocol*	*list of services*
		IN	WKS	128.32.0.10	UDP	who route timed domain
		IN	WKS	128.32.0.10	TCP	(echo telnet discard sunrpc sftp uucp-path systat daytime netstat qotd nntp link chargen ftp auth time whois mtp pop rje finger smtp supdup hostnames domain nameserver)

The *Well Known Services* record, *WKS*, describes the well known services supported by a particular protocol at a specified address. The list of services and port numbers come from the list of services specified in */etc/services*. There should be only one *WKS* record per protocol per address.

CNAME - Canonical Name

aliases	*{ttl}*	*addr-class*	*CNAME*	*Canonical name*
ucbmonet		IN	CNAME	monet

Canonical Name resource record, *CNAME*, specifies an alias for a canonical name. An alias should be unique and all other resource records should be associated with the canonical name and not with the alias. Do not create an alias and then use it in other resource records.

PTR - Domain Name Pointer

name	*{ttl}*	*addr-class*	*PTR*	*real name*
7.0		IN	PTR	monet.Berkeley.Edu.

A *Domain Name Pointer* record, *PTR*, allows special names to point to some other location in the domain. The above example of a *PTR* record is used in setting up reverse pointers for the special *IN-ADDR.ARPA* domain. This line is from the example *hosts.Pv* file. *PTR* names should be unique to the zone.

MB - Mailbox

name	*{ttl}*	*addr-class*	*MB*	*Machine*
miriam		IN	MB	vineyd.DEC.COM.

MB is the *Mailbox* record. This lists the machine where a user wants to receive mail. The name field is the users login; the machine field denotes the machine to which mail is to be delivered. Mail Box names should be unique to the zone.

MR - Mail Rename Name

name	*{ttl}*	*addr-class*	*MR*	*corresponding MB*
Postmistress		IN	MR	miriam

Main Rename, MR, can be used to list aliases for a user. The name field lists the alias for the name listed in the fourth field, which should have a corresponding *MB* record.

MINFO - Mailbox Information

name	*{ttl}*	*addr-class*	*MINFO*	*requests*	*maintainer*
BIND		IN	MINFO	BIND-REQUEST	kjd.Berkeley.Edu.

Mail Information record, *MINFO*, creates a mail group for a mailing list. This resource record is usually associated with a mail group *Mail Group*, but may be used with a *Mail Box* record. The *name* specifies the name of the mailbox. The *requests* field is where mail such as requests to be added to a mail group should be sent. The *maintainer* is a mailbox that should receive error messages. This is particularly appropriate for mailing lists when errors in members names should be reported to a person other than the sender.

MG - Mail Group Member

{mail group name}	*{ttl}*	*addr-class*	*MG*	*member name*
		IN	MG	Bloom

Mail Group, MG lists members of a mail group.

An example for setting up a mailing list is as follows:

Bind	IN	MINFO	Bind-Request	kjd.Berkeley.Edu.
	IN	MG	Ralph.Berkeley.Edu.	
	IN	MG	Zhou.Berkeley.Edu.	
	IN	MG	Painter.Berkeley.Edu.	
	IN	MG	Riggle.Berkeley.Edu.	
	IN	MG	Terry.pa.Xerox.Com.	

MX - Mail Exchanger

name	*{ttl}*	*addr-class*	*MX*	*preference value*	*mailer exchanger*
Munnari.OZ.AU.		IN	MX	0	Seismo.CSS.GOV.
*.IL.		IN	MX	0	RELAY.CS.NET.

Main Exchanger records, *MX*, are used to specify a machine that knows how to deliver mail to a machine that is not directly connected to the network. In the first example, above, Seismo.CSS.GOV. is a mail gateway that knows how to deliver mail to Munnari.OZ.AU. but other machines on the network can not deliver mail directly to Munnari. These two machines may have a private connection or use a different transport medium. The preference value is the order that a mailer should follow when there is more then one way to deliver mail to a single machine. See RFC974 for more detailed information.

Wildcard names containing the character "*" may be used for mail routing with *MX* records. There are likely to be servers on the network that simply state that any mail to a domain is to be routed through a relay. Second example, above, all mail to hosts in the domain IL is routed through RELAY.CS.NET. This is done by creating a wildcard resource record, which states that *.IL has an *MX* of RELAY.CS.NET.

