RISC *iX*
SYSTEM ADMINISTRATOR'S GUIDE

Acorn

Designed and written by Acorn Computers Technical Publications Department

The products described in this Guide are subject to continuous development and improvement. All information of a technical nature and particulars of the product and their use (including the information and particulars in this Guide) are given by Acorn Computers Limited in good faith. However, Acorn Computers Limited cannot accept any liability for any loss or damage arising from the use of any information or particulars in this Guide.

If you have any comments on this Guide, please complete and return the form at the back of this Guide to the address given there. All other correspondence should be addressed to:

Customer Services
Acorn Computers Limited
Fulbourn Road
Cambridge CB1 4JN

Support and information can also be obtained from the Acorn Support Information Database (SID). This is a Viewdata system available to registered SID users. Initially, access SID by dialling directly into the Guest User access (parity 7E1, speed V21/22/23/22bis, telephone number (0223) 243642: this will allow you to inspect the system and use a response frame for registration. Alternatively, access SID via Prestel (type *SID#): this will automatically register you on your first call.

Cover illustration: desert scene scanned using Irlam Instruments' i-mage system.

Note: Various third party monitors and other peripheral equipment are depicted or described in this Guide. The illustration or description of these or any other third party equipment does not imply any recommendation or endorsement or approval by Acorn Computers Ltd., and users must satisfy themselves as to the suitability of any peripheral equipment (including monitors) on advice from the equipment supplier.

Edition 2
Published October 1990
ISBN 1 85250 082 4
Published by Acorn Computers Limited
Part Number 0470,702 Issue 1

Contents

About this Guide

Readership of this Guide

This Guide assumes that you have already successfully set up your RISC iX workstation as detailed in the *Installation Guide* supplied with your system.

This Guide moves on from this and tells you how to configure the system software on your machine, including setting your machine up on a network and creating user accounts. If you are familiar with other UNIX† systems, you should have no problem following these instructions. If you have never used a UNIX system before, you should refer to the first part of the *RISC iX User Guide* which explains some of the fundamental aspects of the RISC iX operating system and introduces some basic commands.

RISC iX System Administration is a very large topic to broach in one Guide and there are many aspects that can only be fully appreciated by a thorough knowledge of the RISC iX environment. So do not be put off by sometimes quite detailed information about the system appearing in this Guide. Many example programs are available for you to use that will simplify the job of looking after your system.

Summary of contents

Here is a summary of what you will find in this Guide:

Parts

The Chapters in this Guide are divided into three parts:

- Part 1 – *Basic Procedures and Concepts*, contains initial installation information together with basic information relevant to all types of RISC iX machines. You should read this part thoroughly before attempting any System Administration task.
- Part 2 – *Network Administration*, describes firstly how to set your workstation up on a network and then moves on to describe the additional tasks you will have to perform and concepts that you will have to become familiar with when administering RISC iX workstations on a network.

---

†UNIX is a trademark of AT&T.

- Part 3 – *Advanced Administration*, contains information about more advanced System Administration tasks, that you may be required to perform.

Appendices and Index

The Appendices contain supplementary information that is referred to in the main part of the Guide and the Index at the back of this Guide should help you locate information quickly.

Conventions used in this Guide

The fixed width font, `Courier`, is used for the text of example programs and commands. This font gives the closest approximation to what appears on your screen, enabling you to spot where spaces should appear.

Different Courier font families are also used to describe more fully what you have to type:

Convention	Meaning
`<Delete>`	Press the key indicated.
`<Ctrl-D>`	Hold down the first key and press the second.
`login:`	Text displayed on the screen.
`cat`	Text that you type in.
`filename`	A variable, where you should substitute what the word represents.

For example:

`login:` **`guest`**

Another example:

`cat` *`filename`*

where *filename* is the name of a file; for example, *readme1*:

`cat readme1`

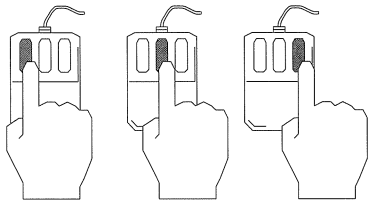
In the main body of the text, *italics* is used for emphasis.

Symbols used

The symbol below, when attached to a section title, indicates that the section contains additional or alternative information, that would otherwise interrupt the flow of a procedure:

This symbol is used extensively throughout the first part of this Guide.

The symbols below are used to indicate the appropriate mouse button to press when describing a procedure that uses the mouse:



Finding out more

If you wish to develop your System Administration skills further, you should dip into some of the books listed in the Bibliography at the back of this Guide.

For information on how to set up your workstation, including connecting the monitor, keyboard and mouse and maintaining the hardware components of your workstation, refer to the *Installation Guide* supplied with your workstation.

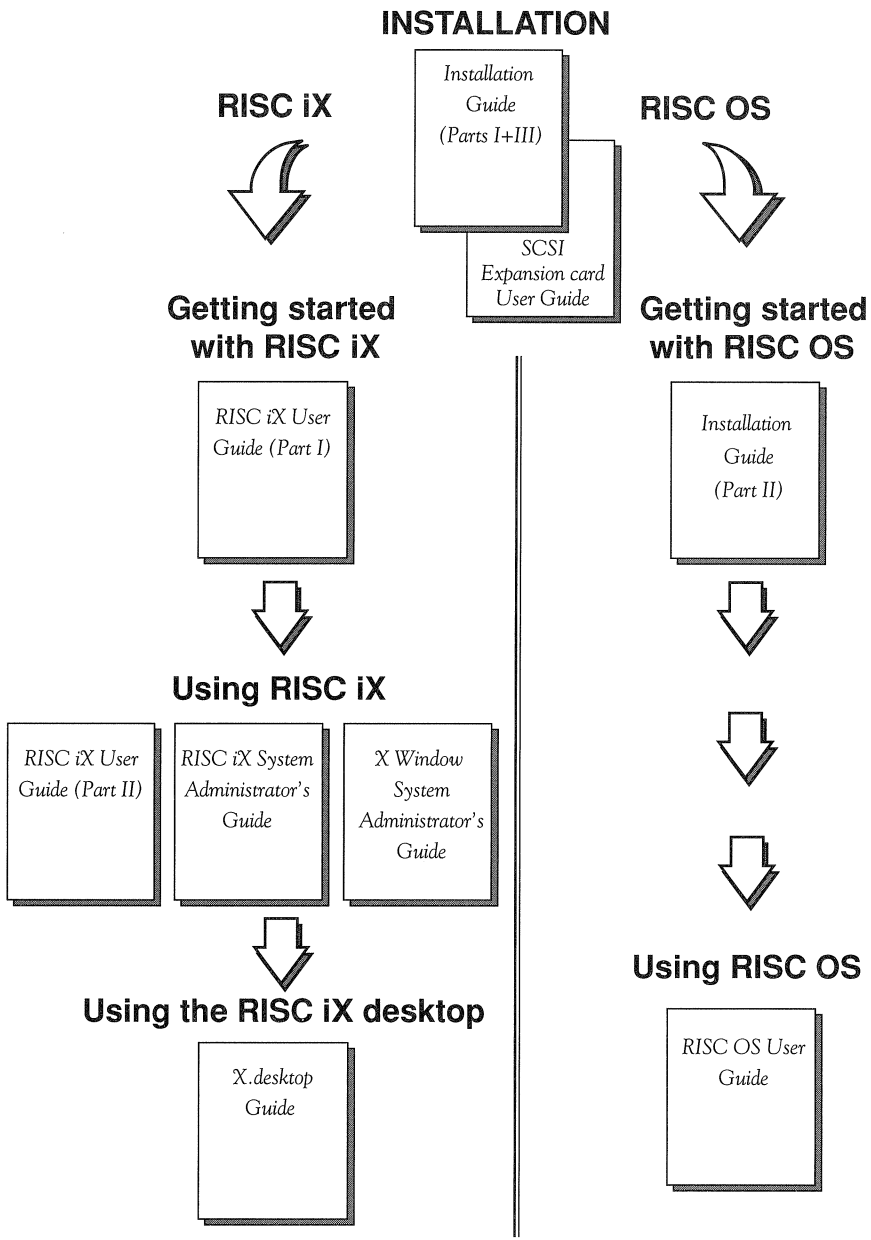
Details about using the utilities and commands of the RISC iX operating system are contained in the *RISC iX User Guide*, also supplied with your workstation.

If you wish to write programs in C, Fortran or Pascal then you will find information about all these languages contained in the *RISC iX Programmer's Reference Manual, Volume 1 – Kernel and Languages*. Details about network programming is contained in the *RISC iX Programmer's Reference Manual, Volume 2 – Network File System*. Both of these volumes are available from your Acorn supplier.

If your workstation is supplied with the X Window System, then refer to the *X Window System Administrator's Guide* for details about setting it up. For details on using the desktop manager supplied with the X Window System, refer to the *X.desktop Guide*.

For information about the RISC OS operating system, refer to the chapter entitled *Introduction to RISC OS* in the *Installation Guide*.

Documentation roadmap



Introduction to System Administration

What is System Administration?

UNIX is a more complex system than single-user operating systems such as MS-DOS. This is because UNIX was developed originally on machines intended for simultaneous multi-user access, with a team of operators assigned to attend to tapes and printers located well away from each user's terminal.

One of the benefits is that the system is *multi-tasking*, and many activities are simultaneously maintained by the kernel. Thus typical UNIX systems can be driving a printer, communicating with another system down a serial line and running a variety of tasks on a number of virtual terminals all at the same time.

However, there are some drawbacks to this type of operating environment. For example:

- You cannot just *pull the plug* to shut the machine down – it must be halted properly leaving everything in a tidy state.
- You cannot totally forget about what goes on behind the scenes and what *housekeeping* needs to be done from time to time to keep the system running smoothly.
- You will also need to know about avoiding data loss and safeguarding your data by performing backups.
- In future, you may also want to configure and adapt some of the aspects of the system to meet your own particular requirements, bypassing parts of the system you never use and avoiding questions to which you always give the same answer. In many cases you will find that the system is quite configurable, but you will need to find out a lot about the system before you start customising it.

System Administration covers all these topics and more. The following chapters in this manual will attempt to guide you through these activities by firstly giving you enough background information to enable you to appreciate the task at hand, then taking you methodically through each activity.

Guidelines to follow

When performing any task on your system, always try to observe the following guidelines:

- Never do anything with more privilege than is required.
- Try to be conscious of security on the system.
- Read through the whole procedure before starting any part of a task.
- Do not do anything without having a pretty good idea why you are doing it and what the consequences of your action will be.
- If you are trying something for the first time, write down everything you type and what the response of the system is. If anything unexpected occurs, you will have a valuable record of what you did.

If you follow the above rules, you should never run into any difficulties.

Starting up and shutting down the system

Introduction

This chapter contains information on how to start up and shut down your system. The system is initially configured to start up automatically as a standalone workstation, not on a network. If you have a discless workstation or you wish to connect your workstation onto a network, refer to the second part of this Guide beginning with the chapter entitled *Networking System Administration* on page 127.

You should already be familiar with the procedures for turning your workstation on as detailed in the *Installation Guide*. This chapter moves on from this and explains what happens when the system starts up and how to shut it down safely. The end of the chapter describes how you can configure the system to start up and shut down in different ways.

Starting up the system

If your workstation was installed by somebody else, it may already be switched on and working. If this is so read this section for information only.

To switch on your workstation:

1 Make sure that the mains power lead is plugged into the mains supply and that the mains power is switched on.

2 Turn on the workstation by pressing the power switch located on the rear left-hand side of the workstation to ON (I).

3 Switch the power switch on the monitor to ON (I). See the manual supplied with your monitor.

Start-up procedure

As the workstation starts up, you will notice:

- the POWER lamp on the front panel lights up and the disc light (if fitted) flashes.
- messages are displayed on the screen.

If need be, you can adjust the contrast and brightness, using the controls on the monitor. See your monitor manual for details.

Note: At this point your workstation is about to start up the RISC iX operating system. If your supplier has installed a third party application on the workstation, your start-up procedure may be different. For more information see the additional documentation packaged by your supplier.

Autoboot

When first shipped, RISC iX boots automatically (autoboots). However, your supplier may have configured your system so that the system is booted from the RISC OS desktop. To find out more about starting up via the RISC OS desktop, refer to the chapter entitled *Using the RISCiXFS module* on page 283.

When the autoboot procedure starts, the following messages are displayed on the screen. (The screen messages may vary slightly from those shown in this Guide.)

```
RISC OS 4096K
Acorn ADFS
Supervisor
```

After about 10 seconds the screen clears and the RISC iX start-up screen is displayed:

```
RISC iX Release 1.2
real mem = 8388608
avail mem = 7307264
48 buffers (384 Kbytes)
sd0: internal controller
```

When the system has finished starting up, you will see the *login:* prompt, as shown below. Alternatively, you may see a prompt welcoming you to the X Window System and inviting you to log in, if your system has been has been configured to start up the X Window System. (For more information about configuring your system to start up in the X Window System, refer to the *X Window System Administrator's Guide.*)

```
RISC iX Release 1.2
unix login:
```

Checking the date and time

If, when you switch on your workstation, it has not been used for several days (eg when first delivered) you may get the warning message:

```
Check and reset date
```

This is a reminder to you to make sure that the date and time are correct. For details about correcting either the time or date, refer to the chapter entitled *First things to do after booting* on page 15.

Problems starting up?

If the workstation does not appear to work, go through this checklist to see if you can identify the problem:

- Is the workstation plugged into the mains power socket?
- Is the mains power socket turned on?
- Is the monitor connected to the system unit properly?
- Is the monitor power cable plugged in?
- Is the workstation turned on? (Look for the power light).
- Is the monitor turned on or has the screen been temporarily blanked by Screen blank? (Press any key to find out.)
- Is the monitor's brightness control correctly adjusted?

If you can't solve the problem, contact your supplier.

What happens when the system starts?

As normally supplied, RISC iX comes up automatically when switched on and requires little intervention, unless a fatal error occurs. The main tasks that are performed during the start up procedure are as follows:

- the boot program is executed
- various system processes are started
- consistency checks are performed on the filesystem
- the system enters multi-user mode

The following sections will describe the default start up procedure for your system and the part played by each of the above tasks in this procedure.

Note that the start up procedure described is the procedure that the system runs through as originally supplied. However, remember that your supplier may have altered your system so that it enters RISC iX via RISC OS, or RISC iX is entered and then a window environment is started automatically or extra peripheral devices have

been added and the start up procedure has been suitably changed. For more information on how to alter the way your workstation starts up refer to the chapter entitled *Using the RISCiXFS module* on page 283.

The boot procedure

The term 'boot' is derived from the expression '*to pull yourself up by your own bootstraps*' and is a quite accurate description of what happens immediately following power on.

Firstly, start up information which is resident in CMOS RAM is read. This contains various parameter settings that tell the system where to find the various files and devices that are to be used for starting up RISC iX. These settings are not lost when the mains power is switched off, since CMOS RAM is supported by batteries in the system unit.

A vital part of this information is the location of the boot device that contains the *boot program* which is used to load the RISC iX kernel.

By default, the kernel is a file called */vmunix* (short for *virtual memory unix*) and is located in the root filesystem on the boot device.

If your workstation is discless, then the kernel will be downloaded across the network from the workstation that is acting as the server for the workstation. For more information about how this operation works, refer to the section entitled *Setting up discless workstations* on page 150.

Once the kernel is loaded by the boot program, the kernel image is then executed and by default defines the root filesystem to be partition 0 on the boot device and the swap area as partition 1 on the boot device.

For information about how to change the CMOS RAM parameter settings that the boot program uses, refer to the chapter entitled *Using the RISCiXFS module* on page 283.

System processes

Once */vmunix* is running and the system has booted successfully, a series of *system processes* are started. The first and probably most important system process that is started is *init* (short for *initial process*). *init* spawns a shell and executes the commands contained in the shell script */etc/rc*, which contains all the other commands and system processes that need to be executed to boot the workstation.

Checking the filesystem

One of the commands contained in */etc/rc* that is executed by *init* is *fsck* (short for *file system consistency check*). This program checks out the disc and ensures that the structures are fully consistent, before the kernel starts to access them.

If something goes wrong, or errors are found in the disc structures, the system will not enter multi-user mode, but abort with a suitable error message, and revert to *single-user mode* (see below).

Multi-user mode

After *fsck* has completed successfully, the main system processes (*daemons*) are started up by *init* as indicated by messages appearing on the screen. For example, *daemons* are started for logging in and printing. Eventually the normal login prompt will be displayed on the console terminal screen, indicating that RISC iX has been successfully booted as a stand-alone system and is ready to be used. At this point, the system is said to be in *multi-user mode*.

In the early stages of starting up, when only the console terminal is active, the system is said to be in single-user mode. When RISC iX is fully operational, a number of system tasks are running, a *login:* prompt is output with log ins ready to be accepted on each terminal, and various disc partitions are activated. This is said to be multi-user mode.

In multi-user mode, the main role of *init* is to create a terminal port where a user may log in and execute commands. To facilitate this, *init* reads the file */etc/ttys* during start up and executes a given command for each terminal specified in the file. The command executed is usually */etc/getty* which opens up and initialises a specified terminal line creating a *login:* prompt for the user. On your system, there are three ports that can have a *getty* invoked on them.

You can arrange for the system to stop in single-user mode with a shell prompt after the initial boot using the RISC iX filing system module. For more information, refer to the chapter entitled *Using the RISCiXFS module* on page 283.

In single-user mode, not all of the disc partitions are available, and some of the programs, even if they are available may complain that they cannot open files which they normally expect to find. Hence you should avoid doing normal work in this mode, and limit activities to filesystem checking and repair as well as some simple System Administration tasks.

To come out of single-user mode and enter multi-user mode, terminate the single-user shell by pressing <Ctrl-D> or issue the *reboot*(8) command. The system then runs through a procedure given in the shell script */etc/rc* before outputting a *login:* prompt on all the virtual terminals.

There is a further invocation of *reboot*, called *fastboot*, that you can use to bring the system up quickly without checking the filesystem. This is normally used to either:

- test any minor changes you have made to the filesystem, or

- boot the workstation quickly after it has been correctly shut down according to the details given in the next section.

For more information, refer to *reboot*(8) and *fastboot*(8).

Some parts of the start up procedure are put into separate shell scripts and then invoked from */etc/rc*. For example, start up operations specific to the workstation are put into */etc/rc.local*, those relating to network communications are put into */etc/rc.net* and those relating to the network information service (NIS) are put into */etc/rc.yp*.

The file *rc.local* is read after the filesystem has been successfully checked. Therefore, you can place start up commands in here without fear of disrupting the boot procedure. For more information, refer to *rc*(8).

Your system is initially configured to start up as a standalone workstation as defined in the additional *rc* file */etc/rc.config*. For more information about this file and how to change it for different network environments, refer to the chapter entitled *Setting up a local Ethernet network* on page 129.

Shutting down the system

The workstation is capable of running continuously and does not need to be turned off unless you are not going use it for a lengthy period (several days).

If you do need to turn off your workstation you cannot safely just '*pull the plug*'. This is because the kernel keeps parts of the disc structures and the files on the disc in memory, so as to save continually reading and writing commonly-used parts of the disc.

Similarly, processes which are running may create temporary files, or leave other files in a half-completed state and it may not be straightforward to clean up the mess.

There are three commands you can use to shut down the system:

- *halt*
- *fasthalt*
- *shutdown*

These commands are discussed in the following sections.

halt

If the workstation is configured stand-alone you should use the following procedure to turn off the workstation:

1 Finish working and save any files you were working on.

2 Log out of RISC iX.

3 Log in using the username *halt* – by default, no password is required. Instead of getting a log in shell, the program *halt* is invoked (as specified in the password file */etc/passwd*) which shuts down the system. This involves firstly writing out all cached information resident in memory onto disc, and then stopping execution of RISC iX. This process is complete when the following messages are displayed:

```
Syncing disks... done
halted
```

4 Switch the system off by turning the power switch on the back of the workstation unit to OFF(0), and the power switch on the monitor to OFF.

Always wait until the above message appears, before switching off the power. This ensures that the system has completely shut down and that the discs are left in a consistent state. Refer to the beginning of this chapter to find out how to turn the workstation on again.

Note that the CMOS RAM settings in your workstation may have been altered so that the system returns to the RISC OS desktop or Command Line. (If you are returned to the desktop, type <F12> to access the Command Line).

At the RISC OS Command Line, type:

***Bye**

This command 'parks' any hard discs that may be attached to your workstation. You can then safely switch off the workstation.

fasthalt

The command *fasthalt* can also be used to close down the system in a similar fashion to *halt*. To use *fasthalt*, just log in as *root* and type the command:

fasthalt

and the system will be shut down.

The only difference between *halt* and *fasthalt* is that *fasthalt* creates a temporary file called */fastboot* to ensure that the next time the system is rebooted no disc checks are performed. For more information, refer to *halt*(8), *fasthalt*(8) and *sync*(8).

If you feel that it is slightly dangerous to allow people to log in with the username *halt* and shut the workstation down in this way, you can protect this userid with a password. For more information, refer to the section entitled *Setting the password for various users of the system* on page 17.

shutdown

To bring the system down into single-user mode, use the *shutdown* command. For example, at the normal shell prompt for *root* (#), type:

```
shutdown +5 Closing down for the week-end
```

A message is displayed on the screen of each user, warning users that the system is about to be brought down in five minutes.

After the time specified has elapsed, the system will be brought down into single-user mode. Only the console terminal window will now be active.

Note, that an important security issue is raised here. When the system is shut down in this way, the shell on the console window will have *root* privileges. Therefore, it is important that you never leave your machine in this unguarded state for any longer than you have to. Always leave the machine in multi-user mode.

There are other options that can be used with *shutdown* to bring down the workstation then bring it up by invoking one of the boot commands. For more information, refer to *shutdown*(8).

Configuring the start-up procedure

This section takes you through the alternative start-up procedures for your workstation, which may already be operating if your workstation was configured by your supplier.

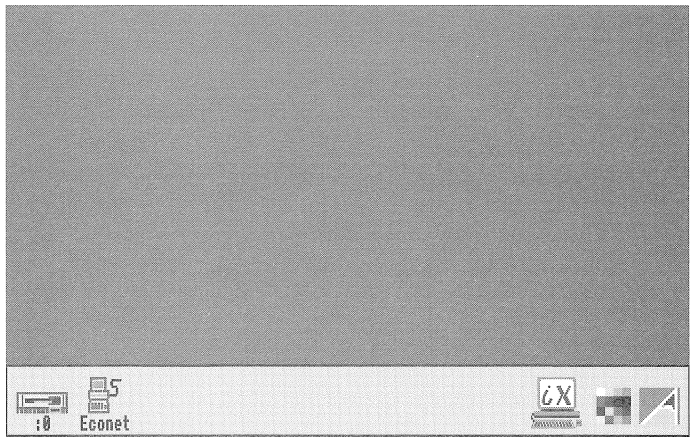
The default RISC iX start-up procedure

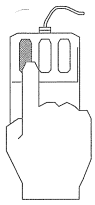
The default start-up procedure has already been described at the beginning of this chapter.

Starting RISC iX from the RISC OS desktop

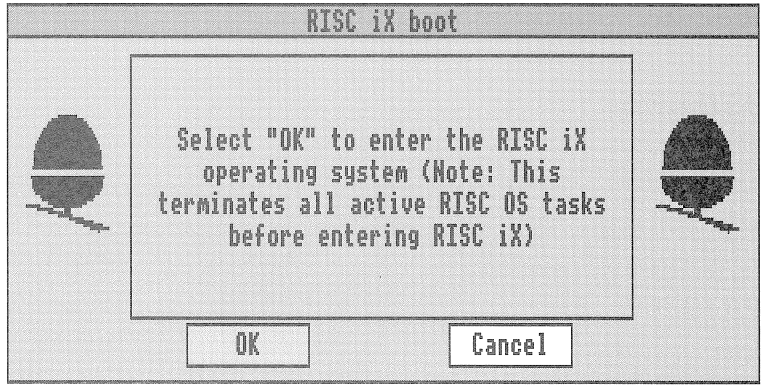
If your workstation has been configured so that the RISC OS desktop is first displayed on start-up, follow this procedure to start RISC iX:

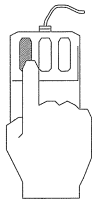
1 Switch on your workstation and monitor. The start-up begins and the monitor displays the RISC OS desktop with the RISC iX '*iX*' icon on the icon bar:





2 Use the mouse to click on the RISC iX icon with the left mouse button. The following dialogue box is displayed:





3 Use the mouse to click *OK* with the left mouse button. This starts RISC iX automatically.

4 Another dialogue box is displayed for a few seconds telling you that you can terminate the RISC iX bootstrap by pressing <Ctrl>-<RESET>. (The <RESET> key is at the rear of the keyboard (next to the mouse connector)). If you press these keys, the bootstrap will stop and you will be returned back to normal RISC OS display.

5 RISC iX starts and you see these messages displayed on your screen:

```
RISC iX Release 1.2
real mem = 8388608
avail mem = 7307264
48 buffers (384 Kbytes)
```

6 When the system has finished booting (this takes several minutes), the login prompt will be displayed on the screen, RISC iX is now operating correctly, and you can log in.

Starting RISC iX from the RISC OS command line

If you want to start RISC iX from the RISC OS command line, type:

***boot**

This will start RISC iX automatically.

Error messages from RISCiXFS

For more information about any error messages that you may receive when using the RISCiXFS module refer to the section entitled *RISCiXFS module error messages* on page 302.

Configuring the close down procedure

This section describes the alternative ways you can shutdown your system, using the following commands:

- *reboot* – this closes down the workstation and then automatically starts it up again
- *halt –RISCOS* – this closes down the workstation and explicitly exits you back to the RISC OS command line or desktop, irrespective of the setting of the 'noRISCOS' option.
- *shutdown* – this closes down the workstation to single-user mode
- *shutdown –h <time>* – this closes down the workstation in the same way as *halt*, but it also allows you to specify the time of the close down in advance and, optionally, to send other users a warning message prior to shutdown.

For more information about close down procedures, read the *halt*(8) and *shutdown*(8) manual pages.

Selecting RISC OS from RISC iX

If your workstation is configured to start up in RISC OS and uses the RISC iX filing system module to invoke RISC iX, then the halt command automatically returns you to RISC OS.

If your workstation is running RISC iX and is configured so that it starts up into RISC iX automatically, you can bring the workstation down to RISC OS using the command:

```
halt -RISCOS
```

This command halts RISC iX and brings the workstation down to RISC OS, irrespective of the configuration of the boot parameter settings in CMOS RAM.

From RISC OS you then have the option of bringing the system up in multi-user or single-user mode as well as being able to perform some simple system administration tasks.

For more information, refer to the chapter entitled *Using the RISCiXFS module* on page 283.

First things to do after booting

Introduction

This chapter contains descriptions of all the tasks that may need to be performed immediately following installation and subsequent booting of your machine. After performing these tasks your system will be fully operational. The tasks include:

- Setting the date and time
- Setting a password for various users of the system
- Setting the name of the workstation
- Editing the message of the day
- Performing a level 0 *dump* backup

Setting the date and time

If you have just installed your system, you may find that the date and time, displayed when you type in the command *date*, are incorrect.

It is particularly important under RISC iX to have the date and time set correctly. This is because:

- Files listed using *ls -lt* will show incorrect modification times.
- The command *make*(1) uses the file modification dates to determine which modules need to be rebuilt.
- If you are running external mail systems which use a modem attached to a telephone, calls may be placed at the wrong times.
- Backup procedures will not operate properly.

On your RISC iX workstation the date is held in battery-backed CMOS RAM, so once set it shouldn't be necessary to correct the date very often, except when you have to change the batteries in the system unit. (For information on changing the batteries, refer to the *Installation Guide* supplied with your machine.)

Check the date and time by typing:

```
date
```

If the date and time displayed by this command are correct you can skip the rest of this section. If the date and time are incorrect, follow the procedure below for setting the date and time:

1 If you are logged in as someone other than *root*, log out by typing *exit*.

2 Log in as user *root*.

If you have just installed your system, *root* will not have a password.

3 Type *date* followed by a space and then a 10 (or twelve) digit figure representing the date and time, in the following format:

date yymmddhhmm [.ss]

- yy represents the last two digits of the year
- *mm* is the number of the month
- *dd* is the number of the day in the month
- *hh* is the hour number (24 hour system)
- *mm* is the number of minutes
- .*ss* is the number of seconds (this parameter is optional).

For example, the command:

date 8806131627

sets the date to June 13 1988, 4:27 pm.

4 Log out.

If you made a mistake setting the date and time, you can always correct it by using the *date* command again. Don't worry about whether you are on standard or daylight saving time as this is automatically changed by the computer.

On RISC iX systems the time is maintained in GMT and programs which read/write the time convert this to and from local time. However, RISC OS is initially set to print and enter the date and time in its local form and does not understand daylight-saving time.

This discrepancy can be partially resolved by setting the date under RISC iX. The penalty incurred is that when you use RISC OS the time displayed is GMT time, which will be an hour out during the summer.

For more information, refer to *date*(1).

Setting the password for various users of the system

This section explains how to set a password for the following users of the system:

- *root*
- *halt*
- *guest*

These user ids are already present on the system when it is first installed. They are initially supplied with no passwords, therefore, it is important that you set passwords for these ids as quickly as possible after the system has been installed.

Once you have set passwords for *root* and *halt* you should keep them secret. It is important that you prevent inexperienced or malicious users from accessing the system using either of these ids, especially *root*. If somebody logs in as *root*, they automatically have superuser privileges. This gives them access to every file and command available, and it is therefore very dangerous to leave these usernames without a password.

The userid *root* should only be used for System Administration purposes. In everyday work sessions, you should log in using your own username. Never use *root* for your normal work session.

Setting a password for root

Follow this procedure to set a password for *root* or if you wish to change the password that *root* is currently assigned.

1 If you are logged in as someone other than *root*, log out by typing *exit*.

2 Log in as user *root*.

3 The *root* password is set with the *passwd* (short for *password*) command. Type the command:

```
passwd
```

The following text is displayed:

```
Changing password for root
New password:
```

4 Type in the password that you want to use for *root*. The password you type in will not appear on the screen.

Choose a password that you can remember easily, but that others can't easily guess. The best passwords are those that mean something to you but which are meaningless to others. It should be at least six characters long and should consist of both upper and lower-case letters and/or numbers.

5 For verification you are prompted to type your password again:

Retype new password:

Type in the new password again. It will not be shown on the screen.

root is given the new password and you are returned to the command prompt.

However, if you mis-typed the password you will receive the message:

Mismatch- password unchanged

and be returned to the command prompt. Try again from the beginning.

6 Log out.

Setting a password for halt

The *halt* userid is used to automatically shutdown the system. Therefore it is a good idea to have this id password protected. To set a password for *halt*, log in as *root* and type:

```
passwd halt
```

Then follow the instructions for changing the password as described in the previous section.

Setting a password for guest

The *guest* userid is used primarily by first time users of the system who have never used RISC iX before. The home directory of the user *guest* contains many sample files and directories that first time users can work through, following instructions in the *RISC iX User Guide*.

To set a password for *guest*, log in as *guest* and follow the instructions previously outlined for setting the password for *root*.

Setting the name of the workstation

The login prompt that is displayed by the system following power on, displays a message about the version number of the software and also the name of the machine, referred to as the *hostname*.

By default, the generic name *unix* is used as the hostname of the machine. You can alter this hostname to something more meaningful by editing the following line in the file */etc/rc*.

```
...
if [ ${STANDALONE} = TRUE ]; then
            hostname "unix"
...
```

For example, to change the hostname of your machine to *tp6*, edit the above line to:

```
...
if [ ${STANDALONE} = TRUE ]; then
            hostname "tp6"
...
```

To change the hostname of a machine connected to a network, edit the following line in the file */etc/rc.net* to add the desired hostname of the machine:

```
HOSTNAME=
```

The next time you reboot the machine, you will see the new hostname displayed in the login prompt. For more information about setting your machine up on a network, refer to the chapter entitled *Setting up a local Ethernet network* on page 129.

Editing the message of the day

When you first log in on any of the virtual terminals, you are greeted with a message that tells you about the version of the software that the system is running. For example:

```
RISC iX release 1.2 made Fri Jan 13 15:19:07 1990
```

The text of this message resides in the file */etc/motd* (short for *message-of-the-day*) and is read by */etc/rc*, each time the system is started up.

Using your favourite text editor, you can edit this file to add extra information after the version information line. For example, to inform other users about recent changes you have made to the system.

Performing a level zero dump backup

It is recommended that at this stage you perform a level zero 'dump' backup of your filesystem. This is a routine precautionary measure; it ensures that you have a backup copy of the original file structure to fall back on in the unlikely event that your filesystem gets totally corrupted.

For information about how to perform a level 0 backup on your system, refer to the chapter entitled *Backing up the filesystem* on page 65.

Finding out about the filesystem

Introduction

This chapter provides you with some background information about the structure of the filesystem, how it works and where users fit in. The concepts described in this chapter will crop up repeatedly throughout the rest of this Guide, so it is essential that you digest this information fully.

The chapter is split up into four main sections:

- Layout of the filesystem
- Users of the filesystem
- How the filesystem works

Layout of the filesystem

The RISC iX filesystem is distributed with utilities and libraries in locations in the file directory tree that conform to the filesystem layout for X/Open and UNIX System V Release 4.0 (SVR4.0). If you are familiar with standard UNIX filesystem layouts then you will be surprised to find that some of the standard directories are no longer present in this new filesystem. For example, there is no */bin* directory any more.

Basic structure of the filesystem

The rationale behind the new layout was to create a filesystem structure that more closely reflected how UNIX machines are now being used. UNIX has evolved from running on standalone microcomputers supporting a small number of text-based terminals, to running on smaller but more powerful workstations integrated on a network, sharing resources and running graphics-based applications software.

The physical organisation of files and directories that comprise the RISC iX filesystem has therefore been changed:

- to facilitate sharing in network environments between machines with differing CPU architectures
- to clearly separate differing sub-trees
- to rationalise add-on package installation

To incorporate these considerations, the entire filesystem has been divided into three basic categories:

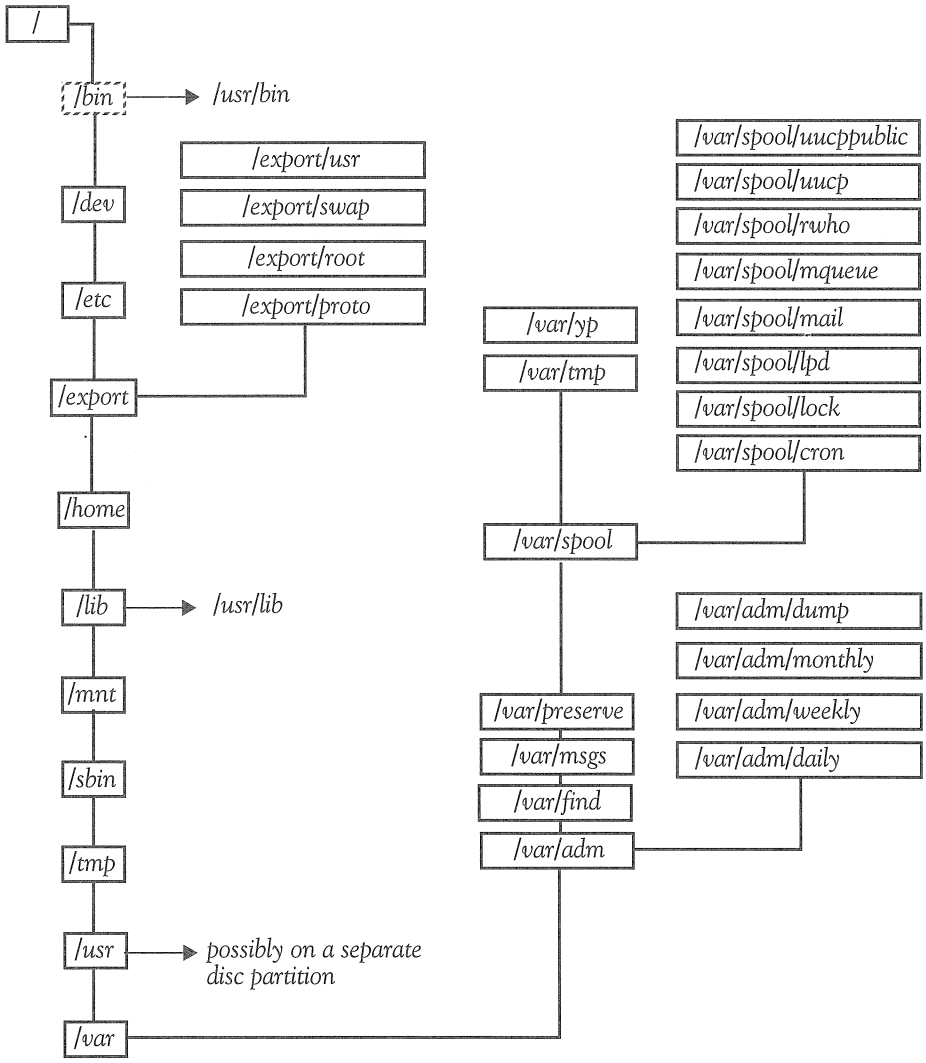
- Machine private configuration files – this category includes those files that should not be shared with other machines, regardless of CPU type. For example, */etc/rc* scripts for inherently machine-specific boot-up procedures or accounting logs. These types of files are contained in the root directory of the filesystem.
- Architecture dependent binary and library files – files that are can be shared across a network between machines of the same CPU type. For example, binary executable files. These types of files are contained in the */usr* directory of the filesystem.
- Architecture independent ASCII databases – files that can be shared across a network regardless of CPU type, eg the on-line manual pages. Files of this type are contained in the */usr/share* directory of the filesystem.

Where possible, symbolic links have been included to retain a degree of compatibility with the old filesystem layout, but it is recommended that you quickly modify any existing shell scripts that you have created to refer to the actual files, rather than symbolic links to these files.

The remainder of this section contains information about many of the major files and directories that are contained in the filesystem. However, you may find that the filesystem on your machine does not exactly match up with this description. The filesystem described is only the default filesystem layout. The layout of the filesystem on your machine may have been modified by your supplier to provide you with just enough of the filesystem to run a specific application.

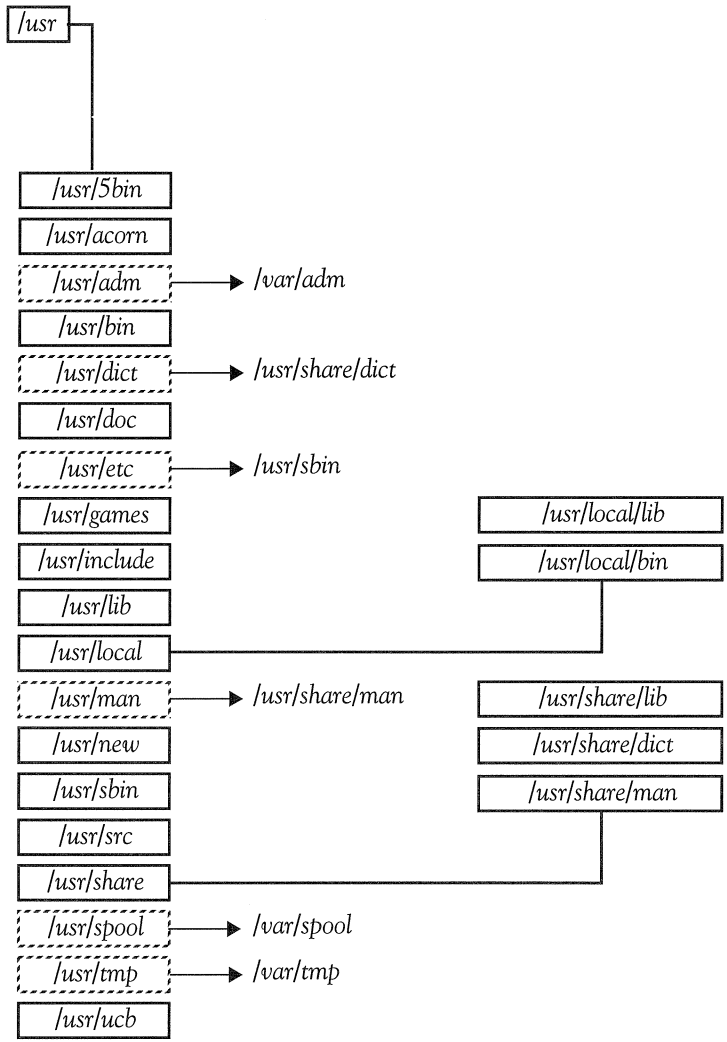
General structure of the root filesystem

A schematic of the overall structure of the most important files and directories in the root filesystem is shown below:



General structure of the /usr filesystem

A schematic of the overall structure of the most important directories in */usr* is shown below:



Detailed structure of the filesystem

The following list describes the contents of the directories shown in the previous section:

- */dev* – in this directory live a number of so-called *special files*. These are pseudo-files which give direct access to various devices. This includes the keyboard (*/dev/kbd*) and the three virtual terminals (*/dev/console*, */dev/ttyv0* and */dev/ttyv1*). When you attach a new peripheral device to your system you will have to create a special file in here to refer to the device. See *mknod*(8).
- */etc* – this directory is where essential System Administration and configuration textual files live. You will have to edit the files in here whenever you wish to make a change to the configuration of your machine. For example, when configuring the machine on a network, for starting new processes when the machine boots up. Files in here should be changed with extreme care.
- */export* – this directory will only exist if your workstation is acting as a server for one or more discless workstations. The directory will contain all the directories that a discless workstation will need. For example:
  - */export/proto* – contains a master copy of the root directory that is copied to */export/root/<hostname>* for each discless workstation supported by the server.
  - */export/root* – small root partitions, one for each discless machine.
  - */export/swap* – swap partitions, one for each discless machine.
  - */export/usr* – the *usr* directory shared by all discless machines

  For more information about discless machines, refer to the chapter entitled *Setting up a local Ethernet network* on page 129.
- */home* – this is the directory which contains the home directories for all the users of your machine. For example, the home directory for user *guest* is located in the directory */home/guest*. Any new user you create on the machine should have their home directory located in here.
- */mnt* – the conventional location for temporarily mounting a filesystem. For more information about mounting filesystems, refer to the chapter entitled *The Network Filesystem Service* on page 185.
- */sbin* – contains program files which are essential for the normal booting, mounting and general administration of the machine. For example, the command *reboot* lives here. Never remove or modify anything in this directory. Many other system software programs assume normal operation of everything here.

- */tmp* – this is a directory, that can be written to by all users, where various programs create temporary files. Files left in here are likely to be deleted without notice. For example, the files contained in this directory are cleared out every time the system is rebooted.
- */usr* – this directory and all sub-directories in it contain files that can be shared between machines and that remain static over the life of the system, or until a new release of the system software is installed. (Files and directories previously located in this directory that are subject to change are now located in */var*.)

  To retain a degree of backwards compatibility and ensure that various shell scripts still work, symbolic links are used to preserve the previous directory naming conventions. For example, you can refer to the electronic mail directory located in */var/spool/mail* as */usr/spool/mail*. For more information about symbolic links, refer to *ln*(1).

  The following directories in */usr* contain different sets of system binary commands that are included for compatibility with other versions of the UNIX operating system and for conformance with UNIX operating system standards. The directories are:

  - */usr/bin*
  - */usr/5bin*
  - */usr/ucb*

  The contents of these binary directories are very similar – commonly used commands are contained in each directory. However, there are subtle differences in usage between similarly named commands contained in any of the binary directories. For more information about the contents of these directories and the differences between them, refer to the chapter entitled *Alternative system commands* on page 331.

  Some of the other main directories contained in */usr* are described below:

  - */usr/acorn* – contains utility programs that are specific to Acorn RISC iX workstations.
  - */usr/games* – contains games programs for you to play with during your spare time, if you are fortunate enough to have spare time!
  - */usr/include* – contains standard *#include* files used by the C compiler.
  - */usr/lib* – contains standard programming and language libraries plus any additional language libraries or architecture dependent databases that may have been installed on your system.

- */usr/local* – contains programs and files local to your site. Software developed and used on your machine should normally go here, (except you may want to have your own directory of personal commands for your login id).
- */usr/local/bin* – is used to contain local binary files.
- */usr/local/lib* – is used to contain local libraries etc.
- */usr/new* – contains miscellaneous 'new' software.
- */usr/sbin* – contains program files similar to those contained in */sbin*, but not required for booting the machine. For example, the *dump* command lives in this directory. The files in here should only be used by you to periodically maintain the machine, therefore this directory should not be specified in normal user's search paths.
- */usr/share* – contains files that can be shared by machines with different hardware architectures. For example, the on-line manual pages are probably installed on your machine in the directory */usr/share/man*.
- */usr/src* – is ordinarily used to contain source code for utilities and libraries. For example, the source code for the RISC iX kernel may be held in */usr/src/sys* if you are licensed to hold sources. If not, this directory will be empty or may not even be present at all.

- */var* – contains files that tend to change throughout the life of a system, usually growing in size. For example, electronic mail message files, system accounting files etc. It is one of your responsibilities as a System Administrator to ensure that these files do not grow too large.
  - */var/adm* – contains various logs of system usage. These are *per-process accounting* records, which are records of each command run to completion, with their I/O and CPU usage. For example, a record of logins and logouts may be contained in */var/adm/lastlog*.
  - */var/adm/daily*, *weekly* and *monthly* – a set of three shell scripts for System Administration tasks. These housekeeping scripts can help you to keep the filesystem tidy. For more information, refer to the chapter entitled *Maintaining the filesystem* on page 51
  - */var/adm/dump* – a directory containing a set of shell scripts for performing incremental backups of your system. For more information about using these scripts, refer to the chapter entitled *Backing up the filesystem* on page 65.
  - */var/find* – a directory containing data files used by the *find* command.
  - */var/msgs* – a directory containing system message files and log files.

- */var/preserve* – contains copies of files that were being edited with *vi* or *ex* when the system crashed or was rebooted.
- */var/spool* – contains sub-directories which contain files used by continually-running system processes to store their output.
- */var/spool/cron/atjobs* – is where files are saved for the *at*(1) command to execute.
- */var/spool/mail* – contains individual files for each user who receives electronic mail on the system. The files are named after the username of the user and are also owned by the user.
- */var/spool/lpd* – is where print spool files are saved.
- */var/spool/uucp* – is where data and control files for pending *uucp*(1C) requests are stored.
- */var/spool/uucppublic* – is a standard place where files arrive from remote systems, via *uucp*.

  The originator should normally send you a message in the mail (*uucp* can be made to generate its own rather cryptic message using the *-m* option) saying that something has arrived, and thereafter you should move it as soon as possible into your own home directory.
- */var/spool/rwho* – a directory containing information about users on other machines. Used by *rwhod*, the *rwho* daemon.
- */var/spool/mqueue* – a directory containing information used by the *mail* program.
- */var/spool/lock* – a directory containing lock files created by *lockd*, the lock daemon
- */var/tmp* – like */tmp*, this is another directory for the storage of temporary files. However, it differs from */tmp* by not having its contents deleted when the machine is rebooted. It is a good location for storing temporary files.
- */var/yp* – contains the files used by the network information service (NIS) if the machine is acting as either a NIS master server or a NIS slave server on a network running NIS. For more information about NIS, refer to the chapter entitled *The Network Information Service* on page 219.

  Note, that NIS was originally called *Yellow Pages*, but has since changed it's name. However, the reference manual pages may still refer to NIS as Yellow Pages.

See *hier*(7) for a more extensive profile of the filesystem.

Administration files

The following lists summarise some of the more important administration files that are normally present in various directories on the system, and with which you should become familiar.

Files in /dev

The files in */dev* are all either *block special* (disc blocked) devices or *character special* (terminals and physical disc) devices.

Care should be taken to preserve the access modes that are originally set on these files. By widening the access, malicious or careless users can destroy the system completely by randomly writing over disc or memory.

However, you must always have some minimal form of access on each device; for example, *ps* requires access to probe kernel memory using the memory access device */dev/kmem*.

The files contained in this directory are listed below:

File	Function
console	Console terminal
*eco**	Econet (Acorn's proprietary networking system)
fb	Frame buffer
*fd**	Floppy disc block devices
kbd	Keyboard
kmem	Kernel memory
lp	Parallel port
mem	Physical memory
mouse	Three-button mouse
null	Sink for unwanted output; immediate EOF on input
*pty{p,q,r}**	master pseudo terminal devices (used by network software)
*rfd**	Raw floppy disc devices
*rst**	Raw hard disc devices (various units and partitions)
*sd**	SCSI devices
serial	Serial port
sound	Sound output
*st**	Hard disc (various units and partitions)
tty	Generic terminal (converted to current terminal)
*tty{p,q,r}**	Slave pseudo terminal devices
*ttyv**	Virtual terminal devices

The access modes on */dev/null* and */dev/tty* should be generally accessible, ie 666. This is because */dev/null* is used everywhere as a character sink, and */dev/tty* is translated to mean the 'current terminal'.

Files in /etc

The files contained in here are textual ordinary files containing configuration and descriptive information about various aspects of the system. Some of these files have corresponding manual pages in section five, which give a complete description of their use.

The administration files contained in this directory are listed below:

File	Function
*auto.**	Configuration files used by the *automounter*
disktab	Disc device description file
dumpdates	Description file containing dates of backups
ethers	Description file containing a list of Ethernet address to Internet number mappings for all the machines on a network. This may be empty on some systems.
exports	Description file containing a list of the directories that can be exported to other machines on a network
fstab	Description file containing a list of local and remote filesystems to be mounted on the machine
gettytab	Database description file used by *getty*
group	Description file containing information about user groups on the machine
hosts	Description file containing a list of the Internet addresses and host names of machines on a network
hosts.equiv	Configuration file containing a list of machines with equivalent password files. One of the files used to govern login access across a network.
hosts.lpd	Configuration file used to govern printer access across a network
inetd.conf	Configuration file used by *inetd*
motd	File containing the message of the day
mtab	Description file containing a list of currently mounted filesystems
networks	Database description file containing a list of known networks
passwd	Password file
passwd.dir	Password database description file
passwd.pag	Password database description file
printcap	Configuration file used for setting up a printer
protocols	Description file containing a list of known network protocols

psdatabase	Database description file of kernel name information. Used to speed up the operation of *ps*(1).
rc	Configuration file containing the commands that are executed at start-up
rc.config	Configuration file for setting the machine up on a network or to run NIS
rc.local	Configuration file containing machine-specific commands that are executed at start-up
rc.net	Configuration file containing commands that are executed at start-up for networked machines
rc.yp	Configuration file containing commands that are executed at start-up for networked machines which are also running NIS
remote	Description file for identifying remote hosts
rmtab	Description file containing a list of machines that have done remote mounts of filesystems from the machine
rpc	Description file containing a list of remote procedure calls that the machine recognises
sendmail.cf	Configuration file for *mail*
services	Description file containing a list of network services that the machine recognises
syslog.conf	Configuration file controlling the output of system logging information
termcap	Description file containing a list of the terminal control codes for a variety of terminals
ttys	Configuration file for terminal lines
utmp	Description file containing a list of currently logged-in users
xtab	Description file containing a list of currently exported filesystems

The use of these files will be described in the relevant chapters of this Guide.

Files in /sbin

The program files that are essential for the correct operation of your machine and for performing important System Administration duties are held in this directory. For example, programs such as *fsck* and *mount* are held in here, as are the system start-up software and *daemon* processes, such as *init* and *reboot*.

A list of the program files held in this directory is given below:

File	Function
bbutil	Disc utility for mapping out bad blocks on st506 devices
chgrp	Change group ownership on a file or directory

chmod	Change the access mode on a file or directory
chown	Change the ownership on a file or directory
cp	Copy files
date	Print and set the date
echo	Echo arguments
econetup	Econet initialisation program
ed	Simple line-based text editor
ex	Powerful line-based text editor with a display editing facility (*vi*)
fsck	Filesystem consistency checking program
halt	Halt or reboot the system
hostname	Set or print the name of the current host system
ifconfig	Configure network interface parameters
init	Initial system process
ls	List contents of directory
mkdir	Make a new directory
mknod	Make a special file in */dev*
mount	Mount filesystems
mv	Move or rename files or directories
ps	Print information about running processes
reboot	Halt or reboot the system
restore	Filesystem retrieval program
rm	Remove or unlink files or directories
route	Manually manipulate network routing tables
rrestore	Filesystem retrieval program across a network
sh	Bourne shell
test	Evaluate a conditional expression
umount	Unmount filesystems

You should already be familiar with most of the commands listed above:

- *chmod*, *cp*, *ls*, *mkdir*, *mv* and *rm* are described in the chapter entitled *Using RISC iX* in the *RISC iX User Guide*
- *echo*, *ps* and *sh* are described in the chapter entitled *Using the shell* in the *RISC iX User Guide*
- *ed* and *ex* are described in the chapter entitled *Text editing* in the *RISC iX User Guide*.

The remainder of the programs contained in this directory will be introduced in the relevant chapters of this Guide.

Files in /usr/lib

The following list details the administration files in */usr/lib*:

File	Function
aliases	Aliases file used by *sendmail* (if this is running on your system)
aliases.dir	Aliases database description file used by *sendmail*
aliases.pag	Aliases database description file used by *sendmail*
atrun	Command to run a script at a specific time
*Mail.*help*	Help files for *mail*
Mailrc	Configuration file for *mail*
more.help	Help file for *more*

Files in /usr/lib/uucp

The following list details the administration files in */usr/lib/uucp*:

File	Function
USERFILE	Remote UUCP access permissions
uucico	File transfer program used by UUCP
uuclean	Program to tidy up a UUCP spool directory
uuxqt	Program to interpret UUCP execution files

For more information about UUCP and how to set it up on your system, refer to the chapter entitled *Setting up UUCP* on page 305.

Files in /usr/sbin

The following list details the administration files in */usr/sbin*:

File	Function
ac	Print information about logged-in users
accton	Turn on accounting information for all processes
arcat	Create an archive of manual pages
arp	Display and modify Ethernet to Internet translation tables
automount	Automatically mount NFS filesystems
biod	NFS daemon
catman	Create formatted versions of the manual pages
clri	Clear all un-allocated inodes
comsat	Notify users about incoming mail
cron	Execute commands at specific dates and times
dmesg	Collect and print system diagnostic messages
dump	Filesystem backup program
*dump**	Filesystem information used by *dump*

exportfs	Export and un-export directories to NFS clients
fastboot	Reboot the system and prevent disc checking
fasthalt	Halt the system. Do not check the disks on next boot
fingerd	Daemon used by *name* and *finger*
fsirand	Inode generation program
ftpd	Daemon used by *ftp*
gettable	Get NIC format host tables from a host
getty	Open and initialise a terminal line. Used by *init*
groupadmin	Update group files
htable	Convert NIC format host tables
inetd	Internet daemon
lpc	Line printer control program
mkfs	Construct a filesystem. Used by *newfs*.
mkhosts	Generate a hosts database description file
mklost+found	Make a *lost+found* directory. Used by *fsck*.
mkpasswd	Generate a password database description file
named	Internet domain name daemon
newfs	Construct a new filesystem
nfsd	NFS daemon
nfsstat	Collect and print NFS filesystem statistics
ntalkd	Remote user communication daemon
pac	Printer/plotter accounting information
packageadmin	Install or remove software packages
ping	Send Internet packets to a machine on a network
portmap	Convert RPC program numbers into DARPA protocol port numbers
pstat	Prints information about the system, derived from certain system tables
rdump	Filesystem backup program across a network
renice	Alter the schedule priority of currently running processes
rexecd	Remote execution daemon for network services
rlogind	Remote login daemon for logins across a network
rmt	Remote magnetic tape control program
routed	Network routing daemon
*rpc**	Protocols for communication between programs across a network
rshd	Daemon for remote shells
rwalld	Daemon used by *rwall* and *shutdown*
sendmail	Electronic mail program
spray	Send packets of information to a machine on a network
swapon	Specify additional device for paging and swapping
syslogd	Daemon used to log system messages

telnetd	Daemon used by *telnet*
tftpd	Daemon used by *tftp*
timed	Daemon used to synchronise the time of machines on a network, also used by *date*
timedc	Controls the operation of *timed*
tunefs	Tunes a filesystem, by changing certain dynamic parameters of the filesystem
update	Daemon that keeps the filesystem up to date, by running the command *sync* every 30 seconds
useradmin	Updates user files
uucpd	Daemon used by UUCP
vipw	Allows you to edit the password file and prevents others from editing it at the same time
yp	Directory containing NIS programs for managing and controlling machines on a network
ypbind	Daemon used by NIS
ypserv	Daemon used by NIS

Files in /var/spool

The following list details the administration files in */var/spool*:

File/directory	Function
/var/spool	Various spooling functions
cron	Used to run *at* command
lpd	Printer spool directory
mail	*mail* directory, used to store user's mail
uucp	Used by *uucp* to save pending jobs
uucppublic	Used by *uucp* to receive files

Users of the filesystem

The RISC iX operating system permits you to control access to files on the filesystem so that several people can share access to the machine, but no one can do anything to anyone else's files which is unintended.

For each of your files you can control whether different users can *read* it, *write* to it, or *execute* it – ie run it as a program file.

To implement this, users can have the following things associated with them:

- *username* – usernames are the unique strings of characters (usually all lower case letters) which people log in with. You should think of yourself and other users in terms of these names.

There are some reserved names, called 'system users' built into the system, such as *root*, *daemon*, *uucp* and *operator*. These are discussed in more detail in the section entitled *System users and groups* on page 37.

Apart from this, as System Administrator you may assign whatever names you like to each user on the machine to distinguish each user, however consistency is recommended. Usual conventions are the initials of the user or initial plus surname.

- *userid* – this is a number, between 0 and 65534, unique to the user, which the kernel uses to identify the user.

  Numbers between 0 and 99 are normally reserved for system users by convention, and 0 represents the userid of *root*.

- *groupid* – this is a second number between 0 and 65534 which is associated with each user, but which may be shared between different users, who are all said to be in the same group.

- *groupname* – corresponding to the groupid is a group name, held in a separate file from the list of user names. Group names are unique names that are issued to a certain set of users. For example, a group of programmers working on a new windowing system could be in a group called *windows*.

  There are usually fewer groups than there are users, and on many systems with only a small number of users, the whole concept of groups is somewhat redundant, all the users being put into one group.

Where groups are used, it is normal to permit users in the same group to access more files than users not in the group. You might want to permit users in the same group as you to have different access to files from users not in the same group.

When you create a file, it is given your numeric userid plus the groupid of the directory in which it was created. These two ids are stored by the system, and subsequent attempts to access the file are controlled by comparing the user and group ids of the user attempting to access the file with these stored ids.

The user ids are also used to decide who can send signals to running background processes using the *kill* command. Only if the user ids of all users (except *root*) and the process, match is this permitted; group ids are not considered. For more information, refer to *kill*(1).

System users and groups

There are a number of standard user and group names which are 'built-in' to the system.

The system users are the owners of system files and programs. Some names may share a common userid, and some are not actual users at all, but a convenient means for quickly performing system operations from the *login:* prompt.

- *root* – is the name of the so-called *super-user*. This user is discussed more fully in the section entitled *Root* on page 37.
- *sync* – is a pseudo-login name to synchronise the discs from the *login*: prompt.
- *halt* – is a pseudo-login name to halt the system from the *login*: prompt.
- *daemon* – is a system user under whose userid various processes needing to restrict access to data files operate. For example, line printer spooling, maintaining games score files.

  You should not need to log in as this user and you are normally prevented from doing so by a '*' being placed in the password field of the entry for *daemon* in the */etc/passwd* file. This prevents logins on that account.
- *operator* – is a system user who owns many of the system files and may have special access to some of them.
- *uucp* – is the name of the owner of the files concerned with the UNIX-to-UNIX copy system.

  You do not normally log in as *uucp*. Instead other UNIX systems calling into your machine to run *uucp* will log in using this userid. Instead of running the shell, a program called *uucico*(8C) is run instead. For more information, refer to the chapter entitled *Setting up UUCP* on page 305.

The following group names are installed on the system:

- *wheel* – this group contains users who may execute the *su* command – your userid should be added to this group, see the section entitled *The su command* on page 38.
- *daemon* – this is the group name corresponding to the daemon user.
- *operator* – this is the group name corresponding to the operator user.
- *staff* – this is the group into which the majority of users are placed.

Root

root is the user with the numeric userid zero. *root* has all protection removed from file access, allowing read or write access to any file on the system. In addition, root can perform additional administrative functions, such as sending messages to system processes, setting the time, and updating the CMOS RAM.

Owing to the power of this login name, you should not get in the habit of logging in as root. It is all too easy to destroy files and corrupt the system. You should only use root access when absolutely necessary. For example, when performing a System Administration task such as creating a new user on your system.

Likewise, the root password should be a closely guarded secret on systems shared by several people who may want to save important or confidential data which only they should access, as *root* can read or modify any of these files regardless of the access permissions.

The command prompt (which by default is '$' for the Bourne Shell and '%' for the C Shell) is replaced by a '#' for *root* as a warning that *root* access is in progress. You should never leave the machine unattended with this prompt displayed.

By default *root* uses the Bourne shell. If you prefer to use the C shell, you should change the final field in the password file entry for root to use *csh* rather than *sh*.

The su command

Rather than logging in as *root* every time you need to perform a System Administration task, you can use the *su* (short for *set-user*) command to temporarily impersonate *root* (or another user) if you wish.

To impersonate *root*, one of your group ids must be 0 (group *wheel*) ie your username should be added to the entry for group 0 in the file */etc/group*. For example, to enable a user with the username *jbloggs* to *su* to *root*, change the line:

```
wheel:*:0:root
```

to read:

```
wheel:*:0:root,jbloggs
```

jbloggs can then become root, as long as he knows the password, by typing:

```
su root
passwd
#
```

Whilst running the *su* command you are still logged in as you were, and the terminal 'belongs' to you. If someone runs the *who* command on another terminal your name will continue to be displayed, but you have the access of the user given with the *su* command (or *root* if none is specified).

This can be confusing – you will still be warned about mail due to you, but if you reply the mail message may seem to come from you, or the other user, or may get confused, depending on the mailer.

The main function of *su* is to invoke a single administrative task, and then return back to the logged-in user. Note that the shell presented to you is a 'sub-shell' of your logged-in shell, so after you have completed your System Administration task, you can drop back into your logged-in shell by typing <Ctrl-D>.

For more information, refer to *su*(1) and *who*(1).

The login command

The *login* command enables you to switch completely to another user as though you had logged out and back in again. This is appropriate where you want to perform a number of administrative tasks as *root*, or having done so, to go back to normal access, using your normal userid.

For more information, refer to *login*(1).

Set-user and set-group programs

RISC iX has a feature enabling utilities to be developed with special privileges normally associated in with *root* or another user, but for general use in a controlled fashion. Examples are *ps* and *su*, which have access to files which the general user cannot normally access.

The feature is the ability to make program files set-user id and/or set-group id. Programs such as these are called *set-user* or *set-group* programs. Whilst running, these programs have the privileges of the owner of the file (and not the user running the program). The owner of the file is said to be the *effective user* and the user running it is the *real user*.

In some cases the owner of the file is *root*. Thus, as in the case of *su* the unlimited privileges of *root*, once the password is checked, can be deployed to reset the effective and real users as required.

However, other effective users do arise; for example *uucp* is set-user id to user *uucp*, who owns the files which control the UNIX-to-UNIX copy package. But it is important to remember that the effective user *uucp* is no more privileged than other users, so *uucp* is unable to access files whose permissions only permit access by the invoking user.

In a similar fashion, but much more rarely, program files may be *set-group* so that there is an *effective* groupid in effect whilst the program is run.

You can recognise set-user and set-group when the file permissions are displayed using the command *ls* with the *-l* option, thus:

```
-rws--x--x 1  root     16044 Jun 18 16:59 setuserfile
-rwxr-sr-x 1  operator 23588 Jun 6 1986 setgroupfile
```

The *s* in place of the access permission *x* denotes that the set-user or set-group bit is set, in fourth position for set-user, and the seventh for set-group.

For more information about the relevance of these permissions in a secure network environment, refer to the section entitled *Making the network more secure* on page 205.

How the filesystem works

The filesystem structure of RISC iX, with its hierarchy of directories, is one of its key features. It is very important to keep the file structure intact, as the system assumes that some files are present and correct, and if certain key files are damaged the system will not boot and will have to be completely reloaded.

In other cases, the disc blocks used by files may overlap with consequent loss of data, or other curious effects may occur.

For these reasons you should be aware of the correct procedures to adopt, to recover the structure of the filesystem if anything goes wrong.

How files are made up

A file in the filesystem consists of two parts:

- The actual blocks of data of the file.
- The *inode*, which contains information regarding the whereabouts of the disc blocks that make up the file, and the access permissions, owner, group, modification and access dates etc.

  Each inode has a corresponding *inode number*. This number is the means by which the kernel refers to a file.

A directory just associates a list of names with corresponding inode numbers. Given the name, the corresponding inode number allows the kernel to access the file itself. It is not uncommon for several names to refer to the same inode number and hence the same file. See *ln*(1).

How files are stored

A directory is itself stored as a file containing this list of names and inode numbers. The operation of searching for a file by specifying a pathname, consists of examining directories to find the directory name, and then looking up corresponding inode numbers, to find the blocks of the next directory specified in the pathname, until the target file is found.

For example, to look up the name */home/tp/fred*, the kernel first looks at the root directory '/', whose inode number is conventionally 2 (numbers 0 and 1 are reserved) to find the name *home*. This has a corresponding inode number, and the inode is then looked at to find the blocks of the */home* directory, which is then scanned for the name *tp* and so on.

The inode has room to store the whereabouts of only the first few blocks of each file. Therefore, extra blocks need to be used to store where subsequent blocks of the file are held, which means that there is an escalating overhead on disc space for larger files, and of course slower access, as these extra blocks (known as *indirect blocks*) are referred to.

Blocks on the disc which aren't actually allocated to files are stored on the *free list*. In addition a free list of available inodes is maintained to cope with new files. It is possible to run out of inodes before running out of disc blocks, but the number of inodes allocated is usually quite generous, and the opposite is usually the case.

Identifying files

For identification purposes, each file has associated with it three 16-bit quantities:

- *Mode* – four bits determine the *file format*. The most important two types are regular file and directory, but in the */dev* directory are also encountered *block special* and *character special* files, and in various places *symbolic links* and *sockets*.

The next two bits are the set-user and set-group bits.

- *Owner* – this is the numeric *userid* as found in the password file, and denotes the owner of the file, and the user to be set in the case of set-user files.
- *Group* – this is the numeric *groupid* as found in the group file, and denotes the group of the file, and the group to be set in the case of set-group files.

The next bit is the *sticky bit* (short for *s*ave *t*ext *i*mage after execution). This is a special bit that can only be set by *root*. It has the following meaning when set on files and directories:

- Files with the sticky bit set increase the speed of execution of a regularly-run program, such as a screen editor.

  When a command is issued, the program files pertaining to this command are found and read off the disc and into memory. If there is not enough memory to hold all these files along with other tasks that may be running, parts of other program files are sequentially written out to a special area of the disc called the *swap area*.

For large programs this process of reading in files can sometimes take a while (remember that files are stored in separate blocks that may be scattered all over the disc). A way of circumventing this problem would be to leave the text parts of regularly-run programs permanently in the swap area so that they can be quickly read into memory when they are needed. This can be done by setting the sticky bit for a particular program.

- A file in a directory with the sticky bit set may only be removed or renamed by a user who has write permission on the directory, and either owns the file, owns the directory, or is the super-user. This is useful for directories such as */tmp*, which must be able to be written to by all users, but should deny users permission to arbitrarily delete or rename the files of others.

For more information refer to *sticky*(8).

Access permissions for files

The last nine bits are in three groups of three bits, and give read, write and execute permission for the owner (referred to as the *user* of the file), group and others. These bits can be set using the *chmod* command (as discussed in the chapter entitled *Using RISC iX* in the *RISC iX User Guide*).

The current mode for a file is displayed by *ls -l*, in the form:

```
-rwxr-xr-x
```

and for directories as:

```
drwxr-xr-x
```

The hyphen indicates absence of permissions.

The *x* bits are replaced by *s* for set-user or set-group files, and the final *x* by a *t* for sticky text files or sticky directories.

To change the mode of a file, you use the *chmod* command. This command takes a list of one or more files preceded by instructions about which permissions to add or remove, for example:

```
chmod u+w file1
chmod o-rw file2 file3
```

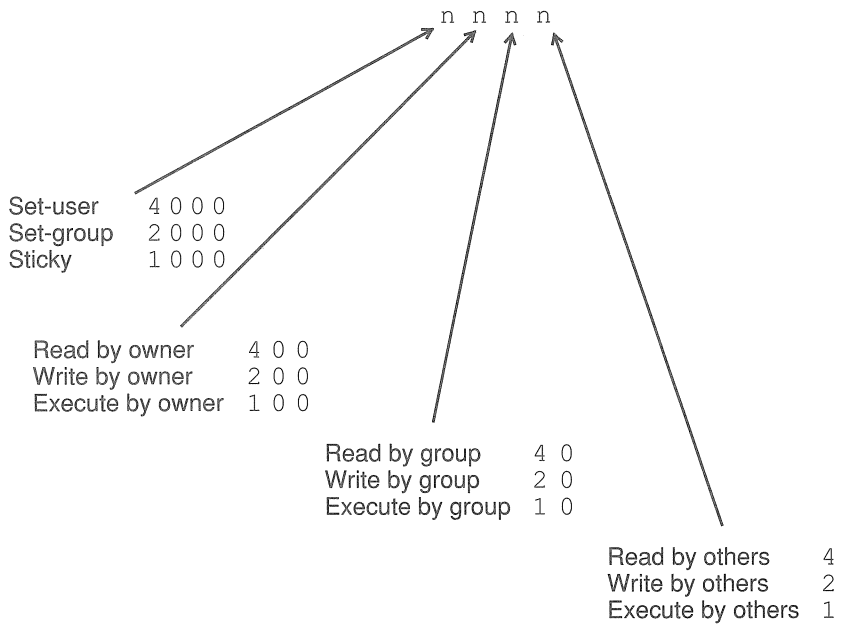
In the first example this would turn on the user's (designated by *u*) write (designated by *w*) permission for *file1*. In the second example the 'other users', not in the group, (designated by *o*) read and write permission bits would be turned off.

These options can be strung together using commas. For example:

```
chmod u+w-s,o-r file4
```

would turn on the write permission and turn off the set-user bit for the owner, and turn off the read permission for other users on *file4*.

You can also change access modes using four octal digits. Each digit denotes a specific permission, as shown below:



The octal notation is used extensively in the reference manual pages so you should get used to using this notation for changing access permissions on files.

For example, typing:

chmod 4755 *filename*

would set the set-user bit, the read, write and execute permission for the owner, and the read and execute bits for the group and other users as indicated below:

```
4 0 0 0     Set-user
  4 0 0     Read by owner
  2 0 0     Write by owner
  1 0 0     Execute by owner
```

4 0	Read by group
1 0	Execute by group
4	Read by others
1	Execute by others

The settings for *filename* would be shown, using the *ls -l* command, as:

```
-rwsr-xr-x
```

In the case of directories, the mode bits mean slightly different things:

- *Read* permission alone on a directory lets you list the contents, ie the file and directory names within that directory using *ls* for example, but prevents you from using *ls -l* or anything which would tell you about the type or contents of each file. For this, you need *execute* permission.
- *Write* permission on a directory does not permit even *root* to directly write to the blocks of the directory. What this means is that files can be created in the directory or deleted from the directory. However, this doesn't necessarily mean that you can read or write to the file. As long as you have write permission to the directory it is possible to delete a file you cannot read or, more importantly, write to a file.
- *Execute* permission on a directory (sometimes known as search permission) means that you can use it in a file pathname to get to a file in that or a sub-directory of the directory. For example in the pathname */a/b/c/d* all of the directories */*, */a*, */a/b* and */a/b/c* must all have execute permission and do not need read permission. But */a/b/c/d* must, as appropriate, have read or write permission to access this file. *Execute* permission also allows you to perform *ls -l* commands on a directory.

For example a directory marked as:

```
drwx--x--x ... dir1
```

would allow other users to pass through it to access files and directories (possibly with wider access) in that directory, provided they knew the name of the files or directory. However, the directory would not allow other users to list its contents.

For example any of the following commands typed by another user would be acceptable:

```
cat dir1/filename
ls -l dir1/dir2/dir3
cd dir1 ; cat filename
```

But the following command would produce an error message:

```
ls dir1
dir1 unreadable
```

Note that *r* access to a directory is not required to change directory to it – only *x* access is required. However, once you have changed directory, you will not be able to use *ls* to see its contents.

For example, another user typing the following commands would have access denied:

```
cd dir1
ls -l
. unreadable
```

Default access permissions on files

In the discussion of file modes it was described how, in many cases, the permission bits were represented by four octal digits. For example setting a file to mode *755* would cause *ls -l* to display the file as having mode *-rwxr-xr-x*.

Since many programs create files, it is helpful to have these files created with standard permissions. However, the permissions that are given to newly created files and directories will vary from one machine to another, and from one application to another.

The *umask* is a field of nine bits passed with every process that controls which bits in the last nine bits of the permission requested by the creating program should be left on for a newly created file. In fact the effect is only to *reduce* the permission, so that if the program only requests certain bits when it creates files, at most those bits will be left on.

For example if a program requests that a file be created with mode 666 (read/write permission for all users, ie *-rw-rw-rw-*) and the *umask* is set to turn off read and write permission for 'others' and write permission for other members of the group, then the file will be created with *640* permission instead.

Slightly confusing to new users is the fact that the *umask* bits set *on* in the *umask* are the bits to be turned *off* in the access permission. Thus the *umask* in the above example is *026*. Another example would be *002* which permits everything except writing by 'others'.

Do remember that if the program chooses not to specify bits when it creates file modes then permission in the *umask* field does not turn them on. Thus many programs create data files with mode 666, which will be converted by the above *umask* values to *664* and *640* respectively. Only language compilers and the link editor *ld* commonly create files with execute permission; ie with mode *777*.

Note that many items of RISC iX software will be confused by values of the *umask* which are particularly restrictive (especially if 'owner' bits are turned off) or which permit wider access for 'group' or 'other' than the owner.

For more information, refer to *chmod*(1) and *umask*(1).

Dates on files

Three dates are recorded on all files in the filesystem:

- The date the file was last read. A backup of the filesystem will probably affect this date which is very easily changed. It can be displayed using the command *ls -lu*.

  Execution of the file does not count as a read operation for this purpose.
- The date the file was last written, or its contents modified in any way. This is the date which *ls -l* displays, and is the most used.
- The date that some changes were made to the file, and *not* as described in some books as the creation date of the file. A change is a modification to the file contents (thus anything which affects the modification date will also affect this) *or* some changes to the inode. Examples of changes to the inode are addition or removal of a hard link to the file, change of the mode or the owner.

  This date is displayed by *ls -lc*.

The *touch* command lets you reset all these dates on a file. For example, typing the command:

```
touch psfile
```

will reset all the above dates of the file *psfile* to the current time and date. This is fine for updating access or modification dates but is misleading for the change date which is also affected. Unfortunately *touch* has no option for adjusting the access or modification dates to a file without also causing the change date to be updated.

How the filesystem is updated

As blocks of a file are written, and inodes updated, they are not written immediately to the disc, but are held in a *buffer cache* of memory, to be written to the disc later. This not only saves disc I/O if only part of a block is written in successive operations, but enables read requests to be serviced without reading the disc, if the blocks happen to be in the cache.

However, it does mean that the disc is not necessarily always up-to-date with the state of the filesystem, although as the contents of the cache are turned over, the disc is updated.

To tell the system to bring the disc up-to-date with the contents of memory, the command *sync* is provided. You should always run this command if there is any danger of accidents. For example, if someone is working nearby who may upset the electricity supply, or if you are about to slightly reposition the machine on a desk and could jog a cable.

To save you actually having to log in as *root* to do this, the pseudo-userid *sync* is provided. For example:

```
RISC iX 1.2
11:23am on Fri, 20 Jul 1990 on console of hostname
hostname login: sync

RISC iX 1.2
11:23am on Fri, 20 Jul 1990 on console of hostname
hostname login:
```

This login name removes the need to log in first to execute this command; you can just type *sync* to the *login:* prompt instead. The discs are updated and the login prompt is output again. For more information, refer to *sync*(8).

When the system is halted the disc has to be made completely up-to-date with the contents of memory, otherwise parts of files, and more seriously, inodes and directories, may not be written correctly to disc. Normally, the *halt* command takes care of this, as it includes the *sync* command in part of its shutdown procedure. Therefore, if *halt* is always used to shutdown the machine, no problems should arise.

There is additionally a continually-running process, called *update*, which is started up when multi-user state is entered. This comes into play every half-minute or so, and runs *sync*, thus you should not lose more than the last half-minute's worth of data should the system crash.

To deal with that last half-minute, the program *fsck* is provided, to repair the disc structures after a crash. For information about using *fsck*, refer to the chapter entitled *Maintaining the filesystem* on page 51.

How peripheral devices are referred to

Peripheral devices are treated just the same as ordinary files and directories in the filesystem and are referred to in the */dev* directory. The entries in this directory are special, in that no actual data is stored in the files contained in the directory. Instead the file names found here are used to designate peripheral devices instead.

The access permissions are usually set very carefully, so that random users and malfunctioning programs cannot scribble unchecked over discs etc.

Devices are sub-divided into two types:

- block devices
- character devices

Block devices

Block devices are devices such as discs and tapes. They handle data in blocks, usually 512 bytes in size. If you access a block device, the I/O system converts the blocks into smaller units.

In fact, the previously mentioned buffer cache is also used for all references to block devices, so it is possible to read and write individual bytes on discs, even if the disc can only read whole numbers of sectors; the kernel buffers the whole sector and updates the appropriate bytes before rewriting.

Character devices

Character devices are much less precise in definition. They include terminal and printer devices which work in terms of individual characters and also include under the same heading 'raw' or 'physical' access devices to discs and tapes.

You must be careful about using the 'raw' device, when dealing with discs or tapes because often they insist that individual read or write operations must be performed using an exact multiple of the sector size. This may confuse certain programs. If in doubt use the program *dd*(1).

For example to access an ADFS format floppy, you would type:

```
dd if=/dev/rfdf1024 bs=1k | etc...
```

where */dev/rfdf1024* refers to the floppy disc and is being used as the input file in the above example.

For certain special operations on certain discs and tapes you *must* use the raw device, because the system call *ioctl* is used to execute these operations and only uses the raw device. An example is the command to format a floppy, and thus the floppy disc format command uses the raw device.

For example:

```
ffd 1024
Do you really want to format the disc (y/n)? y
Commencing format of /dev/rfdf1024
...
```

There are other important device names in the */dev* directory:

- */dev/null* – is the null device. Output sent to it will be harmlessly thrown away, and attempts to read from it will always give an immediate end-of-file.
- */dev/tty* – is mapped to whatever terminal the user happens to be running from.

  This is useful if you have a program, or a shell script, running with output going to a file. As the script runs it may become necessary to output a message onto the user's terminal. Quoting */dev/tty* saves you from having to work out which one of the various terminal devices (*/dev/console*, */dev/ttyv0*, */dev/ttyv1* or a terminal emulator running under the X Window System) is the correct terminal device. For example:

  ```
  cat > /dev/tty
  hello
  hello
  there
  there
  <Ctrl-D>
  ```

For information about the other devices referred to in the */dev* directory, refer to the chapter entitled *Setting up peripheral devices* on page 93.

Maintaining the filesystem

Introduction

This chapter introduces you to the tasks that you will have to perform to maintain the filesystem. This includes, maintaining the integrity of the data on your filesystem by running *fsck* (the filesystem consistency check program) and also keeping your filesystem tidy by removing unnecessary files that are created during normal working sessions.

There are a set of shell scripts on your system that will perform these tasks automatically for you. Before introducing these scripts and showing you how to use them, the first few sections describe how to perform these tasks manually and also explain why these tasks need to be performed at all.

Checking the filesystem using fsck

The program *fsck* enables you to check the integrity of the RISC iX filesystem in terms of its blocks and sizes, pathnames, link counts, inode format etc.

The RISC iX filesystem contains a number of complicated structures. If any failure occurs, such as a system crash, a power cut or anything strange happens, you should run the program *fsck*, in single-user mode. This program runs over the disc and repairs damaged parts of the file structures. Usually, in the case of a power cut or other disaster, recently-created or modified files will be damaged, but now and again some quite substantial parts of the system may be damaged.

The main cause of damage to the filesystem is because disc blocks are buffered in memory in the *buffer cache* for some time before being written out to disc. Therefore, if the system crashes, there may be some parts of the filesystem that did not get written out to the disc and are still stored in the buffer cache.

On systems with less than 100 MB of disc storage, it is advisable to run *fsck* every time the system goes multi-user – since the time taken to check the discs is only a few minutes.

The existence of the file */slowboot* will cause the */etc/rc* script to use *fsck* to check filesystems every time the system goes multi-user. On larger systems, this file can be removed, but you should remember to do a manual *fsck* at the single-user prompt, before going multi-user, if the system had previously crashed.

Reboot with no sync

If *fsck* discovers and corrects faults in the filesystem, then there will be a problem if the kernel is holding cached copies of the repaired structures. Executing *sync* will cause those faulty structures to be written back, destroying the work which *fsck* has done.

In this instance *fsck* will tell you to '*reboot without doing a sync*', as the *sync*, or indeed the *halt* or *reboot* command, which both include *sync*, will rewrite the erroneous data structures which *fsck* has just carefully corrected. In this instance, you should reboot the machine without doing a *sync*. This can be done by typing:

```
reboot -n
```

The *-n* option causes the machine to be rebooted without doing a *sync*.

Preening

If the machine boots up automatically into RISC iX when it is switched on, then *fsck* will be run in *preen mode*, in which a quick check of the disc will take place looking for common errors.

If *fsck* detects any errors, they will be repaired as indicated by a confirmatory message and the machine will be automatically rebooted. Otherwise, *fsck* stays in single-user mode and tells you to run *fsck* manually.

Running fsck manually

To run *fsck*, bring the system down to single-user mode, by typing:

```
shutdown +5 Going down for filesystem check
```

Once the machine is in single-user mode, type:

```
fsck
```

By default, *fsck* will read the file */etc/fstab* to determine which filesystems to check. For example, if *fstab* contains just one entry:

```
/dev/st0a    /      4.3    rw           1 1
```

fsck with no arguments will check the device */dev/st0a* as if you had typed:

```
fsck /dev/st0a
```

If you want to check out a filesystem that has been created on a floppy disc (which does not have to be done in single-user mode if the disc is not mounted), type:

```
fsck /dev/rfdf1024
```

where */dev/rfdf1024* is the block device that refers to the floppy disc on which *fsck* is to be run.

In most cases you should reply 'y' to any question which *fsck* asks when it is checking the filesystem. You can use *fsck* with the argument '*-y*' to automatically answer '*yes*' to all questions. For example:

```
fsck -y /dev/rfdf1024
```

Common problems

If there are a very large number of errors, or *fsck* gives up altogether, the filesystem is very badly damaged and you are best advised to recover the system completely from backup media. For information on how to do this, refer to the chapter entitled *Backing up the filesystem* on page 65.

If any file names are displayed in the course of the operation of *fsck*, you should note down their names and ensure as soon as the system is running again that they are intact, reloading from backup media if necessary.

Now and again *fsck* will find a file that does not have a directory referring to it. In this case it will say something like:

```
UNREF FILE I=1234 SIZE=3921 USER=abcd
RECONNECT?
```

You should normally agree to this by typing 'y'. When the system is running again, you should look in the special directory */lost+found* into which such orphaned files are placed. The file will be given a name in this directory which is a string of digits based on the inode number displayed in the message displayed by *fsck*. In the above example, the file would be given the name *1234*.

You should periodically check the contents of the *lost+found* directory on your filesystem. Any file found should be recovered back to its rightful place in the filesystem if it is still required.

For more information, refer to *fsck*(8).

Tidying the filesystem

One of your tasks as System Administrator is to check that the disc usage is not increasing so quickly that one or more of the disc partitions suddenly becomes full. One problem is that some system processes run endlessly and freely create data, much of it of little interest, and soon builds up and fills the disc.

Every now and then you should type the command *df* to see how parts of the disc are getting on. You can see from the output the spare capacity of each partition. For example, for the root directory type:

```
df
Filesystem     kbytes    used   avail capacity  Mounted on
/dev/sd0a      34983     32010  2623  92%         /
/dev/rfdf1024 663      10    586   2%        /mnt
```

The above example shows the disc usage of an internal hard disc and a mounted floppy disc. The first column shows the device name on which the filesystem is running; subsequent columns show the total amount of space available, the space used, the space left to be used, how much has already been taken up in percentage terms and finally the directory on which the filesystem is mounted.

Note that the *used+avail* total is less than the amount of total space given in the *kbytes* column. This is because the system reserves a fraction of the space in the filesystem to allow its filesystem allocation routines to work well.

The amount reserved is usually about 10% of the total capacity of the disc, but this can be changed using *tunefs*. For more information, refer to *tunefs*(8).

If the percentage figure in the *capacity* column is approaching 100%, then you will need to delete some files to make more space. In the example above, the internal hard disc is up to 92% capacity. To prevent the disc becoming completely full, files will have to be deleted. This is not as distressing a prospect as it sounds, for you will find that there are many files you can delete first without having to start deleting precious user files or your favourite games programs.

UNIX systems are notorious for creating huge files in all sorts of hidden locations and it is your job as a System Administrator to ensure that these files do not build up to any significant extent.

Files tend to build up in the following places:

- */dev* – sometimes programs change directory to this directory and then crash, leaving large *core* files. Alternatively someone logged in as *root* could misspell a device name for output, perhaps to a floppy disc and leave an enormous file here.

  To check if there are any extraneous files in here, change directory to */dev* and type:

  ```
  du -s
  ```

  A small number (approximately 10KB) should be output as the size of the directory itself. A total of anything greater indicates that the directory probably contains files that need deleting.

- */tmp* – temporary files created by many system programs. The programs should normally remove such files when they have finished, but sometimes they don't. Whenever the system is rebooted all files in this directory are removed; however, directories remain. Therefore, you should make sure that you have an entry in your *crontab* file to remove sub-directories of this directory.
- */var/tmp* – this is another location for storing temporary files. It is not cleared out when the system is rebooted, so it should be checked periodically.
- */var/spool/uucp* – if UUCP is set up on your system then there will be two ever-increasing log files (*LOGFILE* and *SYSLOG*) in this directory which leave a blow-by-blow account of what UUCP activities occurred and how long they took. These should be deleted periodically – the program *uuclean*(8C) can assist in tidying up this directory.
- */var/spool/uucppublic* – UUCP files arriving from remote systems often come here. The intended recipients should be told about them. For more information about UUCP, refer to the chapter entitled *Setting up UUCP* on page 305.
- */var/spool/mail* – electronic mail messages arrive here. The files should be read when they arrive and then moved to the appropriate places. For more information, refer to *sendmail*(8).
- */var/adm* – you may wish to delete files kept in this directory as you will have to keep archiving and/or deleting the ever-increasing amount of data kept in here if it is not to consume endless disc space.

  Two large files in */var/adm* that you can quite safely delete are *acct* and *lastlog*. *acct* is just a file that contains a record of the execution of all processes on the system and *lastlog* just records logins. Accounting is really only needed if you are charging for use of the system, so normally you are quite safe to remove these files. You can then suspend accounting using the command *accton* with no options. To resume accounting at any time, just type:

  ```
  cp /dev/null /var/adm/acct
  cp /var/adm/acct /var/adm/lastlog
  ```

  and reboot the machine. Accounting will be started again.

Finding and removing large files

A handy method of finding large files is to use the *find* command with a couple of options to refer to specific classes or sizes of files. For example:

```
find / -size +100 -print
```

This command will display the names of all the files in the filesystem that are greater than 100 blocks. You can then decide which of these files you wish to delete.

The above command can be further refined so that it only displays large files that have not been used for a specified number of days. For example:

```
find / -atime +14 -size +100 -print
```

The above command will display the names of all the files in the filesystem that are greater than 100 blocks in size and have not been accessed in the last 14 days.

To display files which haven't been modified, rather than haven't been accessed in the above example, use *find* with the *-mtime* option. For example:

```
find / -mtime +14 -size +100 -print
```

This command will display the names of all the files in the filesystem that are greater than 100 blocks in size and which have not been modified in the last 14 days.

Note that the list of files that you receive from each of the above examples will also include program files, such as those that live in */bin*. Running these programs does not affect their access or modify times so they will be included in such a list. Therefore, it is more useful if you use the above commands to search specific directories of your filesystem.

For example, if the machine is left switched on most of the time, the contents of the */tmp* directory, that is usually cleared out during the start up procedure, may need to be deleted occasionally. Other temporary directories (*/var/tmp*) can also be checked.

Therefore, you could further refine your *find* command to search these directories only and also add another option to delete any such files found. For example:

```
find /tmp /var/tmp -ctime +2 -exec rm -rf {} \;
```

This command will delete all the files and sub-directories found in */tmp* or */var/tmp* that have not been changed in the last two days. The *-ctime* option checks whether the file has been changed, where changed means that either the contents of the file or some attribute of the file has been changed in the last two days.

The *-rf* option to *rm* ensures that sub-directories are removed also, and no messages are output.

This type of command could be inserted in your *crontab* file and run every day at 1.10am every morning. For example:

```
10 1 * * * find /tmp /var/tmp -ctime +2 -exec rm -rf {} \;
```

If you want to perform more than one task when you 'clean' the disc it is better to put the tasks to be executed in a shell script and arrange for *cron* to execute that, rather than have a large crontab file for *cron* to repeatedly search through.

For example:

```
10 1 * * * /usr/local/clean-disc
```

In this case the commands would be placed in a shell script called */usr/local/clean-disc*.

For more information about running commands at certain times of the day, refer to *cron*(8).

Filesystem maintenance scripts

To help you to keep your system healthy, there are three shell scripts on your system in the directory */var/adm* called *daily*, *weekly* and *monthly*. These 'housekeeping' scripts remove all unwanted files on your disc, check the state of the filesystem and can be altered quite easily to suit the specific requirements of your system.

You should run the scripts manually at the end of each day, week or month. More favourably, providing you keep your machine on all the time, you can run these files at set times during the night by placing an entry for each of them in the *crontab* file on your system.

The next few sections describes what each shell script does by examining each major part of the script. By understanding what the scripts do, you should then be able to modify them accordingly to suit the specific requirements of your system.

The daily script

The first part of the *daily* script, runs the command:

```
msgs -c
```

This command removes all system messages contained in the file */var/msgs* that are more than 21 days old. For more information, refer to *msgs*(1).

If you have system accounting set up on your system, the next part of the shell script can be used. It is initially commented out on your system by the '#' symbol at the start of each line:

```
#echo ""
#echo "Purging accounting records:"
#/usr/sbin/sa -s > /dev/null
```

If you remove the hash symbols from each line, the command *sa* will be invoked, which will tidy up the accounting files contained in */var/adm*. For more information, refer to *sa*(8).

The next part of the script runs *calendar* for all users on the system that have a file named *calendar* in their home directory:

```
echo ""
echo "Running calendar:"
calendar -
```

The *calendar* file contains a list of single line messages (with dates) which get mailed to the user on the date specified. For more information, refer to *calendar*(1).

The next part of the *daily* script checks to see if the system is logging debugging messages in the file */var/spool/debugging* as determined by the following line in the system configuration file */etc/syslog.conf*:

```
#*.debug                    /var/spool/debugging
```

This is initially commented out. Debugging can be started by removing the hash symbol from the line, creating an empty file called */var/spool/debugging* and rebooting the machine. All system debugging messages issued by the kernel will then be logged in the file */var/spool/debugging*.

If system debugging is running, then the debugging messages for the day are moved to the file */var/spool/debugging.old* and a new debugging file (*/var/spool/debugging*) is created:

```
if [ -f /var/spool/debugging ]; then
      echo ""
      echo "Rotating debugging log"
      mv /var/spool/debugging /var/spool/debugging.old
      cp /dev/null /var/spool/debugging
fi
```

Therefore, the debugging messages for your system are kept for only one day before they are removed. The debugging messages are thus rotated over a one day cycle. For more information, refer to *syslogd*(8).

The next part of the script performs a seven-day rotation of the mail system log file */var/spool/mqueue/syslog*. Again, this may or may not be running depending on the setting of the following line in the system configuration file */etc/syslog.conf*:

```
mail.debug                  /var/spool/debugging
```

This is initially set to be running on your system. For more information, refer to *syslogd*(8). The file *syslog* in */var/spool/mqueue* records all mail messages that passed through the system.

By saving these files each day an accurate record is available of all the mail messages that have passed through the system in the last seven days. This is very useful for debugging purposes when there are problems with the *mail* system:

```
echo ""
echo "Rotating mail syslog:"
cd /var/spool/mqueue
rm syslog.7
mv syslog.6  syslog.7
mv syslog.5  syslog.6
mv syslog.4  syslog.5
mv syslog.3  syslog.4
mv syslog.2  syslog.3
mv syslog.1  syslog.2
mv syslog.0  syslog.1
mv syslog    syslog.0
cp /dev/null syslog
chmod 644    syslog
kill -1 `cat /etc/syslog.pid`
```

The next part of the script runs *fsck* with the *-n* option to prevent it from writing to the filesystem, in the event of an error being found. (As *fsck* is running unattended and in multi-user mode it would be unwise to allow any changes.):

```
echo ""
echo "Checking filesystems:"
sync
/sbin/fsck -n | grep -v '^\*\* Phase'
```

The final three parts of the script run a set of commands to show the state of the system.

The command *df* is run to show disc usage, then the command *mailq* is run to show the activity of the mail daemon and finally the command *uusnap* that will give a quick snapshot of the activity of the UUCP system if it is running on your system:

```
echo "Checking subsystem status:"
echo ""
echo "disks:"
df

echo ""
echo "mail:"
mailq

echo ""
echo "uucp:"
uusnap
```

The final part of the *daily* runs a set of *find* commands that search for and remove scratch and junk files found in the filesystem.

The first *find* searches in */tmp* and deletes all the files found in here that have not been accessed in the past three days:

```
find /tmp          -atime +3 -type f -exec rm -f {} \;
```

The second *find* searches in */tmp* for directories, other than a *lost+found* directory, that have not been modified in the last day and deletes them:

```
cd /tmp; find . ! -name . ! -name lost+found -type d \
                -mtime +1 -exec rmdir {} \;
```

The third *find* searches in */var/tmp* for any files, other than files found in a *lost+found* directory, that have not been modified in the last seven days and deletes them:

```
cd /var/tmp; find . ! -name . ! -name lost+found \
                -mtime +7 -exec rm -f {} \;
```

The fourth *find* searches in */var/tmp* for any directories, other than files found in a *lost+found* directory, that have not been modified in the last day and deletes them:

```
cd /var/tmp; find . ! -name . ! -name lost+found -type d \
                -mtime +1 -exec rmdir {} \;
```

The fifth *find* searches in */var/preserve* for any files that have not been modified in the last seven days and deletes any such files found:

```
find /var/preserve   -mtime +7 -exec rm -f {} \;
```

The sixth *find* searches through the entire filesystem, ignoring NFS mounted directories, looking for the following types of files:

`[#,]*` all files beginning with a '#' or a ',', This is a standard naming convention used to denote temporary files.

`.#*` all files beginning '.#'. This is another naming convention that is used to denote temporary files.

`a.out` all files named *a.out*. This is the default name given to output files produced from C programs.

`core` all files named *core*. This is the default name given to core image files.

`*.CKP` all files ending in '.*CKP*'. This is the name given to temporary checkpoint files created by some editors.

`.emacs_[0-9]*` all files beginning '*.emacs_*' followed by a number between 0 and 9 and then any string of characters. This is the name given to temporary checkpoint files created by *emacs*.

Any of the above files found that have not been accessed in the last three days are deleted, using the command, *rm -f*:

```
#find /  -fstype nfs -prune -o \( -name '[#,]*' -o -name '.#*' -o -name a.out -o
-name core -o -name '*.CKP' -o -name '.emacs_[0-9]*' \)
#                    -a -atime +3 -exec rm -f {} \;
```

This last find is initially commented out to enable you to add to or remove the list of files that are thought to be 'useless'. For example, you may not wish *a.out* files to be deleted.

The weekly script

The first part of the *weekly* shell script performs a four-weekly rotation of the system messages file */var/adm/messages*. This may or may not be running depending on the setting of the following line in the system configuration file */etc/syslog.conf*:

```
kern.debug;daemon,auth.notice;*.err;mail.crit   /var/adm/messages
```

This is initially set to be running on your system. For more information, refer to *syslogd*(8). The file *messages* in */var/adm* records all the messages that are issued by the system including error messages, reboot commands etc. By saving these files every week for four weeks an accurate record is available of all the system messages that were generated by the system in the last four weeks:

```
echo ""
echo "Rotating messages:"
cd /var/adm
rm -f messages.3
mv messages.2 messages.3
mv messages.1 messages.2
mv messages.0 messages.1
mv messages   messages.0
cp /dev/null  messages
chmod 644 messages
kill -1 `cat /etc/syslog.pid`
cd /
```

The final part of the script, rebuilds the *find* database using the shell script *updatedb* in */usr/lib/find*. This reads in all the pathnames in the filesystem, ignoring NFS mounted directories, and stores all these pathnames in a database file named *find.codes* in the directory */usr/lib/find*. For more information, refer to *find*(1):

```
echo ""
echo "Rebuilding find database:"
su nobody << EOF 2> /dev/null
      /usr/lib/find/updatedb
EOF
```

The monthly script

The *monthly* script firstly checks to see if your system is running login accounting, by checking for the existence of the file */var/adm/wtmp*:

```
if [ -f /var/adm/wtmp ]; then
```

If this file exists, then the script runs the command *ac* with the *sort* option to give a neat summary of the use of the machine by each user:

```
echo ""
echo "Doing login accounting:"
/etc/ac -p | sort -nr +1
```

For more information, refer to *ac*(8).

The final part of the script performs a three-monthly rotation of the */var/adm/wtmp* file:

```
echo "Rotating wtmp file:"
cd /var/adm
mv wtmp.0 wtmp.1
mv wtmp   wtmp.0
cp /dev/null wtmp
```

Running the scripts using cron

The *crontab* file located in */var/spool/cron/crontabs/root* on your system already contains entries for the 'housekeeping' scripts just described:

```
...
#
# This is an example of a simple crontab - it is designed for use
# on a machine which is kept switched on all the time. If this
# is not true the System Administrator should run the scripts below as
# necessary to keep the system in a healthy state.
#
# Run the daily script at 1 am every day
#
0 1 * * * /sbin/sh /var/adm/daily 2>&1 | /usr/ucb/mail -s "daily admin script log"
root
#
# Run the weekly script at 2:30 am on Saturday
#
30 2 * * 6 /sbin/sh /var/adm/weekly 2>&1 | /usr/ucb/mail -s "weekly admin script log"
```

```
root
#
# Run the monthly script at 3:30 am on the first of the month
#
30 3 1 * * /sbin/sh /var/adm/monthly 2>&1 | /usr/ucb/mail -s "monthly admin script
log" root
...
```

As indicated by the comments in this file, the *daily* script is run at 01.00 am every day, the *weekly* script is run at 02.30 am every Saturday and the *monthly* script is run at 03.30 am on the first of every month.

In all cases, any error messages produced are redirected to standard output (`2>&1`) and then all this information is sent as a mail message to the user *root*.

Reading the mail messages produced by the scripts

At any time you can check what 'housekeeping' has recently been done on your system by typing, when logged in as *root*:

```
mail
```

You will see the following types of messages displayed, possibly interspersed with other messages that have been sent to you:

```
...
U 1 root   Sat Mar 18 01:03  48/1309 "daily admin script log"
U 2 root   Sat Mar 18 02:31  17/369 "weekly admin script log"
U 3 root   Sun Mar 19 01:02  48/1309 "daily admin script log"
U 4 root   Mon Mar 20 01:02  48/1309 "daily admin script log"
U 5 root   Tue Mar 21 01:03  48/1309 "daily admin script log"
U 6 root   Wed Mar 22 01:02  48/1309 "daily admin script log"
U 7 root   Thu Mar 23 01:02  48/1309 "daily admin script log"
U 8 root   Fri Mar 24 01:02  48/1309 "daily admin script log"
U 9 root   Sat Mar 25 01:03  48/1309 "daily admin script log"
U 10 root  Sat Mar 25 02:31  17/369 "weekly admin script log"
...
```

It is good practice to read these message at the start of each day, just to check that no serious errors were produced by any of the commands executed in the scripts. If no errors were produced, then you are free to delete the messages.

Backing up the filesystem

Introduction

As an efficient System Administrator you will need to take regular backups of all the data which has changed. You will need to take *full backups* of everything on the disc, and *incremental backups* of everything changed since a certain date, usually that of a previous full backup.

Available commands

There are three sets of commands that you can use for performing backups:

- *tar*(1),
- *cpio*(1)
- *dump*(8) and *restore*(8).

All of the above commands are useful for performing backups as they preserve the access permissions, owners and groups of files and even the modification date. In the case of *cpio* the file access date is also remembered and restored.

Of these commands, *tar* and *cpio* are often used for copying data from one machine to another, as they produce a single large file out of several smaller ones, saving information about sub-directories etc.

dump and *restore* are used for performing full and incremental backups of the filesystem, so you will need to become familiar with them and use them to organise a backup schedule for your system.

A collection of sample shell scripts are available on your system that provide you with an easy to use method of performing full and incremental backups of your system. These scripts and how to use them, are described at the end of this chapter.

The following sections discuss each of these commands in turn and show you how to use them.

Using tar

tar (short for *tape archiver*) is the program most often used for copying software and data from one UNIX system to another. It produces one large file out of several files, but the internal headers separating each of the constituent files are composed of text strings in a standard format. These headers are usually insensitive to byte-ordering, word sizes or alignment of the machine on which *tar* is run. Therefore, *tar* archives are said to be 'portable', and versions of it are often available on non-UNIX systems also.

Whilst *tar* files may be portable, any binary files which *tar* copies across will not be, and thus archives of binary files should only be made where the files are to be recovered onto a similar machine.

Archiving files using tar

To archive files onto a floppy disc using *tar*, firstly make sure that you are in the console terminal window so that you can see any error messages that may be displayed during the archiving process.

Before using *tar*, you need to format the floppy disc you are going to use. This is done using the command *ffd*(8). Insert the disc in the disc drive and type:

```
ffd
```

ffd (short for *format floppy disc*) formats the disc by default in the standard ADFS-style D format which stores approximately 800K of data. The device name that corresponds to this format is displayed during formatting. For example:

```
Commencing format of /dev/rfdf1024
Commencing read check
Format completed satisfactorily
```

Once formatted, you can create an archive of any file in the system on floppy disc. For example, to write the contents of the directories *dir1* and *dir2* to floppy disc in *tar* format, as user *root* type:

```
tar cvf /dev/rfdf1024 dir1 dir2
```

The *c* option denotes that an archive is to be created, the *v* option tells *tar* to give a full account of its activities and the *f* option denotes the name of the device onto which the archive is to be made.

It is *most important* not to use 'absolute' pathnames (ie file pathnames starting with '/'), as this will create the corresponding pathnames on the target system to which the files are copied, without giving you any opportunity to divert the files elsewhere.

Pathnames should either be the contents of the current directory, as denoted by '.', or a subdirectory and/or files within the current directory.

Also, pathnames should not be longer than 100 characters (the maximum allowed for in the header), and the directory from or to which files are copied should not have a pathname longer than 50 characters owing to a bug in many versions of *tar*.

To list files you have archived, type:

```
tar tvf /dev/rfdf1024
```

To recover the files in a *tar* archive onto another RISC iX machine, change directory to where you want to put the files, insert the floppy disc and type:

```
tar xvf /dev/rfdf1024
```

The files on the floppy disc will be read into the current directory, provided that the archive does not contain absolute pathnames. If this is the case, then the files will be recovered into the places specified in the pathnames.

To recover the files in a *tar* archive onto a non-Acorn UNIX machine with a 3.5" floppy disc drive, change directory to where you want to put the files, insert the floppy disc and type:

```
tar xvf /dev/fdname
```

where *fdname* is the device name of the floppy disc drive.

For more information, refer to *tar*(1) and *ffd*(8).

Using cpio

Sometimes the program *cpio* (short for *copy in and out*) is used to store archives. This is not available on all UNIX systems, and there are less implementations on non-UNIX systems than with *tar*. The archives are less portable than *tar*, in the sense that the binary headers used to separate the constituent files are sensitive to byte-ordering, word sizes and alignment.

There is a *–c* option which may be used to save text string headers like *tar*, but alas not all versions of *cpio* support it.

On the other hand it is very much easier to recover individual files from a *cpio* archive than from a *tar* archive. With *tar* it is usually easier to recover the whole archive and then to delete unwanted files. (*tar* does have a *–w* 'wait' option that allows selective files to be recovered, but this requires a response from the keyboard for each file.)

Archiving files using cpio

To copy a file onto a floppy disc using *cpio*, firstly format the floppy disc using the command *ffd*(8) as detailed in the previous section. Also, make sure that you are in the console terminal window so that you can see any error messages that may be displayed during the archive.

Although *cpio* can be used in a similar fashion to *tar*, *cpio* archives are usually created using an option to the command *find*(1). As with *tar*, care should be taken to make archives relative to the current directory rather than an absolute directory.

For example, to create a *cpio* archive of the contents of the current directory to a previously formatted floppy disc, type:

```
find . -cpio /dev/rfdf1024
```

To list the files you have archived, type:

```
cpio -itvB </dev/rfdf1024
```

To restore the files you have archived, type:

```
cpio -idmB </dev/rfdf1024
```

To use the character headers *-c* option, the corresponding commands are:

```
find . -ncpio /dev/rfdf1024
cpio -ictvB </dev/rfdf1024
cpio -icdmB </dev/rfdf1024
```

However the `-c` option may be left out of the last two commands, as *cpio* detects and adjusts for the format of the archive.

Archiving selected files

Patterns may additionally be inserted in the save option (ie in *find*) to select which files to archive, thus:

```
find dir1 dir2 dir3 -mtime -5 -cpio /dev/rfdf1024
```

would save files in the specified directories *dir1*, *dir2* and *dir3* that had been modified in the last five days.

Moreover patterns may be inserted in the recover option to select which files are to be recovered thus:

```
cpio -idmB '*acct*' </dev/rfdf1024
```

would recover all files containing the string *acct*.

For more information, refer to *find*(1) and *cpio*(1).

Further processing of tar and cpio files

One of the major drawbacks of using either *tar* or *cpio*, is their inability to cope with archiving files that extend over more than one floppy disc. To circumvent this problem there are a couple of utilities that you can use either singularly or in combination.

Compressing the file using compress

Rather than outputting the archived files immediately to a file or floppy disc, a '–' may be specified to denote the standard output for further processing, in particular to compress the resulting file so that it can fit on one floppy disc.

The command *compress*(1) can be used to compress data such as source code by as much as 60%. For example, to produce a compressed version of the *tar* archive of the directories *dir1* and *dir2* onto a floppy disc, type:

```
tar cvf - dir1 dir2 | compress -v >/dev/rfdf1024
```

The *-v* option used with *compress* will display the amount of compression achieved on the given file.

To recover the archived directories from floppy disc, change directory to where you want to put the files, insert the floppy disc, and type:

```
uncompress < /dev/rfdf1024 | tar xvf -
```

For more information, refer to *compress*(1).

Archiving selected files over multiple floppy discs using dsplit

You can also use the RISC iX utility *dsplit*(8) if the standard output produced from *tar* or *cpio*, even after compression, extends over more than one floppy disc.

dsplit is a general utility that enables you to take a data stream produced by whatever means (usually *tar* or *cpio*) and copy it onto a sequence of floppy discs. *dsplit* is fully interactive, providing prompts to insert new discs and optionally, to format floppy discs as required.

For example, if you have a large directory (*bigdir*) to archive, insert a floppy disc and type:

```
tar cf - bigdir | compress | dsplit -f -t "large dir"
```

where *large dir* is the title given to the inserted floppy disc, as denoted by the *-t* option. The *-f* option by default formats each disc before writing information to the disc.

The above command firstly produces a *tar* format of the directory, which is then piped through the *compress* program. The result of which is written onto several floppy discs, as controlled by *dsplit*.

As the archive proceeds, *dsplit* will prompt you to insert and name new floppy discs, systematically formatting and then writing out the data to floppy discs until the archive is complete.

Note: Make sure that each disc you insert is not write-protected, otherwise *dsplit* will fail and exit with an error message.

To recover the archive made in the above example, insert the first floppy disc and type:

```
dsplit | uncompress | tar xf -
```

`dsplit` will read off the disc the title you gave to the archive (*large dir*) and ask you for confirmation that this is the right disc set before continuing. For example:

```
Disc has title 'large dir'.
Do you want to continue? (Y/N)
```

Type `y` at this prompt. The restore will start and *dsplit* will prompt you to insert the discs in the order in which they were archived, until the restore is complete.

For more information, refer to *dsplit*(8).