Sample Files

The following section contains sample files for the name server. This covers example boot files for the different types of servers and example domain data base files.

Boot File

Primary Master Server

```
;
; Boot file for Primary Master Name Server
;

; type      domain                  source file or host
;
domain      Berkeley.Edu
primary     Berkeley.Edu            /etc/ucbhosts
cache       .                       /etc/named.ca
primary     32.128.in-addr.arpa     /etc/ucbhosts.rev
primary     0.0.127.in-addr.arpa    /etc/named.local
```

Secondary Master Server

```
;
; Boot file for Primary Master Name Server
;

; type      domain                  source file or host
;
domain      Berkeley.Edu
secondary   Berkeley.Edu            128.32.0.4 128.32.0.10 128.32.136.22
cache       .                       /etc/named.ca
secondary   32.128.in-addr.arpa     128.32.0.4 128.32.0.10 128.32.136.22
primary     0.0.127.in-addr.arpa    /etc/named.local
```

Caching Only Server

```
;
; Boot file for Primary Master Name Server
;

; type      domain                  source file or host
;
domain      Berkeley.Edu
cache       .                       /etc/named.ca
primary     0.0.127.in-addr.arpa    /etc/named.local
```

Remote Server

/etc/resolv.conf

domain Berkeley.Edu nameserver 128.32.0.4 nameserver 128.32.0.10

named.ca

```
;
; Initial cache data for root domain servers.
;

.                       99999999    IN    NS    USC-ISIC.ARPA.
                        99999999    IN    NS    USC-ISIB.ARPA.
                        99999999    IN    NS    BRL-AOS.ARPA.
                        99999999    IN    NS    SRI-NIC.ARPA.
; Prep the cache (hotwire the addresses).
SRI-NIC.ARPA.           99999999    IN    A     10.0.0.51
USC-ISIB.ARPA.          99999999    IN    A     10.3.0.52
USC-ISIC.ARPA.          99999999    IN    A     10.0.0.52
BRL-AOS.ARPA.           99999999    IN    A     128.20.1.2
BRL-AOS.ARPA.           99999999    IN    A     192.5.22.82
```

named.local

```
@    IN    SOA    ucbvax.Berkeley.Edu. kjd.ucbvax.Berkeley.Edu. (
                  1       ; Serial
                  3600    ; Refresh
                  300     ; Retry
                  3600000 ; Expire
                  3600 )  ; Minimum
     IN    NS     ucbvax.Berkeley.Edu.
1    IN    PTR    localhost.
```

Hosts

```
;
;   @(#)ucb-hosts   1.1   (berkeley)   86/02/05
;

@               IN      SOA     ucbvax.Berkeley.Edu. kjd.monet.Berkeley.Edu. (
                                        1.1     ; Serial
                                        3600    ; Refresh
                                        300     ; Retry
                                        3600000  ; Expire
                                        3600 )  ; Minimum
                IN      NS      ucbarpa.Berkeley.Edu.
                IN      NS      ucbvax.Berkeley.Edu.
localhost       IN      A       127.1
ucbarpa         IN      A       128.32.4
                IN      A       10.0.0.78
                ANY     HINFO   VAX-11/780 UNIX
arpa            IN      CNAME   ucbarpa
ernie           IN      A       128.32.6
                ANY     HINFO   VAX-11/780 UNIX
ucbernie        IN      CNAME   ernie
monet           IN      A       128.32.7
                IN      A       128.32.130.6
                ANY     HINFO   VAX-11/750 UNIX
ucbmonet        IN      CNAME   monet
ucbvax          IN      A       10.2.0.78
                IN      A       128.32.10
                ANY     HINFO   VAX-11/750 UNIX
                IN      WKS     128.32.0.10 UDP syslog route timed domain
                IN      WKS     128.32.0.10 TCP ( echo telnet
                                discard sunrpc sftp
                                uucp-path systat daytime
                                netstat qotd nntp
                                link chargen ftp
                                auth time whois mtp
                                pop rje finger smtp
                                supdup hostnames
                                domain
                                nameserver )
vax             IN      CNAME   ucbvax
toybox          IN      A       128.32.131.119
                ANY     HINFO   Pro350 RT11
toybox          IN      MX      0  monet.Berkeley.Edu
miriam          ANY     MB      vineyd.DEC.COM.
postmistress    ANY     MR      Miriam
Bind            ANY     MINFO   Bind-Request kjd.Berkeley.Edu.
                ANY     MG      Ralph.Berkeley.Edu.
                ANY     MG      Zhou.Berkeley.Edu.
                ANY     MG      Painter.Berkeley.Edu.
                ANY     MG      Riggle.Berkeley.Edu.
                ANY     MG      Terry.pa.Xerox.Com.
```