Using dump and restore

Unlike *tar* and *cpio*, which are useful for storing specified parts of the directory structure, possibly for transfer to other machines, `dump` and *restore* are used for performing full or incremental backups and recovery of the physical devices that comprise the filesystem.

dump

Using *dump* to perform backups is a very thorough means of ensuring the integrity of the information on your system. However, you will find there are problems when you try to use *dump* with floppy discs as the backup media. For example:

- You will use a very large number of discs and waste a lot of time. A full backup of the entire filesystem takes a long time and uses a lot of floppy discs.
- You have to beware of inserting previously used discs of any sort during a backup. *dump* doesn't complain and will happily write over the disc. Therefore, you must separate the discs which you have used from blank discs by labelling them immediately after you have removed them from the drive.
- Restoring information from a backup is time consuming if you are searching for just one file. You have to restore the whole backup, then selectively pick out the file you need.

Due to the above problems, it is recommended that you only use *dump* for making regular backups of your system if an alternative backup media is attached to your system with a large storage capacity, for example, a hard disc or tape streamer.

If you have to backup your system using floppy discs, it is recommended that you use *dump* once, to backup the entire filesystem with a *level 0* backup. From then on, make backups of directories and files that are regularly changing using *tar* or *cpio* together with *compress* and *dsplit*. Although these programs lack the functionality of *dump* for performing backups, they are much more suited for use with floppy discs.

dump has a system of 'level numbers', from 0 to 9, which may be used to control whether a full or incremental backup is to be performed. A file is saved if it has been modified since the last backup of the same or a lesser level.

When you first receive your system, you need to perform a level 0 backup, which will make a copy of everything on the filesystem. Although this takes a long time and uses a lot of floppy discs, you ensure that you have a backup copy of the original file structure to fall back on should you accidentally delete an important file.

When you use *dump*, the access permissions, owners and groups of files saved are preserved, so you can only use this program if you are *root*.

For example, to perform a level 0 backup of your filesystem, using floppy discs as the backup media, type:

```
dump 0Fu /dev/rsd0a
```

The *0* option indicates that you are doing a level 0 backup, the *F* option that you are dumping to the floppy disc device (*/dev/rfdf1024*) and the *u* option writes a record of the backup to the file */etc/dumpdates*.

A level 0 backup can be taken more efficiently by using *dump* in conjunction with *dsplit* and *compress*. For example:

```
dump 0fu - /dev/rst0a | compress | dsplit -f -t "level 0 backup"
```

There is also a shell script in */var/adm/dump* called *zerodump* which you can use to perform level 0 backups of your system. For more information, refer to the section entitled *Sample backup scripts* on page 72.

For more information about using *dump*, refer to *dump*(8).

restore

To restore files from a set of floppies, use the *restore* command in the following form:

```
cd /
restore rf /dev/rfdf1024
```

If the backup was taken using *compress* and *dsplit*, type:

```
cd /
dsplit | uncompress | restore rf -
```

If the disc has been so badly corrupted that you have to restore a level 0 backup in this way, you will not be able to run *restore*. In such cases you should contact your support person to obtain a new copy of the system files, and thereafter reload your incremental backups in this way.

restore is more commonly used for retrieving individual files that have been accidentally deleted. To recover single files, use *restore* with the *-i* option set. For example:

```
cd /
restore if /dev/rfdf1024
```

After reading in the information from the disc, *restore* goes into interactive mode, giving you a shell-like interface that allows you to move around the directory tree to find and extract the file you are looking for.

If used in conjunction with *dsplit* and *compress*, type:

```
cd /
dsplit | uncompress | restore if -
```

For more information, refer to *restore*(8).

Sample backup scripts

Once you have performed a level 0 backup of your system, you need to do a series of incremental backups to save the changes you make when you start using the system.

If you have a backup media with a large storage capacity, you are more likely to use *dump* for backing up the information on your system, so it is important that you quickly effect a schedule for performing backups.

To help you organise backups for your system, the directory */var/adm/dump* contains some sample shell scripts, that you may wish to use initially if you are unsure about how to perform backups. After a while, you can begin to tailor these scripts to suit the needs of your system.

The name of the files contained in this directory and their uses are as follows:

- *zerodump* – a simple script that performs a level 0 backup of your system.
- *incdump* – a backup schedule that performs daily and weekly backups of your system. The schedule is initially set up to run over the course of six weeks.
- *nextdump* – tells you the next level of backup to be performed and how the backup media should be labelled.
- *config* – a configuration file for the backup scripts contained in this directory.

- *dumpdates* – a text file that is created when the first backup is carried out on your system using these scripts and subsequently records every backup that is performed. This file will be useful to refer to when you need to restore information that you have backed up.
- *dayno* – a text file that is updated by the backup scripts and contains information about the next daily backup to be performed.
- *weekno* – a text file that is updated by the backup scripts and contains information about the next weekly backup to be performed.

The scripts are initially set up to perform full and incremental backups over the course of six weeks of the root filesystem of the internal hard disc (*/dev/sd0a*), using the floppy disc (*/dev/rfdf1024*) as the backup device.

For information on changing this default state, refer to the section entitled *Configuring the backup scripts* on page 78.

The backup schedule

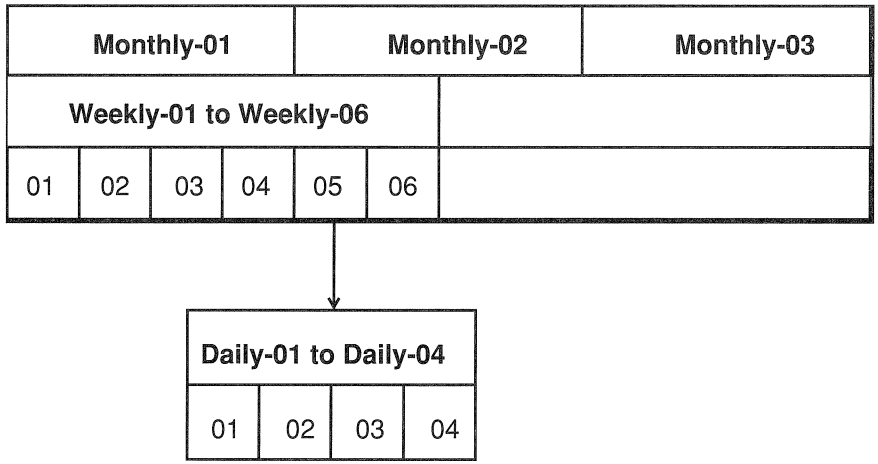
The backup scripts use a backup schedule for your work consisting of incremental backups at various levels according to the 'Tower of Hanoi' backup schedule.

Before using the backup scripts you need to appreciate how this schedule works in order to be able to restore any of the information that you backup and, should the worst happen, be able to restore the entire filesystem using incremental backups.

The schedule begins from the last time a level 0 backup was taken. It is recommended that you take a level 0 backup each month or whenever you make any major changes to the filesystem. A new set of discs should be used for each level 0 backup and labelled from *MONTHLY-01*. These discs should be stored away from your premises and re-used after a reasonable time period. For example, it is recommended that you adopt a three-monthly cycle of level 0 backups (*MONTHLY-01* to *MONTHLY-03*). After three months, you use the *MONTHLY-01* discs again.

The schedule performs daily and weekly incremental backups over the course of a six-week cycle. The weekly backup uses a different set of discs each time labelled *WEEKLY-O1* to *WEEKLY-06*. Different levels of daily backups are taken four times over the course of a week, labelled *DAILY-01* to *DAILY-04*.

The daily sequence restarts after each weekly run and can be up to eight sets, but is usually four. A schematic of the disc sets required, is shown below:



Therefore, ten different sets of discs are required to run the weekly and daily schedules, plus extra sets for each level 0 backup.

At the beginning of the six week cycle, a level 0 backup is taken. Following this, a series of four daily backups from Monday to Thursday are made at various levels from 2 to 5 using disc sets *DAILY-01* to *DAILY-04*. The weekly backup is taken on a Friday, using disc set *WEEKLY-01*. The following week, daily backups are taken as before, using the daily disc sets *DAILY-01* to *DAILY-04* and another weekly backup is taken on Friday, using disc set *WEEKLY-02*.

After another two weeks, a level 0 backup is taken using a fresh set of discs. A further two weeks elapse before the six-week cycle is repeated and the weekly disc sets are re-used.

Although slightly complex in nature this approach has many advantages:

- Daily dumps are kept quite small.
- You always have two recent backup copies of a file that is changing daily.
- Only four tapes are needed at the most to restore any lost information if your system crashes – the most recent MONTHLY tape, followed by the most recent WEEKLY tape and then one, or two, of the DAILY backup tapes, depending upon which day of the week your system crashed.

The levels of backup taken during a six-week cycle are shown below:

Monthly	Daily				Weekly
	Monday DAILY-01	Tuesday DAILY-02	Wednesday DAILY-03	Thursday DAILY-04	Friday WEEKLY
Level 0	Level 3	Level 2	Level 5	Level 4	Level 1 - **01**
	Level 3	Level 2	Level 5	Level 4	Level 1 - **02**
	Level 3	Level 2	Level 5	Level 4	Level 1 - **03**
	Level 3	Level 2	Level 5	Level 4	Level 1 - **04**
Level 0	Level 3	Level 2	Level 5	Level 4	Level 1 - **05**
	Level 3	Level 2	Level 5	Level 4	Level 1 - **06**

For example, if your system crashes on Wednesday of week 2, before the level 5 dump is taken, you can perform an incremental restore of the filesystem using the level 0 backup taken at the beginning of the cycle, followed by the previous weeks' level 1 backup (disc set *WEEKLY-01*) and finally the level 2 backup (disc set *DAILY-02*).

Using nextdump

To help you find out which disc set is needed on a particular day, use the utility *nextdump*. This will display the next backup to be taken in the schedule and also the disc set to be used. For example, at your normal shell prompt, type:

```
cd /var/adm/dump
nextdump
Daily dump required, level 3, use discs DAILY-01
```

This utility is best used by placing it in your crontab file and invoking it at the time of the day you normally do your backups. For more information, refer to the section *Adding the backup scripts to the crontab file*, later on in this chapter.

Using zerodump

To perform a level 0 backup of the filesystem using floppy discs as the backup media, you will need at least 45 formatted discs and a lot of time – a level 0 backup takes about five hours to perform.

To begin a level 0 backup of the filesystem, log in as root in the console window and type:

```
cd /var/adm/dump
zerodump
```

The following messages will be displayed:

```
Zero dump

Enter the disc set name for / (eg: monthly-01) :
```

The disc set name refers to a generic name that you should label on all the discs for this particular backup. For example if you are performing a level 0 backup for the second month of the three-month cycle, the disc set could be labelled *MONTHLY-02*.

Enter a name for the disc set and press ↵. The following messages will be displayed:

```
About to dump / to disc series MONTHLY-02
Press return when the first disc is ready
```

Insert the first backup disc and press ↵. The standard set of messages issued by the *dump* program are displayed, detailing the date of the last level 0 backup and estimating the number of discs needed to perform this new backup.

The backup then begins; information is read from the internal hard disc and copied onto the inserted floppy disc. When the disc becomes full, you will be prompted to insert a new disc. For example:

```
DUMP: Change Tapes: Mount tape #2

Message from the dump program to all operators at 11:33 ...

CHANGE TAPES!

DUMP: NEEDS ATTENTION: New tape mounted and ready?: ("yes" or "no")
```

Do not be put off by *dump* asking for tapes instead of floppy discs. *dump* can now be used with all sorts of backup media. The term 'tape' just refers to any removable magnetic media, such as a floppy disc.

Remove the first backup disc and label it as disc 1 of the set (ie *MONTHLY-01/1*). Insert a new formatted disc (labelled *MONTHLY-01/2*) and type *yes* followed by ↵. The backup proceeds until the disc is full when you will again be prompted to insert a fresh disc.

Continue inserting discs in this way until the backup is complete. This will be indicated by the following messages being displayed:

```
Message from the dump program to all operators at 16:42 ...
DUMP IS DONE!

zerodump: completed

#
```

Using incdump

To run *incdump* from the command line, log in as *root* in the console window and type:

```
incdump
```

The following messages will be displayed:

```
Incremental dump

About to dump / to disc series DISC-SET
Press return when the first disc is ready
```

Where *DISC-SET* refers to the disc set to be used, as specified by *nextdump*. For example, *DAILY-04*.

Insert the first backup disc with the correct label and press ↵. The standard set of messages issued by the *dump* program are displayed, detailing the date and level of this backup along with the date of the last backup made.

The backup then begins; information is read from the internal hard disc and copied onto the inserted floppy disc. When the disc becomes full, you will be prompted to insert a new disc.

For example:

```
DUMP: Change Tapes: Mount tape #2

Message from the dump program to all operators at 11:33 ...

CHANGE TAPES!

DUMP: NEEDS ATTENTION: New tape mounted and ready?: ("yes" or "no")
```

Remove the first backup disc and label it as disc 1 of the disc set. Insert a new formatted disc and type *yes* followed by ↵. The backup proceeds until the disc is full when you will again be prompted to insert a fresh disc.

Continue inserting discs in this way until the backup is completed. This will be indicated by the following messages being displayed:

```
Message from the dump program to all operators at 11:42 ...
DUMP IS DONE!
incdump: completed

#
```

Interrupting a backup

If you wish to interrupt a backup at any time, you can do so by pressing <Ctrl-C>. This will be interpreted by the *dump* program as a request to abort the backup, and you will be asked if you wish to continue. For example:

```
^C DUMP: Interrupt received.
DUMP: NEEDS ATTENTION: Do you want to abort dump?: ("yes" or "no")
```

Typing *yes* to this prompt, aborts the program, as indicated by the following message:

```
DUMP: The ENTIRE dump is aborted.
```

Following this, you are returned back to your normal login prompt.

If you abort a backup in this way, the whole backup is invalidated and you will need to start the backup from scratch.

Configuring the backup scripts

The *config* file allows you to change certain parameters that are used by the backup scripts. The parameters and their default values are listed below:

- *dumplist* (/) – an ordered list of the filesystems to be backed up.
- *maxweekno* (6) – the number of weekly discs in the backup cycle.

dumplist

Initially, the root filesystem (/) of the internal hard disc is backed up. If you attach an additional hard disc to your system and mount it on the directory */b* and decide that this directory will contain most of the information that will be regularly changing on your system, you can change *dumplist* to this parameter:

```
set dumplist = (/b)
```

In future, the backup scripts will backup the directory */b*. If you expand your system further, by attaching another hard disc (mounted on */c*) that will also have files changing regularly you can add this to *dumplist*. For example:

```
set dumplist = (/b /c)
```

The backup scripts will now backup both of the above filesystems, starting with */b*.

maxweekno

The *maxweekno* parameter controls the number of weekly discs in the backup cycle. This is initially set to six weeks., but you can change this to a longer cycle or a shorter cycle depending on the backup schedule you adopt for your system.

Adding the backup scripts to the crontab file

The backup scripts *incdump* and *nextdump* can most easily be used by placing an entry for them in the *crontab* file for *root* located in the directory */var/spool/cron/crontabs*. A typical entry appears below:

```
# Here are a few sample entries to try out the backup scripts in /var/adm/dump
#
# Run nextdump at 5pm each weekday to warn about the impending dump for that evening
0 17 * * 1-5 /usr/bin/csh /var/adm/dump/nextdump 2>&1 | /usr/ucb/mail -s "impending
dump" root
#
# Run incdump at 9pm each weekday to do various levels of backups
#
0 21 * * 1-5 /usr/bin/csh /var/adm/dump/incdump 2>&1 | /usr/ucb/mail -s "record of
the backup" root
#
```

At 5pm each weekday evening, *nextdump* is run and the output produced is mailed to *root*.

At 9pm each weekday evening, *incdump* is run. The status messages produced plus any error messages are sent as mail to *root*.

Note, it is really only suitable to have an entry for *incdump* in the *crontab* file if you have a large capacity backup media. If you are using floppy discs as your backup media, it is very unlikely that an incremental backup will fit on one floppy disc. Therefore, you will have to perform *incdump* manually.

If you have access to a cartridge tape drive and you wish to use this as your backup media then you will need to alter the options used by the *dump* command in the two shell scripts */var/adm/dump/incdump* and */var/adm/dump/zerodump*. For more information, refer to *dump*(8).

Adding and removing users

Introduction

This chapter describes how to add and remove users on your system. As with most System Administration tasks, there is more than one way of performing these tasks.

- *useradmin* and *groupadmin* – two utilities written especially for RISC iX that allow you to administer the password file (for users) and the group file (for groups).
- *vipw* – the conventional utility for editing the password file. This is more suited for experienced System Administrators and should be used with care.

For information on the implications of creating new users on systems in a secure network environment, refer to the section entitled *Making the network more secure* on page 205.

Using useradmin

The program *useradmin* is provided to create and remove users from the password file (*/etc/passwd*), executing all the appropriate steps in the right order and displaying suitable error messages when incorrect values are given.

Creating a new user

To create a new user on your system, log in as *root* and type:

```
useradmin
```

A graphical representation of a copy of the password file is displayed on your screen.

To add a new user, press *u*. The cursor drops to the bottom of the screen and you are prompted to type in a user name for the new user:

```
User name:
```

Enter a valid user name and press ↵. The user name should be one word with no spaces or gaps and consist only of lower-case letters. If you choose a user name that already exists, the original prompt is displayed again.

With a valid user name entered, you will then be prompted to insert the full name of the new user:

```
Full user name:
```

Enter the full name of the new user and press ↵. You will then be prompted to insert a user id:

```
User id:
```

If you are unsure which number to enter, press the space-bar; *useradmin* will display a suitable value that you can use (the first unused number greater than or equal to 100). Repeated pressing of the space-bar increments the number by one. To return to the original value suggested, press <Del> followed by the space-bar. To select a user id, press ↵.

With a user id selected, you will then be prompted to insert a group name:

```
Group name:
```

Likewise, if you are unsure which group to enter, press the space-bar; *useradmin* will suggest a group name. To cycle round all the valid group names, keep pressing the space-bar. To return to the original value suggested, use the <Del> key to delete the entire word and press the space-bar again. To select a user id, press ↵.

With a group name entered, you will then be prompted to insert the location of the home directory for the new user:

```
Directory:
```

If you are unsure where on the filesystem the new user should be placed, press the space-bar; *useradmin* will select the default home directory location, ie */home/user_name*.

You will then be prompted to enter the type of login shell used:

```
Shell:
```

The choices for this field are either */usr/bin/sh* or */usr/bin/csh*. Pressing the space-bar selects the default shell */usr/bin/sh*.

Finally, you are prompted to enter an initial password:

```
Initial password:
```

Enter a password that is easy to remember.

After entering the password you are then prompted to re-enter the password:

```
Reenter initial password:
```

Type the password in again. *useradmin* checks that the two passwords are the same. If they differ, you will be prompted to type the password in again.

Following a successful check, *useradmin* writes all the information you have entered to the password file displayed on your screen. The home directory specified for the new user is also created in the filesystem.

Scroll down through the display using <Ctrl-F> to move forwards and <Ctrl-B> to move backwards, until the new entry appears and check that each field in the entry is correct.

If you are satisfied with the entry, type *q* to quit from the program. If not, refer to the next section that shows you how to delete an entry in the password file and start again.

After typing *q*, the changes made are written out to the password file. The user is now created on the system.

If at any time you wish to exit from adding a new user, press <Esc>. This will discard everything you have entered and take you back to the original display of the password file.

To abandon *useradmin* altogether, without writing out the changes to the password file, press <Ctrl-C>. Note, however that any new user directories created while you were using *useradmin*, will not be deleted. You will have to delete these directories manually.

Removing an existing user

An existing user can also be removed from your system using *useradmin*. This will remove their entry in the password file and also provide you with an option to remove all the files and directories in their home directory.

To remove an existing user from your system, at your normal shell prompt type:

```
useradmin
```

A graphical representation of a copy of the password file is displayed on your screen.

Find the name of the user you wish to delete by paging through the password file using <Ctrl-F> to move forwards and <Ctrl-B> to move backwards, until the user name appears on the screen.

Move the cursor alongside the user name (using *j* and *k* to move down or up), and type *d*. You will be asked to confirm your selection:

```
Are you sure you want to delete user 'username' (Full name)
```

Type y to confirm your choice. If you are trying to delete a system user (user id below 100), you will be asked for further confirmation before proceeding:

```
System user - Do you really want to proceed?
```

Type y again at this prompt.

You are then asked if you wish to delete the home directory of the named user:

```
Do you wish to delete user files?
```

If you are sure that the files in the home directory of the user are of no use to you or anyone else that uses your system, type y. If you type *n* only the password entry for the user is deleted.

If you type y, the home directory to be deleted is displayed and you are asked to confirm your choice:

```
OK to proceed?
```

Confirm your choice by again typing y. The password entry and the home directory of the user are deleted.

Type *q* to quit from the program; the changes made are written out to the password file.

If at any time you wish to exit from deleting a user, press <Ctrl-C>. This will abandon the *useradmin* program altogether and take you back to your normal shell prompt, without affecting the password file.

Editing the password file

useradmin allows you to change two of the fields in the password file for a user that has already been created. The two fields are:

- the *Full user name* field
- the *Shell* field

Changing the 'Full user name' field

To change the *Full user name* field, position the cursor on the line that contains the field you wish to change and type *n*. You will be prompted to enter a new name:

```
New name for user 'username', currently "Full name":
```

Type in the new name and press ↵. The old name is replaced by the new name in the display.

Type *q* to quit from the program and save your changes.

Changing the 'Shell' field

To change the *Shell* field, position the cursor on the line that contains the field you wish to change and type *s*. You will be prompted to enter a new login shell for the user:

```
New shell for user 'username', currently "/usr/bin/<shell>":
```

Type in the new shell (either */usr/bin/sh* or */usr/bin/csh*) and press ↵. The old login shell is replaced by the new login shell in the display.

Type *q* to quit from the program and save your changes.

If at any time you wish to exit from editing the password file to change the user name or login shell, press <Esc>. This will discard everything you have entered and take you back to the original display.

Using groupadmin

The program *groupadmin* is provided to create and remove groups of users from the group file (*/etc/group*), in a similar fashion to *useradmin*.

Only occasionally will you need to edit the group file. For example, to add a new group or possibly to add a user to a group additional to the standard group ids used.

To invoke *groupadmin*, either

- Type:
  `groupadmin`
- Alternatively, if you are using *useradmin*, just type *g*. (The two utilities *useradmin* and *groupadmin* use the same program so you can switch freely from editing the password file to or from editing the group file. The name just determines the initial screen which you start with. To return to the password file, just type *u*.)

Either action produces a graphical representation of a copy of the master group file (*/etc/group*).

Creating a new group

To create a new group, type *g*. The cursor drops to the bottom half of the screen and you are prompted to type in the group name:

```
Group name:
```

Enter the name and press ↵. The name you enter should be one word with no spaces or gaps and consist only of lower-case letters. If you choose a group name that already exists, the original prompt is displayed again.

With a valid group name entered, you will then be prompted to insert a group id (*Gid*) number:

```
Gid:
```

If you are unsure which number to enter, press the space-bar; suitable values that you can use will be displayed, starting with the first unused number greater than or equal to 100. Repeated pressing of the space-bar increments the number by one. To return to the original value suggested, press <Del> followed by the space-bar.

If you choose a group id that already exists, the original prompt is displayed again.

With a valid group id entered, press ↵. The new entry will be added to the group file.

Type *q* to quit from the program and save your changes.

Editing the group file

The group file can be edited to:

- add new member(s) to a group
- set-up a new password or change an existing password for a group

Adding new members to a group

To add a new member to an existing group, position the cursor on the line that contains the group you wish to add to and type `m`. You will be prompted to enter the members of this group:

```
Members of 'groupname':
```

Type in the names of the existing members of this group plus any new members you wish to add, and press ↵. The old list of members is replaced by the new list.

If you type in the names of any unknown users, an error message is issued and the original prompt is displayed again.

Type *q* to quit from the program and save your changes.

Setting a password for a group

To set a password for an existing group, position the cursor on the line that contains the group you wish to set a password for and type *p*. You will be prompted to enter a password for this group:

```
Enter new group password:
```

Type in the password and press ↵.

You will then be prompted to re-enter the password:

```
Reenter new group password:
```

Type the password in again. The program checks that the two passwords are the same. If they differ, you will be prompted to type the password in again.

Following a successful check, the password is processed for that group.

Type *q* to quit from the program and save your changes.

For more information, refer to *useradmin*(8).

Using vipw

The program *vipw* forms only the first stage in the process of either creating a new user or removing a user. The remaining steps for each process need to be carried out by hand.

Creating a new user

To create a new user on your system using *vipw*, the following steps are taken:

- The password file is edited.
- The password database is updated to include details of the new user.
- If necessary, the group file is edited.
- A home directory is created for the new user, and set to be owned by the new user.
- A password is assigned to the new user.

The next few sections describe each of the above steps:

Editing the password file

To edit the password file, and also ensure that no-one else is changing it under your feet, type:

```
vipw
```

this invokes the editor defined by the environment variable $EDITOR (default is *vi*) and reads in a copy of the password file (*/etc/passwd*) called */etc/ptmp*.

The password file is also 'locked' to prevent anyone else from editing it at the same time. If someone else tries to use *vipw* to edit the password file while you are using *vipw*, the following message is displayed:

```
vipw: password file busy
```

If you get this message when you know for certain that no-one is editing the password file, it is possible that the system crashed while the password file was being edited using *vipw* and the file */etc/ptmp* was left lying around. In this situation, delete the file */etc/ptmp* and try again.

If *vipw* is successfully invoked, the password file (*/etc/passwd*) is displayed consisting of separate lines for each user, of the form:

```
user:password:user_id:group_id:name:directory:shell
```

For example:

```
psmith::101:100:Paul Smith:/home/paul:/bin/csh
```

The fields in these lines have the following meanings:

- *user* – is the character login name of the user. It is an error for two or more entries to appear in the password file with the same name. The user name is normally up to ten lower-case letters. It is a good idea to have a consistent scheme for devising user names on any one system, as this enables the author of outgoing external mail to be recognised and helps people on external machines who want to send mail to guess what a particular user's user name might be.

  Possibilities are: the user's initials, surname preceded by first initial, or some other similar variation.

- *password* – is either empty, to indicate that the user does not have a password (this is not recommended), or contains a '*' to disable logins on that account, or contains a string of characters inserted automatically by the *passwd*(1) program. The characters displayed represent an encrypted version of the password.

- *User_id* – is the numeric user id of the user. The user id should always be at least 100 for non-system user names, (system users are the standard anonymous owners of program and system files such as *daemon*, *operator*, *uucp* etc).

  User ids should normally be assigned consecutively (to make the password file intelligible and to avoid mistakes).

- *Group_id* – is the group id of which the user is by default a member. This should normally correspond to a group id in the */etc/group* file. Users can belong to more than one group in order for them to have access to files owned by different groups and also to have special system privileges. For example, adding a user to group wheel, enables that user to *su* to user *root*.

  As with user ids, non-system group ids should usually be at least 100. To execute the *su*(1) command, the user has to be in the group *wheel* (0).

- *Full user name* – this field (sometimes referred to as the *gecos-field*) is usually used to hold the full name of the user. An *&* character may be used to denote that the user name should be substituted with capitalisation. For example, if user is *fred*, *& Bloggs* as a name field is interpreted to mean *Fred Bloggs*.

- *Directory* – this designates the home directory of the user. The home directory is normally owned by the user in question. This directory should exist before the user logs in.
- *Shell* – this field gives the full pathname of the login shell. Standard path names are */usr/bin/sh* and */usr/bin/csh* for the Bourne shell and C shell respectively. If left blank, then the Bourne Shell is assumed.

  If you wish you can enter the name of a command to be executed here. For example, the user *halt* runs the command */sbin/halt* instead of logging in to a shell.

Using the above information about each field, add in a line for the new user. It is usually easiest to copy an existing line and then adjust the fields.

When you have entered the line for the new user, exit from the editor in the normal way. *vipw* automatically copies the changes made, to the password file */etc/passwd*.

Updating the password database

As well as updating the password file. *vipw* also automatically updates the password database files */etc/passwd.pag* and */etc/passwd.dir*. These two files are used by various standard utilities such as *ls* for rapid access to the information contained in the password file.

To update these two files, *vipw* deletes the existing files, then invokes the program */usr/sbin/mkpasswd* which creates the two files again from the new password file.

For more information, refer to *vipw*(8).

To update the password database by hand, you could type the following commands:

```
rm -f /etc/passwd.pag /etc/passwd.dir
/usr/sbin/mkpasswd /etc/passwd
```

If you edit the password file without using *vipw* you must always run *mkpasswd* after you have edited the file, to keep the password database file up to date.

Editing the group file

The group file, */etc/group*, contains lines of the form:

```
group name:group password:groupid:user list
```

for example:

```
wheel:*:0:root
daemon:*:1:daemon
kmem:*:2:root
```

```
bin:*:3:root
tty:*:4:root
operator:*:5:root
cmos:*:6:root
staff:*:10:root
guest:*:9999:guest
+:
```

This means that when a user logs in they are placed in the group corresponding to their group id entry in the password file (*/etc/passwd*) and also any groups in the group file (*/etc/group*) that contain their user name.

The */etc/group* file has several default groups. The group *staff*, for instance, is the default group to which all local users belong. For more information on groups, refer to the *group*(5) and *chgrp*(1) manual pages.

To add any members to this file just edit it and save the changes, using a normal editor. Group passwords are normally not used, so the password field in the file is set to '*'.

Creating the home directory

To create a home directory for a user called *paul*, type:

```
mkdir /home/paul
```

As the directory *paul* is contained in a directory owned by you (*/home*), it is also owned by you. Therefore, you also need to change the ownership of this directory to reflect the owner and group given in the password file.

To change the ownership of the home directory to be owned by user *paul*, type:

```
chown paul /home/paul
```

To change the group ownership of the directory, type:

```
chgrp users /home/paul
```

The above two commands could also be combined:

```
chown paul.users /home/paul
```

Following this, any files created within this home directory will be owned by the user but will still have the group id of the directory and not the group id of the user.

For more information, refer to *chown*(2) and *chgrp*(1).

Setting a password

It is desirable that every user should have a password to avoid unauthorised access to the machine. To set a password for user *paul*, use the command *passwd*. For example:

```
passwd paul
New passwd:
```

Type in the password for the user. You will then be prompted to re-enter the password:

```
Retype new password:
```

Type the password in again. A check is run to see if the two passwords are the same. If they are, the password is set. If the passwords differ, an error message is displayed and the password is not set.

For more information, refer to *passwd*(1).

The new user is now on the system.

Removing an existing user

An existing user can be removed from your system by firstly deleting any files or directories owned by that user and then deleting the entry for this user in the password file, and any entries in the group file.

To find and delete all files and directories owned by a particular user, type:

```
cd /
find . -user username -exec rm -fr {} \;
```

Alternatively you may prefer to keep the files but change the ownership on them to be owned by another user. To do this type:

```
cd /
find . -user username -exec chown newusername {} \;
```

Following this, delete any references to the specified user in the password file using *vipw*. Also, edit the group file to remove any references to the user.

Setting up peripheral devices

Introduction

This chapter describes how to configure your workstation so that it knows about and can communicate with peripheral devices that have been attached to it.

There are two main stages involved in attaching any type of peripheral device to your system:

- **Hardware connection** – physically connecting the peripheral device to your system by attaching a cable from one of the ports on your system to a port on the peripheral device, and ensuring that the correct signals are being sent through this cable so that data can be transmitted.
- **Software connection** – altering existing system configuration files and creating new configuration files so that your system knows about, and can communicate with, the peripheral device.

This chapter assumes that you have successfully completed the first stage, following instructions contained in the *Installation Guide* and the documentation accompanying your peripheral device.

The devices that can be attached to your system include:

- terminals
- printers
- modems
- SCSI peripherals (if you have a SCSI expansion card installed on your system).

This chapter describes how to adjust the software on your workstation in order to describe the peripheral device that is attached to it. Where possible, examples are given for configuring your workstation for specific peripheral devices, but be aware that there will be times when only general explanations can be given and it will be up to you to determine the exact action required. The documentation supplied with your peripheral device should help you in this respect.

Available ports

There are two or (optionally) three ports available on your system that are used to attach peripheral devices:

- Serial port – device name */dev/serial*

  In RISC iX each serial line you use is configured either as a login line or as an outgoing port, but never simultaneously both. If you do need to switch between these modes then the solution is to have the line normally in login mode but temporarily suspend the login prior to outgoing use and then restore it when you finish.

  Unfortunately there is no standard utility to do this; if you really must have both way use of the line you will have to write one yourself or obtain and adapt one from elsewhere. Another problem may be that unless the equipment is highly configurable, you may need different cables for login and outgoing use.
- Parallel port – device name */dev/lp*
- SCSI port (only available if your system has a SCSI expansion card installed)

Terminals

Attaching an extra terminal to your system is a relatively painless task that greatly enhances the capability of your workstation, as it enables an extra person to use the system at any one time. A wide variety of different terminals can be connected to your system using the serial line port (*/dev/serial*).

Hardware connection

Follow the instructions in the *Installation Guide* and the documentation supplied with your terminal to physically connect your terminal to your workstation. An important point to watch is that the terminal monitors the status of the lines DSR (data set ready) and DCD (data carrier detect). It also raises and lowers DTR (data terminal ready) appropriately. In many cases, where 'modem control' is not required, which is the case with most terminals, it is appropriate to strap all these lines together on the plug into the system, and thus only use the three lines of transmitted data, received data and common return.

If the terminal does raise the DTR signal, then this can be connected to the DCD line. The advantage of doing this is that if the terminal is disconnected, then the lowering of DCD will be noticed by the driver and a hang-up signal will be sent to all processes run from the terminal. This provides additional security by preventing unauthorised users from obtaining access by plugging a different terminal into the line whilst a session is in progress.

Both the computer and a terminal are described as DTEs (Data Terminal Equipment). This means that in order to interconnect them, certain connections need to be transposed in the cable. Thus both send data on the line 'Transmitted Data' and receive data on the 'Received Data' line, which means that the transmitted data line at one end should be connected to the received data line at the other.

Likewise the line CTS, 'Clear to Send' is paired with RTS 'Ready to Send', and DTR with DCD. A cable with connections thus paired is often referred to as a *null modem cable*, but the number of connections thus paired varies, and often only the three transmitted data, received data and common return connections are passed through the cable with straps at each end.

After checking the cable, you must also check that the configuration of the terminal as detailed in the manual supplied with your terminal, agrees with the entry for this terminal type as defined in */etc/gettytab*. The usual configuration is 9,600 baud, eight bits with no parity, one start bit and one stop bit.

Software connection

Assuming that you have connected your terminal successfully to the serial port you now need to set up your system to recognise the new terminal.

The first step in this stage involves editing the file */etc/ttys* to include a line describing the terminal. After you have done this, you will have to make *init* (the process which has overall control of terminal access), notice the new line.

The easiest way to do this is to type:

```
kill -HUP 1
```

init creates a process called *getty* for each serial line and virtual terminal that it reads from the */etc/ttys* file and sets up the environment for each terminal so that you can log in. It then outputs the *login:* prompt, then listens out for a reply.

When it receives a reply, it firstly reads the entry in *gettytab*, describing that terminal line and then invokes the program *login* to read in and check the password entry and group entry for the new user. If there is a valid entry for this user, the user's environment is set-up and a shell is created to interpret commands.

When this process exits, *init* notices that the terminal is now unused, and sends the *SIGHUP* signal to all processes that are attached to the terminal line and starts a new getty for the terminal line.

The terminals on which *getty* is activated, permitting logins, are listed in the file */etc/ttys*. Lines in this file take the following form:

```
console  "/usr/sbin/getty std.avc"  avc  on  secure
```

The first field, in this case *console*, denotes the device name corresponding to the terminal, ie */dev/console*.

The second field (*/etc/getty std.avc*) denotes the command which is to be executed on the terminal line. In the above example, a default *getty* command is issued on the terminal line. This is usually the command invoked on all terminal lines, however sometimes other commands may be invoked relating to window servers. For example, if you have the X Window System installed on your workstation, the command *xterm* (the terminal emulator for the X Window System) can be invoked on the line *ttyvf*. For more information about invoking the X Window System refer to the *X Window System Administrator's Guide*.

The third field denotes the terminal type. In the above example *avc* refers to the *Acorn Video Console*. A full list of the terminals that are supported by your system are detailed in the file */etc/termcap*.

The fourth field (*on*) denotes that the line is active – changing this to *off* will mean that no *getty* and hence no logins will be possible on this terminal.

The fifth field (*secure*), denotes that it is *root* log ins are allowed. To disable this feature, delete the word *secure* from the line. Following this, anyone trying to log in as *root* on this line will be denied access.

Using this information, a suitable entry for a new *vt100* terminal, connected to the serial port of your system, would be:

```
serial        "/usr/sbin/getty std.avc"  vt100
```

For more information about setting up terminals, refer to *getty*(8), *gettytab*(5), *init* (8), *login*(1) and *ttys*(5).

Printers

Printers can be connected to your system via:

- the serial port – device name */dev/serial*
- the parallel port – device name */dev/lp*

Connection of printers to each of these two ports is described below along with the printer database description file */etc/printcap*.

Hardware connection

Follow the instructions in the *Installation Guide* and the documentation supplied with the printer to physically connect the printer to your workstation.

Printers on the serial port

If the printer is to be attached to the serial line then care should be taken to ensure that there is no mention of the serial line in */etc/gettytab*, or this is commented out first. This is to prevent *getty* trying to 'log' the printer in.

The user should beware of the fact that each time the serial line is opened with no other processes using the serial line, the baud rate and other parameters are reset to a standard setting, which includes 9,600 baud. This usually precludes (for example) the user sending successive shell commands with output to the serial line if the default parameters are unsuitable, as the serial line will be closed after each command.

To overcome this, a common method is to have a background task running holding the serial line open but without performing any I/O on it, thus:

```
sleep 10000 < /dev/serial &
```

and then to set the baud rate and other parameters thus:

```
stty 1200 > /dev/serial
```

Thereafter, if a command opens the serial line and resets the parameters, for example the command *stty* is run, the parameters will not be reset if the line is closed (by *stty* terminating) and re-opened, as the serial line has been held open.

Most printer spoolers assume nothing about the serial line and reset the parameters each time they operate, and in the long run this approach should be used in every case, as it is possible for the 'sleep' time to elapse or the process to otherwise terminate.

See the manual page for *stty*(1) for details of how to change modes, control characters etc on the terminal.

Printers on the parallel port

Connecting printers via the parallel port is much simpler, as the communications protocols are already pre-defined. All you have to do is make your system aware that it is attached to a printer.

The way you do this for printers connected to the parallel port or serial port is outlined in the next section.

Software connection

Firstly you need to create a new entry for your printer in */dev* using *mknod*(8); setting the appropriate minor device number to define the characteristics of your printer – for more information refer to *lp*(4). Then, edit the */etc/printcap* file to describe your new printer.

No matter what sort of printer you are connecting up, note that large printer buffers are a disadvantage when running the print spooler. This is because the printer becomes so far behind the workstation that the workstation has long forgotten the print job before the printer catches up, which means that you can lose track of what you have printed if the paper is fed into the printer incorrectly. If possible, the printer should be chosen to have a small buffer.

In order to run the print spooler, the file */etc/printcap* may need to be edited.

Editing the /etc/printcap file

The file */etc/printcap* is used to set up details of the printer for the print spooler. The contents include details of various filters, carriage width etc which should be appropriately adjusted.

A typical printcap entry that is suitable for connecting an Apple LaserWriter to your RISC iX workstation is shown below. (This entry is included in the */etc/printcap* file on your system, initially commented out.):

```
# Serial PostScript printer
lp|serial|PostScript|local serial printer:\
        :lp=/dev/serial:br#9600:sd=/usr/spool/lpd:\
        :fc#00374:fs#00003:xc#0:xs#0040040:mx#0:if=/usr/lib/lpf/psfilter:\
        :lf=/usr/adm/lpd-errs:pw#86:sh:sb:sf:pl#72:
```

Let's look at each of the above lines in turn and see what they are describing:

- *lp | serial | PostScript | local serial printer* – refers to the various names that the printer is known by.
- *lp=/dev/serial* – specifies the device name to which all data will be sent. */dev/serial* specifies the serial port. If the printer is connected to the parallel port this would be */dev/lp*.
- *br#9600* – sets the baud rate for the serial line.
- *sd=/usr/spool/lpd* – specifies the name of the spooling directory. This already exists on the standard distribution.
- the next four entries – *fc*, *fs*, *xc* and *xs*, control the operation of the serial line driver code.
- *mx#0* – sets maximum file size, in blocks. When set to `0`, the file size is unlimited.
- *if=/usr/lib/lpf/psfilter* – sets the name of the printer text filter.
- *lf=/usr/adm/lpd-errs* – sets the name of the file where errors are to be logged.
- *pw#86* – sets the page width, in characters.
- *sh* – suppresses printing of burst page header.

- *sb* – outputs a short banner when printing (one line only).
- *sf* – suppresses form feeds.
- *pl#72* – sets the page length, in lines.

For more details, see *printcap*(5).

The spool directory (*/usr/spool/lpd*) and the device file (*/dev/serial*) should be accessible to the user *daemon*, either by setting the modes to *777* and *666* respectively, or by making *daemon* the owner of the directory and giving the files access modes of at least *700* and *600* respectively.

Printer filters

The directory */usr/lib/lpf* on your system contains the printer filter *psfilter* which is used in connecting your RISC iX workstation to an Apple LaserWriter, as described in the previous example.

The filter takes an input file and converts it into PostScript in Courier font suitable for printing on a LaserWriter. If the first two characters of the input file are `%!`, then the file is assumed to be a PostScript file produced from a pre-processor such as *psroff*(1) and is sent directly to the printer without any processing.

The source of this filter (a C program called *psfilter.c*) is also provided in */usr/lib/lpf*. You can use this program as a starting point for writing a printer filter for use with another type of printer you wish to connect to your system.

Modems

Follow the instructions in the *Installation Guide* and the documentation supplied with the modem to physically connect the modem to your workstation.

Hardware connection

Note that for login use, the RISC iX workstation keeps DTR and RTS on but won't respond until it sees DCD from the modem. If DCD falls (due to a line problem or the remote user hanging up) any processes still attached to */dev/serial* are killed. This is the recognised way to handle DCD, and a login modem should be configured to supply DCD only when there is a carrier.

For outgoing use, DTR and RTS are raised when the line is opened and dropped when the line is closed. Note however they are both left raised by RISC OS at boot time and won't drop until the line is opened then closed. Unless DCD is immediately reflected back, communications programs like *uucico* and *tip* will fail, so unless your modem can be configured to always supply DCD, then leave out the DCD wire and instead strap DCD to DTR at the RISC iX workstation end. Irrespective of the DCD setting, an outgoing modem should be configured to drop any call in progress if DTR falls.

Another point to watch with modems is that some of them have an option to set a character or character sequence which causes the line to be hung up. If at all possible this option should be disabled, as *uucp* and other communications software which passes data in binary will sooner or later send this sequence, and it will be very hard to detect why the line is continually being dropped. (Hayes compatible modems have such a sequence, but it is only recognised if no data has been passed for a given time, which should be set to at least one second.)

Software connection

Programs that use modems to communicate, like *uucp* (short for *unix to unix copy*) and *tip* (short for *terminal interface processor*), will only work if there is no *getty* (login) enabled on the serial line. So you must check that there is no mention of the serial line in */etc/gettytab*.

For more information about setting up UUCP, refer to the chapter entitled *Setting up UUCP* on page 305.

SCSI peripherals

There are a number of SCSI peripherals that can be attached to your workstation. The SCSI device driver in the kernel contains support for the following types of *direct access* devices and *sequential access* devices:

- magnetic and optical disc drives (direct access devices)
- magnetic cartridge tapes and EXABYTES (sequential access devices)

For more information on how to attach any other type of SCSI peripheral to your workstation, such as CD-ROM drives, refer to the chapter entitled *Programming SCSI devices* in the *RISC iX Programmer's Reference Manual, Volume 1 – Kernel and Languages*.

Hardware connection

Connecting SCSI peripherals is a simple process, as the communications protocols are already pre-defined. All you have to do is make your workstation aware that a new device is attached to it.

For information about how to physically connect a SCSI device to your workstation, refer to the *Installation Guide* and the documentation supplied with your SCSI device.

Software connection

This section provides a lot of general information about how to set up a SCSI device on your system. Where specific examples are necessary, a CONNER CP3100 hard disc drive will be used as an example of a typical SCSI device.

The SCSI data bus uses a *logical slot* naming scheme. The actual expansion card slots are searched in the order 0, 1, 2 and 3. The first SCSI expansion card found has logical slot 0, the next SCSI expansion card found has logical slot 1 and so on. References in the rest of this section to slot 0, refer to logical slot 0.

This scheme prevents any problems occurring through incorrectly placed SCSI expansion cards. However, problems may arise should the SCSI expansion card subsequently be removed. The next time a call is made to the device supported by the removed SCSI expansion card, the card referred to will be the next SCSI expansion card found in any slot, which may be supporting an entirely different SCSI device.

Also note that the machine can only be booted into RISC iX from a SCSI disc that is attached to a SCSI expansion card in logical slot 0. This is because when booting from RISC OS only a number ranging from 0 to 7 can be used to specify the boot device. For more information, refer to the *Boot command in the chapter entitled *Using the RISCiXFS module* on page 283.

The stages that are required to set up a SCSI device on your system are as follows. Note that some of these stages may be irrelevant depending upon the type of SCSI device you are attaching to your system:

- Log in as *root*
- Create suitable entries in the */dev* directory for the device using the command *mknod*.
- Format and verify the device
- Section the device to define a RISC OS section and a RISC iX section
- Partition the RISC iX section of the device
- Add a suitable entry in the */etc/disktab* file to enable filesystems to be created on selected partitions of the device
- Make a new filesystem on selected partitions of the device using the command *newfs*.
- Add a suitable entry in the */etc/fstab* file for mounting selected partitions of the device onto your system.

Each of these tasks are described in the following few sections.

Creating suitable entries in the /dev directory

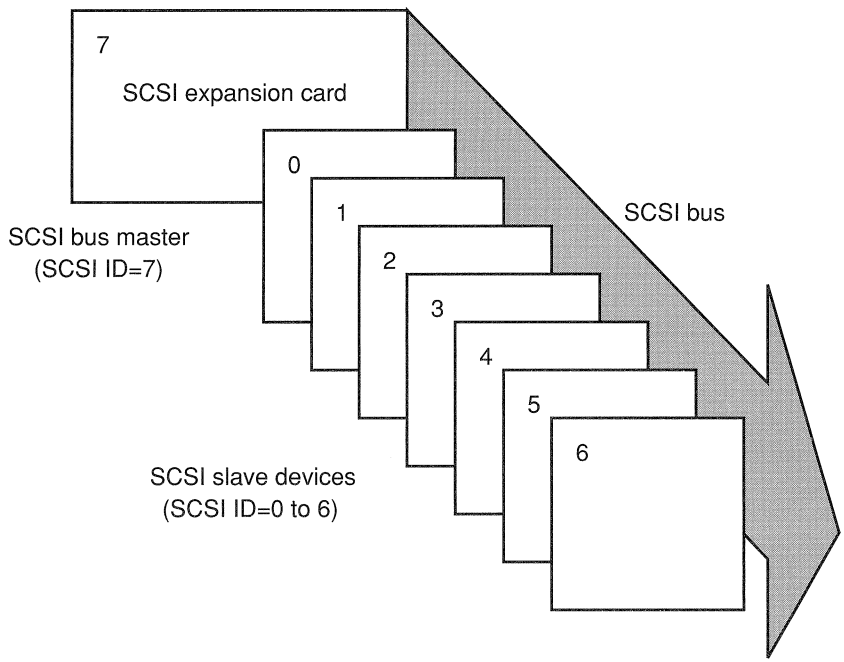
When a new SCSI device has been attached to your system the first requirement is to inform the kernel where the new device is to be found, so that the device can be accessed by the SCSI device driver in the kernel. This entails using the command *mknod* to create suitable entries in the */dev* directory for the device.

The syntax for the command *mknod* is as follows:

mknod *device-name device-type major-number minor-number*

The meaning of each of the above fields is described below.

- *device-name* – a total of eight devices are supported by the SCSI bus. The SCSI expansion card located at the back of your computer unit is designated as the SCSI target device number 7 to which can be attached seven SCSI slave devices ranging from 0 to 6. For example:



To avoid confusion, Acorn recommends the adoption of systematic names for SCSI devices, using *sd* for SCSI disc devices and *ct* for SCSI cartridge tape devices, numbering each device from 0 in both cases.

If you have an integral SCSI hard disc device in your workstation, its device name will be */dev/sd0*. The next SCSI device you attach to your system should be given the name */dev/sd1*.

To confuse matters slightly, each of the seven devices (0-6) may internally support a further eight SCSI devices, called *logical units*. So in theory, there are potentially 56 devices that can be attached to one SCSI bus. However, currently most SCSI devices just support one logical unit number (LUN for short), usually LUN 0.

- *device-type* – this corresponds to the two ways that different types of devices handle I/O data. The *device-type* can either be `b`, to denote a *block special device* or `c` to denote a *character special device*.

  Block devices are normally discs that write data as a fixed string of bytes or blocks, with each block normally 512 bytes long (1/2K blocks). Character devices include tape drives and work in terms of individual characters. Although hard discs are normally referred to by the block device, there are certain operations where it is more appropriate to use the character or *raw* device; for example when you are performing backups it is more efficient to specify the raw device.

  Therefore, you will need to create at least two files in the */dev* directory for a hard disc device: one file to refer to the block device and another file to refer to the raw device.

- *major-number* – the kernel contains a variety of different device drivers for different types of devices, to which it interfaces. When an action is requested on a device, the kernel needs to be able to reference the device using the appropriate driver so that the correct driver, and moreover, the correct routine for that particular device is used.

  For this purpose, the kernel contains two tables:

  - a block device switch table
  - a character device switch table

  Each device type has entries in these tables that direct the kernel to switch to the appropriate driver interfaces for whatever action has to be taken for the device.

  The major device number allotted to a device, indicates a device type in one of these switch tables. The first entry to the SCSI driver in the character device switch table is 21, and in the block device switch table the first entry is 12.

The algorithm for computing the major device number for a character device is as follows:

```
21+(X*7)+Y
```

For block devices the algorithm is:

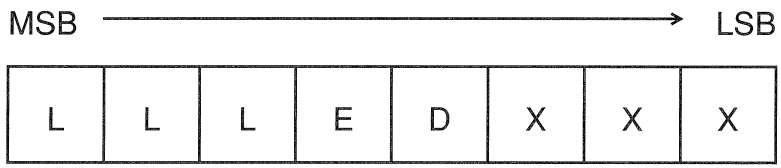
```
12+(X*7)+Y
```

where:

X – is the expansion card number, determined by the logical expansion card slot. This means that irrespective of where the SCSI expansion card is placed, if it is the only one installed, it will be the first found, ie logical device 0.

Y – is the target number, which is set by links on the SCSI device.

- *minor-number* – corresponds to the unit number of a particular device. The format of the minor device number is as follows:



LLL – the top three bits of the minor device number, refer to the SCSI Logical Unit Number (LUN) which is usually 0.

E – exclusive open; will ensure that the device is only open for one activity at any one time.

D – the DIY bit. This causes the SCSI driver to assume no in-built knowledge of the SCSI unit. The system call *ioctl*, which enables processes to interface to the device, is limited to the *SCSIDO* request. No read or write requests will be accepted.

XXX – the bottom three bits of the minor device number are device-class dependent.

For a direct access device, *XXX* refers to the partition number.

For a sequential access device, the first two most significant bits are ignored. The third bit, when set to 1, enables rewind on close:

X=1, ie rewind on close

For a direct access device, the algorithm for computing the minor device number is as follows:

Q + (Z*32)

where:

Q – is the partition, set from 0 to 7 (*sd1a* to *sd1h*)

Z – is the Logical Unit Number, usually 0

By convention, device names for block devices are denoted using a letter postfix (*a* to *h*) for partitions 0 to 7 respectively. A character interface for the block device is denoted by the prefix *r*.

Here are a few sample *mknod* entries for an example SCSI hard disc drive. The device is assumed to be connected to a SCSI expansion card located in the first SCSI slot:

```
mknod /dev/sd1a b 13 0
mknod /dev/rsd1a c 22 0
mknod /dev/sd1b b 13 1
mknod /dev/rsd1b c 22 1
mknod /dev/sd1c b 13 2
mknod /dev/rsd1c c 22 2
mknod /dev/rsd1h c 22 7
```

The above commands all create suitable entries in the */dev* directory for the device named */dev/sd1*, ie SCSI ID=1.

The first and second *mknod* commands specify a block device and character device entry for the SCSI device partition 0, with major number 13 and minor number 0 (block device) and major number 22 and minor number 0 (character device).

The third and fourth *mknod* commands specify a block device and character device entry for the SCSI device partition 1, with major number 13 and minor number 1 (block device) and major number 22 and minor number 1 (character device).

The fifth and sixth *mknod* commands specify a block device and character device entry for the SCSI device partition 2, with major number 13 and minor number 2 (block device) and major number 22 and minor number2 (character device).

The seventh *mknod* command specify a character device entry for the SCSI device partition 7, with major number 22 and minor number 7 (character device).

The device names for character devices such as cartridge tapes are denoted using a number postfix that refers to the entire device. To refer to the non-rewinding character device the letter *n* is added to the device name. For example:

```
mknod /dev/rct4 c 25 1
mknod /dev/rct4n c 25 0
```

The above commands create suitable entries in the */dev* directory for the device named */dev/rct4*, ie SCSI ID=4.

The two *mknod* commands specify two character device entries for a cartridge tape drive */dev/rct4*. The first entry is for the cartridge tape drive with major number *25* and minor number *1*. The second entry is similar but instead specifies the cartridge tape drive with no rewind on close; major number *25* and minor number *0*.

For more information, refer to *mknod*(8).

After typing these *mknod* commands, do an *ls -l* of the */dev* directory to display each of the *mknod*s that you have created for the SCSI device. Check that each file is owned by *root* and the group is *operator* and ensure that *root* has read and write access to the device and group *operator* has read access.

To make certain that only these permissions are in operation for each entry, type the following two commands for each of the entries that you have created in */dev*:

chmod 640 /dev/*device-name*
chgrp operator /dev/*device-name*

Once suitable entries have been created in the */dev* directory, the SCSI device is now accessible.

If the SCSI device is a sequential access device, such as a tape streamer, then it can now be used.

If the SCSI device is a direct access SCSI device, ie a hard disc, then you also need to run the command *scsidm* (short for *SCSI disc manager*) to configure the disc for use as a RISC iX filesystem.

The sequence of activities for configuring a SCSI hard disc device, using *scsidm* is:

- format and verify the disc
- section the disc
- partition the disc
- generate an example entry for the disc, to be included in */etc/disktab*

All the above tasks can be accomplished using *scsidm* in a single session.

scsidm has a user interface that provides on-line help. If at any time you do not understand a question press '?' and further information will be given. For information on any particular command you can also type:

Help *command*

where *command* is a valid *scsidm* command.

scsidm also helps you by providing suitable answers in brackets at the end of many questions it asks you. If you type ↵ the suggested answer is used as your answer.

Format and verify the disc

1 Firstly, type the command *scsidm*. Once the program starts, you will see the prompt:

```
scsidm>
```

2 Define the device name of the SCSI disc by typing:

```
scsidm>device device-name
```

As you are going to format the entire disc, you need to specify the *mknod* entry that refers to all of the disc, ie partition h. Also ensure you specify the raw device. *scsidm* assumes that the device name is located in the directory */dev* therefore you only need to type:

```
scsidm>device rsd1h
Current device is /dev/rsd1h,type 0(direct access)
Device identifies itself as a CONNER CP3100-100mb
-3.50A15
```

3 You can now format the disc using the command *format*:

```
scsidm>format
Keep the existing bad block list? >yes
```

This is a list of the known bad blocks on the disc and should normally be included by answering *yes*.

```
Interleave? (1) >1
```

The parameter *Interleave* determines the spacing of data on the disc. It is usually given the value *1*, the tightest spacing. Some very fast discs may require a different interleave factor. Refer to the documentation accompanying your SCSI device for details.

You are then asked to confirm that you wish to format the disc:

```
scsidm>Really format /dev/rsd1h? >yes
```

The whole disc is then formatted.

4 Once the disc is formatted it needs to be verified. To do this type:

```
scsidm>verify
No. of iterations? (1) >1
```

The number of times the disc may be verified can be changed by altering the number of iterations. This is *1* by default.

Bad blocks on the disc are reassigned to spare blocks on the disc by answering *yes* to the next question:

```
Reassign bad blocks (yes)? >yes
Verifying...verify OK
```

The disc now needs to be sectioned and then partitioned for use with RISC iX. There are further commands available that allow you to re-map out and redefine any subsequent bad blocks that may occur on the disc.

A full explanation of all of the commands is given in the manual page *scsidm*(8).

Section the disc

Sectioning the disc involves splitting the disc up into two discrete sections; one section for use with RISC OS and the second section for use with RISC iX. The size of section you choose to allocate for RISC OS is entirely dependent upon how often you will use your workstation to run RISC OS applications. However, even if you have no interest in running RISC OS, you should allocate at least 1MB of disc space on one of your discs to hold the RISC iX filing system module (!RISCiXFS) along with some RISC OS applications that will enable you to run and, if necessary, edit the module.

1 To section the disc, at the *scsidm* prompt, type:

```
scsidm>section
Include RISC iX partitions? (yes) >
```

Answering *no* to this question means that you wish to use the entire disc for use with RISC OS. Type ↵ to include RISC iX partitions.

2 You will then be asked to specify the size of the RISC OS section you wish to have on the disc, which will be used by the RISC OS SCSI filing system (SCSIFS) to store files. As explained above, a 1MByte RISC OS section is recommended, ie 2048 blocks:

```
SCSIFS area must be between 264 and 200640 blocks long
Size of SCSIFS Area (264)? >2048
Rounded no. of blocks in SCSIFS partition up to 2112
```

The number of blocks specified are then rounded up to equate with a whole number of cylinders on the disc. If you had chosen the maximum figure (200640 blocks), a small partition of approximately 2Mbytes would have been allotted to RISC iX and the rest of the disc would have been given over to RISC OS.

Note that these figures will vary for different types and sizes of hard discs.

3 You are then asked to confirm that you wish to section the device in this way:

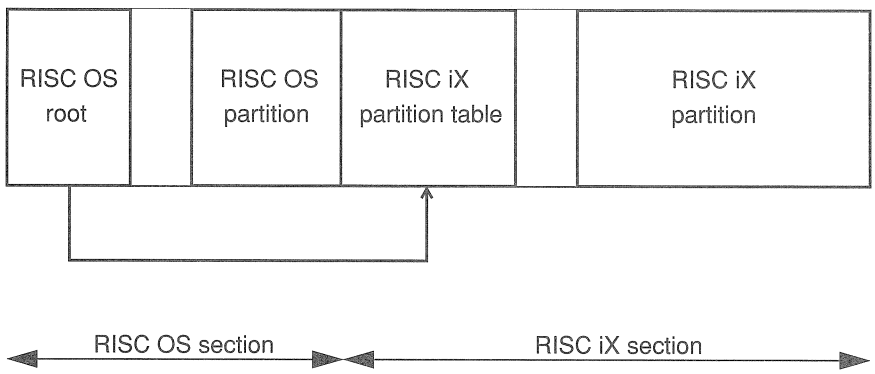
```
Do you really want to section device /dev/rsd1h?> yes
Writing SCSIFS partition .... done
scsidm>
```

The RISC OS section is created and the remainder of the disc area is given over for use with RISC iX. Note that *scsidm* automatically sets up partition 6 (*/dev/rsd1g*) to refer to the RISC OS section of the disc.

Partition the disc

Partitioning the hard disc means dividing the total disc section that has been allotted for use with RISC iX into several logical partitions that can be used for various purposes. For example, one partition can be used for storing user files and another partition can be used as a swap area.

Although the sizes of the sections and partitions will vary, the overall layout of the disc after it has been formatted and sectioned is as follows:



Information about any partitions that you create on the disc are held in the form of a partition table in the 'RISC iX information' area of the disc. This partition table is also replicated between disc partitions that are created on the RISC iX section of the

disc to ensure that at least one other copy of the partition table is available from another area on the disc. In the event of a disc crash there is a good chance of retrieving the disc layout information, which is vital for reading the data off the disc.

To partition the RISC iX section of the disc follow these instructions:

1 At the *scsidm* prompt, type:

```
scsidm>partition
RISC iX area has 202752 logical blocks, each 512 bytes
long
define which partitions ? >
```

You are then prompted to define the partitions on the disc. The maximum number of partitions supported (per device) is eight, ranging from partition 0 to partition 7. You can only choose partitions from 0 to 5 as partition 6 is used to refer to the RISC OS section of the disc and partition 7 is used to refer to the entire disc. Both these entries are automatically written to the partition table.

2 To partition the disc up into three partitions, 0, 1 and 2, corresponding to the sample *mknod* entries that were created for the device at the beginning of this section, type:

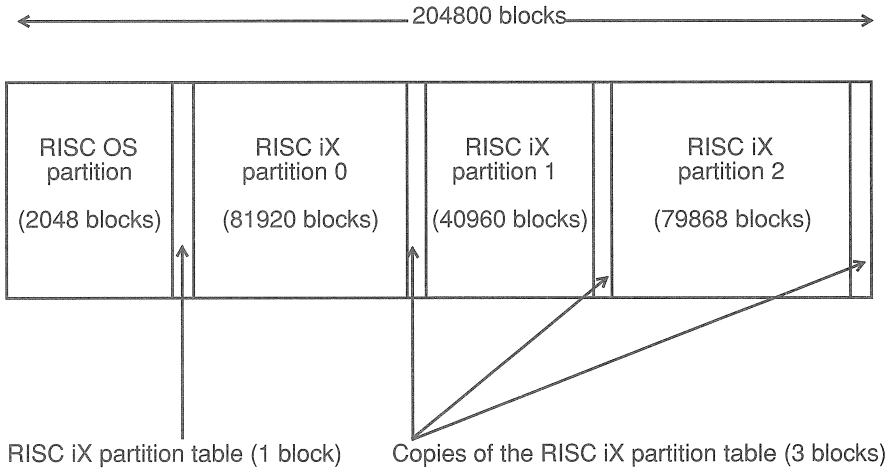
```
Define which partitions? >0,1,2
Starting block for partition 0, (2113) ? > ↵
The maximum allowable length of this partition is 202748
blocks
Length of partition 0 ? >81920
Starting block for partition 1, (84034) ? > ↵
The maximum allowable length of this partition is 120828
blocks
Length of partition 1 ? >40960
Starting block for partition 2, (124995) ? > ↵
The maximum allowable length of this partition is 79868
blocks
Length of partition 2 ? >79868
Writing RISC iX partition table ... done
```

The previous sequence of commands specifies three partitions to be placed on the disc area given over to RISC iX. The first partition is specified as approximately 40MBytes long (81920 blocks), the second partition is specified as approximately 20MBytes long (40960 blocks) and the remaining space on the disc is given over to the third partition, approximately 40 MBytes long (79868 blocks).

Note that the total length of the disc has decreased by four blocks. One block is used at the start of the RISC iX partitions to hold the partition table and an extra three blocks are used at the end of each partition specified to contain another copy of the partition table.

The reason for keeping so many copies of the partition table is to safeguard the information on the disc. In the event of the block containing the partition table becoming corrupt, you can use the command *scanptable* from *scsidm* to find another copy of the partition table on the disc.

For example:



If more partitions were specified, the end of each partition would contain another copy of the partition table.

After completing the partitioning of the RISC iX section of the disc, the partition table and a simple diagram showing the partitioning of the disc is displayed.

Generate a sample /etc/disktab entry

The next stage is to use the *mkdisktab* option of *scsidm* to generate a sample entry for this disc in a temporary file that you can then use as the basis for creating a suitable entry in the disc description file */etc/disktab*.

For example:

```
scsidm>mkdisktab
filename ? >/tmp/disktable
scsidm>
```

Leave *scsidm* by typing:

```
scsidm>quit
```

For more information, refer to *scsidm*(8).

Now edit the file */etc/disktab* to include the file */tmp/disktable* which you created using *scsidm*. All that you need to change is the name field, which is used to refer to the device. For example, you may wish to change the name chosen to refer to the device or to add some more names of your own or extra information about the partitioning of the disc. Also, check if there are any other entries for the device in this file. If there are, then these entries should be deleted.

For example:

```
# sd1a: ~40 Mb
# sd1b: ~20 Mb
# sd1c: ~40 Mb
#
CONNER|Conner|conner:\
      :ty=winchester:ns#33:nt#8:nc#776:se#512:rm#1:\
      :pa#81920:ba#8192:fa#1024:\
      :pb#40960:bb#8192:fb#1024:\
      :pc#79868:bc#8192:fc#1024:\
      :pg#2048:bg#8192:fg#1024:\
      :ph#204800:bh#8192:fh#1024
```

Each line in the disktab file describes a separate area of the disc. The first three entries in the disktab file, refer to the three RISC iX partitions of the disc */dev/sd1a*, */dev/sd1b* and */dev/sd1c*; the fourth line refers to the RISC OS section of the disc */dev/sd1g*; the last line refers to the whole of the disc */dev/sd1h*.

Make a filesystem on the partitions using newfs

At this stage it is possible to use *newfs* on the individual disc partitions to create RISC iX filesystems on selected partitions, which are then suitable for mounting. In our example, you could make filesystems on partitions */dev/sd1a*, */dev/sd1b* and */dev/sd1c*. For example to make a RISC iX filesystem on partition */dev/sd1a*, type:

```
newfs /dev/sd1a conner
```

and check the integrity of the new filesystem, using *fsck*:

```
fsck /dev/sd1a
```

For more information, refer to *fsck*(8), *mkfs*(8) and *newfs*(8).

Note, never create a RISC iX filesystems on the 'all' partition (*sd1h* in our example) as this will destroy all the RISC iX partition information as well as the RISC OS partition on the disc. Also, do not try to make a filesystem on the RISC OS section of the disc (*/dev/sd1g*).

Add suitable entries in the /etc/fstab file

If the disc partitions are to be mounted when the system is initialised then an entry must also be made in */etc/fstab*. For more information about the format of these entries, refer to *fstab*(5).

With a suitable entry created in */etc/fstab*, the SCSI disc partitions are then ready for mounting.

To mount the partitions, reboot the machine in the normal way. If everything works correctly you should be able to access the SCSI disc partitions you have entered in */etc/fstab*.

Using the floppy disc utilities

Introduction

This chapter describes how to use the floppy disc utilities available on your system for performing the following tasks:

- Transferring information from your system onto floppy discs in different formats for use by other types of workstations. For example:
  - UNIX workstations, ie RISC iX workstations, Sun systems, Xenix systems etc.
  - Acorn ADFS compatible machines
  - MS-DOS compatible machines

  This is especially useful for transferring information to other UNIX workstations that you cannot access directly using standard network communications utilities.
- Expanding the storage capacity of your system by using floppy discs as mountable RISC iX filesystems.

For information about taking care of floppy discs and how to write-protect them, refer to the *Installation Guide*. For information about using floppy discs to backup your system, refer to the chapter entitled *Backing up the filesystem* on page 65.

Floppy disc device names

The device names that your workstation uses to refer to its disc drive are shown below. The difference between the device names is the format of the floppies which are written or expected with each device name. The expected format listed alongside the device name:

Device name	Initial value	Meaning
`/dev/rfdf256`	256 bytes/sector	Acorn Master Compact ADFS format
`/dev/rfdf512`	512K bytes/sector	MS-DOS 720K format
`/dev/rfdf1024`	1024K bytes/sector	Acorn ADFS 800*1K 'D' format
`/dev/rfdv256`	256 bytes/sector	variable format
`/dev/rfdv512`	512 bytes/sector	variable format
`/dev/rfdv1024`	1024 bytes/sector	variable format

There is an almost limitless variety of possible formats in which floppy discs (or floppies for short) may be recorded, in terms of sector sizes, number of sectors etc and most are supported by RISC iX. Although there are some exceptions, notably Apple 3.5" discs.

However, most of the time you will probably only use the ADFS 'D' format (*/dev/rfdf1024*), unless you need to exchange data with MS-DOS based workstations, in which case you will need to access the device name which writes and expects MS-DOS format (namely */dev/rfdf512*).

The remaining device names include *oldadfs* 'L' format (640K) – */dev/rfdf256*, and the three variable-specification formats – */dev/rfdv256*, */dev/rfdv512* and */dev/rfdv1024*, which have downloadable format descriptions, but are initialised at boot time to be similar to the three fixed formats.

You can use the program *flpop* to reset the physical layout parameters for these variable-specification formats. For more information, refer to *flpop*(8).

Formatting discs

Brand new discs, or discs which have been written on using a different device name, cannot be used in any way until they have first been formatted.

The command to format floppy discs is *ffd* (short for *format floppy discs*). This command only writes initial track layout information onto the disc and does not write out any file structure information, which will be needed if you are going to store files on the disc.

ffd is quite adequate if you are going to use programs such as *tar* or *cpio* to transfer files between UNIX workstations as they do not expect to find any information about file structures on the disc.

To format a disc in the standard ADFS-style 'D' format, log in as *root*, insert the disc into the drive (having made sure that the write-protect slide tab is covering the hole), and type:

```
ffd 1024↵
Do you really want to format the disc (y/n)? y
Commencing format of /dev/rfdf1024
Commencing read check
Format completed satisfactorily
```

ADFS-style 'D' format is the best format to use if you are going to use the floppy disc with *tar* or *cpio* and it is also the default format that is used if *ffd* is issued with no options.

If you intend to use the floppy disc to transfer information to either ADFS machines or MS-DOS machines then you do not need to use a separate command just to format the disc – the formatting command can be included as an option with the ADFS and MS-DOS transfer commands described in the next section.

Transferring information between workstations

The following sections show you how to transfer information from your system onto floppy discs for use by other types of workstations and from other types of workstations back to your system.

Note that the utilities described are only suitable for copying textual files between your RISC iX workstation and another manufacturer's workstation. You can only transfer binary files between two RISC iX workstations.

Transferring information to and from other UNIX workstations

The following procedure describes how to transfer files between two UNIX workstations. This includes RISC iX workstations and any other UNIX workstation that supports *tar* (short for *tape archiver*) or *cpio* and also has a 3.5" floppy disc drive.

In order to be able to transfer files between two workstations, you need to be able to log in as *root* on both workstations.

First, you need to format the disc using the command *ffd* as described in the previous section. With the floppy disc formatted, you can now copy files onto the floppy disc using either *tar* or *cpio*. The following examples use *tar* which is the most popular command used to transfer files and is supported by more machines than *cpio*.

For example, if you have two textual files (*file1* and *file2*) that you wish to transfer to another UNIX workstation, type:

```
tar cvf /dev/fdf1024 file1 file2
a file1 1 blocks
a file2 1 blocks
```

The command *tar* can be used for saving and restoring files between workstations that may not use the same file formats. *tar* produces one large file in a standard format containing the files specified, which is written onto your floppy disc and can then be transferred to the remote UNIX workstation.

You can check that the files have been copied successfully, by typing:

```
tar tvf /dev/fdf1024
rw-r--r-- 0/0  291 Nov 21 16:19 1988 file1
rw-r--r-- 0/0   23 Nov 21 16:19 1988 file2
```

To copy the archived files from the floppy disc to the remote UNIX workstation, log into the remote workstation as *root*, change directory to where you want to put the files, insert the floppy disc containing the two files and type:

```
tar xvf /dev/fdname
```

where *fdname* is the floppy disc device name of the remote workstation. For example, if you were copying the files on to another RISC iX workstation you would type:

```
tar xvf /dev/rfdf1024
x file1, 291 bytes, 1 tape blocks
x file2, 54 bytes, 1 tape blocks
```

As indicated by the system messages above, this command extracts the two files *file1* and *file2* from the floppy disc and copies them into your current directory on the remote RISC iX workstation.

For more information, refer to *tar*(1), *cpio*(1) and *ffd*(8).

Transferring information to and from ADFS machines

To transfer information from your system to a floppy disc that can subsequently be read by an Acorn RISC OS computer, use the utility *wradfs* (short for *write to adfs*).

For example, if you wish to transfer two files named *file1* and *file2* to a brand new floppy disc, insert the disc into the drive (having made sure that the write-protect slide tab is covering the hole) and type:

```
wradfs -fia -n "ADFS-files" file1 log.file
Do you really want to format the disc? y
748K bytes free
```

This command firstly formats and initialises the disc so that it can store ADFS files (the *-fi* options). The *-a* option displays the amount of free space left on the disc after the transfer has been completed (`748K` in the above example). The *-n* option sets the disc name for the floppy disc (`ADFS-files`).

To check that the transfer worked satisfactorily, type:

```
adfsls -l
Disc name: 'ADFS-files'

Directory title: Initialised by Unix
--RW fff 13/04/89 13:01:00 000ae 000c-000f file1
--RW fff 13/04/89 13:01:00 00057 0010-0013 log-file
```

adfsls (short for *adfs list*) lists out the contents of the currently inserted floppy disc.

Notice that the '.' in the file *log.file* has been changed to a '–'. This is because ADFS uses full stops to specify directory pathnames. Thus the file is now called *log-file*.

The information on this disc can now be read by any type of Acorn RISC OS computer.

You can use the utility *adfscat* (short for *adfs catenate*) to type the contents of an ADFS file on the floppy disc and *adfsrm* (short for *adfs remove*) to delete an ADFS file from the floppy disc.

To transfer information saved onto disc from a RISC OS computer to your system, use the utility *adfscp* (short for *adfs copy*). For example, to transfer the ADFS files *file1* and *log-file* used in the previous example, back to the current directory and into their initial UNIX format, type:

```
adfscp -V file1 log-file .
Created file ./file1: 174 bytes
Created file ./log-file: 87 bytes
```

adfscp copies all the named files from the floppy disc to the current directory (specified by the '.'). The *–V* option causes *adfscp* to give an account of its activities. In the above example two files are copied to the current directory. Notice that *log-file* is not changed back to its original name, as *log-file* is a valid name in UNIX.

To check that the files and directories were successfully copied, list the contents of the current directory, using the *ls* command, immediately after using *adfscp*.

As indicated in the previous examples, there are certain restrictions concerning file names when transferring between ADFS and RISC iX. These restrictions are fully described in the manual pages for the ADFS utilities.

For more information about the ADFS floppy disc utilities and the various options that can be used with them, refer to the following manual pages; *adfscat*(1), *adfscp*(1), *adfsls*(1), *adfsrm*(1) and *wradfs*(1).

Transferring information to and from MS-DOS machines

The MS-DOS floppy disc utilities are virtually identical to the ADFS floppy disc utilities in terms of their functionality and the options that can be used with them.

To transfer information from your system to a floppy disc that can subsequently be read by an MS-DOS machine, use the utility *wrmsdos* (short for *write to ms-dos*).

For example, if you wish to transfer two files *file1* and *log.file* to a brand new floppy disc, insert the disc into the drive (having made sure that the write-protect slide tab is covering the hole) and type:

```
wrmsdos -fia file1 log.file
Do you really want to format the disc? y
728064 bytes free
```

This command firstly formats and initialises the disc so that it can store MS-DOS files (the *–fi* options). The *-a* option displays the amount of free space that is left on the disc after the transfer has been completed (`728064` bytes in the above example).

To check that the transfer worked satisfactorily, type:

```
msdosls -l
FILE1       :AR          14:05:56 13/04/89   2   103
LOG.FIL     :AR          13:54:20 13/04/89   3   92
```

msdosls (short for *ms-dos list*) lists out the contents of the currently inserted floppy disc.

Notice that both file names have been converted to upper-case characters to comply with MS-DOS file naming conventions and also that the file *log.file* has been changed to *LOG.FIL* as MS-DOS only allows three characters after a full stop has been used in a file name.

The information on this disc can now be read by an MS-DOS machine capable of reading 3.5" 720K floppy discs.

You can use the utility *msdoscat* (short for *ms-dos catenate*) to type the contents of an MS-DOS file on the floppy disc and *msdosrm* (short for *ms-dos remove*) to delete an MS-DOS file from the floppy disc.

To transfer information from an MS-DOS machine to your system, use the utility `msdoscp` (short for *ms-dos copy*). For example, to transfer the MS-DOS files *FILE1* and *LOG.FIL* used in the previous example, back to your current directory and into their initial UNIX format, type:

```
msdoscp -V  FILE1 LOG.FIL .
Created file ./FILE1: 183 bytes (174 bytes copied)
Created file ./LOG.FIL: 92 bytes (87 bytes copied)
```

msdoscp copies all the named files from the floppy disc to the current directory (specified by the '`.`'). The *–V* option causes *msdoscp* to give an account of its activities. In the above example two files are copied to the current directory. Notice that *LOG.FIL* is not changed back to its original name, as *LOG.FIL* is a valid name in UNIX.

To check that the files and directories were successfully copied, list the contents of the current directory, using the `ls` command, immediately after using *msdoscp*.

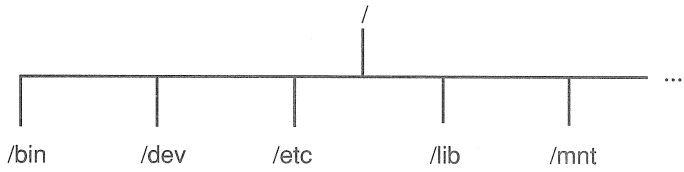
Similar to the ADFS utilities, there are certain restrictions concerning file names when transferring between MS-DOS and RISC iX. The MS-DOS utilities also try to determine between text file types and binary file types. These restrictions are detailed in the manual pages for the MS-DOS utilities.

For more information about the MS-DOS floppy disc utilities and the various options that can be used with them, refer to the following manual pages; *msdoscat*(1), *msdoscp*(1), *msdosls*(1), *msdosrm*(1) and *wrmsdos*(1).

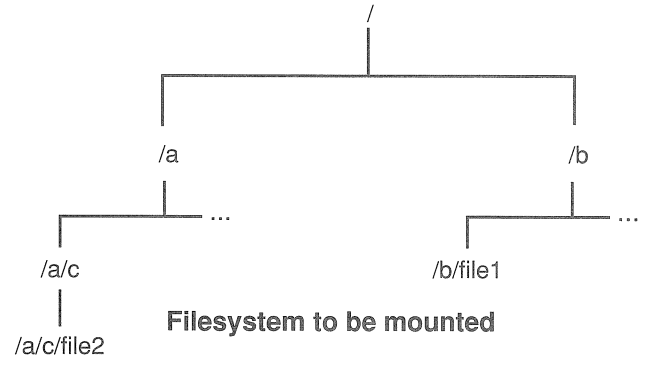
Using floppy discs as mountable filesystems

RISC iX provides facilities for adding to the existing filesystem by enabling you to access external discs that contain RISC iX filesystems and making this filesystem appear to become part of the current filesystem. This is known as *mounting* discs. The command use for this purpose is called *mount*(8).

There are a number of trivial restrictions which apply, but the effect is to make the contents of a given sub-directory become the contents of the mounted disc. For example:



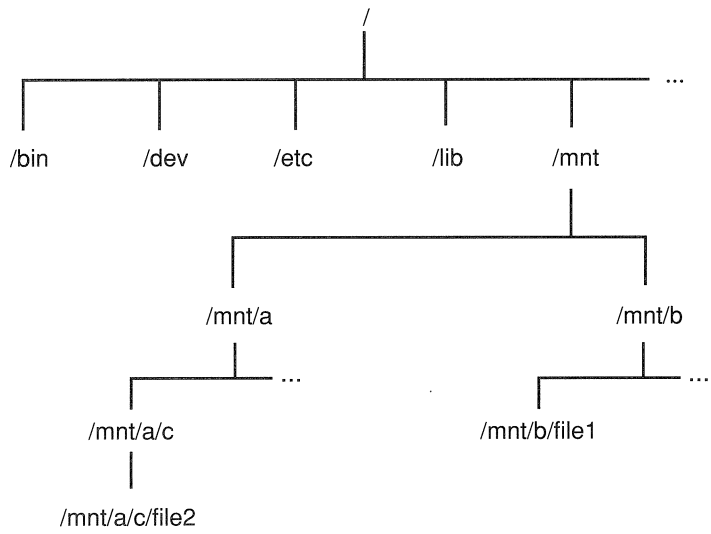
Existing filesystem



Filesystem to be mounted

In this example, the disc to be mounted has a root directory, corresponding to '/' on the existing filesystem, with sub-directories */a*, */b* and */a/c* and files */b/file1* and */a/c/file2*.

If this is mounted onto the directory */mnt* for example, the tree representing the mounted filesystem is now available as */mnt*, and the files and directories can be referred to as */mnt/a*, */mnt/b*, and */mnt/a/c* and as */mnt/b/file1* and */mnt/a/c/file2* respectively.



Example of a mounted filesystem

A disc partition is 'mounted' using one of the *mount* commands. This takes a *block device name* (in the */dev* directory) corresponding to the partition, and a directory name, and connects the two so that future references to the directory name correspond to files on the disc partition.

If the directory mounted upon contains files, then these files cease to be available until the filesystem is unmounted. This is done with the *umount* command. However if any files or directories are open on the mounted filesystem, the *umount* command will not work. A directory is open if anyone has made some directory on the mounted disc their current directory. (Note that they might have done this before they started some background process, which will still count).

The file */etc/mtab* is updated automatically when discs are mounted and unmounted.

In the case of floppy discs, you can chose whether to create a RISC iX filesystem on them and mount them as part of the overall directory structure, or create ADFS or MS-DOS structures on them. There are arguments for both approaches. It is certainly much easier to access individual files if a RISC iX filesystem is created on them, but for reasons of portability, non-RISC iX file structures are usually created. (Note that floppies which are mountable filesystems are very rarely portable between different versions of UNIX).

Any level of directory may be mounted upon, including directories on previously mounted filesystems, but usually a number of standard names are reserved in the root directory, such as */mnt*, as mount points. The files contained therein are invisible once items are mounted, so these directories should not be used to store files.

Using floppy discs as mountable RISC iX filesystems

The floppy discs can be used as mountable RISC iX filesystems for transferring information to and from RISC iX workstations. To use floppy discs as mountable RISC iX filesystems, you need to do three things:

- format the floppy disc using the command *ffd*.
- create a RISC iX filesystem on the formatted floppy disc by initialising the file structures and the root directory on the floppy disc, using the command *newfs*.
- mount the disc in an appropriate directory on the filesystem, using the command *mount*.

Format the disc in ADFS-style 'D' format using the command *ffd* as previously described.

Creating a RISC iX filing system involves initialising the file structures and the root directory on the floppy disc. There are two commands you can use to do this:

- *mkfs* (short for *make filesystem*)
- *newfs* (short for *new filesystem*), a user-friendly front-end to *mkfs*

It is recommended that you use *newfs* for creating a RISC iX filing system. But it also is helpful to have an idea about how *mkfs* works. An appropriate *mkfs* command is as follows:

```
mkfs /dev/rfdf1024 800@1024 5 2 8192 1024
```

The raw device, */dev/rfdf1024* is used because the block device limits transfers to multiples of 1024 byte. The next argument (*800@1024*) gives the size of the disc. As the disc has a capacity of 800K, this argument corresponds to *800* sectors at *1024* bytes (1K) per sector.

The remaining arguments give the number of sectors per track on the disc (5), the number of tracks per cylinder on the disc (*2*), the primary block size for files on the new filesystem (*8192*), and finally the 'fragment' size for files on the filesystem in bytes (*1024*). This last value represents the smallest amount of disc space that will be allocated to any file that is created on the filesystem, ie 1024 bytes.

There are other parameters that can be specified but they can be safely omitted as sensible default values are provided. For more information, refer to *mkfs*(8).

Although *mkfs* works satisfactorily, its syntax is somewhat complex. It is far easier to use the command *newfs*, which does all the spadework for you by selecting the correct options for the type of disc you are initialising and passing these options on to *mkfs*. For example, the filesystem can be initialised in the same manner in the previous example, by using the following *newfs* command:

```
newfs /dev/rfdf1024 adfs
```

newfs reads the disc description file */etc/disktab* to decipher what type of disc is being initialised. It then calculates the optimum parameters to use in calling *mkfs* for this type of disc, then executes the command *mkfs* with these parameters to initialise the filesystem.

You can see the *mkfs* command that is actually used to create the filesystem by invoking *newfs* with the *-v* option set. For example:

```
newfs -v /dev/rfdf1024 adfs
```

The final stage in making your floppy disc a mountable RISC iX filesystem is to mount it in an appropriate directory on the filesystem.

To mount the disc on the directory */mnt*, type:

```
cd /
mount /dev/fdf1024 /mnt
```

/mnt is usually an empty directory that is reserved for mounting discs. If there are any files or directories stored in this directory they will disappear temporarily, but will reappear when the disc is unmounted.

To check that the disc is mounted, type:

```
df
Filesystem  kbytes    used   avail capacity  Mounted on
/dev/sd0a    34983   32010    2623    92%     /
/dev/fdf1024  703       5     627     1%    /mnt
```

The above display shows that the floppy disc device */dev/rfdf1024* is mounted on the directory */mnt* and has just under 600K of space available for storing RISC iX files. Note that the other 200K of space on the disc has been used up in formatting the disc and initialising the filesystem.

The disc can now be referenced as the directory */mnt*, just as you would reference any other directory on the filesystem. Files can be saved in this directory just as they would in any other area of the filesystem.

After use, you can dismount the disc, by typing:

```
cd /
umount /mnt
```

Don't forget to type '*cd /*'. If you are in */mnt* when you type the second line, you will get the error message:

```
/mnt: Device busy
```

This is because unmounting a filesystem while you are in it is rather like sawing a branch off a tree while you are perched on its extremity!

After typing the above commands, you are free to remove the disc. Note that you should never remove the disc before you first *unmount* it from the filesystem. There may still be some information in memory that has not yet been written out to the disc.

Dismounting the disc first ensures that all data is preserved, as *umount* performs a *sync*.

For more information, refer to *mount*(8).

Networking System Administration

Introduction

This part of the Guide contains information about the Network Services that are available on RISC iX workstations. The information is applicable for all types of RISC iX workstations and covers only those aspects of Network Services that are necessary for performing the duties of System Administration on a network.

The chapters in this part of the Guide include:

- *Setting up a network* – a step by step guide to setting up a local network of RISC iX workstations.
- *How NFS works – An overview* – provides a general description of NFS and the services it provides. For a simple introduction to networking in general and the NFS in particular, refer to the chapter entitled *Networking and NFS* in the *RISC iX User Guide*.

Subsequent chapters introduce each of the Network Services that are available under NFS, and describes the System Administration duties required for each service:

- Network File System (NFS) Service
- Network Information Service (NIS)

  Note, the former name given to the Network Information Service was *Yellow Pages*. While this term has been expunged from the documentation, it may still be mentioned in certain parts of the networking software. For example, in message text and source code comments.
- Using the Automounter
- Secure Networking

Within each of the above chapters you will also find information about periodic maintenance and trouble-shooting for the service under discussion.

While some of the material in these chapters tends to be theoretical, its specific implications will be seen again and again as you become familiar with Network System Administration. For example, if you run the Network Information Service, you need to be aware that some typical RISC iX procedures will have changed in this environment. This is also true of the Network File System Service.

Setting up a local Ethernet network

Introduction

This chapter contains information that should enable you to set up a local Ethernet network of RISC iX workstations from scratch, or alternatively to incorporate RISC iX workstations into an existing Ethernet network, supporting the Internet communications protocol.

RISC iX workstations can also be connected together using Econet, which also implements the standard Internet communications protocol. This enables RISC iX workstations networked by Econet to use the Berkeley networking commands (described in the chapter entitled *Networking and NFS* in the *RISC iX User Guide*).

In a normal configuration the workstations networked using Econet would also be connected to an Ethernet network via a *gateway* workstation that is fitted with an Econet and an Ethernet card. For more details about setting up a network using Econet, refer to the chapter entitled *Setting up an Ethernet/Econet network* on page 313.

The remainder of this chapter discusses setting up a local Ethernet network for all types of RISC iX workstations and gives information on configuring different network environments.

This chapter contains information on:

- Networking protocols supported by RISC iX
- Designing the network
- Assigning Host and Interface names
- Assigning Internet addresses
- Setting up network hardware
- Setting up network software for all types of workstations.

Networking protocols supported by RISC iX

RISC iX workstations support the Internet communications protocol as defined by the *Internet Engineering Task Force* which includes support for the Ethernet physical layer standard as defined by the IEEE 802.3 local area network protocol for CSMA/CD (Carrier Sense Multiple Access/Collision Detection) networks.

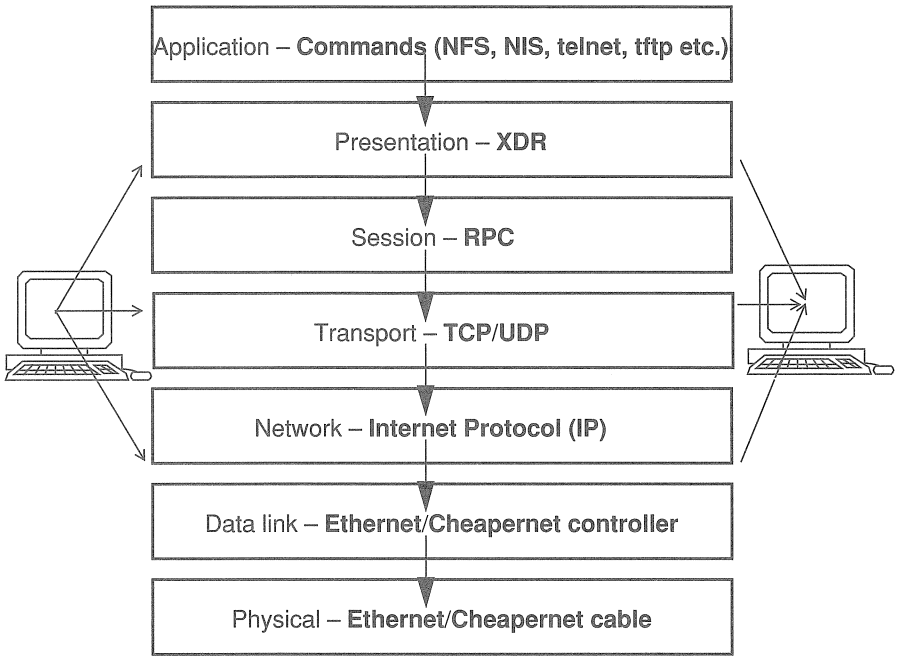
An existing network must support these protocols to enable RISC iX workstations to be able to communicate with the other workstations on the network.

Conversely, if you create a network from scratch made up entirely of RISC iX workstations, then in future you must ensure that any other types of workstations you add to the network also support the Internet Communications protocol.

The ISO Communications Protocol model is made up of seven layers. Each layer defines a specific feature of a network at progressively increasing levels of complexity. The first level defines the actual physical connection between two workstations on a network and subsequent layers build on this layer to describe how information is to be transferred across the network.

This section will describe each of the layers in turn and explain what they do. Again although this material is not essential for your System Administration duties, a clear understanding of how the network operates is very useful when trying to solve problems with the network.

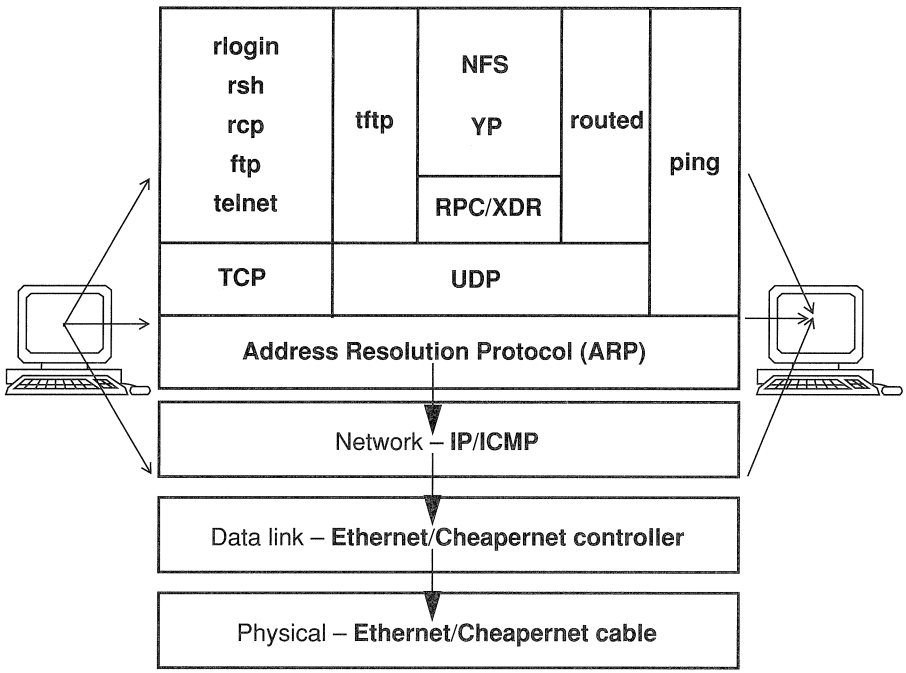
The diagram below shows how the ISO layered model is implemented for RISC iX workstations:



Theoretically, when information is transferred across the network it passes through each layer until it arrives at the workstation it was sent to. In practice, many commands do not need to travel through each layer and bypass some of the layers. For example, *tftp* bypasses the presentation and session layers of the above model. Most generic UNIX workstations support the above model at least up to the transport layer, therefore programs like *tftp* can run under many different versions of UNIX.

Other network services like NFS and NIS use the presentation and session layers (ie RPC and XDR) for more high-level interprocess communication facilities like transparent file access. Therefore, these services are restricted to UNIX workstations that support RPC/XDR protocols. These protocols are described in more detail later on in this chapter.

A more precise schematic of the networking model is shown below:



Physical – Ethernet/Cheapernet cable

The IEEE 802.3 Local Area Network Protocol defines the physical requirements of a cable to support the Internet Communications protocol. The following list describes the cables that you can use to connect your RISC iX workstations together:

- *Ethernet* is the full specification Ethernet cable. This type of cable is usually referred to as *Thick Ethernet* (other terms used include *broadband Ethernet* and *10 base 5 Ethernet)*. Thick Ethernet is a thick cable which allows a relatively large maximum distance between *nodes* or workstations on a network. It has the disadvantage however of being expensive and difficult to lay as it has to be routed through walls and ceilings and must meet certain fire and safety standards.
- *Thin Ethernet* is a thinner coaxial cable and overcomes the problems of Thick Ethernet by being less expensive and more flexible. This type of cable is usually referred to as *Cheapernet* (other terms used include *baseband Ethernet* and *10 base 2 Ethernet*). The penalties for this flexibility are fewer nodes and shorter maximum allowable distances between nodes.

  For example, the full Local Area Network Protocol (IEEE 802.3) Ethernet standard for Thin Ethernet cable states that the distance between two workstations should be less than 185m but greater than 1m, and also that no more than 8 nodes should be installed.

Data link – Ethernet/Cheapernet controller

The first two layers comprise firstly, the Ethernet cable itself. On top of the physical Ethernet layer lies the Ethernet controller layer which prepares the information for transmission across the network, breaking it up into appropriate chunks and attaching header information about its size, destination etc.

Network – IP/ICMP

The Internet Protocol handles the routing of information across the network, using the Internet Control Message Protocol (ICMP) to return any error messages.

Address Resolution Protocol (ARP)

Layered on top of this is the Address Resolution Protocol (ARP) which dynamically maps between the Internet and Ethernet addresses of a workstation. In this way the physical address of a workstation is translated into a meaningful Internet address which identifies it to the rest of the network. The Internet addresses known to a workstation are located in */etc/hosts*.

TCP/IP and UDP/IP

The transport layer comprises two separate protocols that both use Internet protocols to provide a guaranteed data channel between workstations on a network:

- Transmission Control Protocol (TCP/IP)
- User Datagram Protocol (UDP/IP)

They both use error checking and flow control to ensure the reliability of information and can re-transmit information if necessary.

TCP/IP provides a reliable data stream by establishing a virtual connection between two workstations and so is used by network services that rely on data integrity. For example, *ftp*, *rlogin* etc.

UDP/IP is a simple but less reliable transport protocol. It is a 'connectionless' protocol that provides a basic datagram service. Less elaborate network services such as *tftp* and *rwall*, use this protocol.

The file */etc/services* contains a list of the network services available in the network and shows which of the two transport protocols they use. */etc/inetd.conf* tells the Internet daemon (*inetd*) which of the services it should monitor. Refer to *inetd*(8), *inetd.conf*(5) and *services*(5) for more information about the format of these files.

RPC/XDR

RPC and XDR protocols provide a higher level of communications facilities for advanced network services such as NFS and NIS.

The Remote Procedure Call (RPC) protocol is made up of a library of procedures that provide a means whereby one process (the client process) can have another process (the server process) execute a procedure call, as if the client process had executed the procedure call in its own address space. Because the client and the server are now two separate processes, they no longer have to live on the same physical workstation.

The External Data Representation (XDR) is a protocol specification for the portable data transmission standard. Together with RPC, it provides a kind of standard I/O library for interprocess communication.

The *portmapper* is a utility service that RPC network services listed in the previous diagram use. It's a kind of registrar that keeps track of the correspondence between ports (logical communications channels) and RPC services on a workstation, and provides a standard way for a client to look up the port number of any remote program supported by the server.

There are other Network Services that could also be termed network services in the broad sense. This section, however, is intended as an introduction, and it covers only the fundamental networking services as shown in the previous diagram.

Network Services are added by means of server processes (usually daemons) that are based upon the RPC (Remote Procedure Call) mechanism. These daemons are executed on all workstations that provide the service. Each server communicates with the kernel proper and with its fellows on other workstations as necessary to get its job done.

The daemons differ significantly from those that were inherited from Berkeley in that they are all based on RPC. As a consequence, they automatically benefit from the services provided by RPC, and the External Data Representation (XDR) that it, in turn, is built upon – for example, the data portability provided by XDR and RPC's authentication system.

Anything built with RPC/XDR is automatically a network application, as is anything that stores data in NFS files, even if it doesn't use RPC explicitly. Furthermore, insofar as network applications can presume the functionality of other network applications and call upon their services, all network applications are network services as well. The XDR/RPC environment then, is inherently *extensible*. New network services can be easily added by building upon the foundation already in place.

The Networking Services are analogous to UNIX commands – anyone can add one, and when they do they are effectively extending the "system".

Designing the network

Before you begin setting it up, you'll need to consider the design of your network. For example:

- the different ways you can connect together your network
- the names you have to assign to each computer on the network
- the numerical addresses you need to assign to each computer on the network
- the different ways you can configure the software

Network topologies

The most fundamental part of your network installation is how you design the topology (or layout) of your network. These are some of the major points to consider:

- Computers that communicate frequently with each other should ideally be on the same physical network. However, if you put too many computers on the same physical network you will overload it, and its performance will degrade.
- You may wish to connect together separate local networks to make a single larger network. You can do this by using *gateways*. These bring various advantages and disadvantages which are considered below.
- You may also wish to connect your local network into larger wide area networks. These may cover all of your site or organisation, or they may even be worldwide in scope. If your network is connected to others you'll need to closely conform to certain standards, so that you don't bring other sites' or organisations' networks to a halt because of an incompatibility between your network and theirs.

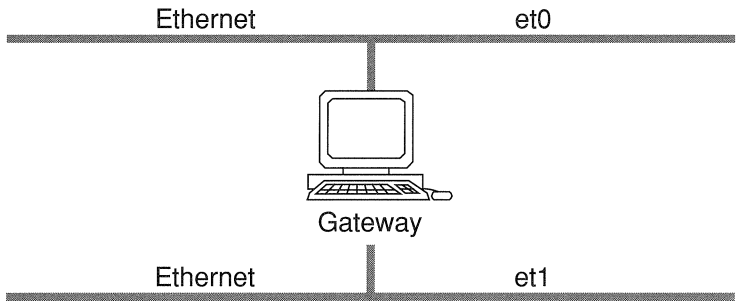
The rest of this section outlines some of the different topologies that you can use.

Simple network topology

If you only wish to create a small network of workstations (between 5-15) then you have very few decisions to make. Your RISC iX workstations can all be connected directly using one main Ethernet cable.

Using gateways

A gateway workstation is simply a workstation that enables you to split up a single network into two separate networks. The gateway workstation sits between two local networks and transmits packets of data between them. The gateway workstation has two interfaces which are normally Ethernet controllers. Each interface allows it to communicate with a separate network, so the gateway workstation has a separate identity on each network. For example:



There are many reasons why you should consider splitting your network using a gateway workstation:

- If you intend to network together a lot of workstations (between 15-25).
- Physical restrictions on the length of Ethernet cable. If you exceed the restrictions on the length of cable that is needed to connect all the workstations together on a single network.
- If the traffic over a single network will cause the network to operate very slowly due to constant data collisions.

There will be an overhead involved in using a workstation as a gateway workstation (approximately 5% of CPU time), but this is a small sacrifice to make on the resources of one workstation if it increases the overall performance of all the other workstations on the network.

Bottlenecks

When a network packet goes through a gateway there are inevitable delays as it's merged with the existing traffic on the other side. Consequently these can cause bottlenecks in the system. In general:

- The fewer gateways a network packet has to cross, the quicker it will reach its destination
- If too many network packets are using the same gateway, you can get a bottleneck there too. The more gateways you provide, the better performance will be; although there comes a point where you'll have enough gateways that they're almost always working at peak efficiency anyway.

It's hard to give more precise guidelines, because of the wide range in how network-intensive computer usage is at different sites. If you're already running a network, you'll have a good feel for how things work out at your site.

Domains

A further consideration when setting up networks, is whether to establish a *domain* name or a series of domain names for your network. A domain name is a naming technique that you can use to allow a collection of workstations to be administered as a single entity, thereby greatly easing your System Administration duties.

Within a domain, all named objects such as workstations or users, must have different names. However, two workstations on different domains can have the same hostname, but each workstation is still unique by being in a separate domain.

Domains are normally an important consideration, when a network contains a large number of workstations. They enable you to sub-divide the network into logical divisions using either geographical or physical boundaries.

If you have a small site, it is normal to have just a single administrative domain and a single local area network. Larger sites might have two or more local area networks that communicate with each other via a router, thus forming an *internetwork*. If all networks in the internetwork are under a single administration, you can group them together under a single domain name. Larger sites might need both multiple networks and multiple domains.

In RISC iX domain names are only used by the Network Information Service (NIS) environment to define a NIS domain. A NIS domain defines a group of workstations that access a particular set of NIS maps. For more information on setting up domain names in the NIS environment refer to the chapter entitled *The Network Information Service* on page 219.

Assigning host and interface names

The next stage in planning your network is to assign a name to each workstation and each interface on your network, if you decide to split the network up. You'll use these later when you come to set up the network software on your workstations.

If you've already got a network running...

If you've already got a network running on your site, you should have a naming scheme set up. Make sure that the names you assign conform to this scheme, and that you first contact anyone who centrally administers their allocation, if you are not directly responsible.

Assigning host names

First assign a name to each workstation on your network. Your users will use this name to refer to the workstation. The name must be unique on your network – you can't have two workstations with the same name. The name you give to each workstation is known as the *principal hostname*, or *hostname* for short.

It helps your users if each host name is easy for them to remember. One way to do this is to use a theme, such as planets (eg *saturn*, *uranus*); another way is to give names that have some relationship to the workstation's function on your network (eg *accounts1*, *accounts2*). You can combine these ideas – so you might name the graphics department's workstations after famous artists (eg *turner*, *vangogh*).

Assigning interface names

You also have to give a name to each interface in each workstation, whether it be an Ethernet or Econet interface, or two Ethernet interfaces. Again, this name must be unique on your network – you can't have two interfaces with the same name. The name you give to each interface is known as the *interface name*.

If there's only a single interface in a workstation it's normal to use just the principal host name as the interface name. If there are two interfaces in a workstation it's normal to refer to the principal hostname in each interface name: so a workstation named *saturn* may have interfaces named *saturn_ether1* and *saturn_ether2*.

Assigning Internet addresses

Likewise you have to assign a unique numerical address to each interface. This address is known as the network interface's *Internet address*. It is this address that the IP protocol uses to communicate; if a user specifies a host name or interface name, the software automatically converts it to an Internet address.

An Internet address is four bytes long. These four bytes are split into two fields: one identifies the network, the other identifies a host on that network.

If you're already connected to other sites...

If you're already connected to other sites over the Internet, you should have a numbering scheme set up for your site. Make sure that the Internet addresses you assign conform to this scheme, and that you first contact anyone who administers their allocation.

If you don't plan to connect to other sites...

If you don't plan to connect to other sites over the Internet, all you need to ensure is that each interface's Internet address is unique. Here is a suggested scheme.

Suggested scheme

<network number>	<host number> (high byte)	<host number> (middle byte)	<host number> (low byte)

Number your networks from one: for example, you might number one Ethernet network as net 1, and your second Ethernet network as net 2. Likewise, number your hosts (not your interfaces) from one. Your available Internet addresses and their meanings would then be:

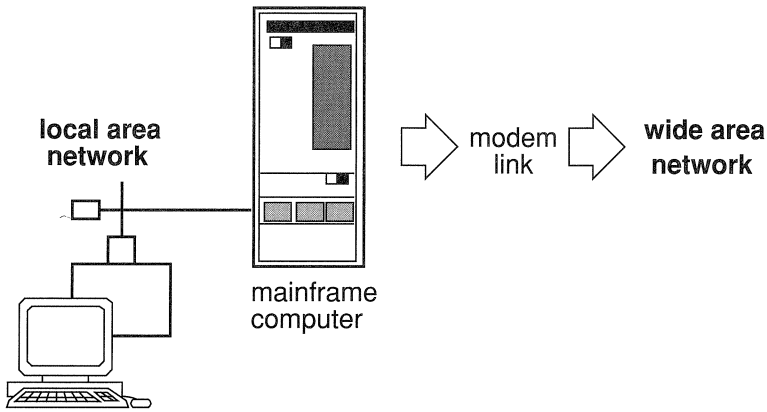
Ethernet 1	Meaning	Ethernet 2	Meaning
1.0.0.1	host 1 on Ethernet 1	2.0.0.1	host 1 on Ethernet 2
1.0.0.2	host 2 on Ethernet 1	2.0.0.2	host 2 on Ethernet 2
1.0.0.3	host 3 on Ethernet 1	2.0.0.3	and so on up to...
1.0.0.255	host 255 on Ethernet 1	2.0.0.255	host 255 on Ethernet 2
1.0.1.0	host 256 on Ethernet 1	2.0.1.0	host 256 on Ethernet 2
1.0.1.1	host 257 on Ethernet 1	2.0.1.1	and so on...

Of course, if a workstation has only got one interface fitted, you'll only use one of the addresses assigned to it; one of the addresses will be 'wasted'. But if you later upgrade the workstation to add a second interface, you'll already have a meaningful Internet address reserved for it.

If you plan to connect to other sites...

If you plan to connect to other sites over the Internet, you need to ensure not only that Internet addresses are unique to your site, but also that they are unique to the entire Internet.

You don't need an official network number unless you are planning to connect to other existing public domain TCP/IP networks in the future, using a leased line such as X.25 or PSS. For example:



In this example, the mainframe acts as the gateway workstation between the two networks, by being able to transfer the information from one type of network and communication line to another type of network and communication line.

As with a local area network, the physical link must be supported by an appropriate communications protocol for the workstations to be able to talk to each other. Due to the variety of workstations that use these links, there is a wide variety of protocols.

To circumvent this problem a series of programs exist that support a range of communications protocols. One of the most popular of these programs is *UUCP* (*UNIX to UNIX copy program*) which permits communication between all types of UNIX systems, typically over serial lines. For information about setting up UUCP on RISC iX workstations, refer to the chapter entitled *Setting up UUCP* on page 305 .

If you only intend to use UUCP for outgoing communication, then you can set up a local network using an Internet number of your choice. However, it is worthwhile applying for an official number, should you decide to extend your local network out into a wider area network in the future.

The Internet already connects together thousands of sites, each with many hosts. Clearly it's impossible to keep so many Internet addresses unique on an informal basis. Consequently, there is an administrative body responsible for allocating unique network numbers.

To apply for an official Internet number you need to fill in a short questionnaire about yourself and the type of network you have at your site, and send this information to the DDN Network Information Centre. A copy of this questionnaire and the address to send it to once completed is given in *Appendix A: Internet form* on page 337.

Depending upon your answers to the questionnaire, you will be allocated a class A, B or C network. The difference between each type of network is the number of bits that are allocated for identifying the network.

Internet addresses are composed of 32 bits. The bits are split up into four bytes. In the initial scheme, only the first octet was used to identify the class of network:

```
00000000.00000000.00000000.00000000
```

In a a class A network, only the first byte is allocated for the network, giving a maximum of 127 networks (1-126).

Due to the small number of networks that could be allocated using this scheme, the addressing format was changed to use the second byte for the network address – this is known as a class B network (128-191).

As it became apparent that there would be more networks than there would ever be workstations on a single network, this addressing scheme was modified again to also use the third byte for the network address – known as a class C network (192-223). All the other Internet addresses are reserved.

Depending on the size of your network, you will have been left up to three bytes free for the host field, identifying your network as a class A, B or C network. Unless you are installing a very large network, you will probably be allocated a class C network address.

Once you have been allocated a unique network number, it is your responsibility to allocate a unique host number to each computer on your network. Note that you should not allocate any workstation to have a host number of 0 or 255.

A note about physical addresses

As well as the Internet address, each interface has a six byte *physical address* (also known as its *MAC address*). For example an Ethernet card's physical address is unique worldwide, and is set in the hardware at the time of manufacture.