host.Pv

```
;
;   @(#)ucb-hosts.Pv   1.1   (Berkeley)   86/02/05
;

@       IN   SOA   ucbvax.Berkeley.Edu. kjd.monet.Berkeley.Edu. (
                        1.1     ; Serial
                        3600    ; Refresh
                        300     ; Retry
                        3600000 ; Expire
                        3600 )  ; Minimum
        IN   NS    ucbarpa.Berkeley.Edu.
        IN   NS    ucbvax.Berkeley.Edu.
4.0     IN   PTR   ucbarpa.Berkeley.Edu.
6.0     IN   PTR   ernie.Berkeley.Edu.
7.0     IN   PTR   monet.Berkeley.Edu.
10.0    IN   PTR   ucbvax.Berkeley.Edu.
6.130   IN   PTR   monet.Berkeley.Edu.
```

Domain Management

This section contains information for starting, controlling and debugging *named*.

/etc/rc.local

The hostname should be set to the full domain style name in */etc/rc.local* using *hostname (1)*. The following entry should be added to */etc/rc.local* to start up *named* at system boot time:

```
if [ -f /etc/named ]; then
        /etc/named [options] & echo -n ' named' >/dev/console
fi
```

This usually directly follows the lines that start *syslogd*. **Do Not** attempt to run *named* from *inetd*. This will continuously restart the name server and defeat the purpose of having a cache.

/etc/named.pid

When *named* is successfully started up it writes its process id into the file */etc/named.pid*. This is useful to programs that want to send signals to *named*. The name of this file may be changed by defining *PIDFILE* to the new name when compiling *named*.

/etc/hosts

The *gethostbyname ()* library call can detect if *named* is running. If it is determined that *named* is not running it will look in */etc/hosts* to resolve an address. This option was added to allow *ifconfg (8C)* to configure the machines local interfaces and to enable a system manager to access the network while the system is in single user mode. It is advisable to put the local machines interface addresses and a couple of machine names and address in */etc/hosts* so the system manager can rcp files from another machine when the system is in single user mode. The format of */etc/host* has not changed. See *hosts (5)* for more information. Since the process of reading */etc/hosts* is slow, it is not advised to use this option when the system is in multi user mode.

Signals

There are several signals that can be sent to the *named* process to have it do tasks without restarting the process.

Reload

SIGHUP - Causes *named* to read *named.Boot* and reload the database. All previously cached data is lost. This is useful when you have made a change to a data file and you want *named*'s internal database to reflect the change.

Debugging

When *named* is running incorrectly, look first in */usr/adm/messages* and check for any messages logged by *syslog*. Next send it a signal to see what is happening.

SIGINT Dumps the current data base and cache to */usr/tmp/named_dump.db* This should give you an indication to whether the data base was loaded correctly. The name of the dump file may be changed by defining *DUMPFILE* to the new name when compiling *named*. *Note:* the following two signals only work when *named* is built with *DEBUG* defined.

SIGUSR1 Turns on debugging. Each following USR1 increments the debug level. The output goes to */usr/tmp/named.Run* The name of this debug file may be changed by defining *DEBUGFILE* to the new name before compiling *named*.

SIGUSR2 Turns off debugging completely.

For more detailed debugging, define DEBUG when compiling the resolver routines into */lib/libc.a*.

ACKNOWLEDGEMENTS

Many thanks to the users at U.C. Berkeley for falling into many of the holes involved with integrating BIND into the system so that others would be spared the trauma. I would also like to extend gratitude to Jim McGinness and Digital Equipment Corporation for permitting me to spend most of my time on this project.