Consequently you shouldn't need to do anything to set up these physical addresses. The physical address of each workstation is normally displayed by the kernel when the workstation is first turned on:

```
et0: slot 0: iss1, address 00:00:a4:00:08:13
```

You can also re-read this address when the workstation is running using the command *dmesg*(8), which prints out recently printed diagnostic messages.

Setting up the network hardware

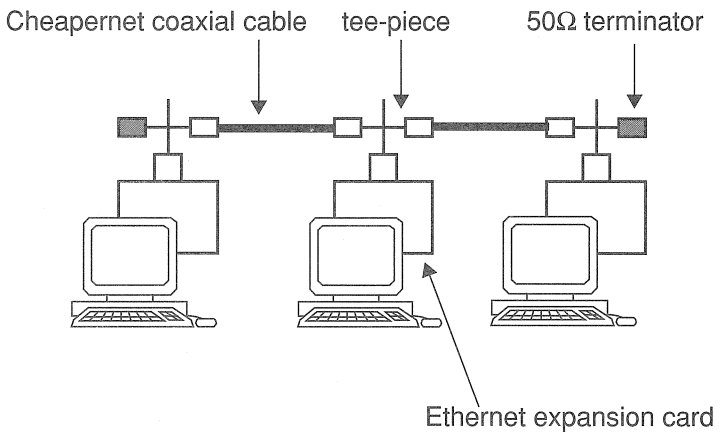
This section briefly describes the hardware that comprises a basic local Ethernet network. The information in this section assumes that you have already installed an Ethernet card in each of the workstations that you intend to network. (Ethernet cards are available from your supplier and come complete with fitting instructions.)

When installing Ethernet cards make sure that you configure the card correctly according to the type of Ethernet cable you are going to use with it. This involves setting a couple of links on the card as detailed in the fitting instruction leaflet.

Individual workstations and other devices are plugged into the Ethernet's cable system. The Ethernet controller located on the Ethernet expansion card inside each workstation has its own, unique Ethernet physical address. This is displayed as soon as you connect an Ethernet cable to the Ethernet card and boot the workstation up into RISC iX.

You can move the workstation around and plug it into any convenient outlet on a single Ethernet cable. You can attach different types of equipment to an Ethernet cable (Sun workstations, Apollo workstations etc), but they may not necessarily be able to communicate with one another. For example, VMS systems using DECNET and RISC iX workstations can be attached to the same Ethernet cable, but they cannot communicate with one another without special software on one or both of the systems.

The diagram below shows a typical Ethernet setup for three RISC iX workstations using Cheapernet coaxial cable:



The basic parts of a typical local Ethernet network, as shown above, include the following:

- Ethernet expansion card (containing the Ethernet controller) in each workstation.
- tee-pieces to connect the cable to the expansion cards.
- 50Ω terminators, with insulated outside covers, to terminate the network at both ends.
- Suitable lengths of Cheapernet coaxial cable.

File transfer over Ethernet

Here is a breakdown of file transfer, from the hardware perspective, over the Ethernet local area network:

1. A workstation on the network uses a command to specify that a file be transferred between a sending and receiving workstation.
2. The command makes requests to the operating system. The data is copied through a *virtual circuit* or as a series of *datagrams*.
3. The virtual circuit or datagram software breaks the character stream into packets for transmission.
4. The packets are passed to the Ethernet driver software.

5 The Ethernet device driver copies the packet into a packet buffer and tells the Ethernet controller to transmit it.

6 The Ethernet controller waits until the coaxial cable is not in use and then transmits the packet.

7 The packet's bit stream is 'injected' into the coaxial cable.

8 The receiving workstation recognises its address in the packets, and reverses the above procedure: bits received are fed to the controller, passed to software that reassembles the packets, maps the characters and stores the data.

Extending the network

There are many hardware devices available that you can attach to your network to expand its capabilities. Some of these are described below:

- *Repeater* – This is a device used at the physical layer of the network that copies each bit of a packet from one segment of a network to another. In Ethernet terminology, a single length of coaxial cable is considered a physical segment. Therefore, a large physical Ethernet local area network can consist of several segments connected by repeaters.
- *Bridge* – This is a device used at the data link level of the network protocol model. It selectively copies packets from one segment of the network to another segment of the same type. This is reasonable to do, in theory, on an Ethernet because the first few bytes of an Ethernet packet contain its destination address. A bridge that is watching all Ethernet traffic can determine which hosts are sending which packets. Therefore, it will know which packets to copy from one segment to another and more importantly, which packets to ignore.
- *Router* (sometimes called a gateway) – This is a device that forwards packets of a particular protocol family, for example, IP, from one *logical network* to another. A logical network is understood only by a particular protocol, and has a single network number. Usually there is a one-to-one mapping between physical network and logical network, but things can get blurred by subnets and bridges.

  The router may forward packets between different physical types of networks, eg from an Ethernet to a ring local area network, or from the ARPANET to a phone line. It can also forward packets between two logical networks of the same type, eg between two Ethernet networks on different floors of a building.

  During forwarding, a router looks inside the packet to find the destination address, then consults its routing table, which is normally kept up to date by having routers talk to each other via some routing protocol.

- *Gateway* (sometimes called a relay or forwarder) – This is a device and its associated software that enable networks using different protocols to communicate with each other. Because it translates protocols existing at all layers of the protocol model, you can use gateways to connect networks that differ on all layers from each other.

  For example, a RISC iX workstation can be set up to act as a gateway workstation to connect an Ethernet network to an Econet network. For more information refer to the chapter entitled *Setting up an Ethernet/Econet network* on page 313.

Once you have installed the Ethernet cards in your workstations, bring each workstation with a disc up into single-user mode. You should see a similar start-up message to the one shown below, confirming that the Ethernet card has been installed correctly:

```
...
et0: slot 1: isa1, address address
...
```

If you have installed a discless workstation, when you turn the workstation on, you should see the following:

```
Looking for IP address ...
Physical address is address
```

From here you will then need to edit a few network configuration files on each workstation, to describe the characteristics of the network and the type of workstation you are connecting to the network.

If you are installing a disc-based workstation (ie a workstation with a disc containing RISC iX), refer to the section entitled *Setting up disc-based workstations* on page 146.

If you are installing a discless workstation (ie a workstation with no disc), refer to the section entitled *Setting up discless workstations* on page 150.

Setting up disc-based workstations

This section assumes that you have connected together every disc-based workstation as described in the *Installation Guide* and successfully installed every disc-based workstation onto the network as described in the section entitled *Setting up the network hardware* on page 142 of this Guide.

This section now describes how to set up the software for disc-based workstations on the network, which may or may not be acting as servers.

The set up procedure involves editing the following configuration files on each disc-based workstation:

- */etc/rc.config*
- */etc/rc.net*
- */etc/hosts*
- */etc/ethers*
- */etc/networks*

Editing /etc/rc.config

RISC iX workstations are initially set up to boot standalone, not attached to a network. This configuration can be altered by changing suitable lines in the file */etc/rc.config*. The lines to be changed and their initial settings are as follows:

```
...
STANDALONE=TRUE
FULLNETWORK=FALSE
NIS=FALSE
...
```

The lines to change depend on the type of network environment that you are connecting your workstation to.

If you are connecting your workstation to an Ethernet network, edit */etc/rc.config* and change the settings to:

```
...
STANDALONE=FALSE
FULLNETWORK=TRUE
NIS=FALSE
...
```

If your Ethernet network is also running the NIS distributed network database, change the settings to:

```
...
STANDALONE=FALSE
FULLNETWORK=TRUE
NIS=TRUE
...
```

For more information on setting up NIS, refer to the section entitled *NIS Installation and Administration* on page 223.

Editing /etc/rc.net

The hostname of the workstation needs to be set in this file. Look for the following line:

```
HOSTNAME=
```

The example below sets the hostname of a workstation to *saturn*:

```
HOSTNAME=saturn
```

Editing /etc/hosts

Enter the hostnames and Internet addresses of each workstation, according to the guidelines given in the section entitled *Assigning host and interface names* on page 138 and the section entitled *Assigning Internet addresses* on page 138.

The following example shows a typical entry for a local network that will not be communicating with any public domain networks. An extra entry for each workstation is also defined just in case a second interface (for example, Econet) is ever added to a workstation:

```
# Ethernet 1 network
1.0.0.1      saturn
1.0.0.2      jupiter
1.0.0.3      mars
#
# Secondary network
2.0.0.1      saturn_e
2.0.0.2      jupiter_e
2.0.0.3      mars_e
```

If your network is running NIS, then the new workstation names and Internet addresses should also be added to the */etc/hosts* file on the NIS master. The NIS *hosts* maps should then be updated. For more information on how to update the NIS database, refer to the chapter entitled *The Network Information Service* on page 219.

Editing /etc/ethers

Enter the Ethernet addresses of all hosts on the network in this file. For each host on an Ethernet, a single line should be present containing the following information:

```
# Ethernet address        hostname
00:00:a4:00:06:3a         saturn
00:00:a4:00:08:10         jupiter
00:00:a4:00:08:3c         mars
```

Again, if your network is running NIS, then the Ethernet addresses should also be added to the */etc/ethers* file on the NIS master. The NIS *ethers.byname* map should then be updated. For more information on how to update the NIS database, refer to the chapter entitled *The Network Information Service* on page 219.

For more information about */etc/ethers*, refer to *ethers*(5).

Editing /etc/networks

Enter information in this file about all the known networks which comprise the Internet network. This file, or the details that need to be placed in this file, should normally be supplied to you when you receive notification about your official Internet network name and address.

Entries in this file take the following form:

```
# Network name            Internet address
Acorntech                 192.12.206
```

Again, if your network is running NIS, then the Internet networks and addresses should be added to the */etc/networks* file on the NIS master. The NIS *networks.byaddr* map should then be updated. For more information on how to update the NIS database, refer to the chapter entitled *The Network Information Service* on page 219.

For more information about */etc/networks*, refer to *networks*(5).

Booting the workstations

Once you have configured the network software, bring all the workstations up into multi-user mode by typing <Ctrl-D>. The following start-up messages should be displayed on the workstation *saturn*:

```
ethernet configured as address saturn
...
starting network daemons: inetd.
```

If these appear, then your workstations have been successfully set up on the network.

Testing the network

To test that the workstations are operating normally on the network, use the command *ping*(8). This transmits a packet of data from one workstation to another along the network and records various statistics about how successful the transmission was.

For example, to check the connection from the workstation *mars* to the workstation *saturn*, at the workstation *mars*, type:

```
ping saturn
PING saturn: 56 data bytes
64 bytes from 1.0.0.3: icmp_seq=0. time=4. ms
64 bytes from 1.0.0.3: icmp_seq=1. time=10. ms
64 bytes from 1.0.0.3: icmp_seq=2. time=4. ms
64 bytes from 1.0.0.3: icmp_seq=3. time=5. ms
64 bytes from 1.0.0.3: icmp_seq=4. time=4. ms
64 bytes from 1.0.0.3: icmp_seq=5. time=5. ms
<Ctrl-C>
----saturn PING Statistics----
6 packets transmitted, 6 packets received, 0% packet loss
round-trip (ms) min/avg/max = 4/5/10
```

This command can also be used during normal use, for testing if a workstation that you are trying to communicate with is up. For more information, refer to *ping*(8).

Adding extra disc-based workstations

To add extra disc-based workstations to the network, all you need to do is install the workstation on the network, then edit the following set of configuration files on the new workstation, according to the instructions given in the previous sections:

- */etc/rc.config*
- */etc/rc.net*
- */etc/hosts*
- */etc/ethers*

Setting up discless workstations

This section assumes that you have connected together every discless workstation as described in the *Installation Guide* and successfully installed every discless workstation onto the network as described in the section entitled *Setting up the network hardware* on page 142.

This section now describes how to set up the software for discless workstations on the network.

On the discless workstation ...

The first part of setting up a discless machine involves setting a few configuration parameters to discover the Ethernet address of the discless workstation.

When you firstly turn the workstation on, either the RISC OS desktop will be displayed or you will see the command prompt. If the desktop is displayed press <F12> to go to the command prompt. Then type the following:

```
*Configure boot
*Configure netboot on
*Configure autoboot on
```

then press <Ctrl>–<RESET>. You should then see the following displayed on your monitor:

```
Looking for IP address ...
Physical address is address
```

where *address* is a series of digits of the form *00:00:00:00:00:00*. Note this number down, you will have to enter it in a file later in this procedure.

For more information about the meaning of these options, refer to the section entitled *Configuring the discless booting procedure* on page 158.

On the server workstation ...

Before attempting to set up or boot the discless workstation, you will need to designate another workstation on the network to act as its server. The basic requirements of a server are:

- It must be able to export the Sun Network File System, version 4.0 (with or without *secure* mount options).
- It must have enough space on its disc to hold all the necessary files and directories needed for discless booting. Approximate values are:
  - Standard RISC iX 1.2 distribution (40MB shared between all discless workstations)
  - Root partition (3MB for each discless workstation)
  - Swap file (10MB for each discless workstation)

- Enough space to store the files of users of the discless workstations.

In addition to these requirements, a number of extra facilities are required by the discless workstations:

- machine address server (*bootpd* or *revarp*)
- bootstrap code server (*tftpd*)
- boot parameters server (*bootparamd*)

All these facilities can be provided by any Acorn RISC iX workstation if the server workstation is not able to provide them. For more information about all these services and how they are used in the discless booting procedure, refer to the section entitled *How discless booting works* on page 160.

Assuming that your server meets the above requirements, the remainder of this section describes how to setup a server to support discless workstations, using *bootpd* as the address server.

The following steps need to be followed to set up a server workstation:

- Allocate a valid hostname and an Internet address for the discless workstation.
- Transfer the discless software provided on cartridge tape to the server using the command *install_fs*
- Run the command *setup_client*
- Create the file */etc/bootpd.conf*
- Edit the file */etc/inetd.conf*
- Restart *inetd*

Allocating a hostname and Internet address

To facilitate discless booting, the server workstation needs to be able to respond to a discless workstation when it requests information from the server, it needs to know its hostname and its Internet address. Therefore, a valid *hostname* and Internet address for the discless workstation needs to be added to the */etc/hosts* file of the server. For example:

```
89.0.4.23    maxwell
```

This line defines *maxwell* with an Internet address of *89.0.4.23*. This should also be added to the *hosts* database if your network is running NIS.

Transferring the discless software to the server

The discless software is supplied as three separate *tar* archives on a 1/4" cartridge tape. The first archive contains the various shell scripts that will be used to set up the workstation as a server and the final two archives contains the *root* and */usr* filesystems used by the discless workstations.

To extract the first *tar* archive, insert the cartridge tape in your local tape drive and type the following:

```
cd /usr/local/bin
tar xf /dev/ntapedevice .
```

Remembering to specify the non-rewinding tape device name. The scripts will then be taken off the tape and copied into the directory */usr/local/bin*.

Once this has completed, run the script *install_fs* from */usr/local/bin*, to extract the other two *tar* archives from the tape:

```
# install_fs /dev/ntapedevice

Path for /: /export/proto
Path for /usr: /export/usr


Ready to start unload (y/n)? y

Extracting root filesystem
Extracting /usr

Updating /etc/exports to export /export/usr.

Client Filesystems have been unloaded
Don't forget to edit /etc/inetd.conf on address server
#
```

Run the command setup_client

The command *setup_client* creates and edits the necessary files and directories on the server that are used by each discless workstation. The command needs to be used with the following arguments

```
setup_client name yp size homepath addrscheme addrserver
```

where:

name is the name of the discless workstation

nis specifies whether the discless workstation is to be part of the NIS service. This field can either be *client* to denote a NIS client or *none* to denote that it will not use the NIS service.

size specifies the size of the *swap* file that will be created for the discless workstation. The size of file you choose depends on the application that you are going to be running on the discless workstation, but in general it should be at least as big as the amount of physical memory that you have in your discless workstation.

homepath defines the pathname of the home directory that will be used by the discless workstation to store user's files. For example, */home* specifies */home* on the server workstation. *homeserver:/home* specifies */home* on the workstation called *homeserver*.

addrscheme specifies the address resolution scheme that is to be used. This can either be *bootp* or *revarp*, but is normally *bootp* (if the address server is a RISC iX workstation).

addrserver specifies the location of the address server workstation which can either be *local* or *remote*. *local* signifies the server workstation and *remote* signifies another workstation on the network. You should normally choose *local*.

If set to *local*, then the file */etc/bootparams* on the server will be automatically edited by *setup_client*. If set to *remote* then you will have to edit the */etc/bootparams* file yourself on the workstation you wish to use as the address server (*setup_client* will remind you to do this when it has finished). For more information about the *bootparams* file, refer to *bootparams*(5).

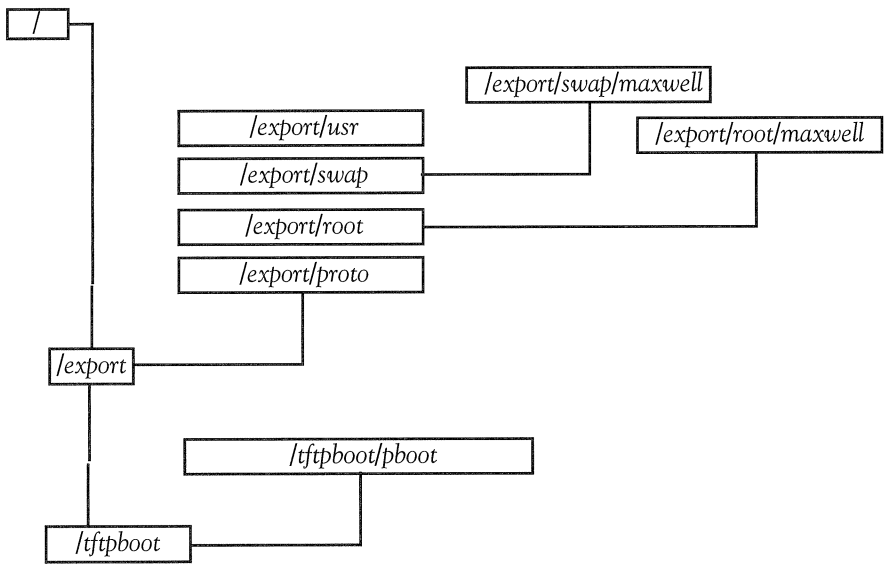
For example, here is the typical usage of *setup_client* for a discless workstation called *maxwell*:

```
# setup_client maxwell client 9M elm:/home bootp local

Installing discless client 'maxwell'
Root filesystem: /export/root/maxwell
Swap file: /export/swap/maxwell

OK to start installation (y/n)? y
```

```
Creating root filesystem.
Copying prototype directory to /export/root/maxwell
Making devices in dev
Editing files in etc

#
```

The above command sets up the discless workstation *maxwell* as a NIS client with 9MB of swap space and a home directory called */home* on the workstation *elm*. The address scheme used is *bootp*, run from the *local* workstation.

After running the two commands *install_fs* and *setup_client*, the server workstation will now contain the following files and directories:



Creating /etc/bootpd.conf

The file */etc/bootpd.conf* needs to be created on the server workstation to contain configuration information which tells the discless workstation it's Internet address and it's hostname plus the location of the discless booting program on the server workstation.

/etc/bootpd.conf is subdivided into two sections, which are delimited by a percent character (%) in column 1. Fields are delimited by one or more spaces or tabs. Lines which are empty, or which begin with the hash symbol '#' are ignored. For example, the server workstation should have the following entry in */etc/bootpd.conf* for the discless workstation *maxwell*:

```
.

unix_boot pboot
%
maxwell 1 00:00:a4:00:08:33 89.0.4.23
```

The first line in section one of the file contains a single field which specifies the default pathname for the boot file. In the above example this is given as the current directory, denoted by '.'. As no filename is specified the default directory */tftpboot* will be searched for.

All other lines have two fields, which are the generic name, and the associated boot file, which is given as *hostname:pathname*. If the optional hostname is not given (as in the above example), it defaults to the current hostname. Thus if the name of the server is *murdoch* the boot file (*pboot*) will be searched for in *murdoch:/tftpboot*.

Entries in section two of the file contain four mandatory fields which map hardware addresses of the given type to hostname and Internet address.

The first field defines the hostname of the workstation (*maxwell* in the above example). This should have already been defined in the file */etc/hosts* on the server, or in the NIS hosts database.

The second field defines the hardware address type used by the discless workstation. (This value is used by *bootp* to interpret the hardware address of the discless workstation.) In our example, *maxwell* uses 10 Mbit/s Ethernet. The hardware address type for this is 1. For information about other hardware types, refer to *bootpd.conf*(5).

The third field specifies the hardware address of the discless workstation. You should already have discovered the Ethernet address of the discless workstation, as described at the start of this section (page 150). The number should be of the form *00:00:00:00:00:00*. Enter this number in the file.

The fourth field specifies the Internet address of the discless workstation, as defined in the */etc/hosts* file on the server or as defined in the NIS hosts database.

Therefore, in the above example, the workstation with a hardware address of 00:00:a4:00:08:33 (*maxwell*) will come up as host *maxwell*, and will load its boot file from */tftpboot/pboot* on the local workstation. For more information, refer to *bootpd.conf*(5) and *bootpd*(8C).

Editing /etc/inetd.conf

The *inetd.conf* file contains the list of servers that *inetd*(8C) invokes when it receives an Internet request over a socket.

Initially the *bootp* server is commented out in this file. As *bootp* is the server that will be used to boot the discless workstation, this needs to be included. Thus the following line in */etc/inetd.conf*:

```
#bootp dgram udp wait root /usr/sbin/bootpd bootpd /etc/bootpd.conf
```

needs to be changed to:

```
bootp dgram udp wait root /usr/sbin/bootpd bootpd /etc/bootpd.conf
```

For more information, refer to *bootpd.conf*(5) and *bootpd*(8C).

bootparamd is also commented out. This needs to be included. Thus the following line:

```
#bootparam/1 dgram rpc/udp wait root /usr/sbin/rpc.bootparamd rpc.bootparamd
```

also needs to be changed to:

```
bootparam/1 dgram rpc/udp wait root /usr/sbin/rpc.bootparamd rpc.bootparamd
```

For more information, refer to *inetd.conf*(5).

Restarting inetd

inetd needs to be restarted so that it will notice the changes that have been made to the *inetd.conf* file which is read every time *inetd* starts up. The easiest way to restart *inetd* is to find the process number of *inetd* and then send the process the hang-up signal so that *inetd* is firstly killed and then restarted.

To find the process number of *inetd*, type:

```
ps t\? | grep inetd
 141 ? I 0:02 /usr/sbin/inetd
```

This shows the process id number of *inetd* to be *141*. Thus, to restart *inetd* in this example you would type:

```
kill -HUP 141
```

For more information, refer to *inetd*(8).

Booting the discless workstation

Once you have followed these steps you are now ready to try booting the discless workstation. Turn the workstation on. You should see something like the following:

```
Looking for IP address ..
Physical address is 00:00:a4:00:08:33
IP address is 89.0.4.23
Looking for NetMask ...
NetMask is ff000000, Broadcast address is 89.255.255.255
Making TFTP request to 89.0.4.41..
TFTP transfer in progress
TFTP transfer completed
Acorn RAMFS
*|> !Boot
*|
*| Boot file for discless bootstrap
*|
*| Load the NFS filer & set path for Internet files
*|
*rmload nfs
*set InetDBase$Path $.
*|
*| away we go ...
*|
*nfsboot
Loading page n
```

The above sequence downloads the discless booting program from the server workstation. After this has been downloaded, the screen will clear and the discless workstation begins to boot. Finally the login prompt will be displayed.

Adding extra discless workstations

To add extra discless workstations to the same server workstation, all you need to do is:

- Install the discless workstation on the network
- Allocate a valid hostname and an Internet address for the discless workstation.
- Run the command *setup_client*
- Edit the file */etc/bootpd.conf*
- Restart *bootpd*, if it is currently running

The first three steps of this procedure can be followed according to the instructions given in the previous sections of this chapter for setting up discless workstations.

Editing the file /etc/bootpd.conf

This file needs to be carefully edited to include the information about the new discless workstation. The first section of the file needs no editing. The second section of the file needs the following extra line added to it:

```
%
maxwell 1 00:00:a4:00:08:33   89.0.4.23
shah    1 00:00:a4:00:09:13 89.0.4.24
```

This identifies a new discless workstation to the server workstation called *shah* with an Ethernet interface (address 00:00:a4:00:09:13) and Internet address 89.0.4.24. For more information, refer to *bootpd.conf*(5).

Restarting bootpd

Once the *bootpd.conf* file has been edited, *bootpd* needs to restarted if it is running. To check if it is running, type:

```
ps t\? | grep bootpd
```

If *bootpd* does not appear to be running, then there is no need to do anything further. If *bootpd* is running then kill it, using the *kill* command. For example:

kill *pid_number*

The next time a *bootp* request is received, *inetd* will start *bootpd*.

After completing the above tasks, you should then be ready to try booting the discless workstation. Turn the workstation on. You should see the normal start-up messages as detailed in the section entitled *Booting the discless workstation* on page 157.

Configuring the discless booting procedure

There are two CMOS RAM configuration options that the discless workstation uses when booting over the network. The options are:

```
*Configure netboot
*Configure autoboot
```

The configuration options are provided to enable the discless workstation to differentiate between the *reboot* and *halt* commands that were used to shutdown the workstation:

*Configure netboot ON

If set to *on*, then when the discless workstation is brought down using *reboot*, the discless workstation automatically tries to reboot from over the network. This is known as *default booting*. When the discless workstation is brought down using *halt*, no booting is attempted.

If default booting is enabled, then the discless workstation will continue to perform a network bootstrap until either it is successful, or both <*Alt*> keys are held down simultaneously (which will cause the bootstrap to terminate).

*Configure netboot OFF

When *netboot* is set to *off*, then if the workstation is brought down using either of the commands *halt* or *reboot*, no booting of the workstation will be attempted.

*Configure autoboot ON

When *autoboot* is set to *on* (and *netboot* is also set to *on*), then if the workstation is brought down using either of the commands *halt* or *reboot*, it will be rebooted. (Unless *halt* is used with the *–RISCOS* option.)

*Configure autoboot OFF

When *autoboot* is set to *off* (regardless of the setting of *netboot*), then if the workstation is brought down using either of the commands *halt* or *reboot*, no booting of the workstation will be attempted.

*Configure netboot <boot suffix>

netboot can also be configured to enable the discless workstation to boot from an alternative boot file, where the *boot suffix* is a character in the range {A-Za-z} which is used as an extension to the base boot file name. For example, typing:

```
*Configure netboot x
```

will cause the boot file *pbootx* to be loaded rather than *pboot*. For more information about the base boot file name, refer to the section entitled *How discless booting works* on page 160.

Overriding the Configuration options

To override the settings of the two configuration options and boot the discless workstation, just type the command:

```
*netboot
```

and the discless workstation will attempt to boot over the network.

How discless booting works

This section describes in a bit more detail how the discless booting process actually works. This description should help you sort out any problems that you may be having with installing any discless workstations or alternatively assist you in changing how you have configured your servers and discless workstations.

To support discless workstations two classes of network service are required:

- A filing system.
- Information services to allow the discless workstation to configure itself and find its file space.

The files that will be present on a server supporting a discless workstation (assuming that it holds the boot code and the filesystems for the discless workstations), are as follows:

- */tftpboot* – is the directory containing the first stage bootstrap modules, loaded by the code contained in the bootstrap ROM.

  With *bootpd* the file names (and indeed their location) can be chosen by you. However, if *tftpd* is running with the *-s* secure option (as defined in */etc/bootpd.conf*) then *pboot* will live in this directory.

  If *revarp* is used, then there will be files contained in here of the form:

  `nnnnnnnn.ARM`

  where *n* stands for a character in the range 0-9, A-F. It is likely that the bootstrap program (*pboot*) will be the same for most workstations, so it is probable that the server workstation will have each of these names linked to *pboot*. The file will therefore just be a symbolic link to the *pboot* file in the directory – one file per discless workstation supported by the server.

- */export* – the directory holding the files used by the discless workstations. This consists of:
  - */export/proto* – is the directory holding the RISC iX release. The script *setup_client* extracts files from here as appropriate when creating the root filesystem for a new discless workstation.
  - */export/root/<hostname>* – is the root filesystem that is used by the discless workstation with hostname *hostname*. The default kernel loaded into the discless workstation will be */export/root/<hostname>/vmunix*.
  - */export/swap/<hostname>* – is the swap file used by the discless workstation with hostname *hostname*.

The discless booting procedure

After the primary bootstrap has completed, the RISC OS RAM filing system will have been initialised with files, and if RAMFS is the CMOS RAM configured filing system, and if *!Boot* is present, then it will be executed when RISC OS initialisation is complete. This is how the secondary bootstrap is initiated.

Discless booting requirements

To enable the boot process to proceed, the following things must be set up correctly:

- The configuration file */etc/bootparams* which is used by the server *rpc.bootparamd*
- If *bootpd* is used, the configuration file */etc/bootpd.conf*.
- The name/address of the discless workstation must be entered into the hosts database that is used on the network.
- The filing system(s) required by the client must be exported from the server.

In addition to these requirements, a number of extra facilities need to be provided on the network. These facilities are required by the discless workstations whenever they are rebooted:

- machine address server (*bootpd* or *revarp*)
- bootstrap code server (*tftpd*)
- boot parameters server (*bootparamd*)

All these facilities can be provided by any Acorn RISC iX workstation if the server workstation is not able to provide them. The following sections describe these functions in more detail.

- *Machine Address Server* – the firmware bootstrap on the discless workstation determines the Internet identity by issuing both *bootp* and *revarp* Internet requests. You can specify which Internet number is to be associated with each Ethernet address (unique for each Ethernet expansion card) in the relevant file (or via NIS). This file is referenced by the *bootpd* or *revarpd* daemons.

  If neither of these services are available from the server workstation, then any Acorn RISC iX 1.2 workstation can be configured to run *bootpd* to supply this service.

- *Bootstrap Object server* – once the firmware has obtained the Internet identity, it will broadcast a request for a single bootstrap object from a TFTP server. The name of the object requested is either the filename returned with the *bootp* response, or (in the case of a *revarp* response) the file

/tftpboot/<nnnnnnnn>.ARM<postfix>, where *nnnnnnnn* is the Internet address of the discless workstation as returned from the *revarp* server, in character form using the characters *0-9*, *A-F*.

Note that this bootstrap object is an encapsulation of:

- The *nfsboot* discless bootstrap program
- The RISC OS NFS module
- Some RISC OS executable files (eg *!Boot*)

If the main fileserver does not (or does not wish to) support broadcast bootstrap TFTP requests, then any Acorn RISC iX 1.2 workstation can be configured to run *tftpd*.

- *Bootstrap configuration server* – when the kernel has been loaded into the discless workstation it will request configuration parameters from a *bootparamd* server. You specify (in a text file) or from NIS the location for each discless workstation of the directory to be mounted as *root* and the file to be used for swapping.

  If this service is not available from the server workstation, then any Acorn RISC iX 1.2 workstation can be configured to run *bootparamd* to supply this service.

For more information about maintaining workstations that have been set up on a network, refer to the chapter entitled *The Network Filesystem Service* on page 185.

How NFS works – An overview

Introduction

The next few sections define some of the terms used in NFS and explains some of the underlying concepts behind the operation of NFS.

Terminology

A workstation that provides resources to the network is called a *server*. A single workstation may provide more than one service. In fact, a typical configuration would be for one workstation to act as an NFS server and a NIS server. In each of the Network Services, servers are entirely passive. The servers wait for clients to call them, they never call the clients.

A workstation that employs the resources provided by a server is called a *client*. A workstation may be both a server and a client, and when NFS resources (files and directories) are at issue, often is.

A person logged in on a client workstation is a *user*, while a program or set of programs that run on a client is an *application*.

The Network File System (NFS) is an operating system-independent service which allows users to mount directories, even root directories, across the network, and then to treat those directories as if they were local. There is also an option for a secure mount using an encryption standard for authentication of user and host. For more information, see the chapter entitled *Secure networking* on page 269.

The Network Information Service(NIS) is a network service designed to ease the job of administering the large networks that NFS encourages. The NIS is a replicated, read-only, database service. Network file system clients use it to access network-wide data in a manner that is entirely independent of the relative locations of the client and the server. The NIS database typically provides password, group, network and host information for all the workstations on the network.

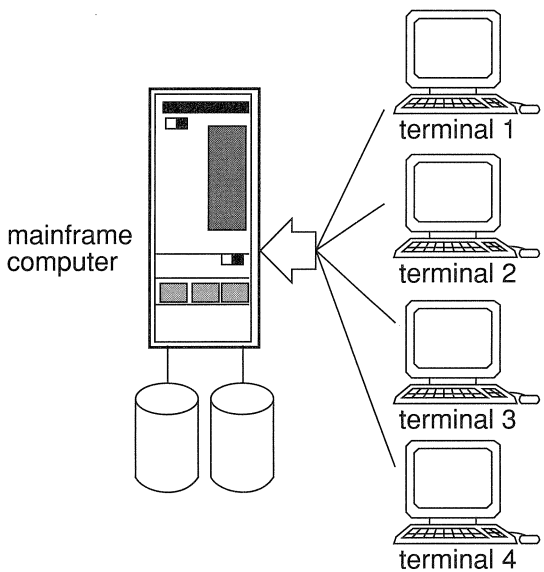
The *Secure Networking Service* provides a means to greatly improve the security of network environments. The service is general enough to be used by other UNIX and non-UNIX systems.

NFS environment

NFS uses Internet protocols plus some advanced communications protocols to provide a facility for sharing files in a heterogeneous environment of workstations, operating systems and networks.

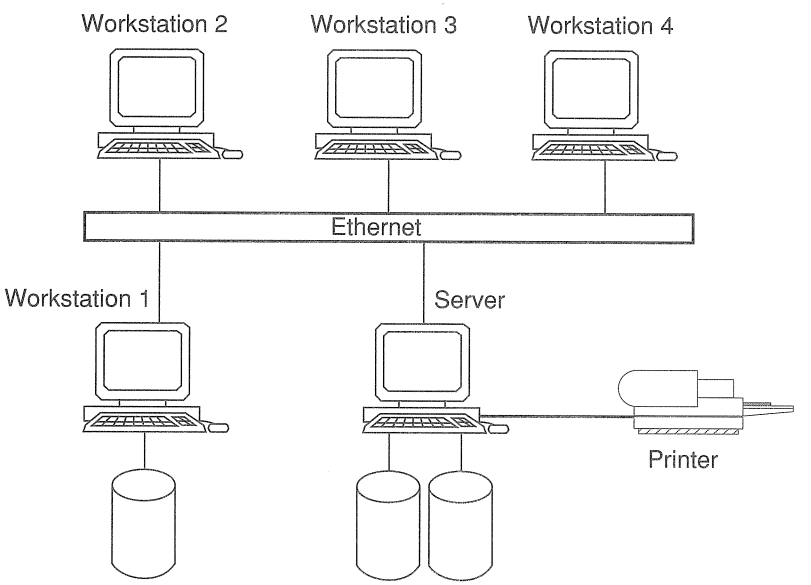
Sharing files is accomplished by mounting a remote filesystem, then reading or writing files in place. The NFS is open-ended, and users are encouraged to interface it with other systems.

The traditional time-sharing environment looks like this:



The major problem with this environment is competition for CPU cycles; users on each of the four terminals are competing for access to the discs on the mainframe. The result is a slow response time for all users.

The workstation environment solves that problem, but requires more disc drives. A network environment looks like this:



NFS makes all discs available as needed. Individual workstations have access to all information residing anywhere on the network. Printers and supercomputers may also be available somewhere on the network.

Example NFS usage

This section gives three typical examples of NFS usage, which highlight the benefits of working in an NFS environment.

Example 1: Mounting a remote filesystem

Suppose that from your workstation (named *client*) you wish to read some on-line manual pages, and that these pages are not available on your server workstation, named *server* but are available on another workstation on the network named *docserv* in the directory */usr/man*.

To mount the directory containing the manual pages, firstly ensure that a directory called */usr/man* exists on the client workstation. If it doesn't, create it. Then simply type:

```
client# /sbin/mount -o ro docserv:/usr/man /usr/man
```

where the directory */usr/man* exists on the client workstation.

Now you can use the *man* command whenever you want. Try running the *mount -p* command on *client* after you've mounted the remote filesystem. The output will look something like this:

```
server:/roots/client   /            nfs  rw,hard       0 0
server:/usr            /usr         nfs  ro            0 0
server:/home/server    /home/server nfs  rw,bg         0 0
server:/usr/local      /usr/local   nfs  ro,soft,bg    0 0
docserv:/usr/man       /usr/man     nfs  ro            0 0
```

You can remote mount not only filesystems, but also directory hierarchies inside filesystems. In this example */usr/man* is not a filesystem root on *docserv*, it's just a subdirectory within the */usr* filesystem.

Example 2: Exporting a filesystem

Suppose that you and a colleague need to work together on a programming project. The source code is on your workstation, in the directory */usr/proj*. Suppose that after creating the proper directory your colleague tried to remote mount your directory. Unless you have explicitly exported the directory, your colleague's remote mount will fail with a 'access denied' error message. This means that the directory has not been exported for mounting by other workstations.

To export a directory, you need to edit the */etc/exports* file. If your colleague is on a workstation named *cohort* then you need to run *exportfs*(8), after putting the following line in */etc/exports*:

```
/usr/proj   -access=cohort
```

If no explicit access is given for a directory, then the system allows anyone on the network to remote mount your directory. By giving explicit access to *cohort* you have denied access to others. (For more details about the */etc/exports* file see *exports*(5).)

The NFS mount request server *mountd* (see the section entitled *The NFS interface* on page 172) reads the */etc/xtab* file whenever it receives a request for a remote mount. Now your colleague can remote mount the source directory by issuing the command:

```
cohort# /sbin/mount client:/usr/proj /usr/proj
```

This, however, isn't the end of the story, since NFS requests are also checked every time a read or write request is issued. If you do nothing, the accesses that you've established in your */etc/exports* file will stay in effect, but you (and your programs) are free to change them at any time with the *exportfs* command. For more information, refer to *exportfs*(8).

Example 3: Administering a server workstation

You need to know how to set up NFS server workstations so that client workstations can mount all the necessary filesystems. You export filesystems (that is, make them available for mounting) by placing appropriate lines in the */etc/exports* file. Here is a sample */etc/exports* file for a typical server workstation:

```
/                   -access=systems
/exec               -access=engineering:joebob:shilling
/usr                -access=engineering
/home/server        -access=engineering
/home/local.ac1     -access=engineering:athena
/home/local.ac2     -access=engineering
```

Workstation names or net groups, such as *staff* (see *netgroup*(5)) may be specified after the filesystem, in which case remote mounts are limited to workstations that are a member of this net group. For the complete syntax of the */etc/exports* file, see *exports*(5).

At any time, you can see which filesystems are remote mounted using the *showmount*(8) command.

NFS Architecture

There are a number of advantages to be gained by using NFS. These are discussed in this section.

Transparent information access

Users are able to get directly to the files they want without knowing the network address of the data. To the user, all NFS-mounted filesystems look just like private discs. There's no apparent difference between reading or writing a file on a local disc, and reading or writing a file on a disc in the next building. Information on the network is truly distributed.

Different workstations and operating systems

No single vendor can supply tools for all the work that needs to get done, so appropriate services must be integrated on a network. NFS provides a flexible, operating system-independent platform for such integration.

Easily extensible

A distributed system must have an architecture that allows integration of new software technologies without disturbing the extant software environment. Since the NFS network-services approach does not depend on pushing the operating system onto the network, but instead offers an extensible set of protocols for data exchange, it supports the flexible integration of new software.

Ease of network administration

The administration of large networks can be complicated and time-consuming, yet they should (ideally) be *at least* as easy to administer as a set of local filesystems on a time-sharing system. The UNIX system has a convenient set of maintenance commands developed over the years, and the Network Information Service (NIS) has allowed them to be adapted and extended for the purpose of administering a network of workstations.

The NIS also allows certain aspects of network administration to be centralised. The NIS interface is implemented using RPC and XDR, so it can also be used with non-UNIX operating systems. NIS servers do not interpret data, so it is easy for new databases to be added to the NIS service without modifying the servers. NIS is discussed more fully in the chapter entitled *The Network Information Service* on page 219.

Reliability

NFS's reliability derives from the robustness of the 4.3BSD filesystem, from the stateless NFS protocol† and from the daemon-based methodology by which network services like file and record locking are provided. See the chapter entitled *The Network Filesystem Service* on page 185 for more details on locking.

In addition, the file server protocol is designed so that client workstations can continue to operate even when the server crashes and reboots. Continuation is achieved after reboot without making assumptions about the reliability of the underlying server hardware.

The major advantage of a stateless server is robustness in the face of client, server, or network failures. Should a client fail, it is not necessary for a server (or System Administrator) to take any action to continue normal operation. Should a server or the network fail, it is only necessary that clients continue to attempt to complete NFS operations until the server or network gets fixed. This robustness is especially

---

†The NFS protocol is *stateless* because each transaction stands on its own. The server doesn't have to remember anything – about clients or files – between transactions.

important in a complex network of heterogeneous systems, many of which are not under the control of a disciplined operations staff, and which may be running untested systems that may be often rebooted without warning.

High performance

The flexibility of the NFS allows configuration for a variety of cost and performance trade-offs. For example, configuring servers with large, high-performance discs, and clients with no discs, may yield better performance at lower cost than having many workstations with small, inexpensive discs. Furthermore, it is possible to distribute the filesystem data across many servers and get the added benefit of multiprocessing without losing transparency. In the case of read-only files, copies can be kept on several servers to avoid bottlenecks.

There are also additional performance enhancements to the NFS, such as 'fast paths' for key operations, asynchronous service of multiple requests, disc-block caching, and asynchronous read-ahead and write-behind. The fact that caching and read-ahead occur on both client and server effectively increases the cache size and read-ahead distance. Caching and read-ahead do not add state to the server; nothing (except performance) is lost if cached information is thrown away. In the case of write-behind, both the client and server attempt to flush critical information to disc whenever necessary, to reduce the impact of an unanticipated failure; clients do not free write-behind blocks until the server confirms that the data is written.

The NFS implementation

In the NFS implementation, there are three entities to be considered:

- the operating system interface
- the *virtual file system* (VFS) interface, and
- the network file system (NFS) interface.

The operating system interface

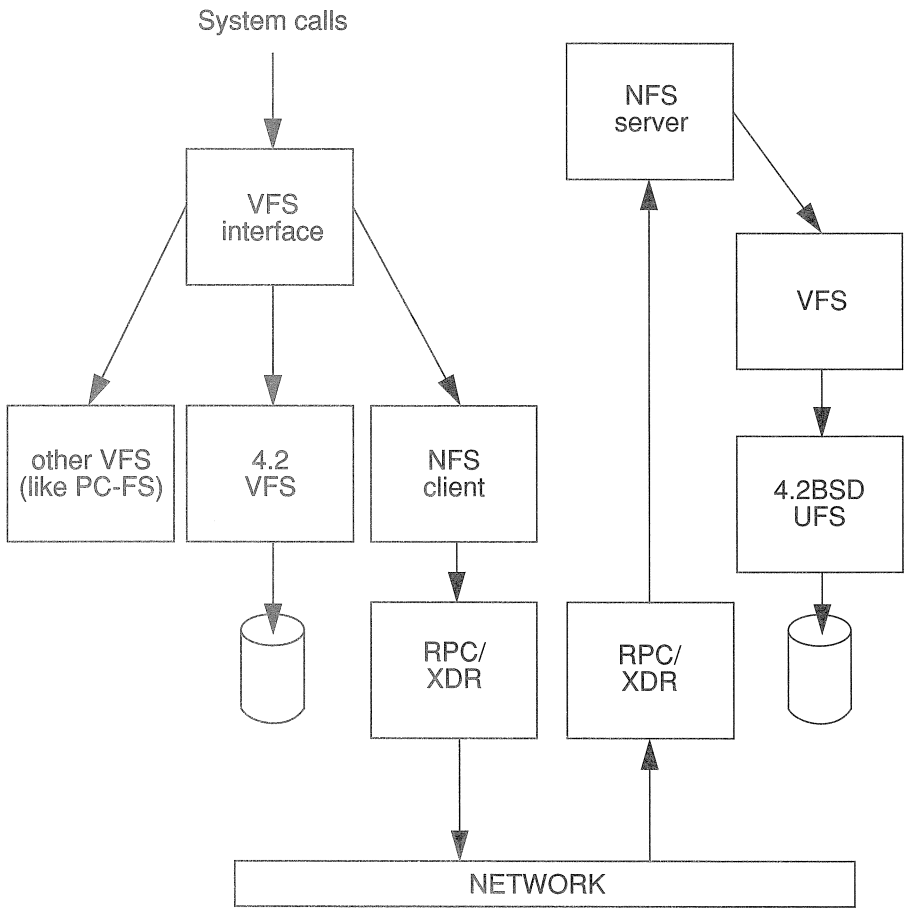
The UNIX operating system interface has been preserved in the implementation of the NFS, thereby ensuring compatibility for existing applications.

The Virtual File System interface

The VFS is best seen as a layer that wraps around the traditional UNIX filesystem. This traditional filesystem is composed of directories and files, each of which has a corresponding *inode* (index node), containing administrative information about the file, such as location, size, ownership, permissions, and access times. For more information about the composition of the filesystem refer to the chapter entitled *Finding out about the filesystem* on page 21.

Inodes are assigned unique numbers within a filesystem, but a file on one filesystem could have the same number as a file on another filesystem. This is a problem in a network environment, because remote filesystems need to be mounted dynamically, and numbering conflicts would cause havoc. To solve this problem, the VFS was designed, based on a data structure called a *vnode*. In the VFS, files are guaranteed to have unique numerical designators, even across a network. Vnodes cleanly separate filesystem operations from the semantics of their implementation. Above the VFS interface the operating system deals in *vnodes*, below this interface the filesystem may or may not implement *inodes*. The VFS interface can connect the operating system to a variety of filesystems (for example, 4.2 BSD or MS-DOS). A local VFS connects to filesystem data on a local device.

The remote VFS defines and implements the NFS interface on the basis of the RPC and XDR mechanisms. The figure below shows the flow of a request from a client (at the top left) to a collection of filesystems.



In the case of access through a local VFS, requests are directed to filesystem data on devices connected to the client workstation. In the case of access through a remote VFS, the request is passed through the RPC and XDR layers onto the net. In the

current implementation, the UDP/IP protocols are used along with Ethernet. On the server side, requests are passed through the RPC and XDR layers to an NFS server; the server uses *vnodes* to access one of its local VFSs and service the request. This path is retraced to return results.

NFS provides five types of transparency:

- *Filesystem Type* – the *vnode* in conjunction with one or more local VFSs (and possibly remote VFSs) permits an operating system (hence client and application) to interface transparently to a variety of filesystem types.
- *Filesystem Location* – since there is no differentiation between a local and a remote VFS, the location of filesystem data is transparent.
- *Operating System Type* – the RPC mechanism allows interconnection of a variety of operating systems on the network, and makes the operating system type of a remote server transparent.
- *workstation Type* – the XDR definition facility allows a variety of workstations to communicate on the network and makes the workstation type of a remote server transparent.
- *Network Type* – RPC and XDR can be implemented for a variety of transport protocols, thereby making the network type transparent.

Simpler NFS implementations are possible at the expense of some advantages of this version. In particular, a client (or server) may be added to the network by implementing one side of the NFS interface.

Users at client workstations with discs can arrange to share over the NFS without having to appeal to you as a System Administrator to configure a different system on their workstation.

The NFS interface

As mentioned in the preceding section, a major advantage of the NFS is the ability to mix filesystems. In keeping with this, other vendors are encouraged to develop products to interface with these Network Services. RPC and XDR have been placed in the public domain, and serve as a standard for anyone wishing to develop applications for the network. Furthermore, the NFS interface itself is open and can be used by anyone wishing to implement an NFS client or server for the network.

The NFS and the Mount Protocol

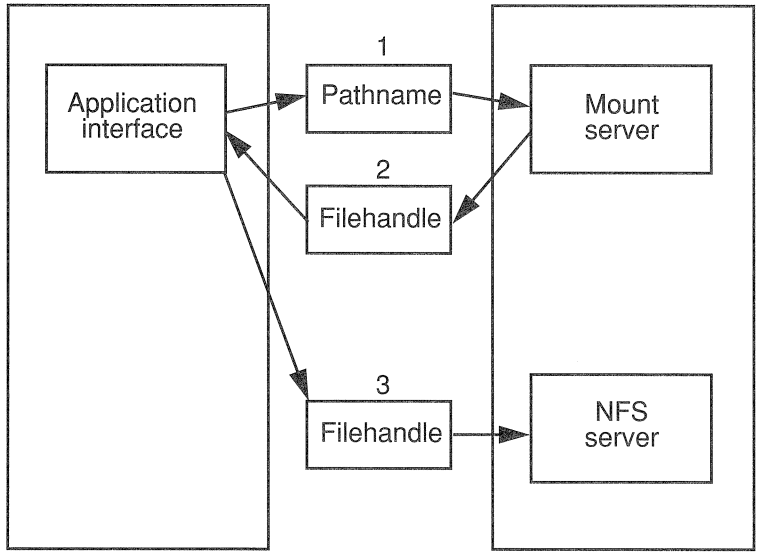
The NFS interface defines traditional filesystem operations for reading directories, creating and destroying files, reading and writing files, and reading and setting file attributes. The interface is designed so that file operations address files with an

uninterpreted identifier called a *filehandle*, a starting byte address, and a length in bytes. NFS *never* deals with pathnames, only with filehandles. It gets those filehandles from mount.

More precisely, NFS *never* interprets pathnames. Some NFS procedures take pathname arguments, but they are just strings to NFS.

Given a filehandle for a directory, a client program can use NFS procedures to get other filehandles and thereby navigate throughout the directories and files of a filesystem. A client must, however, get its first filehandle for a filesystem by using RPC to call the mount server. Mount will return a filehandle that grants access to the filesystem.

The diagram below illustrates the interaction between a client program, a mount server, and an NFS server. Note that the only interface between a mount server and an NFS server is a common filehandle.



The sequence of operations that occur in the above diagram are:

1 Client sends pathname to mount server

2. Mount server returns corresponding filehandle

3. Client sends filehandle to NFS server to subsequently access that path

Pathname Parsing

Although many operating systems have analogies to the hierarchical NFS directory and file structure, the conventions used by operating systems to formulate pathnames vary considerably. To accommodate the many possible path naming conventions, the mount procedure is not defined in the NFS protocol but in a separate mount protocol.

At present, there is one mount protocol, the UNIX mount protocol, but others can be defined as necessary. The *mount* procedure in the UNIX mount protocol converts a UNIX pathname into a filehandle. If local pathnames can be reasonably mapped to UNIX pathnames, an NFS server developer may wish to implement the UNIX mount protocol, even though the server runs on a different operating system. This approach makes the server immediately usable by clients that use the UNIX protocol and eliminates the need to develop a new *mount* command for UNIX-based clients.

The mount protocols remove pathname parsing from the NFS protocol, so that a single NFS protocol can work with multiple operating systems. This means that users and client programs need to know the details of a server's path naming conventions only when mounting a filesystem. Different server path naming conventions therefore typically have little impact on users.

Because mounts are relatively infrequent operations, mount servers can be implemented outside of operating system kernels without materially affecting overall file system performance. Because user-level code is easier to write and far easier to debug than kernel code, mount servers are fairly simple to put together.

Export and Mount Lists

Technically, a mount protocol needs to define only a *mount* procedure that bootstraps the first filehandle for a filesystem. (By convention, a mount protocol should also define a NULL procedure). However, adding other procedures can simplify network management. As a convenience to clients, a mount protocol might provide a procedure that returns a list of filesystems exported by a server. Another useful item is a mount list, a list of clients and the pathnames they have mounted from the server. The UNIX mount protocol defines a mount list and a procedure called *readmount()* that returns the list. With the help of *readmount* you can notify the clients of a server that is about to be shut down.

Note that a mount list makes a mount server stateful. Recall, however, that the business of a mount server is to translate pathnames into filehandles; the state represented by a mount list does not affect a server's ability to operate correctly.

Neither servers nor clients need take any action to update or rebuild a mount list after a crash. Mount server users should regard the mount and export lists provided by a mount server as "accessories" that are usually, but not necessarily, accurate.

UNIX Mount Protocol Procedures

The mount protocol consists of the six remote procedures listed below. The *mount()* procedure transforms a UNIX pathname into a filehandle which the client can then pass to the associated NFS server. The pathname passed to the mount procedure usually refers to a directory, often the root directory of a filesystem, but it can name a file instead.

In addition to returning the filehandle, *mount* adds the client's host name and the pathname to its mount list. The *readmount()* procedure returns the server's mount list. *unmount()* removes an entry from the server's mount list and *unmountall()* removes all of a client's mount list entries. The *readexport()* procedure returns the server's export list.

Number	Name	Description
0	null	Do nothing
1	mount	Return filehandle for pathname
2	readmount	Return mount list
3	unmount	Remove mount list entry
4	unmountall	Clear mount list
5	readexport	Return export list

A Stateless Protocol

The NFS interface is defined so that a server can be *stateless*. This means that a server does not have to remember from one transaction to the next anything about its clients, transactions completed or files operated on. For example, there is no *open()* operation, as this would imply state in the server; of course, the UNIX interface uses an *open()* operation, but the information in the UNIX operation is remembered by the client for use in later NFS operations.

An interesting problem occurs when a UNIX application unlinks an open file. This is done to achieve the effect of a temporary file that is automatically removed when the application terminates. If the file in question is served by the NFS, the call to *unlink()* will remove the file, since the server does not remember that the file is open. Thus, subsequent operations on the file will fail. In order to avoid state on the server, the client operating system detects the situation, renames the file rather than unlinking

it, and unlinks the file when the application terminates. In certain failure cases, this leaves unwanted "temporary" files on the server; these files are removed as a part of periodic filesystem maintenance.

Another example of the advantages gained by having the NFS interface to the UNIX system without introducing state is the *mount* command. A UNIX client of the NFS "builds" its view of the filesystem on its local devices using the *mount* command; thus, it is natural for the UNIX client to initiate its contact with the NFS and build its view of the filesystem on the network with an extended *mount* command. This *mount* command does not imply state in the server, since it only acquires information for the client to establish contact with a server. The *mount* command may be issued at any time, but is typically executed as a part of client initialization. The corresponding *umount* command is only an informative message to the server, but it does change state in the client by modifying its view of the filesystem on the network.

The major advantage of a stateless server is robustness in the face of client, server or network failures. Should a client fail, it is not necessary for a server (or human administrator) to take any action to continue normal operation. Should a server or the network fail, it is only necessary that clients continue to attempt to complete NFS operations until the server or network is fixed. This robustness is especially important in a complex network of heterogeneous systems, many of which are not under the control of a disciplined operations staff and may be running untested systems and/or may be rebooted without warning.

An NFS server can be a client of another NFS server. However, a server will not act as an intermediary between a client and another server. Instead, a client may ask what remote mounts the server has and then attempt to make similar remote mounts. The decision to disallow intermediary servers is based on several factors. First, the existence of an intermediary will impact the performance characteristics of the system; the potential performance implications are so complex that it seems best to require direct communication between a client and server. Second, the existence of an intermediary complicates access control; it is much simpler to require a client and server to establish direct agreements for service. Finally, disallowing intermediaries prevents cycles in the service arrangements; this is preferred to detection or avoidance schemes.

The NFS currently implements UNIX file protection by making use of the authentication mechanisms built into RPC. This retains transparency for clients and applications that make use of UNIX file protection. Although the RPC definition allows other authentication schemes, their use may have adverse effects on transparency.

Note that the NFS, although very UNIX-like, is *not* a UNIX filesystem per se – there are cases in which its behaviour differs from that which would be expected of the UNIX system proper:

- The guaranteed APPEND_MODE is the most striking of these differences, for it simply is not supported by NFS.

  Note that network access to devices such as tape drivers, using the terminology */dev/<device-name>* is a good idea, but is not available under NFS. It is best implemented as a separate network service whose requirement for stateful operation is kept separate from network access to files.

- There are also minor incompatibilities between NFS and UNIX file-system interfaces that are dictated by the very nature of remote NFS mounts. For example, a local NFS system simply can't tell that a remote disc partition is full until the remote NFS system tells it so. Rather than wait for a positive confirmation on every write – a strategy that would impose unacceptable performance problems – the local NFS code caches writes and returns to its caller. If a remote error occurs, it gets reported back as soon as possible, but not as immediately as would a local disc.

File locking and other inherently stateful functionality has been omitted from the base NFS definition. In this way, a simple, general interface has been preserved that can be implemented by a wide variety of customers.

File locking has been provided as a NFS-compatible network service. Many other features that inherently imply state and/or distributed synchronization may also be provided this way in future. These features, too, will be kept separate from the base NFS definition. In any case, the open nature of the RPC and NFS interfaces means that customers and users who need stateful or complex features can implement them "beside" or "within" NFS.

Non-NFS Network Operations

A small number of non-NFS networking operations are also supported that are useful for temporary inter-host connections, isolated file transfers, and access to non-UNIX systems. These operations include *rcp*, *rlogin*, *rsh*, *ftp*, *telnet* and *tftp*.

- *rcp* is a remote copy utility program that uses BSD networking facilities to copy files from one workstation to another. The *rcp* user supplies the path name of a file on a remote workstation, and receives a copy of the file in return. Access control is based on the client's login name and host name.

The major problem with *rcp* is that it's not transparent to the user, who gets a redundant copy of the transferred file. With the NFS, by contrast, only one copy of the file is necessary.

Another problem is that *rcp* does nothing but copy files. To use it as a model for additional network services would be to introduce a remote command for every regular command: for example, *rdiff* to perform differential file comparisons across workstations. By providing for the sharing of filesystems, NFS makes this unnecessary.

- *rlogin* allows the user to log in to a remote workstation, directly accessing both its processor and its mounted file systems. It remains useful in NFS-based networks because, with it, users can directly execute commands on remote workstations over the network.
- *rsh* allows the user to execute a command on a remote workstation. If no command is specified, *rsh* is equivalent to *rlogin*. Unlike other remote execution utilities, *rsh* does not make a great effort to copy the users local environment to the remote workstation before executing the command.
- *ftp* is very much like *rcp* in that it supports file copying between workstations. However, *ftp* is more general than *rcp* and is not restricted to copies between two UNIX systems. This makes it one of the most useful networking programs.
- *telnet* communicates with another host using the TELNET protocol. It isn't used much because *rlogin* is the standard mechanism for local inter-host communication. However, it is useful for logging into non-UNIX systems.
- *tftp* is like *ftp* except that it is simpler and less reliable. This is because *tftp* transfer protocol is very simple; it is less robust than *ftp* protocol, and offers fewer options. But it is still used for certain network operations like discless booting.

The above commands are described in the chapter entitled *Networking and NFS* in the *RISC iX User Guide*.

The Portmapper

Client programs need a way to find server programs; that is, they need a way to look up and find the *port* numbers of server programs†. Network transport services do not provide such a service; they merely provide process-to-process message transfer across a network. A message typically contains a transport address which contains a network number, a host number, and a port number. (A port is a logical communications channel in a host – by waiting on a port, a process receives messages from the network).

How a process waits on a port varies from one operating system to the next, but all provide mechanisms that suspend a process until a message arrives at a port. Thus, messages are not sent across networks to receiving processes, but rather to the ports at which receiving processes wait for messages. Ports are valuable because they allow message receivers to be specified in a way that is independent of the conventions of the receiving operating system.

The portmapper protocol defines a network service that provides a standard way for clients to look up the port number of any remote program supported by a server. Because it can be implemented on any transport that provides the equivalent of ports, it provides a single solution to a general problem that works for all clients, all servers and all networks.

Port registration

Every portmapper on every host is associated with port number 111. The portmapper is the only network service that must have such a well-known (dedicated) port. Other network services can be assigned port numbers statically or dynamically so long as they register their ports with their host's portmapper. For example, a server program based on the RPC library typically gets a port number at run time by calling an RPC library procedure.

Note that a given network service can be associated with port number 256 on one server and with port number 885 on another; on a given host, a service can be associated with a different port every time its server program is started.

Delegating port-to-remote program mapping to portmappers also automates port number administration. Statically mapping ports and remote programs in a file duplicated on each client would require updating all mapping files whenever a new

---

†The naming of services by way of the port-number segment of their IP address is mandated by the Internet protocols. Given this, clients face the problem of determining which ports are associated with the services they wish to use

remote program was introduced to a network. (The alternative of placing the port-to-program mappings in a shared NFS file would be too centralized, and if the fileserver went down the whole network would go down with it).

The port-to-program mappings which are maintained by the portmapper server are called a *portmap*. You can use the command *rpcinfo*(8) to view these mappings.

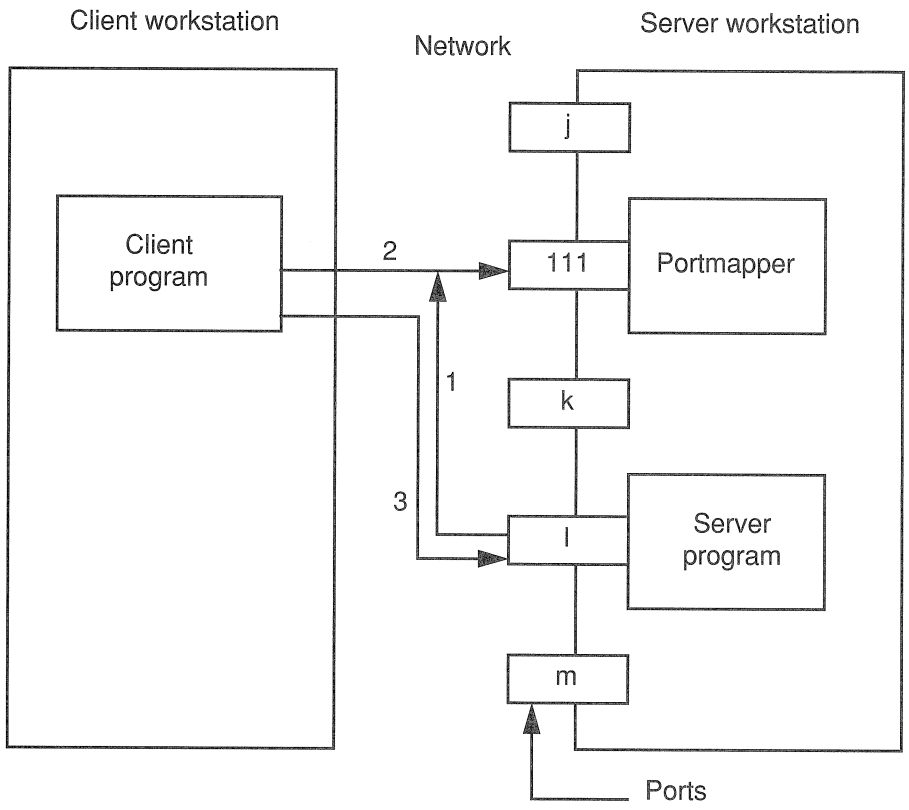
The portmapper is started automatically whenever a workstation is booted. As shown in the following diagram, both server programs and client programs call portmapper procedures†. As part of its initialization, a server program calls its host's portmapper to create a portmap entry. Whereas server programs call portmappers to update portmap entries, clients call portmappers to query portmap entries.

To find a remote program's port, a client sends an RPC call message to a server's portmapper; if the remote program is supported on the server, the portmapper returns the relevant port number in an RPC reply message. The client program can then send RPC call messages to the remote program's port. A client program can minimize its portmapper calls by caching the port numbers of recently called remote programs.

---

†Although client and server programs and client and server workstations are usually distinct, they need not be. A server program can also be a client program, as when an NFS server calls a portmapper server. Likewise, when a client program directs a "remote" procedure call to its own workstation, the workstation acts as both client and server

Note that the portmapper provides an inherently stateful service because a portmap is a set of associations between registrants and ports.



The sequence of operations that occur in the diagram are:

1 Server registers with the portmapper

2. Client gets servers port from the portmapper

3. Client calls server

The portmapper protocol (for details, refer to the *Remote Procedure Calls protocol specification* in the *RISC iX Programmer's Reference Manual - Volume 2*) provides a procedure, *callit* by which the portmapper can assist a client in making a remote procedure call.

A client program passes the target procedure's program number, version number, procedure number (for a discussion of these numbers, refer to the *Remote Procedure Call Programming Guide* in the *RISC iX Programmer's Reference Manual - Volume 2*) and arguments in an RPC call message. *callit()* looks up the target procedure's port number in the portmap and sends an RPC call message to the target procedure including in it the arguments received from the client. When the target procedure returns results *callit()* returns the results to the client program; also returned is the target procedure's port number so the client can subsequently call the target procedure directly.

Note that, because every instance of a remote program can be mapped to a different port on every server, a client has no way to broadcast a remote procedure call directly. However, the portmapper *callit()* procedure can be used to broadcast a remote procedure call indirectly, since all portmappers are associated with port number 111. One way for a client to find a server running a remote program is to broadcast a call to *callit* asking it to call procedure 0 (by convention, the no-operation request) of the desired remote program. If this call is broadcast to all servers, the first reply received is likely to be from the server with the lightest workload.

The RPC library provides an interface to all portmapper procedures. Some of the RPC library procedures also call portmappers automatically on behalf of client and server programs.

Exporting and Mounting

NFS performs two major functions: mounting and exporting. This section presents some of the general concepts concerning these functions. It also explains how various daemons interact to enable exporting and mounting.

Servers and Exporting

When a server *exports* file systems, it is essentially advertising the directories on its disc(s) that it will permit other computers to access. The server's */etc/exports* file lists these available directories, which clients are allowed to access them, and any access restrictions that must be applied. When you boot an NFS server, the */etc/rc* script automatically runs a program called *exportfs*. This program then looks at the */etc/exports* file and informs the server's kernel about the permissions applicable to each exported file system.

You can also explicitly export and un-export a file system after the server is up by using the *exportfs*(8) command. Explicit exporting is explained in the chapter entitled *The Network Filesystem Service* on page 185.

Clients and Mounting

Clients access files on the server by *mounting* the server's exported directories. When a client mounts a directory, it does not make a copy of that directory. (Do this by using the *rcp* or *tftp* commands.) Rather, the mounting process uses a series of remote procedure calls to enable a client to transparently access the directories on the server's disc.

RPCs running on the server receive the information in XDR format, and translate it to a form recognizable to the server's kernel. The kernel then processes the information and permits (or does not permit) the client to mount the file system.

Once a client mounts a remote directory, it appears to the users that all they have to do to go to a mounted directory is to type *cd* and the directory's pathname, just as they would for a local directory.

The process by which a client locates a server that exports the information it wants, then sets up communication between itself and that server, is called *binding*. NFS binding occurs during an NFS mount. (Clients also initiate binding to NIS servers, as explained in the chapter entitled *The Network Information Service* on page 219).

A client can mount a directory when it boots, or can explicitly mount a directory when a user issues a *mount* command. The */etc/fstab* file lists all file systems that the client mounts at boot time. You can also explicitly mount a file system during a work session using the *mount* and *umount* commands.

What happens when a client mounts a directory

The following list briefly summarizes the activities that take place when an NFS client mounts a directory. Its purpose is to introduce you to the daemons and system programs that handle NFS requests. On a server, NFS service is controlled by the *exportfs* system program and the *rpc.mountd* and *nfsd* daemons. On a client, NFS service is handled by the *mount* system program and a number of *biod* daemons.

Note that it is not critical for you to understand these daemons in depth. You simply need to know that they exist, because you may have to restart them if they stop after a crash. Should you want to know more about a particular daemon, refer to its manual page.

This summary begins as the server and clients boot up:

1 When the server reboots, *rc* executes *exportfs* which reads the server's */etc/exports* file, then tells the kernel which directories the server can export and what access restrictions – if any – are on these directories.

2. The *rpc.mountd* daemon (via the *inetd* daemon) and several *nfsd* daemons (usually about four) are started automatically by *rc* when the server boots.

3. When the client reboots, its *fstab* file is automatically read by the *mount* program.

4. The *mount* program then requests the server to allow the client to access the directories in the client's */etc/fstab* file.
5. The server's *rpc.mountd* daemon handles the client's mount requests.
6. If the requested directory is available to that client (or to the public), the *rpc.mountd* daemon sends the client's kernel an identifier called a *file handle*.
7. When it does a file operation, the client kernel then sends the file handle to the server, where it is read by one of the *nfsd* daemons to process the file request.
8. The *nfsd* daemons know how a directory is exported from the information sent to the server's kernel by *exportfs*. These daemons allow the client to access the directory according to its permissions, by way of the file handle.
9. The client's *biod* daemons improve client performance while the directory is accessed. They are not necessary for NFS to work, but they do increase performance.

The Network Filesystem Service

Introduction

This chapter introduces you to the procedural aspects of NFS, including:

- How to maintain NFS server workstations
- How to maintain NFS discless workstations
- How to set up network security
- How to troubleshoot NFS-related problems

Maintaining server workstations

This section explains the files you will need to edit to successfully maintain a workstation that you set up as a server, plus any other actions you may need to take after its initial set up. As explained in the previous chapter, the following steps were performed to set up a server workstation:

- Identify the workstation as an NFS server.
- Define the server's disc or discs, indicating which partitions and directories would hold the filesystems to be shared among the discless workstations.
- Define parameters for each of the server's discless workstations.

After an NFS server has booted, it has dedicated root directories, home directories, and swap space for each discless workstation on its disc. The */etc/exports* file contains entries for all these directories, which discless workstations need as soon as they boot up.

Nevertheless, at this stage the basic NFS service has been established. Many other NFS-related activities need to be performed, mostly in the area of file creation and modification. This is particularly important if the Network Information Service (NIS) is not used on the network.

The other activities that need to be performed to maintain server workstations include:

- Adding additional directories to the default */etc/exports* file, if needed.
- Changing access permissions to */etc/exports* as needed.

- Directly exporting and unexporting directories during a server's work session.
- Modifying the server's */etc/fstab* file, if more than one NFS server is on the network. Servers can then mount exported filesystems from the other servers on the network.
- Creating mount points in the server's filesystems for directories that it will mount from other servers.
- Editing the following files if you are not going to use NIS on your network:
  - */etc/hosts*
  - */etc/ethers*
  - */etc/netgroup*
- Checking the network status on a regular basis, and updating related files, as necessary.
- Setting up security for your network, as determined by your site's requirements.
- Diagnosing and fixing NFS related problems as they arise.

Exporting directories

You can control how a server exports directories in two ways:

- At boot time, through the */etc/exports* file.
- As needed, by using the *exportfs* command.

The /etc/exports file

The */etc/exports* file advertises all filesystems that a server exports to its discless workstations. Mounting is initiated by the discless workstation. Through */etc/exports* servers can control which discless workstation may mount a directory by limiting access to it to a desired discless workstation or netgroup.

The */etc/exports* file for a server, resembles the following:

```
/usr          access=clients
/home         access=clients
#
/export/root/zelda -root=zelda,access=zelda
/export/swap/zelda -root=zelda,access=zelda
#
/export/root/raks -root=raks,access=raks
/export/swap/raks -root=raks,access=raks
```

```
#
/export/root/samba -root=samba,access=samba
/export/swap/samba -root=samba,access=samba
```

Each line in the file */etc/exports* has the syntax:

```
directory -option[,option]
```

where *directory* is the pathname of a directory, a file, or, in some cases, the name of an entire filesystem. *option* represents a list of options that indicates which workstations are allowed to access the directory, and if read-only restrictions are on that directory.

All options are fully described in the manual page *exports*(5). More frequently used options are defined below.

ro — This option specifies that workstations are limited to read-only access to the specified directory. Therefore, they can read the directory's files but cannot change them. If *ro* is not specified for a directory, workstations are given read-write (*rw*) access by default.

root=hostnames — This option indicates that root access to the directory is given only to superusers from a specified *hostname*. In the previous example, the line

```
/export/root/zelda -root=raks
```

indicates that root access to */export/root/raks* is given only to those users who can become superuser on the workstation *raks*.

Root access is also given for superusers on the workstation *raks* for the other directory used by the workstation (*/export/swap/raks*). Note that a superuser on the workstation *samba* cannot access */export/root/raks* unless local files on the workstation *raks* grant *samba* this type of permission.

access=client — This option indicates that access to the particular directory should be given to the named *client*, *clients*, or *netgroup*. To limit access to a directory to a single workstation, use the parameter *access=client_name*.

For example, the line:

```
-access=raks
```

grants mount access to */export/root/raks* and */export/swap/raks* to all users on the discless workstation *raks*.

You can use the word *clients* to enable all the NFS server's discless workstations to mount a directory. In the example above, the line

```
/                access=clients
```

indicates that all the server's clients are allowed to mount /.

Finally, you can limit mount access to a directory to a designated *netgroup*. A *netgroup* is a network-wide group allowed access to certain network resources for security and organizational reasons. You set up net groups either through NIS, as described in the chapter entitled *The Network Information Service* on page 219 or through the */etc/netgroup* file described later in this section.

Editing */etc/exports*

You will occasionally want to modify the contents of */etc/exports* at some time. For example, you might want to designate that */usr* be exported read-only, designate access of a directory to a *netgroup*, or add more directories for exporting. Once you have modified */etc/exports* the server will automatically export these files every time you boot it up. Therefore, you should modify */etc/exports* so that it contains directories that you want the server to export all the time.

Below is a set of instructions that explain how to edit */etc/exports*:

1 Log in as *root* and edit the */etc/exports* file, using your preferred text editor, for example, *vi*

```
# vi /etc/exports
```

2 Enter the mount point pathname of the directory you want to add into the file. For example, suppose you want all members of the *accounting* netgroup to mount a new billing program that you have installed in its own filesystem, */billing*. Because this information is confidential, you do not want network users who aren't in accounting to mount this directory. The entry in */etc/exports* should look like the following:

```
/billing        -access=accounting
```

3 Save the changes you have made to the file.

4 Run the command *exportfs* as follows, to update the exports information in the server's kernel.

```
# /usr/sbin/exportfs -a
```

The *-a* option tells *exportfs* to send all information in */etc/exports* to the kernel.

Note, that you cannot export a directory that is either a parent or subdirectory of a directory *within the same filesystem* that is already exported. It would be illegal, for example, to export both */usr* and */usr/local* if both directories resided on the same disc partition.

Explicitly exporting directories with exportfs

The program */usr/sbin/exportfs* can be used in a number of ways to export and unexport directories to NFS discless workstations. You have seen how *rc.local* runs *exportfs* at boot time, and how you can use *exportfs* during a work session to update */etc/exports*, you can also run *exportfs* at any time to explicitly export and unexport directories. You probably will want to use *exportfs* in this fashion for directories that you only want to export for a finite amount of time, then unexport.

exportfs has the following syntax:

```
/usr/sbin/exportfs [ -aivu ] [ -o options ] [ directory ]
```

Its arguments are completely described in the man page *exportfs(8)*. The more commonly used arguments are defined below:

-a Export all the directories in */etc/exports* as illustrated in the previous section.

-u Unexport the indicated directories.

-o options Execute the list of options following *-o*. The arguments to *options* are the same as the options you can add to */etc/exports*. For example, typing:

```
/usr/sbin/exportfs -o ro /Frame
```

exports the directory */Frame* with read-only access. You can also use the:

```
root=hostname
```

and:

```
access=client
```

parameters in the same fashion as in */etc/exports*

directory This is the full pathname of the directory that you want to export or unexport.

If you type */usr/sbin/exportfs* without an argument, you receive a display of currently exported directories.

The */etc/xtab* file

Whenever *exportfs* runs, it updates the */etc/xtab* file, which lists currently exported directories. */etc/xtab* has an identical format to */etc/exports* but its contents change whenever you run *exportfs*. On the other hand, */etc/exports* contents remain static until you go in and change them.

Suppose you explicitly exported a directory with *exportfs* then checked the */etc/xtab* file. That directory would be listed, although it would not be listed in */etc/exports*. Moreover, if you then unexported the directory using *exportfs -u* then checked */etc/xtab*, the unexported directory would no longer be listed.

The NFS server's /etc/fstab file

The */etc/fstab* file shows all the directories that a workstation mounts when it boots up. It is completely described in the *fstab* man page (*fstab*(5)). You need to modify a server's *fstab* when you want it to access directories from another server on the network.

Lines in the */etc/fstab* file have the following syntax:

```
filesystem   mount_point   type   options   freq   pass
```

Here is the meaning of each parameter.

filesystem — This is the name of the filesystem that the server mounts. In the default *fstab* file, it is */dev/sd0a* – the server's root filesystem.

mount_point — This is the location within the directory tree where the server accesses the files designated by the *filesystem* parameter.

type — Type indicates the type of mount used to access the particular directory – either 4.3 or NFS. In the default *fstab* file on your workstation, only local 4.3 mounts are done for the local disc. However, if another server is on the local area network, you can also add remote NFS mounts to your workstations's *fstab* file.

options — This is a list of options that you can choose for the type of mount; they are fully described in the *fstab* man page. For example, with a 4.3 mount you can specify a *quota* option to enable the quota system. In the first line of the sample *fstab* file, the *rw* option is given, which enables the server to mount the specified directory with read-write permission – the default.

freq — This parameter indicates the interval in days between dumps.

pass This parameter indicates the pass in the *fsck* program during which this filesystem will be checked. If you indicate a 0 for this parameter, the filesystem is not checked during *fsck*.

The contents of the *fstab* file do not change unless you explicitly modify it. You will want to do this for your server if it is one of several on a local area network. To mount files from a remote server, you add NFS mounts to your server's *fstab* file. These are explained in the next section.

Maintaining discless workstations

This section contains information related to maintaining a discless workstation on a network plus any other actions you may need to take after its initial set up. You should read this if you are responsible for administering discless workstations, or if you are administering both servers and discless workstations on your network.

Activities you need to perform to administer a discless workstation include:

- Making mount points in your discless workstations's directory tree for the directories mounted through the */etc/fstab* file.
- Modifying the workstation's */etc/fstab* file to include directories the workstation is to mount.
- Directly mounting and unmounting directories as needed.
- Updating local files on the workstation.
- Diagnosing and fixing network-related problems on the discless workstation.

Mounting files from an NFS server

The most important service NFS provides to discless workstations is the ability to access directories on a disc on a remote workstation – a process called an NFS *mount*. Both NFS and 4.3 mounting occur during booting, when the *mount* program reads the */etc/fstab* file. You can also dynamically mount and unmount directories at any time by using the *mount* and *umount* commands. This next section explains how to set up discless workstations to perform NFS mounts.

Making mount points

Mount points are locations within a directory tree through which your workstation accesses directories. These directories can be mounted locally from a disc, or remotely from another computer on the network.

Mount points are essentially empty directories, which you create through the *mkdir* command. Simply type:

```
% mkdir mount_point
```

where *mount_point* is the full pathname of the empty directory you want to create. You need to create mount points for any non-default directories mounted through the */etc/fstab* file or through the *mount* command.

An NFS discless workstation's fstab file

When you first set up a workstation to be a server (as described in the chapter entitled *Setting up a local Ethernet network* on page 129), you had to create entries in the */etc/fstab* file for each of the server's discless workstations. The example below shows a typical entry for a discless workstation called *zelda* and a server called *eddison*:

```
eddison:/export/root/zelda     /      nfs    rw     0 0
eddison:/export/swap/zelda     /      nfs    rw     0 0
eddison:/usr                   /usr   nfs    ro     0 0
eddsion:/home                  /home  nfs    rw     0 0
```

The entries in the file have the following syntax:

```
server:server_dir client_mountpoint  type  options  freq
pass
```

Here is how these parameters apply to the sample *fstab* file above:

server — This is the name of the server exporting the directory the discless workstation wants to mount. In the above file, *eddison* is the only server listed.

server_dir — This is the name of the directory to be mounted from the indicated server. In the first entry, the directory is */export/root/zelda*. *zelda*'s individual *root* directory.

Note the only mounted directories needed by the discless workstation to operate are its individual *root*, *swap* and *home* plus */usr* which, along with *root* contains all essential programs.

client_mountpoint — This is the mount point on the discless workstation through which it accesses the directories mounted from the server. In the first entry above, the mount point is the workstation's root directory.

type — This is the type of mount taking place. In the example above, this is an NFS mount.

options This can be any one of a number of options provided for NFS or 4.3 mounts. Refer to the *fstab* man page for a complete list. For example, if you are running a secure network, you may want to select the *secure* option for NFS mounts.

The example file shows the discless workstation *zelda* performing NFS mounts with the common options *rw* or *ro*. *zelda* mounts the *root*, *swap* and *home* directories with read-write access. However, it accesses the server's */usr* directory with read-only permission; thus a user on *zelda* cannot add or modify a file in */usr*.

freq This is the interval in days between dumps of the directory.

pass This indicates the *fsck* pass in which the listed filesystem is checked. If a zero is indicated, the filesystem is not checked during *fsck*

The contents of */etc/fstab* remain the same until you change them.

Mounting and Unmounting directories

This subsection shows how to perform two different mount procedures: statically, through the *fstab* file, or explicitly, through the *mount* and *umount* commands. You can use them to enable both NFS and 4.3 mounts.

Procedures for Modifying the discless workstation's *fstab*

If you want the workstation always to mount a particular filesystem, you should add an entry to its *fstab* file. Follow these instructions:

1 Log in as superuser.

2 Edit */etc/fstab* using your preferred text editor.

If you are editing a discless workstation's *fstab* file while logged in to your server, this file is located in */export/root/client_name/etc/fstab*.

3 Create an entry in the file, using the syntax shown in the previous subsection, to mount the desired filesystem. List the server exporting the directory you wish to mount, that directory's pathname, the mount point on the workstation, and so on.

4 Create the mount point in the workstation's directory tree using the *mkdir* command.

5 Type the following:

```
# /sbin/mount -a
```

to mount everything in the current *fstab* file.

After you have finished these steps, the discless workstation will always mount the files in its */etc/fstab* file until you modify it again.

Explicitly Mounting and Unmounting Directories

Use the *mount* and *umount* commands during the course of using the discless workstation. It is advisable to use these commands for directories that you only want to access for a finite amount of time.

Using *mount*

mount explicitly mounts a directory. It only requires that you can reach the directory's server over the network, and that the server's */etc/exports* file allows you access to the directory.

Here is a form of the *mount* command that you can use for most mount operations. (Refer to the *mount* man pages for the complete syntax of *mount*.)

```
# /sbin/mount -t type [-rv] -o [options] server_name: /directory  /mount_point
```

These parameters are explained below.

`-t type` — This is the type of mount you want to perform – *NFS* or *4.3*.

`[-rv]` — These are two options that you can supply with *mount*. *–r* specifies that you want to mount the directory read-only. *–v* specifies the verbose option, in which *mount* displays messages as it operates.

`-o options` — This is a list of options specified after the *–o* flag. The options are fully described in the *mount* manual page.

One important *–o* option is *soft* or *hard*. In a *hard* mount, the workstation continually tries to mount the directory, even if the server does not respond to its request. In a *soft* mount, the workstation does not retry the mount operation if the server does not respond to its first request.

server_name — This is the name of the server exporting the directory.

directory — This is the pathname on the server of the directory you want to mount

mount_point — This is the mount point on the workstation through which the directory is mounted.

Directories accessed through the *mount* command stay mounted during a work session unless you unmount them with the *umount* command. Moreover, if you reboot the workstation, the directory is automatically unmounted (unless you also edited the *fstab* file to include the mount).

For example, to mount the reference manual pages from the remote workstation *dancer* to the local empty directory */usr/man* on a discless workstation, type the following all on one line:

```
# /sbin/mount -t NFS -o soft dancer:/usr/share/man
/usr/share/man
```

Here are some final points to remember about *mount*:

- Do not forget to make the applicable mount point before issuing the *mount* command.
- Consider soft mounting directories such as the manual pages so that if the server exporting them goes down, the discless workstation will not hang.
- Use the *hard* option with any filesystems you mount read-write.

Using *umount*

You use *umount* to explicitly unmount a currently mounted filesystem. A simple form of *umount* is shown below. (Refer to the *mount* man page for more variations of this command.)

```
/sbin/umount [ -v ] mount_point
```

`-v` This is an option to select verbose mode, wherein *umount* displays messages as it runs.

`mount_point` This is the name of the mount point on the discless workstation where you have mounted the files from the server.

The */etc/mtab* file

Whenever you explicitly mount or unmount a filesystem, the */etc/mtab* file is modified. This file has the same format as the *fstab* for a discless workstation or for a server, whichever applies. You can display */etc/mtab* by using the *cat* or *more* commands, but you should not edit it, as you would */etc/fstab*.

Files to set up for a new discless workstation

When you add new discless workstations to a network, you have to edit certain administrative files to enable the workstation to operate smoothly and securely. Become *root* on the discless workstation and do the following:

- Edit the */etc/passwd* file to add any new users who will be using the discless workstation. Remember not to use */usr/sbin/vipw* to edit the password file as this will load the server's password file into the editor, rather than the password file used by the discless workstation.
- If user groups are in existence at your site, edit the workstation's */etc/group* file. If NIS is not present, move a copy of the NFS server's */etc/group* file into the workstation's root directory. If NIS is present, make the same changes to */etc/netgroup* on the discless workstation's NIS master server.
- To enable the discless workstation to use printers on the network, copy the server's version of this file to the discless workstation's root directory:

```
# rcp server_name:/etc/printcap /etc/printcap
```

- To enable the mail facility on the new client, type the following command on the discless workstation:

```
# rcp server_name:/usr/lib/sendmail.subsidiary.cf /usr/lib/sendmail.cf
```

- Edit the client's */etc/rc.net* file, ensure the line is of the form:

```
HOSTNAME=new_discless_workstation_name
```

- If you are running the NIS service, check that the new client's domain name is specified on the first line of the */etc/rc.yp* file:

```
domainname domain_name
```

- Update the */etc/hosts.equiv* and *.rhosts* files to specify who is allowed to log in to the discless workstation over the network.
- You may define a new users' environment on login in several ways. For example you may give them copies of such files as *.login* and *.cshrc* if they use */usr/bin/csh* or *.profile* if they use */usr/bin/sh*.
- You may also want to give a user *.mailrc* files. Remember to change the ownership of these files to be owned by the user.
- If the user should belong to any mailing lists, add them to */etc/aliases*.

Handling NFS problems

This section describes typical problems that occur on workstations using NFS services. Topics discussed include:

- Strategies for tracking NFS problems
- NFS-related error messages

Before trying to clear NFS problems, you might want to re-read the chapter entitled *How NFS works – An overview* on page 163. Also, consider familiarizing yourself with the following manual pages, if you have not done so already:

- *mount*(8)
- *nfsd*(8)
- *biod*(8)
- *mountd*(8)
- *inetd*(8)
- *inetd.conf*(8)

The information in this section contains enough technical detail to give advanced System Administrators a thorough picture of what happens on the network with their workstations. If you do not yet have this level of expertise, note that it is not important to understand everything in this section (eg daemons, system calls, and files) fully. However, you should be able to at least recognize their names and functions.

Determining where the NFS service has failed

When tracking down an NFS problem, keep in mind that, like all network services, there are three main points of failure:

- the server
- the client
- the network itself

The strategy outlined below tries to isolate each individual component to find the one that is not working.

For example, consider this sample mount request made from an NFS client workstation:

```
% mount dancer:/usr/src /dancer.src
```

Below is a summary of how this command works and where it can fail.

1 The command requests server *dancer* to send a file handle *fhandle* for the directory */usr/src*. This file handle is sent to the client kernel by the *mount* program.

2 The client kernel looks up the directory */dancer.src* and, if everything is okay, it ties the file handle to the directory in a mount record. From now on all filesystem requests to that directory and its subdirectories go through the file handle to server dancer.

3 The *mountd* daemon must be present for a remote mount to succeed. Make sure *mountd* will be available for an *rpc* call by checking */etc/inetd.conf* on the NFS server for this line:

```
mountd/1                dgram       rpc/udp   wait   root
/usr/sbin/rpc.mountd    rpc.mountd
```

If it is not there add it. For details, see *inetd*(8).

4 Remote mount also needs some number, typically 4, of *nfsd* daemons to execute on NFS servers. Check */etc/rc* for the following lines:

```
if [ ${FULLNETWORK} = TRUE -a -f /usr/sbin/nfsd -a -f /etc/exports ];
then
/usr/sbin/nfsd 4; echo -n ' nfsd' >/dev/console
```

You can enable these daemons manually, without rebooting. Type:

```
# /usr/sbin/nfsd 4
```

Debugging Hints

Below are some general pointers for debugging, followed by a list of possible errors and their probable causes.

When the network or server has problems, programs that access hard mounted remote files will fail differently than those that access soft mounted remote files. Hard mounted remote filesystems cause programs to retry until the server responds again. Soft mounted remote filesystems return an error after trying for a while. *mount* is like any other program: if the server for a remote filesystem fails to respond, it retries the mount request until it succeeds. When you use *mount* with the *bg* option, it retries the mount in the background if the first mount attempt fails.

Once a hard mount succeeds, programs that access hard mounted files hang as long as the server fails to respond. In this case, NFS should print on the console, the message:

```
NFS server not responding
```

On a soft mounted filesystem, programs get the message:

```
Connection timed out (ETIMEDOUT)
```

when they access a file whose server is dead. Unfortunately, many programs do not check return conditions on filesystem operations, so you may not see this error message when accessing soft mounted files. An NFS error message should be printed on the console in this case also.

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

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

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

```
program vers proto   port
100000    2   tcp    111  portmapper
100000    2   udp    111  portmapper
100004    2   udp    648  ypserv
100004    2   tcp    649  ypserv
100004    1   udp    648  ypserv
100004    1   tcp    649  ypserv
100007    2   tcp   1024  ypbind
100007    2   udp   1026  ypbind
100007    1   tcp   1024  ypbind
100007    1   udp   1026  ypbind
100029    1   udp    652  keyserv
100028    1   tcp    658  ypupdated
100028    1   udp    660  ypupdated
100009    1   udp   1023  yppasswdd
100035    1   udp    719
100035    1   tcp    722
100005    1   udp    724  mountd
```

If that works, you can also use *rpcinfo* to check if the *mountd* daemon is running:

```
% /usr/sbin/rpcinfo -u server_name mount
```

This should come back with the response:

```
program 100005 version 1 ready and waiting
```

If these fail, try logging in to the server's console and see if it is okay.

If the server is up but your workstation cannot communicate with it, you should check the Ethernet connections between your workstation and the server. The easiest way to do this is to use the *ping*(1) command (described in the section entitled *Testing the network* on page 149).

If the server is okay and the network is okay, use *ps* to check your client daemons. You should have a *portmap* and several *biod* daemons running. For example, running *ps* should result in output similar to the following:

```
% ps ax
  PID TT    STAT   TIME   COMMAND
  0   ?     D      0:00   swapper
  1   ?     I      0:00   /single/init -
  2   ?     D      0:00   pagedaemon
  35  ?     I      0:11   portmap
  38  ?     I      0:00   ypbind
  41  ?     I      0:00   keyserv
  57  ?     S      9:27   in.routed
  59  ?     I      0:00   (biod)
  60  ?     I      0:00   (biod)
  61  ?     I      0:00   (biod)
  62  ?     I      0:00   (biod)
```

Clearing remote mounting problems

This section deals with problems related to mounting. If mount fails for any reason, check the sections below for specific details about what to do. They are arranged according to where they occur in the mounting sequence and are labelled with the error message you are likely to see.

mount can get its parameters explicitly from the command line or from */etc/fstab*. The example below assumes command line arguments, but the same debugging techniques work if */etc/fstab* is used in the *mount –a* command.

Keep in mind the interaction of the various players in the mount request. If you understand this, the problem descriptions below will make a lot more sense.

Consider the following *mount* request from a client on a network where NIS is running:

```
% /sbin/mount dancer:/usr/src /dancer.src
```

Here are the steps *mount* uses to mount a remote filesystem.

1 *mount* opens */etc/mtab* and checks that this mount has not already been done.

2 *mount* parses the first argument into host *dancer* and remote directory */usr/src*.

3 *mount* calls the NIS binder daemon *ypbind* to determine which server workstation to find the NIS server on. It then calls the *ypserv* daemon on that workstation to get the Internet protocol (IP) address of dancers.

4 *mount* calls *dancer*'s portmapper to get the port number of *mountd*.

5 *mount* calls *dancer*'s *mountd* daemon and passes it */usr/src*.

6 *dancer*'s *mountd* reads its */etc/exports* file and looks for the exported filesystem that contains */usr/src*.

7 *dancer*'s *mountd* daemon calls the NIS server *ypserv* to expand the host names and net groups in the export list for */usr/src*.

8 *dancer*'s *mountd* does a *getfh* system call on */usr/src* to get the *fhandle*.

9 *dancer*'s *mountd* returns the *fhandle*.

10 *mount* does a *mount* system call with the *fhandle* and */dancer.src*

11 *mount* checks if the caller is superuser and if */dancer.src* is a directory.

12 *mount* does a *statfs* call to dancer's NFS server *nfsd*.

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

Error Messages Related to Remote Mounts

Any step in the remote mounting process can fail – some of them in more than one way. This section gives a detailed descriptions of the failures associated with specific error messages:

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

The mounted filesystem table is kept in the file */etc/mtab*. This file must exist before mount can succeed.

```
mount: ... already mounted
```

The filesystem that you are trying to mount is already mounted or there is an erroneous entry for it in */etc/mtab*.

```
mount: ...  Block device required
```

You probably did not specify the remote workstation name in the following command:

```
# /sbin/mount remote.workstation:/usr/src /dancer.src
```

The *mount* command assumes you are doing a local mount unless it sees a colon in the filesystem name, or unless the */etc/fstab* specifies that the filesystem is mounted via an NFS mount.

```
mount: ... not found in /etc/fstab
```

If you issue the *mount* command with only a directory or filesystem name but not both, it looks in */etc/fstab* for an entry whose filesystem or directory field matches the argument. For example, the following command:

```
# /sbin/mount /dancer.src
```

searches */etc/fstab* for a line that has a directory name field of */dancer.src*. If it finds an entry, such as:

```
dancer:/usr/src /dancer.src nfs rw,hard 0 0
```

it will do the mount as if you had typed:

```
# /sbin/mount -o rw,hard dancer:/usr/src /dancer.src
```

The default options are read-write, hard, and suid.

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

This message indicates that *mount* tried to look up the directory given it as an argument in the */etc/fstab* file, but could not find */etc/fstab*.

```
... not in hosts database
```

On a network without NIS, this message indicates that the host specified to *mount* is not in the */etc/hosts* file. On a network running NIS, the message indicates that NIS could not find the host name in the */etc/hosts* database or that the NIS daemon *ypbind* has died on your workstation.

Check the spelling and the placement of the colon in your *mount* command. If the command is correct, your network does not run NIS, and you only get this message for this host name, check the entry in */etc/hosts*.

If your network is running NIS, make sure that *ypbind* is running by typing:

```
# ps ax
```

Try to *rlogin* to another workstation, or use *rcp* to remote copy something to another workstation. If this also fails, your *ypbind* daemon is probably dead or hung. If you only get this message for this hostname, you should check the */etc/hosts* entry on the NIS server. For more information about NIS problems, refer to the section entitled *Debugging NIS clients and servers* on page 235.

```
mount: directory path must begin with '/'
```

The second argument to the *mount* is the path of the directory to be used as a mount point. This must be a full pathname starting at root /.

```
mount: ... server not responding: RPC: Port mapper failure - RPC: Timed out
```

Either the server you are trying to mount from is down, or its portmapper is dead or hung. Try rebooting the server to restart the *inetd*, *portmap* and *ypbind* daemons. If you can't *rlogin* to the server but the server is up, you should check your Ethernet connection by trying to *rlogin* to some other workstation. You should also check the server's Ethernet connection.

```
mount: ... server not responding: RPC: Program not registered
```

This means that *mount* got through to the portmapper, but the NFS mount daemon *rpc.mountd* was not registered.

```
mount: ...: No such file or directory
```

Either the remote directory or the local directory does not exist. Check the spelling of the directory names. Try to use *ls* on both directories.

```
mount: not in export list for ...
```

Your workstation name is not in the export list for the filesystem you want to mount from the server. You can get a list of the server's exported filesystems by typing:

```
# showmount -e hostname
```

If the filesystem you want is not in the list, or your workstation name or netgroup name is not in the user list for the filesystem, log in to the server and check the */etc/exports* file for the correct filesystem entry. A filesystem 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, or it could not find the filesystem, or the filesystem name was not a local mounted filesystem. See the *exports*(5) manual page for more information. If the */etc/exports* file looks correct, and your network runs NIS, check the server's *ypbind* daemon. It may be dead or hung.

```
mount: ...: acess denied for ...
```

This message is a generic indication that some authentication failed on the server. It may be that you are not in the export list (see above), that the server could not recognize your workstation (*ypbind* is dead), or that the server does not believe you are who you say you are. Check the server's */etc/exports* file, and, if applicable, *ypbind*. In this case you can just change your hostname with *hostname* and retry the *mount* command.

```
mount: ...: Not a directory
```

Either the remote path or the local path is not a directory. Check the spelling in your command, and try to run *ls* on both directories.

```
mount: ...: Not owner
```

You should have run *mount* as root on your workstation because it affects the filesystem for the entire workstation, not just you.

Fixing hung programs

If programs hang doing file related work, your NFS server may be dead. You may see the following:

```
NFS server hostname not responding, still trying
```

on your console. The message indicates that NFS server *hostname* is down. This indicates a problem with your NFS server or with the Ethernet. Programs can also hang if a NIS server dies.

If your workstation hangs completely, check the server(s) from which you have mounted filesystems. If one of them (or more) is down, do not be concerned. When the server comes back up, your programs continue automatically. No files are destroyed.

If a soft mounted server dies, other work should not be affected. Programs that time out trying to access soft mounted remote files will fail with *errno ETIMEDOUT*, but you should still be able to access your other filesystems.

If all servers are running, go ask someone else using these same servers if they are having trouble. If more than one workstation is having problems getting service, this indicates a problem with the server's *nfsd* daemons. Log in to the server; Run *ps* to see if *nfsd* is running and accumulating CPU time. If not, you may be able to kill and then restart *nfsd*. If this does not work, you will have to reboot the server.

If other systems seem to be up and running, check your Ethernet connection and the connection of the server.

Fixing a workstation that Hung Part Way Through Boot

If your workstation boots up normally until it tries to do remote mounts, probably one or more servers is down, or your network connection is bad. Try to reboot in single user mode. Then edit the */etc/fstab* file to prevent the remote directories from being mounted at boot time by inserting the *noauto* option in each mount line. Then try to boot the machine multi-user again. For more information, refer to *fstab*(5).

Speeding Up Slow Access Times

If access to remote files seems unusually slow, type:

```
# ps aux
```

on the server to be sure that it is not being adversely affected by a runaway daemon, bad *tty* line, etc. If the server seems okay and users on other machines are getting good response, make sure your *biod* daemons are running. Try the following steps:

1 Run *ps ax* and look for *biod* daemons in the display. If they are not running or are hung, continue with these steps.

2 Find the process ids of the *biod* daemons by typing:

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

3 Kill these processes as follows:

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

4 Restart the daemons by typing:

```
# /usr/sbin/biod 4
```

To determine if the *biod* daemons are hung, run *ps* as above, then copy a large file from a remote system, then run *ps* again. If the *biod* do not accumulate CPU time, they are probably hung.

If the *biod* are okay, check your Ethernet connection. The command *netstat*(1) tells you if you are dropping packets. Also, you can use the command *nfsstat*(8) with the options *–c* and *–s* to tell if the client or server is doing a lot of retransmitting. A retransmission rate of 5 percent 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.

Making the network more secure

This section explains how to implement some degree of network security through the use of various administrative files. These files are:

- */etc/passwd*
- */etc/group*
- */etc/hosts.equiv*
- */.rhosts*

For more information, refer to the manual pages associated with these files. More information about making the network more secure can be found in the chapter entitled *Secure networking* on page 269.

The /etc/passwd file

The local password file contains entries for each user that has permission to log in to the workstation to which the file applies. */etc/passwd* is present on all types of RISC iX workstations, from discless client to any type of server.

Here is a sample */etc/passwd* file on a networked workstation:

```
root:uYU722Lg5xk/U:0:10:System Administrator:/:
sync::0:10:& Disks:/:/usr/sbin/sync
halt::0:10:& the machine !:/:/sbin/halt
daemon:*:1:31:The devil himself:/:
operator:*:2:28:System &:/nonexistant:
bin:*:3:3:& file owner:/:
uucp:kserjff:66:1:UNIX-to-UNIX
Copy:/var/spool/uucppublic:/usr/lib/uucp/uucico
guest::32766:9999:Unprivileged & user:/home/guest:
nobody:*:65534:9999:Unprivileged user:/nonexistent:
stefania:1Zm62edDOO8es:3747:20:Stephanie
Brucker:/home/dancer/raks:/bin/csh
+::0:0:::
```

Each entry has the syntax:

```
user:password:uid:gid:gcos-field:home-dir:login-shell
```

For more information about the fields in the password file, refer to the section entitled *Using vipw* on page 87. Briefly, they are:

user	The user's login name.
password	The user's encrypted password.
uid	User's numerical user id.
gid	Numerical id of the group to which the user belongs.
gcos-field	User's real name and other identifying information to be printed in the user's mail message heading.
home-dir	Full pathname of the user's home directory.
login-shell	Shell the user accesses upon login.

In the sample password file above, the users:

- *root*
- *nobody*
- *daemon*
- *bin*

- *sync*

are actually ids created by default for each workstation. When you log in with the *root* user name, the system grants you special access rights to the administrative files in */etc*. Depending on the circumstances, various programs also run with the user names *root*, *bin*, *nobody* or *daemon* as their owners.

The user name:

```
uucp:kserjff:66:1:UNIX-to-UNIX
Copy:/usr/spool/uucppublic:/usr/lib/uucp/uucico
```

in the sample file above is used by *uucp* programs. The entry:

```
stefania:1Zm62edD0O8es:3747:20:Stephanie Brucker:/home/dancer/stefania:/bin/csh
```

is the entry for a user allowed to log in to this workstation.

Setting Up a New User

When you first add a new user, you have to create an entry for them in their workstation's */etc/passwd* file, then create a home directory for them. You should do the following:

1 Obtain basic information from the user, such as their preferred user name (which should be unique within your network domain), full name as the user wants it displayed in mail headers, and their preferred login shell.

2 Become superuser and access the */etc* directory of the user's workstation, for example, *raks*.

```
#cd /export/root/raks/etc
```

3 Edit the local *passwd* file, using your preferred text editor.

4 Create an entry for the new user on a separate line.

5 Type the user's requested login name, for example:

```
shamira
```

The colon after the user name is the delimiter used in the *passwd* file to indicate the end of a field.

6 Leave the password field blank by typing another colon, as in:

```
shamira::
```

7 Add a user id for *shamira*, according to your company's policies. Do not use a user id that already exists in the local password file. Especially do not use 0 or 1, the user ids for *root* and *daemon*. If your company does not have a policy for assigning

user ids, you have to create one. As one suggestion, some companies use a person's employee number in the user id field of */etc/passwd*. Follow the user id with a colon.

8 Add a group id number for the group you want the person to be in, according to your company's policy. If your company does not have a policy, refer to the section entitled *Editing the group file* on page 86, for information on allocating users to groups.

9 Type the person's name as requested by the user for the mail header. Follow it with a colon.

10 Type the full pathname of the user's home directory, which should be: */home/user_name*. Follow it with the delimiting colon.

11 Finally, type the user's preferred login shell. Here is an example of a complete entry for new user *shamira*:

```
shamira::235:12:Marsha Wong:/home/dancer/shamira:/bin/csh
```

12 Close the */etc/passwd* file. Now, create a home directory for the new user.

```
# cd /home/dancer
# mkdir shamira
# chown userid.groupid shamira
```

13 You should then tell the user that you have created a user id for them on the network and advise them to log in immediately and run the *passwd* command to create a password for their login name.

The /etc/group file

The */etc/group* file is a local file that you might want to alter for your server. The groups created consist only of users who are allowed to log in to your server.

The */etc/group* file has several default groups. The group *staff* for instance, is the default group to which all users belong. A second group that you might want to assign users to is *operator*. Members of the *operator* group have special permissions that enable them to run certain commands such as *dump*, *restore* and *shutdown* without being superuser, and to run these commands on a remote workstation without being in its *.rhosts* file. If your site employs certain individuals whose job it is to specifically run daily backups, you might want to add these people to the *operator* group.

The /etc/hosts.equiv file

The */etc/hosts.equiv* file is a list of workstations, or *hosts* whose users are permitted to remote login (*rlogin*) to your workstation without supplying a password. It is a local file, pertaining only to your system, which you modify by logging in as superuser and using your text editor to make changes.

A typical *hosts.equiv* file has the following structure:

```
host1
host2
+@group1
-@group1
```

When you place a simple entry for a host in *hosts.equiv* such as the entry above for *host1* this means that anyone logging in to your workstation from *host1* is trusted. Furthermore, if the user logging in from *host1* also has an entry in the */etc/passwd* file for your workstation, then that user will be granted immediate access – ie they will not be asked to type a password.

Here are some simple guidelines for understanding how the protections in */etc/hosts.equiv* operate:

- If the user has an entry in the remote workstation's *passwd* file and the user's workstation is in the remote workstation's *hosts.equiv* file, then the user can *rlogin* to the remote workstation without a password.
- If the user is in the remote workstation's *passwd* file but the user's workstation is not in the remote workstation's *hosts.equiv* file, then the user must supply a password when logging in to the remote workstation.
- If the user does not have an entry in the remote workstation's *passwd* file but the user's workstation is in the remote workstation's *host.equiv* the user cannot *rlogin* in to the remote workstation.
- If the user does not have an entry in the remote workstation's *passwd* file, and there is no entry for the user's workstation in the remote workstation's *hosts.equiv* file, then the user cannot *rlogin* to the remote workstation.

A single '+' on a line in your *hosts.equiv* file means that everyone logging in over the network is trusted. The '+@' in front of a group name means that all members of that group or netgroup are trusted. The -@ before a group name means that no members of that group are to be trusted.

If your workstation is a printer server, its *hosts.equiv* file has to have real entries in it in order for it to print files from remote workstations. The same is true if your workstation is a mail server.

Refer to the *hosts.equiv*(5) manual page for more information.

The /.rhosts file

The *.rhosts* file is located in the user's home directory. It has the same format as */etc/hosts.equiv*:

```
host1
host2
+@group1
-@group2
```

It is used for additional permission checking when trying to access a remote workstation using *rlogin* or *rsh*.

When you use *rlogin* or *rsh* the *.rhosts* file in your home directory on the remote workstation is added to the end of the *hosts.equiv* file for additional permission checking. If the *host.equiv* file denies you access due to a minus entry, but your workstation is listed as trusted in *.rhosts* then you will be able to access the remote workstation. Additionally, if you are logged in as superuser on your workstation and try to remote log in to another workstation, only the *.rhosts* file of the remote workstation is checked.

How the Administrative files affect network security

Network security is implemented at two levels: first, at the workstation level, and second, at the user level. The */etc/hosts.equiv* and */.rhosts* files, respectively, control access at these levels. If NIS runs on your network, your workstation consults the *yppasswd* file in the NIS database on the NIS server. If the workstation does have a local copy of the password file, the local copy is consulted first. The security-checking process goes like this:

- If you initiate a remote process on another workstation, for example *rlogin*, the system first checks for an entry for your user name in */etc/passwd* on the remote workstation. If no entry is found and NIS is running, the system then checks *yppasswd*. If the system still cannot find an entry, you are denied access. If attempting to *rlogin* to the workstation, you will be prompted for a password and then get:

  ```
  Login incorrect
  ```

  You receive the same message if attempting an *rcp* or *rsh*.
- If an entry for your user name is found in the local */etc/passwd*, the system next checks for your workstation's hostname in the other workstation's */etc/hosts.equiv* file. If the hostname is found, you gain access.
- If no */etc/hosts.equiv* entry is found, the system checks for a line with the your workstation's hostname (and, optionally, your user name) in the *.rhosts* file in the home directory on the workstation you want to access. If the entry is found, you gain access.

- If no entry is found for your workstation in either */etc/hosts.equiv* or *user_name/.rhosts* but an entry for your user name is in */etc/passwd* you can *rlogin* to the workstation after giving the right password. However, you will receive the message:

  ```
  Permission denied
  ```

  when attempting to run remote processes like *rcp* or *rsh*.

- The single exception to this security scenario is the superuser. When you *rlogin* as *root* the system skips */etc/hosts.equiv* (the second level check) and directly checks *.rhosts*.

To allow access to your workstation by all users on another specific workstation, do the following:

1 Include an entry in your workstation's */etc/passwd* file for each user on the remote workstation.

2 Include the remote workstation's hostname in your */etc/hosts.equiv* file.

For example, suppose a workstation's hostname is *samba*, and you want to allow anyone on host *ballet* to gain access to *samba*. Simply edit */etc/hosts.equiv* on *samba* as follows. The file is just a list of hostnames, one per line:

```
raks
dancers
jazz
```

Add host *ballet* to the list:

```
raks
dancers
jazz
ballet
```

Now all users who can gain access to ballet can also freely *rlogin* to *samba* without having that workstation request a password. Furthermore, users on ballet can *rcp* from and use *rsh* on *samba*, provided they are in *samba's* */etc/passwd* file.

Suppose you want certain users on a particular workstation to access your workstation, but not everyone. Here you do not put the workstation's hostname in your workstation's */etc/hosts.equiv*. Instead, put the workstation's name in the *.rhosts* file in each user's home directory on your workstation (*user_name/.rhosts*). Note that to avoid some security problems, this *.rhosts* file must be owned by either the home directory's owner or root and may not be a symbolic link.