Ralph Campbell, Doug Kingston, Craig Partridge, Smoot Carl-Mitchell, Mike Muuss and everyone else on the DARPA Internet who has contributed to the development of BIND. To the members of the original BIND project, Douglas Terry, Mark Painter, David Riggle and Songnian Zhou.

Anne Hughes, Jim Bloom and Kirk McKusick and the many others who have reviewed this paper giving considerable advice.

This work was sponsored by the Defense Advanced Research Projects Agency (DoD), Arpa Order No. 4871 monitored by the Naval Electronics Systems Command under contract No. N00039-84-C-0089. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing official policies, either expressed or implied, of the Defense Research Projects Agency, of the US Government, or of Digital Equipment Corporation.

REFERENCES

[Birrell] Birrell, A. D., Levin, R., Needham, R. M., and Schroeder, M.D., In *Comm. A.C.M. 25,* 4:260-274 April 1982.

[RFC819] Su, Z. Postel, J., *Internet Request For Comment 819* Network Information Center, SRI International, Menlo Park, California. August 1982.

[RFC882] Mockapetris, P., *Internet Request For Comment 882* Network Information Center, SRI International, Menlo Park, California. November 1983.

[RFC883] Mockapetris, P., *Internet Request For Comment 883* Network Information Center, SRI International, Menlo Park, California. November 1983.

[RFC973] Mockapetris, P., *Internet Request For Comment 973* Network Information Center, SRI International, Menlo Park, California. February 1986.

[RFC974] Partridge, C., *Internet Request For Comment 974* Network Information Center, SRI International, Menlo Park, California. February 1986.

[Terry] Terry, D. B., Painter, M., Riggle, D. W., and Zhou, S., *The Berkeley Internet Name Domain Server.* Proceedings USENIX Summer Conference, Salt Lake City, Utah. June 1984, pages 23-31.

[Zhou] Zhou, S., *The Design and Implementation of the Berkeley Internet Name Domain (BIND) Servers.* UCB/CSD 84/177. University of California, Berkeley, Computer Science Division. May 1984.

Chapter 16: Sendmail - Mail Router

Introduction

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. (A considerable part of this work was done while under the employ of the INGRES Project at the University of California at Berkeley.)

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, including old (NCP/RFC733) Arpanet, new (TCP/RFC822) Arpanet, 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 data-base 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, these 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 domain technique separates the issue of physical versus logical addressing.

For example,an address of the form

```
eric@a.cc.berkeley.arpa
```

describes only the logical organization of the address space.

is intended to help bridge the gap between the totally *ad hoc* world of networks that know nothing of each other and the clean, tightly-coupled world of unique network numbers. It can accept old arbitrary address syntaxes, resolving ambiguities using heuristics specified by the system administrator, as well as domain-based addressing. It helps guide the conversion of message formats between disparate networks. In short, *sendmail* is designed to assist a graceful transition to consistent internetwork addressing schemes.

Section 1 discusses the design goals for *sendmail*. Section 2 gives an overview of the basic functions of the system. In section 3, details of usage are discussed. Section 4 compares *sendmail* to other internet mail routers, and an evaluation of *sendmail* is given in section 5, including future plans.

DESIGN GOALS

Design goals for *sendmail* include:

- Compatibility with the existing mail programs, including Bell version 6 mail, Bell version 7 mail [UNIX83], Berkeley *Mail* [Shoens79], BerkNet mail [Schmidt79], and hopefully UUCP mail [Nowitz78a, Nowitz78b]. ARPANET mail [Crocker77a, Postel77] was also required.
- Reliability, in the sense of guaranteeing that every message is correctly delivered or at least brought to the attention of a human for correct disposal; no message should ever be completely lost. This goal was considered essential because of the emphasis on mail in our environment. It has turned out to be one of the hardest goals to satisfy, especially in the face of the many anomalous message formats produced by various ARPANET sites. For example, certain sites generate improperly formated addresses, occasionally causing error-message loops. Some hosts use blanks in names, causing problems with UNIX mail programs that assume that an address is one word. The semantics of some fields are interpreted slightly differently by different sites. In summary, the obscure features of the ARPANET mail protocol really *are* used and are difficult to support, but must be supported.
- Existing software to do actual delivery should be used whenever possible. This goal derives as much from political and practical considerations as technical.
- Easy expansion to fairly complex environments, including multiple connections to a single network type (such as with multiple UUCP or Ether nets [Metcalfe76]). This goal requires consideration of the contents of an address as well as its syntax in order to determine which gateway to use. For example, the ARPANET is bringing up the TCP protocol to replace the old NCP protocol. No host at Berkeley runs both TCP and NCP, so it is necessary to look at the ARPANET host name to determine whether to route mail to an NCP gateway or a TCP gateway.
- Configuration should not be compiled into the code. A single compiled program should be able to run as is at any site (barring such basic changes as 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.
- *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.
- 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).
- 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.

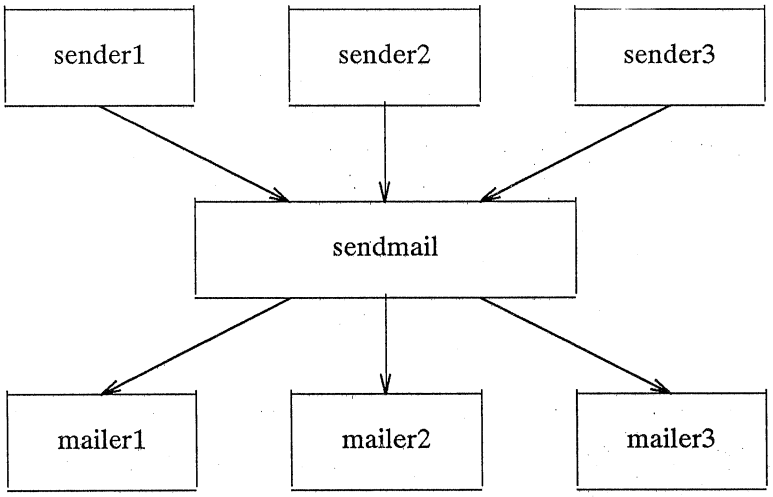


Figure 1 – Sendmail System Structure.

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.

OVERVIEW

System Organization

Sendmail neither interfaces with the user nor does actual mail delivery. Rather, it collects a message generated by a user interface program (UIP) such as Berkeley *Mail*, MS [Crocker77b], or MH [Borden79], edits the message as required by the destination network, and calls appropriate mailers to do mail delivery or queueing for network transmission, except when mailing to a file, when *sendmail* does the delivery directly. This discipline allows the insertion of new mailers at minimum cost. In this sense *sendmail* resembles the Message Processing Module (MPM) of [Postel79b].

Interfaces to the Outside World

There are three ways *sendmail* can communicate with the outside world, both in receiving and in sending mail. These are using the conventional UNIX argument vector/return status, speaking SMTP over a pair of UNIX pipes, and speaking SMTP over an interprocess(or) channel.

Argument vector/exit status

This technique is the standard UNIX method for communicating with the process. A list of recipients is sent in the argument vector, and the message body is sent on the standard input. Anything that the mailer prints is simply collected and sent back to the sender if there were any problems. The exit status from the mailer is collected after the message is sent, and a diagnostic is printed if appropriate.

SMTP over pipes

The SMTP protocol [Postel82] can be used to run an interactive lock-step interface with the mailer. A subprocess is still created, but no recipient addresses are passed to the mailer via the argument list. Instead, they are passed one at a time in commands sent to the processes standard input. Anything appearing on the standard output must be a reply code in a special format.

SMTP over an ICP connection

This technique is similar to the previous technique, except that it uses a 4.2bsd ICP channel [UNIX83]. This method is exceptionally flexible in that the mailer need not reside on the same machine. It is normally used to connect to a sendmail process on another machine.

Operational Description

When a sender wants to send a message, it issues a request to *sendmail* using one of the three methods described above. *Sendmail* operates in two distinct phases. In the first phase, it collects and stores the message. In the second phase, message delivery occurs. If there were errors during processing during the second phase, *sendmail* creates and returns a new message describing the error and/or returns an status code telling what went wrong.

Argument processing and address parsing

If *sendmail* is called using one of the two subprocess techniques, the arguments are first scanned and option specifications are processed. Recipient addresses are then collected, either from the command line or from the SMTP RCPT command, and a list of recipients is created. Aliases are expanded at this step, including mailing lists. As much validation as possible of the addresses is 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 (i.e., 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 file *dead.letter* in the sender's home directory. 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.