The *.rhosts* file has a slightly different format than */etc/hosts.equiv*. */etc/hosts.equiv* accepts only hostnames; *.rhosts* accepts a hostname and, optionally, a user name on each line. Its format is best illustrated by an example. You can allow user *chris* at host *samba* to gain access to host *raks*, and deny access to other users of *samba*:

- Making sure *samba* is not in *raks*' */etc/hosts.equiv* file.
- Adding an entry for chris on *samba* to *~chris/.rhosts*. The entry looks like this:

```
samba chris
```

Note also that this means user *chris* can only remotely access *raks* from host *samba*. If he tries to remotely access *raks* from another host, he must supply a correct password to *raks* in order to *rlogin* and cannot complete remote processes. Of course, if he can *rlogin* he can always modify his *.rhosts* file (if it is not owned by *root* and you gave him a home directory on *raks*).

This brings up two points:

- The only way to achieve anything resembling security in a sensitive environment is to exclude users from the */etc/passwd* file; once someone knows a password, that user can access a workstation. If your site requires tight security, also be especially careful to protect */.rhosts* properly. Make sure only *root* has write permission.
- If your site does not require stringent security measures, the easiest way to administer workstation access at the user level is to give each *trusted* user an account in */etc/passwd* and a home directory on your workstation(s). Then ask the trusted users to create their own *.rhosts* files in their home directories on the workstation(s).

Finally, note that an entry in */etc/hosts.equiv* and *.rhosts* is invalid if it contains trailing white space – blanks or tabs. The presence of trailing white space may not be obvious. You can look at a file through the *cat -e* command or by running *vi* with the *set list* option. Both of these methods show a '$' at the end of each line, before which you can look for trailing white space. Also try running *grep* on a file to search for *tab_character$* or *blank_space$* patterns to get a listing of any lines ending in tabs or blanks.

Setting the user id on execution

When a program executes, the kernel changes the user ID of its owner to that of the person running it – unless the program's permissions have the *suid* (setuid) bit set. When the *suid* bit is set, the program's user ID does not change when it is executed. This is an important security consideration. Many programs on your release tape come with the *suid* bit set, a condition you may want to modify. In addition, people creating programs may ask you to enable or disable this option for their own applications.

You set the user ID of a program by becoming superuser and running the *chmod* command. In the syntax statement:

chmod [-fR] *mode filename ...*

specify *setuid* by substituting for *mode* the absolute mode *nnn* where *nnn* are octal numbers representing the permissions for owner, group, and public. If you prefer, you can use the symbolic mode *s* with *u* and *g* to accomplish this. Then, if necessary, use the *chown* command to specify the login name that will always own the program, regardless of who runs it. Refer to *chmod*(1) and *chown*(1) if you need help with these commands.

Here are the permissions for the */usr/bin/crontab* program, produced by running *ls -l* on the directory */usr/bin*:

```
-rwsr-xr-x  1 root        16384 Feb  5 12:42 crontab
```

Note the letter *s* listed as the execute permission bit for the program's owner. It indicates that the *suid* bit is set. Since the program's owner is root, *crontab* will always run with the *root* user id as owner and thus have *root* permissions and privileges.

You should go through the permissions for the files installed from the release tape. Note any that are run with the *suid* bit set and note who is the owner. Some programs, such as daemons, always need to execute with root as user ID, regardless of who actually executed them. Depending on your site's security considerations, it is sometimes safer to change the ownership of other programs from root to daemon.

Shell scripts can prove dangerous security holes, if they run with root ownership and the *suid* bit set. This means that anyone can make changes to the script.

In NFS4.0 you can mount a directory from the server with the *suid* or the *nosuid* option set. For example, if you specify the command:

```
# /sbin/mount -o server:/home/lauraf /home/lauraf
```

the home directory for *lauraf* will be mounted with the default options *rw* and *suid*. You can also mount a directory with the *nosuid* option, as in:

```
# /sbin/mount -o rw,nosuid server:/home/lauraf
/home/lauraf
```

If you enable this option, then any setuid programs in the mounted directory lose their setuid characteristics when executed on your workstation. For example, a program such as */usr/lib/sendmail* which has the *suid* bit set and is owned by *root*, would run as if the user ID of its executor actually owned it, rather than *root*.

This is a wise precaution if you think the directory you are mounting might have poorly protected setuid programs.

Setting the group id

There is a difference in the way the group ID worked in previous releases and the way it works with Release 4.0 of NFS. If the *setgid* bit on a directory is set, files created within that directory will be owned by the group ID of that directory. If the *setgid* bit is not set, files created in that directory will have the group ID of the creating process. This allows users to choose which type of group permission they want.

Allowing root access over the network

root access over the network is an important consideration if you are trying to implement some degree of network security. This section contains information on some of the considerations.

Allowing users to rlogin as root over the network

Users attempting the following command:

rlogin *hostname* **-l root**

where *hostname* is a RISC iX workstation, will receive the error message:

```
login incorrect
```

and on the RISC iX workstation, the following error message will be displayed (by default in the console window):

```
...: ROOT LOGIN REFUSED ON ttyp0 FROM hostname
```

where *hostname* is the name of the workstation you tried to log in as *root* from.

The reason for this refusal, is because to the *ttyp* entries in */etc/ttys* are configured by default not to be *secure*. This means that *root* access is denied to anyone attempting to log in as *root* over the network. For example:

```
ttyv1 "/usr/sbin/getty std.avc" avc on secure

ttyp0 none network
```

ttyv1 is one of the local virtual terminals that allows *root* logins. However *ttyp0* is initially set to refuse *root* logins. This can be changed if necessary but it is bad practice to allow *root* logins over the network. If you wish to become *root* from another workstation, log in to the remote workstation using your normal user name and then type:

login root

Root access to exported filesystems

Under the NFS, a server exports filesystems it owns so clients can remote mount them. When a client becomes superuser, it is denied *root* access on remote mounted filesystems. When a person logged in as *root* on one host requests access to a particular file from NFS, the user ID of the requester is changed to the user ID of the user name *nobody* (Recall that *nobody* is one of the user names automatically placed in the default */etc/passwd* file.) User *nobody*'s access rights are the same as those given to the public for a particular file. For example, if the public only has execute permission for a file, then user *nobody* can only execute that file.

When you export a filesystem, you can permit *root* access on a particular workstation to have root access to that filesystem by editing */etc/exports* on the server. For example, suppose you wanted the workstation *samba* (but no others) to have superuser access to the exported directory */usr/src* You would enter the following line in */etc/exports*

```
/usr/src      -root=samba
```

If you want more than one client to have root access, you can specify a list as follows:

```
/usr/src      -root=samba:raks:jazz
```

You can also enable superuser access for all clients, again using the */etc/exports* file. Suppose you wanted to allow superuser access for all clients accessing */usr/src*. You add the following line to */etc/exports*

```
/usr/src      -anon=0
```

The *anon* is short for *anonymous*. Its meaning is somewhat involved. An NFS server labels as anonymous any request from a root user (someone whose current user name has user ID 0) who is not in the root list in */etc/exports*. Anonymous requests, by default, get their user ID changed from its previous value to -2, the user ID of *nobody* The line in the sample */etc/exports*:

```
/usr/src      -anon=0
```

tells the operating system to use the value 0 instead of -2 for anonymous requests. The result is that all root users retain their user ID of 0.

See *exports* and *exportfs* for more information on exporting directories.

Privileged Ports

NFS has some Internet domain source ports to which only privileged users can attach. These are known as *privileged ports*. Currently, NFS does not check to see if a client is bound to one of these. That is, an NFS server has no way of knowing whether a client's file request originated from the real client's kernel or from someone's user program. If you get the following error message:

```
NFS request from unprivileged port
```

you should also see the IP address of the client on console screen of the server. You can turn on server port checking by changing the following parameter in the kernel using *adb*(1) (Note, you may have to use *unsqueeze*(1) on *vmunix* before you can run *adb* on it):

```
# adb -w /vmunix
adb>nfs_portmon?W0
_nfs_portmon:0x1          =              0x0
adb><Ctrl-D>
```

The next time you boot your system, the source ports will be checked. If you can trust all of the *root* users on your network, then just doing the above is enough. But be warned: some non-UNIX systems do not enforce the privileged port convention (in particular, PCs with 3Com boards).

The Lock Manager

RISC iX now supports a System V compatible record and file locking service. The lock manager functionality applies to local record and file locking as well as to record and file locking of remote filesystems on the network. File locking is accomplished with the *lockf*(3) and *fcntl*(2) functions. Since the Lock Manager is an extensible service, the user, program, or application can optionally invoke the service on a per-transaction, or per-application basis. Once a record is locked by one user or application, another user or application trying to lock the same record will either receive an error message or be placed in a queue and be appropriately notified when the lock becomes available.

The *Lock Manager* is a System V Interface Definition (SVID) compatible record and file locking service. When multiple users attempt to access the same record at the same time, only one will have access and the rest will be locked out.

System V compatible file and record locking *fcntl* calls support both local and remote files. A new signal has been added to allow recovery in the event that a remote server crashes and locks are lost. Ordinarily, locks will be reclaimed when the remote server recovers and SIGLOST (the signal) will not be issued. The signal notifies the process in the event of a lost lock. The default action is to kill the process, and therefore existing System V programs need not be changed to run on NFS, unless they choose to recognize this failure notification.

See the manual pages *lockf*(3) and *fcntl*(2) for more information.

Clock skew in user programs

Because NFS architecture differs in some minor ways from earlier versions of the UNIX operating system, you should be aware of places where users' programs could run up against these incompatibilities.

Because each workstation keeps its own time, the clocks are out of sync between the NFS server and client. Obviously this might introduce a problem in certain situations. The clock skew problems have been fixed. Here are examples of two problems and how they were fixed.

ls problem

Many programs make the (reasonable) 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 considered old.

Now assume that the time on the server is Jan 22 15:30:31 (three minutes ahead of the local workstation'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 of the two times is huge:

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

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

ranlib timestamp problem

The *ranlib* program was also modified to deal with clock skew. *ranlib* timestamps a library when it is produced, and *ld* compares that timestamp with the last modified date of the library. If the last modified date occurred after the timestamp, then *ld* instructs you to run *ranlib*

If the library is on a server whose clock is ahead of the client running *ranlib ld* will always complain. *ranlib* was fixed to set the timestamp to the maximum value of the current time and the library's modify time.

Remember that if your application depends upon local time or the filesystem timestamps, then it will have to deal with clock skew problems if it uses remote files.

The Network Information Service

Introduction

The Network Information Service (NIS†) is a distributed network look-up service. It maintains a set of files that machines can query as they would a database. These files are known as NIS maps. All NIS maps are fully replicated on several systems known as NIS servers, each of which runs a server process for the maps. It does not matter which server process answers a client request. The response is always the same because the set of NIS maps is identical for each server. This allows you to have multiple servers per network and makes the NIS service highly reliable and available.

Basic NIS concepts

This section gives an overview of NIS, its maps, and its associated commands.

The NIS map

Each NIS map contains a set of keys and associated values. For example, in a map called *hosts.byname* all the host names within a domain are the keys, and the Internet addresses of these host names are the values. Each NIS map has a *mapname* used by programs to access it. Many of the current NIS maps are derived from ASCII files traditionally found in */etc* such as:

- */etc/hosts*
- */etc/group*
- */etc/passwd*

and a few others. The format of the data within the NIS map is identical (in most cases) to the format within the ASCII file. *dbm* files, which are located in the subdirectories of the directory */var/yp* on NIS servers, implement the NIS maps.

A NIS *domain* is a named set of NIS maps, located in a subdirectory of */var/yp* the directory where an NIS server holds all its maps. The name of this subdirectory is the name of the NIS domain. For example, maps for a domain called *literature* would be located in */var/yp/literature*.

---

† NIS was originally called the Yellow Pages Service.

Each machine belongs to a default domain set at boot time by the */etc/rc.yp* file with the *domainname* command. You must set the domain name on all machines (both servers and clients) that will access the maps of a particular domain. You can also display or change your NIS domain name by using the *domainname* command.

In the NIS environment, only NIS servers have a set of NIS maps, which they make available to clients over the network. There are two kinds of NIS servers:

- an NIS slave server
- an NIS master server.

The NIS master server updates the maps of the slave servers. Therefore, you should always modify maps on the NIS master server. The changes propagate from the master server to the NIS slave servers. If you create or change NIS maps on a slave server instead of a master server, this may create two potentially different versions of the NIS map. Furthermore, the NIS master server is the only machine with all the necessary ASCII files.

The NIS clients run processes that request data from maps on these machines. NIS clients do not care which server is the master, since all NIS servers should have the same information. The distinction between master and slave server only applies to where you make the updates.

A server may be a master with regard to one map, and a slave with regard to another. Randomly assigning maps to NIS servers can cause a great deal of confusion. You are strongly urged to make a single server the master for all the maps you create by the *ypinit* command within a single domain. The examples in this chapter assume that one server is the master for all maps in the domain.

Commands used for maintaining NIS

YP offers a series of commands that you can use for setting up and editing maps and performing other NIS-related functions. This subsection briefly describes them. (These commands are described in detail later in this section and in the reference manual pages.)

ypserv — Describes the processes that comprise the NIS service. These are *ypserv*, the NIS map server daemon and *ypbind* the NIS binder daemon. *ypserv* must run on each NIS server. *ypbind* must run on all clients.

ypfiles — Describes the file structure of the NIS service.

ypinit — Automatically constructs maps from files located in */etc* such as */etc/hosts*, */etc/passwd* and others. *ypinit* also constructs initial versions of required maps that are not built from files in */etc* for example,

	ypservers. Use *ypinit* to set up the master NIS server and the slave NIS servers for the first time. You typically do not use it as an administrative tool for running systems.
ypmake	Describes the use of */var/yp/Makefile*, which builds several commonly-changed components of NIS maps. These are the maps built from the files in */etc* on the master NIS server: *passwd*, *hosts*, *group*, *netgroup*, *networks*, *protocols* and *services*.
makedbm	Takes an input file and converts it into a pair of *dbm* files, which then become valid NIS maps. For example, *ypmaps.dir* and *ypmaps.pag* are both *dbm* files. You can use *makedbm* to build or rebuild maps not built from */var/yp/Makefile*. You can also use *makedbm* to *disassemble* a map, so that you can see the key-value pairs that comprise it. You can also edit the disassembled form using editors such as *vi*, *emacs* and *ex* or text processing tools like *awk*, *grep* and *cat*. The disassembled form is in the format required for input back into *makedbm*.
ypxfr	Moves an NIS map from one NIS server to another, using NIS itself as the transport medium. You can run *ypxfr* interactively, or periodically from a *crontab* file.
yppush	Requests each of the *ypserv* processes within a domain to transfer a particular map, waits for a summary response from the transfer agent, and prints out the results for each server. You run it on the master NIS server.
ypset	Tells a *ypbind* process (the local one, by default) to get NIS services for a domain from a named NIS server. This is not for casual use.
yppoll	Asks any *ypserv* for the information it holds internally about a single map.
ypcat	Displays the contents of an NIS map. Use it when you do not care which server's version you are seeing. If you need to see a particular server's map, *rlogin* to that server (or use *rsh* and use *makedbm*).
ypmatch	Prints the value for one or more specified keys in an NIS map. Again, you have no control over which NIS server's version of the map you are seeing.
ypwhich	Use this command to see which NIS server a host is using at the moment for NIS services, or which NIS server is master of a particular map.

ypupdated Daemon used for changing NIS information. This daemon is normally started up by *inetd*. *ypupdated* consults the file *updaters* in the */var/yp* directory to determine which maps should be updated and how to change them. Note that *ypupdated* only works if the network is running secure RPC.

How administrative files are consulted on an NIS

YP can serve any number of maps. Typically these include some files in */etc*. NIS services make updating these files much simpler, since you do not have to make the same change to every machine on the network. For example, on networks that do not run NIS, programs read the */etc/hosts* file to find an Internet address. When you add a new machine, you have to add an entry for this machine to the */etc/hosts* files on every machine on the network. On networks running NIS, programs that need to consult */etc/hosts* now do a remote procedure call to the NIS servers for the same information.

NFS programs do not consult the same system administrative files on a network with NIS that they would on a network without NIS. This is because they consult NIS maps instead. The following list describes how NFS programs on a network running NIS consult the administrative files you have previously learned about:

/etc/passwd Always consulted. If there are + or – entries, the NIS password map is consulted, otherwise NIS is not used. See *passwd*(5).

/etc/group Always consulted. If there are + or – entries, the NIS group map is consulted, otherwise NIS is not used. See *group*(5).

/etc/services Never consulted. The data that was formerly read from this file now comes from the NIS *services* map. However, note that as *ypbind* is started after */etc/services* in the boot-up procedure, the file still needs to be there for the machine to boot successfully. If this file is renamed, then errors will be given about failure to start *syslogd* and *timed* during the boot-up procedure.

/etc/protocols Never consulted. The data that was formerly read from this file now comes from the NIS *protocols* map.

/etc/networks Never consulted. Data is taken from this file to create the NIS *networks* map.

/etc/netgroup Never consulted. The data that was formerly read from this file now comes from the NIS *netgroup* map.

/etc/bootparams Never consulted. The data that was formerly read from this file now comes from the NIS *bootparams* map.

/etc/ethers	Never consulted. The data read from this file comes from the NIS *ethers* map.
/etc/hosts	Consulted only when booting (by the *ifconfig* command in the */etc/rc.boot* file). After that the NIS map is used instead.
/etc/hosts.equiv	(And similarly for *.rhosts*). Always consulted, though neither of these files is in the NIS domain. (See the later section entitled *How Security Is Changed with NIS* for a fuller explanation of these two files.) If there are + or – entries, whose arguments are net groups, the NIS netgroup map is consulted, otherwise NIS is not used. See *hosts.equiv*.
/etc/aliases	Always consulted. Local aliases take precedence over those in the NIS database. See */etc/aliases*.
/etc/netmasks	Never consulted. The data that was formerly read from this file now comes from the NIS *netmasks* map.

NIS Installation and Administration

This section covers nine installation and administration topics:

1. Setting up a master NIS server
2. Altering an NIS client's database to use NIS services
3. Setting up a slave NIS server
4. Setting up an NIS client
5. Modifying individual NIS maps after installing NIS
6. Propagating an NIS map
7. Making new NIS maps after installing NIS
8. Adding an additional NIS server
9. Changing the master server to a different machine

How to set up a Master NIS server

Before setting up the master NIS server, there are several steps you must take. You need to set up the NIS domain name plus the hostname. By default, */etc/rc.yp* sets up the *domainname* and */etc/rc.net* sets up the *hostname*.

To create a new server on an existing network, change directory to */usr/sbin/yp* and run *ypinit*. You will be asked whether you want the procedure to die at the first non-fatal error (in which case, you can fix the problem and restart *ypinit*, recommended if you haven't done the procedure before), or to continue despite non-fatal errors. In this second case you can try to fix all the problems by hand, or fix some, then restart *ypinit*.

ypinit prompts you for a list of other hosts that will also be NIS servers. (Initially, this is the set of NIS slave servers, but at some future time any of them might become the NIS master server.) You need not add any other hosts at this time, but if you know that you will be setting up more NIS servers, add them now. You will save yourself some work later, and there is little runtime penalty for doing it. (However, do not name every host in the network.)

Before running *ypinit* the following files in */etc* should be complete and reflect an up-to-date picture of your network:

- */etc/passwd*
- */etc/hosts*
- */etc/ethers*
- */etc/group*
- */etc/networks*
- */etc/protocols*
- */etc/services*

Also, if you know how */etc/netgroup* is going to be set up, do that before running *ypinit* If you don't know, *ypinit* makes an empty netgroup map. Also, */etc/aliases* should be complete.

For security reasons, you may restrict access to the master NIS machine to a smaller set of users than that defined by the complete */etc/passwd* file. To do this, copy the complete file to some place other than */etc/passwd* and edit out undesired users from the remaining */etc/passwd*. For a security-conscious system, this smaller file should not include the NIS escape entry discussed in the next section.

Finally, edit the file */etc/rc.config*. Change the following line:

```
...
NIS=FALSE
...
```

to:

```
...
NIS=TRUE
...
```

You are now ready to create a new master server. Change directory to */usr/sbin/yp* on the machine which is going to act as the master NIS server and type:

domainname *domainname*

where *domainname* is the name of your NIS domain. Then type:

ypinit -m

This command constructs an NIS database from the files in */etc* mentioned previously for the domain name defined in */etc/rc.yp*. A new directory is created in */var/yp* named after the domain name and the database files are deposited here.

Assuming that the database was constructed satisfactorily, to start providing NIS services, reboot the machine. *ypserv* will start up automatically from */etc/rc.yp* every time the server boots – provided that the NIS database constructed by *ypinit* still exists.

Altering an NIS client's files to use NIS services

Once you decide to run NIS at your site, you should have all hosts on the network access the NIS maps, rather than potentially out-of-date information in their local administrative files.

This policy can be enforced by running a *ypbind* process on the client machine (including machines that may be running NIS servers), and by abbreviating or eliminating the files that traditionally implemented the NIS maps. The files in question are:

- */etc/passwd*
- */etc/hosts*
- */etc/ethers*
- */etc/group*
- */etc/networks*
- */etc/protocols*
- */etc/services*
- */etc/netgroup*
- */etc/aliases*
- */etc/netmasks*
- */.rhosts*

The treatment of each file is discussed below:

- */etc/networks*, */etc/protocols*, */etc/ethers* and */etc/netgroup* need not exist on any NIS clients.

- */etc/services* needs to exist for the workstation to boot successfully, although it is not used by NIS.
- */etc/hosts.equiv* is never served by NIS. However, you can add escape sequences to reference NIS. This reduces problems with *rlogin* or which are sometimes caused by different */etc/hosts.equiv* files on the two machines.

  To let anyone log on to a machine, you can edit */etc/hosts.equiv* to contain a single line, with only the character, + (plus) on it. A line with only a + means that all further entries are retrieved from NIS rather than the local file.

  Alternatively, you can exercise more control over logins by using lines of the form:

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

  Each of the names to the right of the at sign(@) is assumed to be a netgroup name, defined in the global *netgroup* database. The netgroup database is served by NIS.

  If none of the escape sequences is used, only the entries in */etc/hosts.equiv* are used; NIS is not used.
- */.rhosts* also is never served by NIS. As shown in the chapter entitled *The Network Filesystem Service* on page 185, its format is identical to that of */etc/hosts.equiv*. However, because this file controls remote root access to the local machine, unrestricted access to it is not recommended. Make the list of trusted hosts explicit, or use netgroup names for the same purpose. You can not use secondary hostnames in your *.rhosts hosts.equiv* or *netgroup* files. You can, however, use secondary hostnames in */etc/hosts*. All of the above files are related in that they enable local machines to access remote machines in some fashion.
- */etc/hosts* must contain entries for the local host's name, and the local loopback name. These are accessed at boot time when the NIS service is not yet available. After the system is running, and after the *ypbind* process is up, the */etc/hosts* file is not accessed at all. An example of the */etc/hosts* file for an NIS client *raks* is:

  ```
  127.1           localhost
  192.9.1.87      raks  # Stefania
  ```
- */etc/passwd* should contain entries for the root user name and the primary users of the machine, and the + escape entry to force the use of NIS. A few additional entries are recommended: *daemon*, to allow file-transfer utilities to work; and *operator*, to let a dump operator log in.

A sample NIS client's */etc/passwd* file looks like:

```
root:uYU722Lg5xk/U:0:10:System Administrator:/:
sync::0:10:& Disks:/:/usr/sbin/sync
halt::0:10:& the machine !:/:/sbin/halt
daemon:*:1:31:The devil himself:/:
operator:*:2:28:System &:/nonexistant:
bin:*:3:3:& file owner:/:
uucp:kserjff:66:1:UNIX-to-UNIXCopy:/var/spool/uucppublic:/usr/lib/uucp/uucico
guest::32766:9999:Unprivileged & user:/home/guest:
nobody:*:65534:9999:Unprivileged user:/nonexistent:
stefania:1Zm62edDOO8es:3747:20:Stephanie Brucker:/home/dancer/raks:/bin/csh
+::0:0:::
```

The last line informs the library routines to use NIS. If you remove the last line in the *passwd* file, you will disable NIS password access.

A program that calls */etc/passwd* first looks in the password file on your machine; if it doesn't find the relevant information in this file it will then look in the NIS password file only if your machine's password file contains + (plus sign) entries, as shown in the above example. Also, earlier entries in the file take precedence over, or mask later ones with the same user name, or the same user ID.

- */etc/group* may be reduced to a single line:

  ```
  +:
  ```

  which forces all translation of group names and group ids to be made via NIS. This is the recommended procedure.

How to set up a slave NIS server

The network must be working to set up a slave NIS server – in particular, you must be able to *rcp* files from the master NIS server to NIS slaves. This means that the following tasks should have been done:

- the *domainname* needs to have been set
- */etc/rc.config* needs to have been edited to enable NIS
- */usr/sbin/ypbind* must also be running.

To create a new slave server, change directory to */usr/sbin/yp*. From there run *ypinit* with the *-s* option. Name a host already set up as an NIS server as the master. Ideally, the named host really is the master server, but it can be any host that has its NIS database set up. The host must be reachable. The default domain name on the machine intended to be the NIS slave server must be set up, and must be set to the same domain name as the default domain name on the machine named as the master.

Also, an entry for *daemon* must exist in the */etc/passwd* files of both slave and master, and that entry must precede any other entries which have the same user ID. Carefully check that you have not altered the password file to change the order. Note the example shown in the section above. You won't be prompted for a list of other servers, but you will have the opportunity to choose whether or not the procedure gives up at the first non-fatal error.

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

```
nisslave# cp /etc/passwd /etc/passwd-
nisslave# cp /etc/hosts /etc/hosts- etc
```

Edit the original files in accordance with the section entitled *Altering an NIS client's files to use NIS services* on page 225 to ensure that processes on the slave NIS server actually use NIS, rather than the local ASCII files. (That is, make sure the NIS slave server is also an NIS client.) Make backup copies of the edited files, as well. For instance:

```
nisslave# cp /etc/passwd /etc/passwd+
nisslave# cp /etc/hosts /etc/hosts+ etc
```

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

After this has been completed, you will need to update the master yp server's database. For more information, refer to the section entitled *How to add a new NIS server not in the original set* on page 233.

How to set up an NIS client

To set up an NIS client, edit the local files as described in the previous section *Altering an NIS Client's Files to Use NIS*. As well as this, you should also do the following on the NIS client:

- the *domainname* needs to have been set
- */etc/rc.config* needs to have been edited to enable NIS
- */usr/sbin/ypbind* must also be running.

With the ASCII databases of */etc* abbreviated and */usr/sbin/ypbind* running, the processes on the workstation will be clients of NIS. At this point, there must be an NIS server available; processes will hang if no NIS server is available while *ypbind* is running.

Note the possible alterations to the client's */etc* database as discussed above in the section entitled *Altering an NIS client's files to use NIS services* on page 225. Because some files may not be there, or some may be specially altered, it is not always obvious how the ASCII databases are being used. The escape conventions used within those files to force data to be included or excluded from the NIS databases can be found in the following man pages:

- *passwd* (5)
- *hosts* (5)
- *netgroup* (5)
- *hosts.equiv* (5)
- *group* (5)

In particular, notice that changing passwords in */etc/passwd* (by editing the file, or by running *passwd*) only affects the local client's environment. To change the NIS password database, run the command *yppasswd*(1).

How to modify existing NIS maps after NIS installation

Always change databases served by NIS on the master server. The databases expected to change most frequently, like */etc/passwd* may be changed by first editing the ASCII file, changing directory to */var/yp* and running *make*.

Non-standard maps (that is, maps that are specific to the applications of a particular vendor or site, but are not part of the standard RISC iX release), or maps that are expected to change rarely, or maps for which no ASCII form exists (for example, maps not around before you installed NIS), may be modified manually. The general procedure is to use *makedbm*(8) with the *-u* option to disassemble them into a form which can be modified using standard tools (such as *awk*, *sed* or *vi*). Then build a new version again using *makedbm*. This may be done by hand in two ways:

- The output of *makedbm* can be redirected to a temporary file that can be modified, then fed back into *makedbm*, or
- The output of *makedbm* can be operated on within a pipeline that feeds into *makedbm* again directly. This is appropriate if the disassembled map can be updated by modifying it with *awk*, *sed* or a *cat* append, for instance.

Suppose you want to create a non-standard NIS map, called *mymap*. You want it to consist of key-value pairs in which the keys are strings like *al*, *bl*, *cl* etc (the 'l' is for left), and the values are *ar*, *br*, *cr* (the 'r' is for right). There are two possible procedures to follow when creating new maps. One uses an existing ASCII file as input; the other uses standard input.

For example, suppose there is an existing ASCII file named */var/yp/mymap.asc* created with an editor or a shell script on the machine *nismaster*. *home_domain* is the subdirectory where the map is located. You can create the NIS map for this file by typing:

```
nismaster% cd /var/yp
nismaster% makedbm mymap.asc home_domain/mymap
```

But at this point you notice the map really should have included another 2-tuple: (*dl*, *dr*). You can make the modification simply. In all situations like this, remember to modify the map by first modifying the ASCII file. Modifications made to the *dbm* files, not also in the ASCII file, will be lost. Make the modification like this:

```
nismaster% cd /var/yp
nismaster% <make editorial change to mymap.asc>
nismaster% makedbm mymap.asc home_domain/mymap
```

When there is no original ASCII file, create the NIS map from the keyboard by typing input like this (assume the machine name is *nismaster* and the default domain is *home_domain*).

```
nismaster% cd /var/yp
nismaster% makedbm - home_domain/mymap
al ar
bl br
cl cr
<Ctrl-D>
```

When you need to modify that map, you can use *makedbm* (located in the directory */usr/sbin/yp*) to create a temporary ASCII intermediate file which can be edited using standard tools. For instance:

```
nismaster% cd /var/yp
nismaster% makedbm -u home_domain/mymap > mymap.temp
```

At this point you can edit *mymap.temp* to contain the correct information. A new version of the database is created by the commands:

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

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

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

Propagation of an NIS map

To propagate a map means to move it from place to place – in general, to move it from the master NIS server to all slave NIS servers. Initially, *ypinit* moves it, as described above in the section entitled *How to set up a slave NIS server* on page 227. After a slave NIS server has been initialized, updated maps are transferred from the master server by *ypxfr*.

ypxfr is normally invoked on the slave when you perform a *make* in */var/yp* on the master. However, it is a good idea for the slaves to run this command themselves occasionally to ensure that no updates are missed.

You can run *ypxfr* in three different ways:

- periodically by *cron*
- by *ypserv*
- interactively by a user.

Examples of each of these ways are given in the rest of this section.

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

To avoid having to have a *crontab* entry for each map, several maps with approximately the same change characteristics can be grouped in a shell script, and the shell script can be run from a *crontab* file. For more information about editing *crontab* files, refer to the chapter entitled *Maintaining the filesystem* on page 51.

Suggested groupings, mnemonically named, can be found in */usr/sbin/yp* *ypxfr_1perhour*, *ypxfr_1perday* and *ypxfr_2perday*. If the rates of change are inappropriate for your environment, you can easily modify or replace these shell scripts.

These same shell scripts should be run on each NIS slave server in the domain. Alter the exact time of execution from one server to another, so as to prevent the checking from bogging down the master. If you want the map transferred from some particular server, not the master, you can specify that (using *ypxfr -h* option) within the shell

script. You can use the *-s* option to transfer maps from another domain. Finally, maps having unique change characteristics can be checked and transferred by explicit invocations of *ypxfr* within the system *crontab* file, */var/spool/cron/crontabs/root*.

ypxfr also gets invoked by *ypserv* responding to a *Transfer Map* request. That request is made as an RPC call from *yppush*. *yppush* is run on the master NIS server. It enumerates the NIS map *ypserver* to generate a list of NIS servers in your domain. To each of the named NIS servers, it sends a *Transfer Map* request. *ypserv* forks off a copy of *ypxfr* invoking it with the –C flag, and passing it the information it needs to identify the map, and to call back the initiating *yppush* process with a summary status.

In the cases mentioned above, *ypxfr* transfer attempts and the results can be captured in a log file. If */var/yp/ypxfr.log* exists, results are appended to it. No attempt to limit the log file is made. To turn off logging, remove the log file.

Finally, you can run *ypxfr* as a command. Typically, you do this only in exceptional situations – for example when setting up a temporary NIS server to create a test environment, or when trying to quickly get an NIS server that has been out of service consistent with the other servers.

How to make new NIS maps after NIS installation

Adding a new NIS map entails getting copies of the map's *dbm* files into the domain directory on each of the NIS servers in the domain. The actual mechanism has been described in the section entitled *Propagation of an NIS map* on page 231. This section only describes how to get the proper files in place so the propagation works correctly. Things must be set up correctly on both the master and the slaves.

After deciding which NIS server is the master of the map, modify */var/yp/Makefile* on the master server so that the map can be conveniently rebuilt. Actual case-by-case modification is too varied to describe here, but typically a human-readable ASCII file is filtered through *awk*, *sed* and/or *grep* to make it suitable for input to *makedbm* Consult the existing *Makefile* as a source for programming examples. You should use the mechanisms already in place in */var/yp/Makefile* when deciding how to create dependencies that *make* will recognize; specifically, the use of *.time* files allows you to see when the *Makefile* was last run for the map.

To get an initial copy of the map, you can run *yppush* on the NIS master server. The map must be globally available before clients begin to access it. If the map is available from some NIS servers, but not all, you will see unpredictable behaviour from client programs.

How to add a new NIS server not in the original set

To add a new NIS slave server, you start by modifying some maps on the master NIS server. If the new server is a host that has not been an NIS server before, you must add the host's name to the map *ypservers* in the default domain. The sequence for adding a server named *nisslave* to *domain_name* is:

```
nismaster# cd /var/yp/domain_name
nismaster# ../makedbm -u ypservers > /tmp/temp_file
nismaster# vi /tmp/temp_file
nismaster# ../makedbm /tmp/temp_file ypservers
```

Running the *makedbm* command with the *-u* option undoes the *ypservers dbm* file; that is, it converts it from *dbm* format temporarily so you can add the new hostname to the temporary file *temp_file*. Then running the *makedbm* command with *temp_file* as the input file and *ypservers* as the output file converts *ypservers* back into *dbm* format.

The new slave NIS server's databases can then be set up by copying the databases from the NIS master server *nismaster*. To do this, remote log in to the new NIS slave, and run the *ypinit* command:

```
nisslave# cd /usr/sbin/yp
nisslave# ypinit -s nismaster
```

To verify that the *ypservers* file is correct (since there is no ASCII file for *ypservers*), do the following:

```
nisslave# cd /var/yp/domain_name
nisslave# ../makedbm -u ypservers
```

Note: If a host name is not in *ypservers* it will not be warned of updates to the NIS map files.

Then complete the steps described above in the section entitled *How to set up a slave NIS server* on page 227.

How to change the master server

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

```
newmaster# cd /var/yp
newmaster# make NOPUSH=1 jokes.bypunchline
```

/var/yp/Makefile must be set up correctly for this to work. If it isn't, do it now. Also, this is a good time to go back to the old master (if it will remain an NIS server) and edit */var/yp/Makefile* so that *jokes.bypunchline* is no longer made there – that is, comment out the section of *oldmaster:/var/yp/Makefile* which made *jokes.bypunchline*.

If the map only exists as a *dbm* database, you can remake it on the new master by disassembling an existing copy (one from any NIS server will do) and running the disassembled version back through *makedbm*. For example:

```
newmaster# cd /var/yp
newmaster# ypcat -k jokes.bypunchline |\
makedbm - mydomain/jokes.bypunchline
```

After making the map on the new master, you must send a new copy of the map to the other (slave) NIS servers. However don't use *yppush*, the other slaves will try to get new copies from the old master rather than the new one.

A typical method (you may find others) is to become superuser on the old master server and type:

```
oldmaster# /usr/sbin/yp/ypxfr -h newmaster
jokes.bypunchline
```

Now you have a new copy on the old master, and now you may run *yppush*. The remaining slave servers still believe that the old master is the current master, and will attempt to get the current version of the map from the old master. When they do so, they will get the new map, which names the new master as the current master.

If the method above fails, there is a cumbersome but sure-fire option. On each NIS server machine, while superuser, execute the command shown just above. This will certainly work, but should be considered the worst case solution, especially if you have a lot of slave servers on your network.

Note that the following NIS maps all need to be resident on the same master server:

- *passwd*
- *group*
- *hosts*
- *netid*

Debugging NIS clients and servers

This debugging section has two parts – first those problems seen on an NIS client, and then those problems seen on an NIS server.

Before trying to debug an NIS client, read the section entitled *Basic NIS concepts* on page 219, which explains how NIS works.

On client: Commands hang

The most common problem encountered at an NIS client node is for a command to hang and generate console messages which say:

```
yp: server not responding for domain <wigwam>. Still trying
```

Sometimes many commands begin to hang, even though the system as a whole seems okay and you can run new commands.

The message above indicates that *ypbind* on the local machine is unable to communicate with *ypserv* in the domain *wigwam*. This often happens when machines that run *ypserv* have crashed. It may also occur if the network or the NIS server machine is so overloaded that *ypserv* cannot get a response back to your *ypbind* within the timeout period. Under these circumstances, all the other NIS client nodes on your net show the same or similar problems. The condition is temporary in most cases, and the messages will usually go away when the NIS server machine reboots and *ypserv* gets back in business; or when the load on the NIS server nodes and/or the Ethernet decreases.

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

- The NIS client has not set, or has incorrectly set, *domainname* on the machine. Clients must use a domain name that the NIS servers know about. Use the command *domainname*(1) to see the client domain name. Compare that with the domain name set on the NIS servers. The domain name should be set in */etc/rc.yp*. When */etc/rc.yp* fails to set or incorrectly sets *domainname* do the following:
  - Become superuser on the machine in question
  - Edit */etc/rc.yp* to fix the *domainname* line with a proper domain name (this ensures the domain name will be correct every time the machine boots), and set *domainname* manually so it is fixed immediately. To do this, type:

    ```
    # domainname good_domain_name
    ```

    Note that you may have to boot the client in single-user mode to be able to make these changes.
- If your domain name is correct, make sure your local network has at least one NIS server machine. You can automatically bind only to a *ypserv* process on your local network, not on another accessible network. There must be at least one NIS

server for your machine's domain running on your local network. Two or more NIS servers will improve availability and response characteristics for NIS services.

- If your local network has an NIS server, make sure it is up and running. Check other machines on your local net. If several client machines have problems simultaneously, suspect a server problem. Find a client machine behaving normally, and try the *ypwhich* command. If *ypwhich* never returns an answer, kill it and go to a terminal on the NIS server machine. Type:

  ```
  # ps ax | grep yp
  ```

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

  ```
  # /usr/sbin/ypbind
  ```

  If there is a *ypserv* process running, do a *ypwhich* on the NIS server machine. If *ypwhich* returns no answer, *ypserv* has probably hung and should be restarted. Kill the existing *ypserv* process, and start */usr/sbin/ypserv*:

  ```
  # kill -9 [some pid number from ps]
  # /usr/sbin/ypserv
  ```

  If *ps* shows no *ypserv* process running, start one up.

On client: NIS service unavailable

When other machines on the network appear to be okay, but NIS service becomes unavailable on your machine, many different symptoms may show up. Among them: some commands appear to operate correctly while others terminate printing an error message about the unavailability of NIS; some commands limp along in a backup-strategy mode particular to the program involved; and some commands or daemons crash with obscure messages or no message at all. For example, things like the following may show up:

```
my_machine% ypcat myfile
ypcat: can't bind to YP server for domain <domain-name>.
      Reason: can't communicate with ypbind.

my_machine% /usr/sbin/yp/yppoll myfile
RPC: Timed out.
```

When symptoms like those above occur, try

```
my_machine% ls -l directoryname
```

where *directoryname is* a directory containing files owned by many users, including users not in the local machine's */etc/passwd* file, for example */usr*. If the *ls -l* reports file owners not in the local machine's */etc/passwd* file as numbers, rather than names, it is one more symptom that the NIS service is not working.

These symptoms usually indicate that your *ypbind* process is not running. You can do a *ps -ax* to check for one. If it you do not find it, type:

```
my_machine# /usr/sbin/ypbind
```

to start it. The NIS problems should disappear.

On client: ypbind crashes

If *ypbind* crashes almost immediately each time it is started, you should look for a problem in some other part of the system. Check for the presence of the *portmap* daemon by typing:

```
my_machine% ps ax | grep portmap
```

If you don't find it running, reboot the machine.

If *portmap* itself will not stay up or behaves strangely, look for more fundamental problems. Check the network software.

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

```
rigel% rpcinfo -p client
```

If your *portmap* is okay, the output should look something like the following (although the port numbers will be different on your workstation):

```
program vers proto  port
100007  2  tcp  1024 ypbind
100007  2  udp  1028 ypbind
100007  1  tcp  1024 ypbind
100007  1  udp  1028 ypbind
100021  1  tcp  1026 nlockmgr
100024  1  udp  1052 status
100021  1  udp  1054 nlockmgr
100024  1  tcp  1027 status
100020  1  udp  1058 llockmgr
100020  1  tcp  1028 llockmgr
100021  2  tcp  1029 nlockmgr
100012  1  udp  1083 sprayd
100011  1  udp  1085 rquotad
```

```
100005  1  udp  1087 mountd
100008  1  udp  1089 walld
100002  1  udp  1091 rusersd
100002  2  udp  1091 rusersd
100001  1  udp  1094 rstatd
100001  2  udp  1094 rstatd
100001  3  udp  1094 rstatd
100015  6 udp 1365 selection_svc
```

The four entries that represent the *ypbind* process are (again, the port numbers will be different on your workstation):

```
100007  2  tcp  1024 ypbind
100007  2  udp  1028 ypbind
100007  1  tcp  1024 ypbind
100007 1 udp 1028 ypbind
```

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

On client: ypwhich inconsistent

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

Debugging an NIS server

Before trying to debug an NIS server, read the section entitled *Basic NIS concepts* on page 219 which explains how NIS works.

On server: Different versions of an NIS map

Because NIS works by propagating maps among servers, you will sometimes find different versions of a map at servers on the network. This version skew is normal if transient, and abnormal otherwise.

Most commonly, normal update is prevented when some NIS server or some router between NIS servers is down during a map transfer attempt. (Normal update is described in the section entitled *Propagation of an NIS map* on page 231.) When all the NIS servers, and all the routers between them, are up and running, *ypxfr* should succeed.

If a particular slave server has problems updating, log in to that server and run *ypxfr* interactively. If *ypxfr* fails, it will tell you why it failed, and you can fix the problem. If *ypxfr* succeeds, but you think it has been failing sometimes, create a log file to enable logging of messages. Type:

```
nisslave# cd /var/yp
nisslave# touch ypxfr.log
```

This saves all output from *ypxfr*. The output looks much like what *ypxfr* creates when run interactively, but each line in the log file is timestamped. (You may see unusual orderings in the timestamps. That's okay– the timestamp tells you when *ypxfr* began its work. If copies of *ypxfr* ran simultaneously, but their work took differing amounts of time, they may actually write their summary status line to the log files in an order different from the order in which they were invoked.) Any pattern of intermittent failure shows up in the log. When you have fixed the problem, turn off logging by removing the log file. If you forget to remove it, it will grow without limit.

While still logged in to the problem NIS slave server, inspect the system *crontab* file, */var/spool/cron/crontabs/root* and the various *ypxfr* shell scripts it invokes. Typographic errors in these files cause propagation problems, as do failures to refer to a shell script within */var/spool/cron/crontabs/root* or failures to refer to a map within any shell script.

Also, make sure that the NIS slave server is in the map *ypservers* within the domain. If it's not, it will still work fine as a server, but *yppush* won't tell it when a new copy of a map exists.

If the problem is not obvious, you can work around it while you debug using *rcp*(1) or *tftp*(1) to copy a recent version from any healthy NIS server. You may not be able to do this as root, but you can probably do it as daemon. For instance, to transfer map *busted*:

```
nisslave# chmod go+w /var/yp/mydomain
nisslave# su daemon
$ rcp nismaster:/var/yp/mydomain/busted.\*
/var/yp/mydomain
$ <Ctrl-D>
nisslave# chown root /var/yp/mydomain/busted.*
nisslave# chmod go-w /var/yp/mydomain
```

Notice that the * character has been escaped in the command line, so that it will be expanded on *nismaster*, instead of locally on *nisslave*. Also, notice that the map files should be owned by root, so you must change ownership of them after the transfer. Obviously, if you can do the *rcp* as root, it makes the whole thing easier.

On server: ypserv crashes

When the *ypserv* process crashes almost immediately, and won't stay up even with repeated activations, the debug process is virtually identical to that described above in the section entitled *On client: ypbind crashes* on page 237. Check for the *portmap* daemon, by typing:

```
nisserver% ps ax | grep portmap
```

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

```
rigel% rpcinfo
```

and look for output to the screen something like (although the port numbers will be different on your workstation):

```
program vers proto  port
100004  2  udp  1027 ypserv
100004  2  tcp  1024 ypserv
100004  1  udp  1027 ypserv
100004  1  tcp  1024 ypserv
100007  2  tcp  1025 ypbind
100007  2  udp  1035 ypbind
100007  1  tcp  1025 ypbind
100007  1  udp  1035 ypbind
100009  1  udp  1023 yppasswdd
100003  2  udp  2049 nfs
100024  1  udp  1074 status
100024  1  tcp  1031 status
100021  1  tcp  1032 nlockmgr
100021  1  udp  1079 nlockmgr
100020  1  udp  1082 llockmgr
100020  1  tcp  1033 llockmgr
100021  2  tcp  1034 nlockmgr
100012  1  udp  1104 sprayd
100011  1  udp  1106 rquotad
100005  1  udp  1108 mountd
100008  1  udp  1110 walld
100002  1  udp  1112 rusersd
100002  2  udp  1112 rusersd
```

```
100001  1  udp  1115 rstatd
100001  2  udp  1115 rstatd
100001  3 udp 1115 rstatd
```

The four entries that represent the *ypserv* process are (again, the port numbers will be different on your workstation):

```
100004  2  udp  1027 ypserv
100004  2  tcp  1024 ypserv
100004  1  udp  1027 ypserv
100004  1 tcp 1024 ypserv
```

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

How security is changed with NIS

Security on a system running NIS is dependent on how NIS consults the administrative files on which its maps are based. Local files on the host are consulted first, then the system consults maps in the NIS domain. For example, the system checks the */etc/aliases* file for mail aliases, then checks the *mail.aliases* NIS map.

The remaining files on which NIS maps are based are global files:

- */etc/hosts*
- */etc/networks*
- */etc/ethers*
- */etc/services*
- */etc/netmasks*
- */etc/protocols*
- */etc/netgroup*

The information in these files is network wide data, and is accessed only from NIS. However, when booting, each machine needs an entry in */etc/hosts* for itself and needs to have a copy of */etc/services* for use before NIS services are running. In summary, if NIS is running, global files are only checked in the NIS maps; a file on your local machine is not consulted.

Special NIS password change

When you change your password with the *passwd* command, the local password file is firstly searched for your user name. If an entry for you is not found there, then the NIS password file on the NIS master server is consulted (by means of a + entry at the end of the local */etc/passwd* file). If an entry for you is found here then the NIS password map will also be automatically updated by the *yppasswd* server and the new copy of the password file will be propagated to all the NIS slaves on the network.

To explicitly change your password on the NIS password map, ignoring any entry you may have on the local password file, use *yppasswd* or *passwd* with the *–y* option. Again, the NIS password map will be automatically updated by the *yppasswd* server and the new copy of the password file will be propagated to all the NIS slaves on the network.

If you change your password while you are actually logged in to the NIS master server, you need to manually update the NIS password map. To do this change directory to */var/yp* and run the *make* command:

```
# cd /var/yp
# make passwd
```

the NIS password map will be re-*made* containing your new password and will then be propagated to all the NIS slaves on the network by *yppush*. For more information, refer to *passwd*(5).

/etc/publickey

To enable users to use the *secure* option to mount a directory the NIS database *publickey* must exist. *publickey* is the public key database used for secure networking. Each entry in the database consists of a network user name (which may either refer to a user or a hostname), followed by the user's public key (in hexadecimal notation), a colon, and then the user's secret key encrypted with its login password (also in hexadecimal notation).

This file is altered either by the user through the *chkey* command or by using the *newkey* command. The file */etc/publickey* should only contain data on the NIS master machine, where it is converted into the NIS database *publickey.byname*. Also see *ypupdated*, *newkey*, *chkey* and *getpublickey*.

There are two new NIS databases that enhance system security called *publickey* and *netid*. You use the *newkey* program to put new entries into this map. Type the following command:

```
# cd /var/yp
# make publickey
```

to create the NIS database from the */etc/publickey* file. Do not use a text editor to alter the */etc/publickey* file because the file contains encryption keys. To alter the */etc/publickey* file, you need to use the *newkey* command. There are two options to this command:

```
# newkey -u username
```

for a regular user on a host machine, and

```
# newkey -h hostname
```

for a root user on a host machine. The *publickey* database contains every user that has a public key. They are identified by a long string.

You should also be aware that there is a new NIS database called *netid* however, you do not need to administer it. The *netid* database is created from the *passwd*, *host* and *group* files.

Net groups: Network wide groups of machines and users

YP uses the */etc/netgroup* file on the master NIS server for permission checking during remote mount, login, remote login, and remote shell. It uses */etc/netgroup* to generate three NIS maps in the */var/yp/domainname* directory:

- *netgroup*
- *netgroup.byuser*
- *netgroup.byhost*

The NIS map *netgroup* contains the basic information in */etc/netgroup*. The two other NIS maps contain a more specific form of the information to speed up the look-up of net groups given the host or user.

The programs that consult these NIS maps are *login*, *mountd*, *rlogin* and *rsh*. *login* consults them for user classifications if it encounters netgroup names in */etc/passwd*. *mountd* consults them for machine classifications if it encounters netgroup names in */etc/exports*. *rlogin* and *rsh* consult the *netgroup* map for both machine and user classifications if they encounter netgroup names in the */etc/hosts.equiv* or */.rhosts* file.

What if you do not use NIS

If *ypserv* on the master is disabled, you can no longer update any of the NIS maps.

If you choose not to use NIS, you can disable it by simply renaming */usr/sbin/ypbind* to */usr/sbin/ypbind.orig*:

```
# mv /usr/sbin/ypbind /usr/sbin/ypbind.orig
```

To disable NIS on a particular NIS slave or master, do the following:

```
# mv /usr/sbin/ypserv /usr/sbin/ypserv.orig.
```

Refer to the chapter entitled *The Network Information Service* on page 219 for information about the files you need to maintain for a server should you decide to disable NIS.

Adding a new user to an NIS server

To add a new user, add an entry for the new user in the */etc/passwd* file of the NIS server and create a home directory on the new user's machine as described in the steps below.

On the master NIS server add a new line to the password file with the *vipw* command. *vipw* brings the password file into the *vi* editor, and prevents anyone else from editing it until you are finished:

```
# /usr/sbin/vipw
```

The following example shows an entry in the password file for new user *Mr. Chimp* with an account called *bonzo* and no password:

```
bonzo::1947:10:Mr.Chimp:/home/bonzo:/usr/bin/csh
```

Here is how the fields in the password file work when used in a network running NIS:

Login name — Synonymous with user name.

Encrypted password — The user's password. Tell all new users how to add, or change, their password with the *yppasswd* command. Remember, you can make this field empty when a user has forgotten their password, thereby enabling them to log in without a password until such time as a new one is given. Note that an asterisk in this field matches no password.

User id — A number unique to this user. A system knows the user by ID number associated with login name. Therefore a login name must have the same user ID number on all password files of machines that are networked in a local domain. Failure to keep ID's unique prevents files on different machines from being moved between directories because the system will respond as if the directories are owned by two different users. Also, file ownership may become confused when an NFS server exports a directory to an NFS client whose password file contains users with user ids that match those of different users on the NFS server.

Group — Can be used to group users together who are working on similar projects.

User Information

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

Home Directory

The directory the user logs in to, usually */home/user_name*

Login Shell — Initial shell to use on login. If this field is blank, the default */sbin/sh* is used. It is recommended that you place */usr/bin/csh* here, as in the example above, to give the user the C shell for a login shell.

After you have updated the password file and created a password for the new user, be sure to update the NIS maps by changing directories to */var/yp* and running the *make* command:

```
# cd /var/yp
# make passwd
```

To make the new entry available from all the workstations on the network, ensure that all the other workstations include the NIS map in their password files, by placing a '+' at the end of their local password file. For more information, refer to *passwd*(5).

Make a home directory

After adding a new entry to the password file, you should create a home directory for the new user to log in to on the workstation that they will use to store their files. This will be the same as the directory given in the sixth field of the password file entry. In the */home* directory, make a directory for the new user and change ownership to the user's login name and change group to the user's group. For example:

```
# cd /home
# mkdir bonzo
# chown bonzo bonzo
# chgrp 10 bonzo
```

Note that if the NIS maps for the password file have not yet been updated on the machine's NIS server, you will get the following error message when you attempt to do the *chown*:

```
unknown user id: username
```

In that case, you can use the following set of commands:

```
# cd /home
# mkdir bonzo
# chown userid bonzo
# chgrp 10 bonzo
```

You use Mr. Chimp's user ID number (from the password file entry) instead of the login name to change the ownership of his home directory.

Refer to the chapter entitled *Using the Automounter* on page 247 for alternative ways of handling network-wide home directories.

Using the Automounter

Introduction

The automounter is closely related to the Network Information Service (NIS) and provides a facility for automatically mounting file hierarchies shared through NFS. This chapter describes how to use the automounter and shows how to set it up on your network. The final section in the chapter lists some of the error messages you are likely to encounter when you use the automounter.

About the automounter

The *automount*(8) program enables users to mount and unmount remote directories on an as-needed basis. Whenever a user on a client machine running the automounter invokes a command that needs to access a remote file or directory, such as opening a file with an editor, the hierarchy to which that file or directory belongs is automatically mounted and remains mounted for as long as it is needed.

Once a certain amount of time (usually 5 minutes) has elapsed without the hierarchy being accessed, it is automatically unmounted.

The advantages of this are:

- Faster boot time – no mounting needs to be done at boot time
- The user no longer has to know the super-user password to mount a directory or even how to use the *mount* and *unmount* commands. Mounting is done automatically and transparently.

Note that mounting some file hierarchies with *automount* does not exclude the possibility of mounting other directories with *mount*. Also, a discless machine still needs to mount */export/root*, */export/swap* and a */home* directory via the *mount* command and the */etc/fstab* file.

You can invoke the automounter directly from the command line or by adding an appropriate line in */etc/rc*. You can also create a set of database maps, similar to the maps used in NIS, to further extend the functionality of the automounter. These maps can be incorporated with other NIS maps, or exist as separate files in */etc* on each machine.