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.

Configuration File

Almost all configuration information is read at runtime from an ASCII file, encoding macro definitions (defining the value of macros used internally), header declarations (telling sendmail the format of header lines that it will process specially, i.e., lines that it will add or reformat), mailer definitions (giving information such as the location and characteristics of each mailer), and address rewriting rules (a limited production system to rewrite addresses which is used to parse and rewrite the addresses).

To improve performance when reading the configuration file, a memory image can be provided. This provides a *compiled* form of the configuration file.

USAGE AND IMPLEMENTATION

Arguments

Arguments may be flags and addresses. Flags set various processing options. Following flag arguments, address arguments may be given, unless we are running in SMTP mode. Addresses follow the syntax in RFC822 [Crocker82] for ARPANET address formats. In brief, the format is:

- Anything in parentheses is thrown away (as a comment).
- Anything in angle brackets < > is preferred over anything else. This rule implements the ARPANET standard that addresses of the form **user name <machine-address>** will send to the electronic *machine-address* rather than the human *user name.*
- Double quotes (") quote phrases; backslashes quote characters. Backslashes are more powerful in that they will cause otherwise equivalent phrases to compare differently – for example, *user* and "*user*" are equivalent, but *user* is different from either of them.

  Parentheses, angle brackets, and double quotes must be properly balanced and nested. The rewriting rules control remaining parsing.

  Disclaimer: Some special processing is done after rewriting local names; see below.

Mail to Files and Programs

Files and programs are legitimate message recipients. Files provide archival storage of messages, useful for project administration and history. Programs are useful as recipients in a variety of situations, for example, to maintain a public repository of systems messages (such as the Berkeley *msgs* program, or the MARS system [Sattley78]).

Any address passing through the initial parsing algorithm as a local address (i.e, not appearing to be a valid address for another mailer) is scanned for two special cases. If prefixed by a vertical bar |) the rest of the address is processed as a shell command. If the user name begins with a slash mark /) the name is used as a file name, instead of a login name.

Files that have setuid or setgid bits set but no execute bits set have those bits honored if *sendmail* is running as root.

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.

Message Delivery

The send queue is ordered by receiving host before transmission to implement message batching. Each address is marked as it is sent so rescanning the list is safe. An argument list is built as the scan proceeds. Mail to files is detected during the scan of the send list. The interface to the mailer is performed using one of the techniques described in section 2.2.

After a connection is established, *sendmail* makes the per-mailer changes to the header and sends the result to the mailer. If any mail is rejected by the mailer, a flag is set to invoke the return-to-sender function after all delivery completes.

Queued Messages

If the mailer returns a *temporary failure* exit status, the message is queued. A control file is used to describe the recipients to be sent to and various other parameters. This control file is formatted as a series of lines, each describing a sender, a recipient, the time of submission, or some other salient parameter of the message. The header of the message is stored in the control file, so that the associated data file in the queue is just the temporary file that was originally collected.

Configuration

Configuration is controlled primarily by a configuration file read at startup. *Sendmail* should not need to be recomplied except

- To change operating systems (V6, V7/32V, 4BSD).
- To remove or insert the DBM (UNIX database) library.
- To change ARPANET reply codes.
- To add headers fields requiring special processing.

Adding mailers or changing parsing (i.e., rewriting) or routing information does not require recompilation.

If the mail is being sent by a local user, and the file "*.mailcf*" exists in the sender's home directory, that file is read as a configuration file after the system configuration file. The primary use of this feature is to add header lines.

The configuration file encodes macro definitions, header definitions, mailer definitions, rewriting rules, and options.

Macros

Macros can be used in three ways. Certain macros transmit unstructured textual information into the mail system, such as the name *sendmail* will use to identify itself in error messages. Other macros transmit information from *sendmail* to the configuration file for use in creating other fields (such as argument vectors to mailers); e.g., the name of the sender, and the host and user of the recipient. Other macros are unused internally, and can be used as shorthand in the configuration file.

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 (i.e., 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.

COMPARISON WITH OTHER MAILERS

Delivermail

Sendmail is an outgrowth of *delivermail*. The primary differences are:

Configuration information is not compiled in. This change simplifies many of the problems of moving to other machines. It also allows easy debugging of new mailers.

Address parsing is more flexible. For example, *delivermail* only supported one gateway to any network, whereas *sendmail* can be sensitive to host names and reroute to different gateways.

Forwarding and :include: features eliminate the requirement that the system alias file be writable by any user (or that an update program be written, or that the system administration make all changes).

Sendmail supports message batching across networks when a message is being sent to multiple recipients.

A mail queue is provided in *sendmail*. Mail that cannot be delivered immediately but can potentially be delivered later is stored in this queue for a later retry. The queue also provides a buffer against system crashes; after the message has been collected it may be reliably redelivered even if the system crashes during the initial delivery.

Sendmail uses the networking support provided by 4.2BSD to provide a direct interface networks such as the ARPANET and/or Ethernet using SMTP (the Simple Mail Transfer Protocol) over a TCP/IP connection.

MMDF

MMDF [Crocker79] spans a wider problem set than *sendmail*. For example, the domain of MMDF includes a *phone network* mailer, whereas *sendmail* calls on preexisting mailers in most cases.

MMDF and *sendmail* both support aliasing, customized mailers, message batching, automatic forwarding to gateways, queueing, and retransmission. MMDF supports two-stage timeout, which *sendmail* does not support.

The configuration for MMDF is compiled into the code. (Dynamic configuration tables are currently being considered for MMDF; allowing the installer to select either compiled or dynamic tables.)

Since MMDF does not consider backwards compatibility as a design goal, the address parsing is simpler but much less flexible.

It is somewhat harder to integrate a new channel (The MMDF equivalent of a *sendmail mailer.)* into MMDF. In particular, MMDF must know the location and format of host tables for all channels, and the channel must speak a special protocol. This allows MMDF to do additional verification (such as verifying host names) at submission time.

MMDF strictly separates the submission and delivery phases. Although *sendmail* has the concept of each of these stages, they are integrated into one program, whereas in MMDF they are split into two programs.

Message Processing Module

The Message Processing Module (MPM) discussed by Postel [Postel79b] matches *sendmail* closely in terms of its basic architecture. However, like MMDF, the MPM includes the network interface software as part of its domain.

MPM also postulates a duplex channel to the receiver, as does MMDF, thus allowing simpler handling of errors by the mailer than is possible in *sendmail*. When a message queued by *sendmail* is sent, any errors must be returned to the sender by the mailer itself. Both MPM and MMDF mailers can return an immediate error response, and a single error processor can create an appropriate response.

MPM prefers passing the message as a structured object, with type-length-value tuples. (This is similar to the NBS standard.) Such a convention requires a much higher degree of cooperation between mailers than is required by *sendmail*. MPM also assumes a universally agreed upon internet name space (with each address in the form of a net-host-user tuple), which *sendmail* does not.

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.

Sendmail has knowledge of a few difficult environments built in. It generates ARPANET FTP/SMTP compatible error messages (prepended with three-digit numbers [Neigus73, Postel74, Postel82]) as necessary, optionally generates UNIX-style *From* lines on the front of messages for some mailers, and knows how to parse the same lines on input. Also, error handling has an option customized for BerkNet.

The decision to avoid doing any type of delivery where possible (even, or perhaps especially, local delivery) has turned out to be a good idea. Even with local delivery, there are issues of the location of the mailbox, the format of the mailbox, the locking protocol used, etc., that are best decided by other programs. One surprisingly major annoyance in many internet mailers is that the location and format of local mail is built in. The feeling seems to be that local mail is so common that it should be efficient. This feeling is not born out by our experience; on the contrary, the location and format of mailboxes seems to vary widely from system to system.

The ability to automatically generate a response to incoming mail (by forwarding mail to a program) seems useful *I am on vacation until late August....*) but can create problems such as forwarding loops (two people on vacation whose programs send notes back and forth, for instance) if these programs are not well written. A program could be written to do standard tasks correctly, but this would solve the general case.

It might be desirable to implement some form of load limiting. I am unaware of any mail system that addresses this problem, nor am I aware of any reasonable solution at this time.

The configuration file is currently practically inscrutable; considerable convenience could be realized with a higher-level format.

It seems clear that common protocols will be changing soon to accommodate changing requirements and environments. These changes will include modifications to the message header (e.g., [NBS80]) or to the body of the message itself (such as for multimedia messages [Postel80]). Experience indicates that these changes should be relatively trivial to integrate into the existing system.

In tightly coupled environments, it would be nice to have a name server such as Grapvine [Birrell82] integrated into the mail system. This would allow a site such as *Berkeley* to appear as a single host, rather than as a collection of hosts, and would allow people to move transparently among machines without having to change their addresses. Such a facility would require an automatically updated database and some method of resolving conflicts. Ideally this would be effective even without all hosts being under a single management. However, it is not clear whether this feature

should be integrated into the aliasing facility or should be considered a *value added* feature outside *sendmail* itself.

As a more interesting case, the CSNET name server [Solomon81] provides an facility that goes beyond a single tightly-coupled environment. Such a facility would normally exist outside of *sendmail* however.

ACKNOWLEDGEMENTS

Thanks are due to Kurt Shoens for his continual cheerful assistance and good advice, Bill Joy for pointing me in the correct direction (over and over), and Mark Horton for more advice, prodding, and many of the good ideas. Kurt and Eric Schmidt are to be credited for using *delivermail* as a server for their programs *Mail* and BerkNet respectively) before any sane person should have, and making the necessary modifications promptly and happily. Eric gave me considerable advice about the perils of network software which saved me an unknown amount of work and grief. Mark did the original implementation of the DBM version of aliasing, installed the VFORK code, wrote the current version of *rmail*, and was the person who really convinced me to put the work into *delivermail* to turn it into *sendmail*. Kurt deserves accolades for using *sendmail* when I was myself afraid to take the risk; how a person can continue to be so enthusiastic in the face of so much bitter reality is beyond me.

Kurt, Mark, Kirk McKusick, Marvin Solomon, and many others have reviewed this paper, giving considerable useful advice.

Special thanks are reserved for Mike Stonebraker at Berkeley and Bob Epstein at Britton-Lee, who both knowingly allowed me to put so much work into this project when there were so many other things I really should have been working on.

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. IGST/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, Seventh Edition, Volume 2. August, 1978.

[Nowitz78b] Nowitz, D. A., *Uucp Implementation Description.* Bell Laboratories. In UNIX Programmer's Manual, Seventh Edition, Volume 2. October, 1978.

[Postel74] Postel, J., and Neigus, N., Revised FTP Reply Codes. RFC 640, NIC 30843. In [Feinler78]. June, 1974.

[Postel77] Postel, J., *Mail Protocol.* NIC 29588. In [Feinler78]. November 1977.

[Postel79a] Postel, J., *Internet Message Protocol.* RFC 753, IEN 85. Network Information Center, SRI International, Menlo Park, California. March 1979.

[Postel79b] Postel, J. B., *An Internetwork Message Structure.* In *Proceedings of the Sixth Data Communications Symposium,* IEEE. New York. November 1979.

[Postel80] Postel, J. B., *A Structured Format for Transmission of Multi-Media Documents.* RFC 767. Network Information Center, SRI International, Menlo Park, California. August 1980.

[Postel82] Postel, J. B., *Simple Mail Transfer Protocol.* RFC821 (obsoleting RFC788). Network Information Center, SRI International, Menlo Park, California. August 1982.

[Schmidt79] Schmidt, E., *An Introduction to the Berkeley Network.* University of California, Berkeley California. 1979.

[Shoens79] Shoens, K., *Mail Reference Manual.* University of California, Berkeley. In UNIX Programmer's Manual, Seventh Edition, Volume 2C. December 1979.

[Sluizer81] Sluizer, S., and Postel, J. B., *Mail Transfer Protocol.* RFC 780. Network Information Center, SRI International, Menlo Park, California. May 1981.

[Solomon81] Solomon, M., Landweber, L., and Neuhengen, D., *The Design of the CSNET Name Server.* CS-DN-2, University of Wisconsin, Madison. November 1981.

[Su82] Su, Zaw-Sing, and Postel, Jon, *The Domain Naming Convention for Internet User Applications.* RFC819. Network Information Center, SRI International, Menlo Park, California. August 1982.

[UNIX83] *The UNIX Programmer's Manual, Seventh Edition,* Virtual VAX-11 Version, Volume 1. Bell Laboratories, modified by the University of California, Berkeley, California. March, 1983.