Simple use of the automounter

The automounter can be used directly from the command line, by typing:

```
# automount -m /net -hosts
```

This invokes the automounter daemon to make available for mounting all the entries in the special map *-hosts*. This is a built-in map that does not use any external files except the hosts database */etc/hosts* (or the *hosts.byname* NIS map if NIS is running on the machine).

Once the automounter daemon has been started in this way, a user can enter the command:

cd /net/*machinename*

to change directory to the root filesystem of the machine specified (as long as it exists in the hosts database).

The act of specifying a machinename in the */net* directory triggers the automounter to automatically mount the filesystems that have been exported by that machine, as specified in it's */etc/exports* file. Note that if the machine does not have anything exported you will receive the error message:

```
/net/machinename: bad directory
```

The directory */net* is just an arbitrary directory used to specify the automount directory. It will be created by the automounter if it does not exist already.

Following a successful mount you can then copy, edit or save files in the normal way. Although, the first two levels in the full pathname of a file on the remote machine will begin with */net/machinename*.

The mount will remain active while files are being accessed on the remote machine. When no files have been accessed for approximately five minutes, the directory will be dismounted automatically by the automounter.

The automounter is initially started in this way from */etc/rc* when a machine is configured to be on a network:

```
if [ ${FULLNETWORK} = TRUE -a -f /usr/sbin/automount ]; then
 /usr/sbin/automount -m ${AUTOMOUNTDIR} -hosts; echo -n "
automount(${AUTOMOUNTDIR})" >/dev/console
fi
```

The directory */net* is used as the default automount directory as defined in the file */etc/rc.config*:

```
# The directory which the automounter will use for the -hosts map
AUTOMOUNTDIR=/net
```

If you already have a directory called */net* or prefer to use another directory, change the above line to refer to the new directory. The next time the machine is rebooted, the new directory will be used as the automount directory. If the directory does not exist, the automounter will create it the first time that it is invoked.

Using the automounter with several maps

The previous section described how to use the automounter at its most basic level with just the special built-in map (hosts). However, the facilities the automounter provides can be more fully exploited when used in conjunction with the additional maps that you can create.

A server never knows, nor cares, whether the files it shares are accessed through *mount* or *automount*. Therefore, you do not need to do anything different on the server for *automount* than for *mount*.

A client, however, needs special files for the automounter. As mentioned previously, *automount* does not consult */etc/fstab*; rather, it consults the map file(s) specified at the command line (see the section entitled *Invoking automount* on page 263). All automounter maps are located in the directory */etc*, or if NIS is running they reside on the NIS server in */var/yp*. Their names all begin with the prefix *auto*.

There are three kinds of automount maps:

- *auto.master*
- *auto.indirect*
- *auto.direct*

The master map usually contains entries for all the maps that are to be used by the automounter, including built-in special maps (*–hosts*) and direct and indirect maps that you have created.

The master map should be created as a NIS database. However, it is recommended that you firstly create and use a local master map for use on a few machines so that you can test that it works correctly. This map can then be incorporated in the NIS database.

The master map

Each line in a master map, by convention called */etc/auto.master*, has the syntax:

```
Mount-point        Map    [Mount-options]
```

where:

- *mount-point* is the full pathname of a directory.

- *map* is the name of the map the automounter should use to find the mount points and locations.
- *mount-options* is an optional list of options, separated by commas that regulate the mounting of the entries specified in the *map*, unless the *map* entries list other options. For example, the *-f* option enables you to specify a local master map on a machine rather than one that exists on the NIS database.

A line whose first character is '#' is treated as a comment, and everything that follows until the end of the line is ignored. A backslash '\' at the end of a line permits long lines to be split into shorter lines.

Direct and Indirect maps

Lines in direct and indirect maps have the syntax:

```
key    [Mount-options]    location
```

where:

- *key* is the pathname of the mount point.
- *mount-options* are the options you wish to apply to a particular mount.
- *location* is the location of the resource, specified as *server:pathname*.

As in a master map, a line whose first character is '#' is treated as a comment, and everything that follows until the end of the line is ignored. A backslash '\' at the end of a line permits long lines to be split into shorter lines.

The only formal difference between a direct and an indirect map is that the key in a direct map is a full pathname, whereas in an indirect map it is a simple name with no slashes.

For example, the following line would be a valid entry in a direct map:

```
/usr/share/man     -ro,intr     goofy:/usr/share/man
```

and the following would be a valid entry in an indirect map:

```
parsley            -ro,intr     veggies:/usr/greens
```

As you can see, the *key* (*parsley*) in the indirect map is begging for more information; where is the mount point for *parsley* really located? This is why you must either provide that information at the command line or through another map.

For example, if the above line is part of an indirect map called */etc/auto.veggies*, you would have to call it up either as:

```
automount /veggies /etc/auto.veggies
```

or specify, in the master map:

```
/veggies      /etc/auto.veggies          -ro,soft,nosuid
```

In either case, you are associating a mount directory (*veggies*) with the entries (*parsley* in this case) mentioned in the indirect map */etc/auto.veggies*. The end result is that the hierarchy */usr/greens* from the machine *veggies* will be mounted on */veggies/parsley* when needed.

Writing a master map

As stated above, the syntax for each line in the master map is:

```
Mount-point        Map                    [Mount-options]
```

A typical *auto.master* file would contain:

```
#Mount-point       Map                    Mount-options
/net               -hosts
/homedir           /etc/auto.home         -rw,intr
/-                 /etc/auto.direct       -ro,intr
```

The automounter recognises some special mount points and maps, which are explained below.

Mount point /-

In the example above, the mount point '/-' is a 'filler' that the automounter recognises as a directive not to associate the entries in */etc/auto.direct* with any directory. Rather, the mount points are to be those that are specified in the direct map. (Remember, in a direct map the key is a full pathname, so there will be no problem with the location of the directory.)

Mount point /homedir

The mount point */homedir* is to be the directory under which the entries listed in */etc/auto.home* (an indirect map) are to be mounted. They will be mounted by the automounter under */tmp_mnt/homedir*, with a symbolic link created between the directory named */homedir/directoryname* and the directory named */tmp_mnt/homedir/directoryname*.

Mount point /net

This map was introduced earlier in this chapter. The automounter will mount under the directory */net* all the entries in the special map *–hosts*. This is a built-in map that does not use any external files except the hosts database */etc/hosts* (or the *hosts.byname* NIS map if NIS is running on the machine). Note that since the automounter does not mount the entries until needed, the specific order is not important.

Once the automount daemon has been started, a user typing the command:

```
cd /net/gumbo
```

will change directory to the top of the hierarchy of files (ie the root filesystem) of the machine *gumbo*, as long as the machine is in the hosts database.

However, the user may not see under */net/gumbo* all the files and directories. This is because the automounter can mount only the *shared* filesystems of the host *gumbo* that are accessible from their machine, in accordance with the restrictions placed on sharing. For more information about sharing filesystems, refer to *exports*(5).

The actions of the automounter, when the command in the example above is issued are as follows:

1 *ping* the null procedure of the server's mount service to see if it's alive.

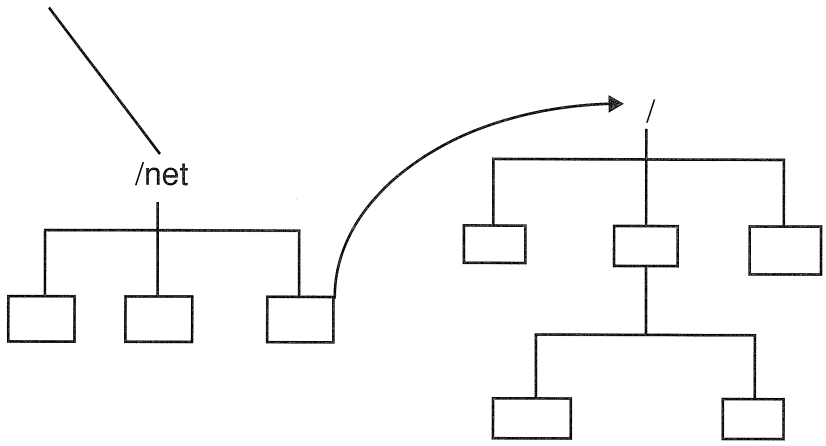
2 Request the list of shared hierarchies from the server.

3 Sort the shared list according to the length of the pathname (to ensure proper mounting order):

- */usr/src*
- */export/home*
- */usr/src/sccs*
- */export/root/blah*

4 Proceed down the list, mounting all the filesystems at mount points in */tmp_mnt* (creating the mount points as needed).

5 Return a symbolic link that points to the top of the recently mounted hierarchy:



Note that, unfortunately, the automounter has to mount all the filesystems that the server in question advertises for sharing. Even if the request is:

```
ls /net/gumbo/usr/include
```

the *automounter* mounts all of *gumbo*'s shared filesystems, not just */usr*.

In addition, any unmounting that occurs after a certain amount of time has passed takes place from the lowest level directory to the highest. This means if one of the directories at the top is busy, the automounter has to remount the hierarchy and try again later.

Nevertheless, the *-hosts* special map provides a very convenient way for users to access directories in many different hosts without having to use *rlogin* or *rsh*. (These remote commands have to establish communication through the network every time they are invoked.) Furthermore, users no longer have to ask you to modify their */etc/fstab* files or mount the directories by hand as super-user, so using the automounter will save you a lot of administration work.

Notice that both */net* and */homedir* are arbitrary names used to specify automount directories. They will be created by the automounter if they do not exist already.

Writing an indirect map

The syntax for an indirect map is:

```
key    [Mount-options]    location
```

where *key* is the name (not the full pathname) of the directory that will be used as the mount point. Once the key is obtained by the automounter, it is suffixed to the mount point associated with it either by the command line or by the master map that invokes the indirect map in question.

For example, one of the entries in the master map shown previously was:

```
/homedir            /etc/auto.home     -rw,intr
```

Here */etc/auto.home* is the name of the indirect map that will contain the entries to be mounted under */homedir*.

Using home directories of the form */home/server/user* rather than */home/user*, a typical *auto.home* might contain:

```
#key          mount-options       location
willow                            willow:/home/willow
cypress                           cypress:/home/cypress
poplar                            poplar:/home/poplar
pine                              pine:/export/pine
apple                             apple:/export/homedir
ivy                               ivy:/home/ivy
peach         -rw,nosuid          peach:/export/home
```

As an example, assume that the map above is on host *oak*. If user *laura* has an entry in the password database on *oak* specifying her home directory as */homedir/willow/laura*, then whenever she logs in to the machine *oak*, the automounter will mount (as */tmp_mnt/homedir/willow*) the directory */home/willow* residing on the machine *willow*.

If one of the directories is indeed *laura*, she will be in her home directory, which is mounted read/write and interruptible (according to the mount options specified in the master map).

Suppose, however, that *laura*'s home directory is specified as */homedir/peach/laura*. Whenever she logs in to *oak*, the automounter will mount the directory */export/home* from the machine *peach* under */tmp_mnt/homedir/peach*. Her home directory will be mounted *read/write*, *nosuid*, as specified in the *auto.home* map. Note that, any option specified in the *auto.home* map overrides all options given in the master map or from the command line.

Now assume that the following conditions occur:

- User *laura*'s home directory is listed in the password database as */homedir/willow/laura*.
- Machine *willow* exports its home directory */home* for the machines listed in *auto.home*.
- All those machines have a copy of the same *auto.home* and the same password database.
- Each user has their home directory specified as */homedir/<machine>/<username>*

Under these conditions, user *laura* can run the commands *login* or *rlogin* on any of these machines and have her home directory mounted in place for her.

Furthermore, now *laura* can also refer to the home directory of another user:

```
cd ~username
```

and the automounter will mount the home directory of the specified user for her (if all access permissions apply and the named user appears in the password file that is used on *laura*'s machine.

To use the automounter in this way, on a network without NIS, you will have to change all the relevant databases (such as */etc/passwd*) on all the machines on the network. On a network running NIS, ensure that all the relevant databases are propagated throughout the network. For more information about building and propagating NIS maps, refer to the chapter entitled *The Network Information Service* on page 219.

Although this example, shows how useful the automounter can be, it can be refined even further by mounting specific user directories only. For more information, refer to the section entitled *Specifying sub-directories* on page 259

Writing a direct map

The syntax for a direct map is:

```
key    [Mount-options]    location
```

where:

- *key* is the full pathname of the mount point. (Remember that in an indirect map, this is not a full pathname)
- *mount-options* are optional but, if present, override – for the entry in question – the options of the calling line, if any, or the defaults. (See the section entitled *Invoking automount* on page 263.)

- *location* is the location of the resource, specified as *server:pathname*.

Of all the maps, the entries in a direct map most closely resemble, in their simplest form, what their corresponding entries in */etc/fstab* might look like. For example, an entry that appears in */etc/fstab* as:

```
dancer:/usr/local /usr/local/tmp nfs ro 0 0
```

appears in a direct map as:

```
/usr/local/tmp -ro dancer:/usr/local
```

At its simplest level, a direct map contains separate lines with entries similar to those above. However, you can also add extra features into the map such as specifying multiple locations for the same mount and having multiple mounts from different locations.

Below is an example of a typical */etc/auto.direct* map, containing entries for both of these features:

```
/usr/local \
      /bin          -ro,soft     ivy:/export/local/arm3 \
      /share        -ro,soft     ivy:/export/local/share \
      /src          -ro,soft     ivy:/export/local/src
/usr/share/man      -ro,soft     oak:/usr/share/man \
                                 rose:/usr/share/man \
                                 willow:/usr/share/man
/usr/games          -ro,soft     peach:/usr/games
/usr/spool/news     -ro,soft     pine:/usr/spool/news
/usr/frame          -ro,soft     redwood:/usr/frame1.3 \
                                 balsa:/export/frame
```

Multiple mounts

A map entry can describe a multiplicity of mounts, where the mounts can be from different locations and with different mount options.

The first entry in the previous example is one long entry whose legibility has been improved by splitting it into three lines, using the backslash character and indenting the lines. This entry mounts */usr/local/bin*, */usr/local/share* and */usr/local/src* from the server *ivy*, with the options *read-only* and *soft*. The entry could also read:

```
/usr/local \
      /bin          -ro,soft     ivy:/export/local/arm3 \
      /share        -rw          willow:/export/local/share\
      /src          -ro,intr     oak:/export/local/src
```

where the options are different and more than one server is used.

Multiple mounts can be hierarchical. When filesystems are mounted hierarchically, each filesystem is mounted on a sub-directory within another filesystem. When the root of the hierarchy is referenced, the automounter mounts the whole hierarchy.

The concept of *root* here is very important. The symbolic link returned by the automounter to the kernel request is a path to the mount root. This is the root of the hierarchy that is mounted under */tmp_mnt*. This mount point should theoretically be specified as:

```
/parsley     /      -ro,intr     veg:/usr/greens
```

In practice, it is not specified because in a trivial case of a single mount as above, it is assumed that the location of the mount point is **at** the mount root or '/'. So instead of the above, it is perfectly acceptable, indeed preferable, to enter:

```
/parsley     -ro,intr     veg:/usr/greens
```

The mount point specification, however, becomes important when mounting a hierarchy; here the automounter must have a mount point for each mount within the hierarchy. The example above is a good illustration of multiple, non-hierarchical mounts under */usr/local* when the latter is already mounted.

A true hierarchical mounting is shown below:

```
/usr/local \
      /           -rw,intr     peach:/export/local \
      /bin        -ro,soft     ivy:/export/local/arm3 \
      /share      -rw          willow:/export/local/share\
      /src        -ro,intr     oak:/export/local/src
```

The mount points used here for the hierarchy are */*, */bin*, */share* and */src*. Note that these mount point paths are relative to the *mount* root, not the hosts's filesystem root. The first entry in the example above has '/' as its mount point. It is mounted **at** the mount root (*/usr/local*).

Note that the mount root has been mounted read/write so that the sub-directories below it can be mounted.

There is no requirement that the first mount of a hierarchy be at the mount root. The automounter will happily issue *mkdir*'s to build a path to the first mount point if it is not at the mount root.

Multiple locations

The example given for a direct map was:

```
/usr/local \
      /bin          -ro,soft    ivy:/export/local/arm3 \
      /share        -ro,soft    ivy:/export/local/share \
      /src          -ro,soft    ivy:/export/local/src
/usr/share/man      -ro,soft    oak:/usr/share/man \
                                rose:/usr/share/man \
                                willow:/usr/share/man
/usr/games          -ro,soft    peach:/usr/games
/usr/spool/news     -ro,soft    pine:/usr/spool/news
/usr/frame          -ro,soft    redwood:/usr/frame1.3 \
                                balsa:/export/frame
```

The mount points */usr/share/man* and */usr/frame* specify more than one location (three for the first, two for the second) for the directory to mount. This means that the mounting can be done from any of the two replicated locations.

This procedure makes sense only when you are mounting a hierarchy read-only, since theoretically, you would like to have some control over the locations of files you write or modify.

This manual page hierarchy is a good example for the use of this feature. In a large network, more than one server may export the current set of manual pages. It does not matter which server you mount them from, so long as the server is up and running and sharing its filesystems.

In the example above, multiple mount locations are expressed as a list of mount locations in the map entry:

/usr/share/man -ro,soft oak:/usr/share/man rose:/usr/share/man \
willow:/usr/share/man

This could also be expressed as a comma separated list of servers, followed by a colon and the pathname (as long as the pathname is the same for all the replicated servers):

```
/usr/share/man -ro,soft oak,rose,willow:/usr/share/man
```

This allows you to mount the manual pages from the servers *oak*, *rose* or *willow*. From this list of servers the automounter first selects those that are on the local network and *ping*'s these servers. This launches a series of RPC requests to the null procedure of the mount service in each server. (Note that the list does not imply any ordering.) The first server to respond is selected, and an attempt is made to mount from it.

This redundancy, very useful in an environment where individual servers may or may not be sharing their filesystems, is 'enjoyed' only at mount time. There is no status checking of the mounted-from server by the automounter once the mount occurs. If the server goes down while the mount is in effect, the filesystem becomes unavailable. An option here is to wait five minutes until the automounter unmounts the directory, and try again. Next time around the automounter will choose one of the available servers. Another option is to use the *unmount* command, inform the automounter of the change in the mount table (as described in the section entitled *The mount table* on page 265), and retry the mount.

Specifying sub-directories

The example *auto.home* file used earlier as an example of an indirect map was:

```
#key          mount-options      location
willow                           willow:/home/willow
cypress                          cypress:/home/cypress
poplar                           poplar:/home/poplar
pine                             pine:/export/pine
apple                            apple:/export/home
ivy                              ivy:/home/ivy
peach         -rw,nosuid         peach:/export/home
```

Given this *auto.home* file, every time a user wants to access a home directory in, say, */homedir/willow*, all the directories under it will be mounted.

A more efficient way of organising this file would be to use the user name as the key instead of the machine name:

```
#key          mount-options      location
john                             willow:/home/willow/john
mary                             willow:/home/willow/mary
joe                              willow:/home/willow/joe
```

The above example assumes that home directories are of the form */home/willow/user* rather than the normal */home/user* and are in the password file as */homedir/user*. If a user now enters the following command:

```
ls ~john ~mary
```

the automounter has to perform the 'equivalent' of the following actions:

```
mkdir /tmp_mnt/homedir/john
mount willow:/home/willow/john /tmp_mnt/homedir/john
ln -s /tmp_mnt/homedir/john /homedir/john
```

```
mkdir /tmp_mnt/homedir/mary
mount willow:/home/willow/mary /tmp_mnt/homedir/mary
ln -s /tmp_mnt/homedir/mary /homedir/mary
```

However, the complete syntax of a line in a direct or indirect map is actually:

```
key    [mount-option]     server:pathname[:subdirectory]
```

Until now, you used the form *server:pathname* to indicate the location. This is an ideal place for you to also indicate the subdirectory, like this:

```
#key          mount-options      location
john                             willow:/home/willow:john
mary                             willow:/home/willow:mary
joe                              willow:/home/willow:joe
```

Here, *john*, *mary* and *joe* are entries in the sub-directory field. Now when a user refers to *john*'s home directory, the automounter mounts *willow:/home/willow* and places a symbolic link between */tmp_mnt/homedir/willow/john* and */homedir/john*.

If the user then requests access to *mary*'s home directory, the automounter sees that *willow:/home/willow* is already mounted, so all it has to do is return the link between */tmp_mnt/homedir/willow/mary* and */homedir/mary*. In other words, the automounter now only has to do:

```
mkdir /tmp_mnt/homedir/john
mount willow:/home/willow /tmp_mnt/homedir
ln -s /tmp_mnt/homedir/john /homedir/john

ln -s /tmp_mnt/homedir/mary /home/mary
```

In general, it is a good idea to provide a sub-directory entry in the location when different map entries refer to the same mounted filesystem from the same server.

Substitutions

If you have a map with a lot of sub-directories specified, like:

```
#key          mount-options      location
john                             willow:/home/willow:john
mary                             willow:/home/willow:mary
joe                              willow:/home/willow:joe
able                             pine:/export/home:able
baker                            peach:/export/home:baker
              [ . . . ]
```

you should use string substitutions. You can use the ampersand character (&) to substitute the key wherever it appears again on the same line. Using the ampersand, the above map can be edited as follows:

```
#key          mount-options      location
john                             willow:/home/willow:&
mary                             willow:/home/willow:&
joe                              willow:/home/willow:&
able                             pine:/export/home:&
baker                            peach:/export/home:&
              [ . . . ]
```

Also, if the name of the server is the name of the key itself:

```
#key          mount-options      location
willow                           willow:/home/willow
peach                            peach:/home/peach
pine                             pine:/home/pine
oak                              oak:/home/oak
poplar                           poplar:/home/poplar
```

the ampersand can also be used to replace the name of the server:

```
#key          mount-options      location
willow                           &:/home/&
peach                            &:/home/&
pine                             &:/home/&
oak                              &:/home/&
poplar                           &:/home/&
```

Finally, notice that all the above entries have the same format. This permits you to use the catch-all substitute character, the asterisk (*). This can be used to reduce the whole thing to:

```
#key          mount-options      location
*                                &:/home/&
```

where each ampersand is substituted by the value of any given key.

You can also use key substitutions in a direct map, in situations like the following:

```
/usr/man      willow,cedar,poplar:/usr/man
```

which can be written as:

```
/usr/man      willow,cedar,poplar:&
```

Notice that the ampersand substitution uses the whole key string, so you cannot do something like:

```
/progs      &1,&2,&3:/export/src/progs
```

because the automounter would interpret it as:

```
/progs      /progs1,/progs2,/progs3:/export/src/progs
```

Special characters

Under certain circumstances you may have to mount directories whose names may confuse the automounter's map parser. An example might be a directory called *rc0:dk1*. This could result in an entry like:

```
/junk       -ro         vmserver:rc0:dk1
```

The presence of the two colons in the location field will confuse the automounter's parser. To avoid this confusion, use a backslash to escape the second colon and remove its special meaning:

```
/junk       -ro         vmserver:rc0\:dk1
```

You can also use double quotes, to hide blank spaces in a name:

```
/smile                  dentist:"/front teeth"/smile
```

Environment variables

You can use the value of an environment variable by prefixing a dollar sign ($) to its name. You can also use curly brackets to delimit the name of the variable from appended letters or digits.

The environment variables can be inherited from the environment or can be defined explicitly with the *-D* command line option. For example, if you want each client to mount client-specific files in the network in a replicated format, you could create a specific map for each client according to its name. The relevant line for the host *oak* would be:

```
/mystuff                pine,ivy.elm:/export/files/oak
```

and on the machine *willow* it would be:

```
/mystuff                pine,ivy.elm:/export/files/willow
```

This scheme is viable within a small network, but maintaining this kind of host specific map across a large network would soon become infeasible. The solution in this case would be to invoke the automounter with a command line similar to the following:

```
automount -D HOST=`hostname` ...
```

and have the entry in the direct map read:

```
/mystuff                    pine,ivy.elm:/export/files/$HOST
```

Now each host would find its own files in the *mystuff* directory, and the task of centrally administering and distributing the maps becomes easier.

Invoking automount

Once the maps are written, you should make sure that there are no equivalent entries in */etc/fstab*, and that all the entries in the maps refer to NFS shared files (see *exports*(5)).

The syntax to invoke the automounter and the options that can be used with it are detailed in the manual page, *automount*(8). The sub-options are the same as those for a standard NFS *mount*, with the exception of *bg* (background) and *fg* (foreground), which do not apply.

This section describes how to invoke the automounter given the following set of maps:

auto.master

```
#Mount-point       Map                Mount-options
/net               -hosts
/homedir           /etc/auto.home     -rw,intr
/-                 /etc/auto.direct   -ro,intr
```

auto.home

```
#key         mount-options      location
willow                          willow:/home/willow
cypress                         cypress:/home/cypress
poplar                          poplar:/home/poplar
pine                            pine:/export/pine
apple                           apple:/export/home
ivy                             ivy:/home/ivy
peach        -rw,nosuid         peach:/export/home
```

auto.direct

```
/usr/local \
      /bin         -ro,soft     ivy:/export/local/arm3 \
      /share       -ro,soft     ivy:/export/local/share \
      /src         -ro,soft     ivy:/export/local/src
/usr/man           -ro,soft     oak:/usr/man \
```

```
                                    rose:/usr/man \
                                    willow:/usr/man
/usr/games          -ro,soft        peach:/usr/games
/usr/spool/news     -ro,soft        pine:/usr/spool/news
/usr/frame          -ro,soft        redwood:/usr/frame1.3 \
                                    balsa:/export/frame
```

You can invoke the automounter (either from the command line or, preferably, from */etc/rc.local*) in one of the following ways:

- You can specify all arguments to the automounter without reference to the master map:

```
automount /net -hosts /homedir /etc/auto.home -rw,intr, /- /etc/auto.direct -ro,intr
```

- You can specify the same in the *auto.master* file, and instruct the automounter to look in it for instructions:

```
automount -f /etc/auto.master
```

  In this case the automounter will also look for an *auto.master* file in the NIS database but the local master map will override any entries that are the same in the NIS version.

- You can specify more mount points and maps in addition to those mentioned in the master map, as follows:

```
automount -f /etc/auto.master /src /etc/auto.src -ro,soft
```

- You can nullify one of the entries in the master map. (This is particularly useful if you use a map that you cannot modify and does not meet the needs of your machine):

```
automount -f /usr/lib/auto.master /homedir -null
```

- You can replace one of the entries with one of your own:

```
automount -f /usr/lib/auto.master /homedir /myown/auto.home -rw,intr
```

  In the example above, the automounter first mounts all items in the map */myown/auto.home* under the directory */homedir*. Then, when it consults the master file */usr/lib/auto.master* and reaches the line corresponding to */homedir* it simply ignores it, since it has already mounted a filesystem on it.

The first two commands used with the example *auto.master* file are equivalent as long as your network does not have a distributed *auto.master* file. This file is only available on networks running NIS.

If your network includes a distributed *auto.master* file, the second example would have to be modified in the following way to be equivalent to the first example:

```
automount -m -f /etc/auto.master
```

The *-m* option instructs the automounter not to consult the master file distributed by NIS. However, if you do not run NIS, you do not have to specify the *-m* option. The automounter is completely silent when it does not find a distributed master file.

You can log in as superuser and type any of the above commands at shell level to start the automounter. Ideally, you should edit */etc/rc* and include your preferred command there.

The temporary mount point

The default name for all mounts is */tmp_mnt*. Like the other names, this is arbitrary. It can be changed at invocation time using the *-M* option:

```
automount -M /auto ...
```

The above command causes all mounts to happen under the directory */auto*, which the automounter will create if it does not exist. Note, that you should not designate a directory in a read-only filesystem, as the automounter would not be able to modify anything.

The mount table

Every time the automounter mounts or unmounts a file hierarchy, it modifies */etc/mtab* to reflect the current situation. The automounter keeps an image in memory of */etc/mtab*, and updates this image every time it performs a mounting or an automatic unmounting.

If you use the *umount* command to unmount one of the automounted hierarchies (a directory under */tmp_mnt*), the automounter will not update */etc/mtab*. However you can force the automounter to do this using the following two commands:

```
ps -aux | grep automount | grep -v grep
root         2672 ...
```

This should return the process id (PID) number of the automounter (2672 in the above example). The automounter is designed so that on receiving a SIGHUP signal it re-reads */etc/mtab*. To send it that signal, type:

kill -HUP *PID*

where *PID* is the process id number you obtained from the previous *ps* command. In the example above, the command would be:

```
kill -HUP 2672
```

Modifying the maps

You can modify the automounter maps at any time, but remember that the automounter looks at the master and indirect maps only when it is invoked. If you want a modification to a map to take effect immediately, you need to either completely reboot the machine or kill the automounter and then restart it again.

However, changes to a direct map should take effect the next time the automounter has to mount the modified entry. For example, if you modify the file */etc/auto.direct* so that the directory */usr/src* is now mounted from a different server. The new entry takes effect immediately (if */usr/src* is not mounted at this time) when you try to access it. If it is already mounted, you can wait until the auto unmounting takes place, and then access it.

Alternatively, you can unmount the directory using the *umount* command then notify the automounter that the mount table has changed (as described in the section entitled *The mount table* on page 265), and then access it. The mounting should now be done from the new server, as specified in the modified map. However, note that changes in any keys cannot be registered in this way. You would need to restart the automounter.

Error messages related to automount

This section lists the error messages you are likely to encounter when using the automounter and suggests their likely causes:

- `no mount maps specified`

  The automounter was invoked with no maps to serve, and it cannot find the NIS *auto.master* map. This message is produced only when the *-v* option is given. Check the command again, or restart NIS.

- *`mapname`*`: Not found`

  The required map cannot be located. This message is produced only when the *-v* option is given. Check the spelling and pathname of the map name.

- `dir mountpoint must start with '/'`

  The automounter mount-point must be given as the full pathname. Check the spelling and pathname of the mount point.

- *`mountpoint`*`: Not a directory`

  The mount point exists, but it is not a directory. Check the spelling and pathname of the mount point.

- `hierarchical mountpoint:` *`mountpoint`*

The automounter will not allow itself to be mounted within an automounted directory.

- `WARNING: mountpoint not empty!`

  The mountpoint is not an empty directory. This message is produced only when the *-v* option is given, and it is only a warning. It means that the previous contents of the mount point will no longer be accessible.

- `Can't mount mountpoint: reason`

  The automounter cannot mount itself as the mount point specified. The reason should be self-explanatory.

- `hostname: filesystem already mounted on mountpoint`

  The automounter has been mounted on an already mounted-on mount point and is attempting to mount the same filesystem there. This will happen if an entry in */etc/fstab* also appears in an automounter map (either by accident or because the output of *mount -p* was re-directed to the file */etc/fstab*). Delete one of the redundant entries.

- `WARNING: hostname: filesystem already mounted on mountpoint`

  The automounter is mounting itself on top of an existing mount point (warning only).

- `couldn't create directoryname: reason`

  The automounter was not able to create a directory. The reason should be self-explanatory.

- `bad entry in map mapname "key"`
- `map mapname, key mapkey: bad`

  The map entry is malformed and the automounter cannot interpret it. Check the entry; perhaps there are characters in it that need escaping.

- `mapname: yp_err`

  Error in looking up an entry in a NIS map.

- `hostname: exports: rpc_err`

  Error getting share list from *hostname*. This indicates a server or network problem.

- `host: hostname not responding`
- `hostname: filesystem server not responding`

- `Mount of hostname: filesystem on mountpoint: reason`

  You will see these error messages after the automounter attempted to mount from *hostname* but either got no response or failed. This may indicate a server or network problem

- `mountpoint - pathname from hostname: absolute symbolic link`

  When mounting a hierarchy, the automounter has detected that the mount point is an absolute symbolic link (begins with '/'). The content of the link is *pathname*. This may have undesired consequences on the client. The contents of the link may be */usr*.

- `Cannot create socket for broadcast rpc: rpc_err`
- `Many_cast select problem: rpc_err`
- `Cannot send broadcast packet: rpc_err`
- `Cannot receive reply to many_cast: rpc_err`

  All these error messages indicate problems attempting to *ping* servers for a replicated filesystem. This may indicate a network problem.

- `trymany: servers not responding: reason`

  No server in a replicated list is responding. This may indicate a network problem.

- `Remount hostname:filesystem on mountpoint: server not responding`

  Attempted remount after unmount failed. This error message indicates a server problem.

- `NFS server (pid@mountpoint) not responding still trying`

  An NFS request made to the automount daemon with PID number serving the mount point specified has timed out. The automounter may be temporarily overloaded or dead. Wait a few minutes; if the conditions persist, the easiest solution is to reboot the client. If not, you have to exit all processes that use automounted directories (or, change to a non automounted directory in the case of a shell), kill the current automounter process and restart it again from the command line. If this does not work, then you have to reboot.

Secure networking

Introduction

NFS 4.0 includes an authentication system that greatly improves the security of network environments. The system is general enough to be used by other UNIX and non-UNIX systems.

The system uses DES encryption and public key cryptography to authenticate both users and workstations in the network. (DES stands for Data Encryption Standard.)

Public key cryptography is a cipher system that involves two keys: one public and the other private. The public key is published, while the private key is not; the private (or secret) key is used to encrypt and decrypt data. This system differs from some other public key cryptography systems in that the public and secret keys are used to generate a common key, which is used in turn to create a DES key.

Administering secure NFS

This section describes what you must do in order to create a secure filesystem over the NFS. Basically, filesystems must be exported secure, then mounted secure.

1. Edit the */etc/exports* file and add the *-secure* option for filesystems that should use DES authentication. Here's how a server might export secure home directories:

   ```
   /home     -secure,access=engineering
   ```

   In this example, *engineering* is the only network group with access to the */home* filesystem.

2. For each client workstation, edit */etc/fstab* (or have users edit their own files) to include *secure* as a mount option on each secure filesystem. Here's how a client might mount secure home directories:

   ```
   serv:/home/serv /home/serv nfs rw,secure,intr,bg 0 0
   ```

   In this example, the */home/serv* filesystem from server *serv* is mounted *hard*, *read/write*, and *secure*. If a client workstation does not mount a secure filesystem as *secure* everything works OK, except that users have access as *nobody* (user id 65534), rather than as themselves.

3. NFS now includes the */etc/publickey* database, which should contain three fields for each user: the user's net name, a public key, and an encrypted secret key. The corresponding NIS map is available to NIS clients as *publickey.byname* but the database should reside only on the NIS master. Make sure that the NIS master server has a *netid* database. As normally installed, the only user is *nobody*. This is convenient administratively, because users can establish their own public keys using *chkey* without administrator intervention. For even greater security, the administrator can establish public keys for everyone using *newkey*. Note that the NIS takes time to propagate a new map, so it's a good idea for users to run *chkey* or for you to run *newkey* at the end of every working day.
4. Verify that the *keyserv* daemon was started by */etc/rc.yp* and is still running. This daemon performs public key encryption and stores the private key (encrypted, of course) in */etc/keystore*:

```
% ps aux | grep keyserv
root  1354 0.0 4.1 128 296 p0 I Oct 15 0:13 keyserv
```

When users log in with *login* or remote log in with *rlogin* these programs use the typed password to decrypt the secret key stored in */etc/publickey*. This becomes the secret key, and gets passed to the *keyserv* daemon. If users don't type a password for *login* or *rlogin* either because their password field is empty or because their workstation is in the *hosts.equiv* file of the remote host, they can still place a secret key in */etc/keystore* by invoking the *keylogin*(1) program, provided that a separate file for storing the secret key for *root* is found. This file is called */.rootkey*.

Note that the user will only have to run *keylogin* once as every time the workstation running secure NFS is rebooted, the secret keys in */etc/keystore* are initialised again by the *keyserv* daemon. Therefore, you should take care not to delete */etc/keystore* or the file */etc/.rootkey*.

5. Note that all users (except *root*) must now invoke *yppasswd* or use *passwd* with the –y option to keep their secret key synchronized with their login password. Out of necessity, *passwd* re-encrypts the secret key for *root*. As a consequence, it is a bad idea to have entries for individual users in local */etc/passwd* files.
6. When you reinstall, move, or upgrade a workstation, save */etc/keystore* and */etc/.rootkey* along with everything else you normally save.

Note that if you *login*, *rlogin* or *telnet* to another workstation, and you are asked for your password, once you type it in, you've given away access to your own account. This is because your secret key is now stored in */etc/keystore* on that remote workstation.

This is only a concern if you don't trust the remote workstation. If this is the case, don't ever log in to a remote workstation if it asks for your password. Instead, use NFS to remote mount the files you're looking for. At this point there is no *keylogout* command, even though there should be.

The remainder of this chapter discusses the theory of secure networking. It is useful for you and also for users working in a secure environment.

Security shortcomings of NFS

The Remote Procedure Call (RPC) mechanism has proved to be a very powerful primitive for building network services. The most well-known of these services is the Network File System (NFS), a service that provides transparent file-sharing between heterogeneous workstation architectures and operating systems. The NFS is not without its shortcomings, however. Currently, an NFS server authenticates a file request by authenticating the workstation making the request, but not the user. On NFS-based filesystems, it is a simple matter of running *su* to impersonate the rightful owner of a file. But the security weaknesses of the NFS are nothing new. The familiar command *rlogin* is subject to exactly the same attacks as the NFS because it uses the same kind of authentication.

A common solution to network security problems is to leave the solution to each application. A far better solution is to put authentication at the RPC level. The result is a standard authentication system that covers all RPC-based applications, such as the NFS and the NIS (a name look-up service). Our system allows the authentication of users as well as workstations. The advantage of this is that it makes a network environment more like the older time-sharing environment. Users can log in on any workstation, just as they could log in on any terminal. Their login password is their passport to network security. No knowledge of the underlying authentication system is required. Our goal was a system that is as secure and easy to use as a time-sharing system.

Several remarks are in order. Given *root* access and a good knowledge of network programming, anyone is capable of injecting arbitrary data into the network, and picking up any data from the network. However, on a local area network, no workstation is capable of packet smashing – capturing packets before they reach their destination, changing the contents, then sending packets back on their original course – because packets reach all workstations, including the server, at the same time. Packet smashing is possible on a gateway, though, so make sure you trust all gateways on the network. The most dangerous attacks are those involving the injection of data, such as impersonating a user by generating the right packets, or recording conversations and replaying them later. These attacks affect data integrity. Attacks involving passive eavesdropping – merely listening to network traffic without

impersonating anybody – are not as dangerous, since data integrity had not been compromised. Users can protect the privacy of sensitive information by encrypting data that goes over the network. It's not easy to make sense of network traffic, anyway.

RPC authentication

RPC is at the core of the new network security system. To understand the big picture, it's necessary to understand how authentication works in RPC. RPC's authentication is open-ended: a variety of authentication systems may be plugged into it and may coexist on the network.

Currently, there are two types of authentication systems: UNIX and DES. UNIX authentication is the older, weaker system; DES authentication is the new system discussed in this chapter. Two terms are important for any RPC authentication system: *credentials* and *verifiers*. Using ID badges as an example, the credential is what identifies a person: a name, address, birth date, etc. The verifier is the photo attached to the badge: you can be sure the badge has not been stolen by checking the photo on the badge against the person carrying it. In RPC, things are similar. The client process sends both a credential and a verifier to the server with each RPC request. The server sends back only a verifier, since the client already knows the server's credentials.

UNIX authentication

UNIX authentication was used by the manufacturers of original network services. The credentials contain the client's workstation-name, *uid*, *gid* and group-access-list. The verifier contains *nothing*! There are two problems with this system. The glaring problem is the empty verifier, which makes it easy to cook up the right credential using *hostname* and *su*. If you trust all root users in the network, this is not really a problem. But many networks – especially at universities – are not this secure. The NFS tries to combat deficiencies in UNIX authentication by checking the source Internet address of *mount* requests as a verifier of the *hostname* field, and accepting requests only from privileged Internet ports. Still, it is not difficult to circumvent these measures, and NFS really has no way to verify the user id.

The other problem with UNIX authentication appears in the name UNIX. It is unrealistic to assume that all workstations on a network will be UNIX workstations. The NFS works with MS-DOS and VMS workstations, but UNIX authentication breaks down when applied to them. For instance, MS-DOS doesn't even have a notion of different user ids.

Given these shortcomings, it is clear what is needed in a new authentication system: operating system independent credentials, and secure verifiers. This is the essence of DES authentication discussed below.

DES authentication

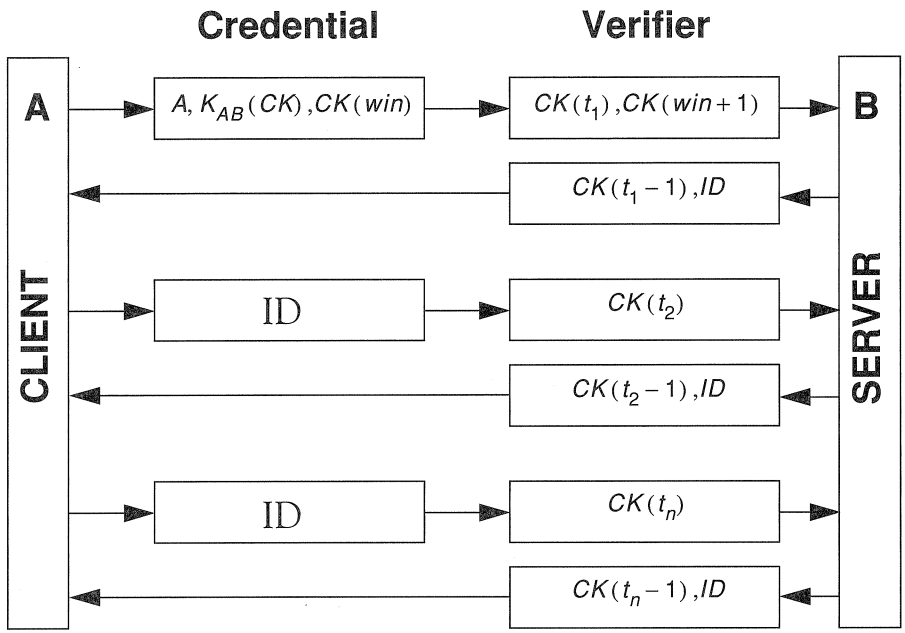
The security of DES authentication is based on a sender's ability to encrypt the current time, which the receiver can then decrypt and check against its own clock. The timestamp is encrypted with DES. Two things are necessary for this scheme to work:

- The two agents must agree on what the current time is, and
- the sender and receiver must be using the same encryption key.

If a network has time synchronization (Berkeley's TEMPO for example), then client/server time synchronization is performed automatically. However, if this is not available, timestamps can be computed using the server's time instead of network time. In order to do this, the client asks the server what time it is, before starting the RPC session, then computes the time difference between its own clock and the server's. This difference is used to offset the client's clock when computing timestamps. If the client and server clocks get out of sync to the point where the server begins rejecting the client's requests, the DES authentication system just re-synchronises with the server.

Here's how the client and server arrive at the same encryption key. When a client wishes to talk to a server, it generates at random a key to be used for encrypting the timestamps (among other things). This key is known as the *conversation key*, CK. The client encrypts the conversation key using a public key scheme, and sends it to the server in its first transaction. This key is the only thing that is ever encrypted with

public key cryptography. The particular scheme used is described further on in this chapter. For now, suffice to say that for any two agents A and B, there is a DES key K_{AB} that only A and B can deduce. This key is known as the *common key*, K_{AB}.



The figure above illustrates the authentication protocol in more detail, describing client A talking to server B. A term of the form $K(x)$ means x encrypted with the DES key K. Examining the figure, you can see that for its first request, the client's credential contains three things:

- its name A,
- the conversation key CK encrypted with the common key K_{AB},
- and a thing called *win* (window) encrypted with CK.

What the window says to the server, in effect, is this:

I will be sending you many credentials in the future, but there may be crackers sending them too, trying to impersonate me with bogus timestamps. When you receive a timestamp, check to see if your current time is somewhere between the timestamp and the timestamp plus the window. If it's not, please reject the credential.

For secure NFS filesystems, the window currently defaults to 30 minutes. The client's verifier in the first request contains the encrypted timestamp and an encrypted verifier of the specified window, *win + 1*. The reason this exists is the following. Suppose somebody wanted to impersonate *A* by writing a program that instead of filling in the encrypted fields of the credential and verifier, just stuffs in random bits. The server will decrypt *CK* into some random DES key, and use it to decrypt the window and the timestamp. These will just end up as random numbers. After a few thousand trials, there is a good chance that the random window/timestamp pair will pass the authentication system. The window verifier makes guessing the right credential much more difficult.

After authenticating the client, the server stores four things into a credential table:

- the client's name *A*
- the conversation key *CK*
- the window
- the timestamp

The reason the server stores the first three things should be clear: it needs them for future use. The reason for storing the timestamp is to protect against replays. The server will only accept timestamps that are chronologically greater than the last one seen, so any replayed transactions are guaranteed to be rejected. The server returns to the client in its verifier an index ID into its credential table, plus the client's timestamp minus one, encrypted by *CK*. The client knows that only the server could have sent such a verifier, since only the server knows what timestamp the client sent. The reason for subtracting one from it is to insure that it is invalid and cannot be reused as a client verifier.

The first transaction is rather complicated, but after this things go very smoothly. The client just sends its ID and an encrypted timestamp to the server, and the server sends back the client's timestamp minus one, encrypted by *CK*.

Public key encryption

The particular public key encryption scheme used is the Diffie-Hellman method. The way this algorithm works is to generate a *secret key* SK_A at random and compute a *public key* PK_A using the following formula (*PK* and *SK* are 192 bit numbers and α is a well-known constant):

$$PK_A = \alpha^{SK_A}$$

Public key PK_A is stored in a public directory, but secret key SK_A is kept private. Next, PK_B is generated from SK_B in the same manner as above. Now common key K_{AB} can be derived as follows:

$$K_{AB} = PK_B^{SK_A} = ((\alpha^{SK_B})^{SK_A}) = \alpha^{(SK_A SK_B)}$$

Without knowing the client's secret key, the server can calculate the same common key K_{AB} in a different way, as follows:

$$K_{AB} = PK_A^{SK_B} = (\alpha^{SK_A})^{SK_B} = \alpha^{(SK_A SK_B)}$$

Notice that nobody else but the server and client can calculate K_{AB}, since doing so requires knowing either one secret key or the other. All of this arithmetic is actually computed *modulo* M, which is another well-known constant. It would seem at first that somebody could guess your secret key by taking the logarithm of your public one, but M is so large that this calculation would require an enormous amount of computing power. To be secure, K_{AB} has too many bits to be used as a DES key, so 56 bits are extracted from it to form the DES key.

Both the public and the secret keys are stored indexed by net name in the NIS map *publickey.byname* the secret key is DES-encrypted with your login password. When you log in to a workstation, the *login* program grabs your encrypted secret key, decrypts it with your login password, and gives it to a secure local keyserver to save for use in future RPC transactions. Note that ordinary users do not have to be aware of their public and secret keys. In addition to changing your login password, the *yppasswd* program randomly generates a new public/secret key pair as well.

The keyserver *keyserv* is an RPC service local to each workstation that performs all of the public key operations, of which there are only three. They are:

- `setsecretkey(secretkey)`
- `encryptsessionkey(servername, des_key)`
- `decryptsessionkey(clientname, des_key)`

setsecretkey() tells the keyserver to store away your secret key SK_A for future use; it is normally called by *login*. The client program calls *encryptsessionkey()* to generate the encrypted conversation key that is passed in the first RPC transaction to a server. The keyserver looks up *servername* public key and combines it with the client's secret key (set up by a previous *setsecretkey()* call) to generate the key that encrypts *des_key*. The

server asks the keyserver to decrypt the conversation key by calling *decryptsessionkey()*. Note that implicit in these procedures is the name of caller, who must be authenticated in some manner. The keyserver cannot use DES authentication to do this, since it would create deadlock. The keyserver solves this problem by storing the secret keys by *uid* and only granting requests to local root processes. The client process then executes a *setuid* process, owned by root, which makes the request on the part of the client, telling the keyserver the real *uid* of the client. Ideally, the three operations described above would be system calls, and the kernel would talk to the keyserver directly, instead of executing the *setuid* program.

Naming of network entities

The old UNIX authentication system has a few problems when it comes to naming. Recall that with UNIX authentication, the name of a network entity is basically the *uid*. These *uid* are assigned per NIS naming domain, which typically spans several workstations. We have already stated one problem with this system, that it is too UNIX system oriented, but there are two other problems as well. One is the problem of *uid* clashes when domains are linked together. The other problem is that the super-user (with *uid* of 0) should not be assigned on a per-domain basis, but rather on a per-workstation basis. By default, the NFS deals with this latter problem in a severe manner: it does not allow root access across the network by *uid* 0 at all.

DES authentication corrects these problems by basing naming upon new names that we call *net names*. Simply put, a net name is just a string of printable characters, and fundamentally, it is really these net names that we authenticate. The public and secret keys are stored on a per-net name, rather than per-user name, basis. The NIS map *netid.byname* maps the net name into a local *uid* and group-access-list, though other non-NFS types of network environments may map the net name into something else.

We solve the Internet naming problem by choosing globally unique net names. This is far easier then choosing globally unique user ids. In the NFS environment, user names are unique within each NIS domain. Net names are assigned by concatenating the operating system and user id with the NIS and ARPA domain names. For example, a UNIX system user with a user id of 508 in the domain *eng.sun*.COM would be assigned the following net name: *unix.508@eng.sun*.COM. A good convention for naming domains is to append the ARPA domain name (COM, EDU, GOV, MIL) to the local domain name. Thus, the NIS domain *eng* within the ARPA domain *sun*.COM becomes *eng.sun*.COM

The problem of having multiple super-users per domain is solved by assigning net names to workstations as well as to users. A workstation's net name is formed much like a user's. For example, a UNIX workstation named *hal* in the same domain as before

has the net name *unix.hal@eng.sun*.COM. Proper authentication of workstations is very important for discless workstations that need full access to their home directories over the net.

Non-NFS environments will have other ways of generating net names, but this does not preclude them from accessing the secure network services of the NFS environment. To authenticate users from any remote domain, all that has to be done is make entries for them in two NIS databases. One is an entry for their public and secret keys, the other is for their local user id and group-access-list mapping. Upon doing this, users in the remote domain will be able access all of the local network services, such as the NFS and remote logins.

Applications of DES authentication

The first application of DES authentication is a generalized NIS update service. This service allows users to update private fields in NIS databases. So far the NIS maps *hosts*, *ethers*, *bootparams* and *publickey* employ the DES-based update service.

The second application of DES authentication is the most important: a more secure Network File System. There are three security problems with the old version of NFS using UNIX authentication.

The first is that verification of credentials occurs only at mount time when the client gets from the server a piece of information that is its key to all further requests: the *file handle*. Security can be broken if one can figure out a file handle without contacting the server, perhaps by tapping into the net or by guessing. After an NFS file system has been mounted, there is no checking of credentials during file requests, which brings up the second problem.

If a file system has been mounted from a server that serves multiple clients (as is typically the case), there is no protection against someone who has root permission on their workstation using *su* (or some other means of changing *uid*) gaining unauthorized access to other people's files.

The third problem with the NFS is the severe method it uses to circumvent the problem of not being able to authenticate remote client super-users: denying them super-user access altogether.

The new authentication system corrects all of these problems. Guessing file handles is no longer a problem since in order to gain unauthorized access, the miscreant will also have to guess the right encrypted timestamp to place in the credential, which is a virtually impossible task. The problem of authenticating root users is solved, since the new system can authenticate workstations. At this point, however, secure NFS is not used for root filesystems provided to discless workstations.

Actually, the level of security associated with each filesystem may be altered by the administrator. The file */etc/exports* contains a list of filesystems and which workstations may mount them. By default, filesystems are exported with UNIX authentication, but you can set them up to be exported with DES authentication by specifying *-secure* on any line in the */etc/exports* file. Associated with DES authentication is a parameter: the maximum window size that the server is willing to accept.

Security issues remaining

There are several ways to break DES authentication, but using *su* is not one of them. In order to be authenticated, your secret key must be stored by your workstation. This usually occurs when you login, with the *login* program decrypting your secret key with your login password, and storing it away for you. If somebody tries to use *su* to impersonate you, it won't work, because they won't be able to decrypt your secret key. Editing */etc/passwd* isn't going to help them either, because the thing they need to edit, your encrypted secret key, is stored in the NIS. If you log into somebody else's workstation and type in your password, then your secret key would be stored in their workstation and they could use *su* to impersonate you. But this is not a problem since you should not be giving away your password to a workstation you don't trust anyway. Someone on that workstation could just as easily change *login* to save all the passwords it sees into a file.

Not having *su* to employ any more, how can nefarious users impersonate others now? Probably the easiest way is to guess somebody's password, since most people don't choose very secure passwords. We offer no protection against this; it's up to each user to choose a secure password.

The next best attack would be to attempt replays. For example, let's say I have been squirreling away all of your NFS transactions with a particular server. As long as the server remains up, I won't succeed by replaying them since the server always demands timestamps that are greater than the previous ones seen. But suppose I go and pull the plug on your server, causing it to crash. As it reboots, its credential table will be clean, so it has lost all track of previously seen timestamps, and now I am free to replay your transactions. There are few things to be said about this. First of all, servers should be kept in a secure place so that no one can go and pull the plug on them. But even if they are physically secure, servers occasionally crash without any help. Replaying transactions is not a very big security problem, but even so, there is protection against it. If a client specifies a window size that is smaller than the time it takes a server to reboot (5 to 10 minutes), the server will reject any replayed transactions because they will have expired.

There are other ways to break DES authentication, but they are much more difficult. These methods involve breaking the DES key itself, or computing the logarithm of the public key, both of which would take months of computing time on a supercomputer. DES authentication does not seek to provide super-secure network computing, but rather gives something as secure as a good time-sharing system.

There is another security issue that DES authentication does not address, and that is tapping of the net. Even with DES authentication in place, there is no protection against somebody watching what goes across the net. This is not a big problem for most things, such as the NFS, since very few files are not publically readable, and besides, trying to make sense of all the bits flying over the net is not a trivial task. For logins, this is a bit of a problem because you wouldn't want somebody to pick up your password over the net. As has been mentioned before, a side effect of the authentication system is a key exchange, so that the network tapping problem can be tackled on a per-application basis.

Performance

Public key systems are known to be slow, but there is not much actual public key encryption going on in the system. Public key encryption only occurs in the first transaction with a service, and even then, there is caching that speeds things up considerably. The first time a client program contacts a server, both it and the server will have to calculate the common key. The time it takes to compute the common key is basically the time it takes to compute an exponential modulo M. This takes roughly 1 second, which means it takes 2 seconds just to get things started, since both client and server have to perform this operation. This is a long time, but you have to wait only the first time you contact a workstation. Since the keyserver caches the results of previous computations, it does not have to recompute the exponential every time.

The most important service in terms of performance is the secure NFS, which is acceptably fast. The extra overhead that DES authentication requires versus UNIX authentication is the encryption. A timestamp is a 64-bit quantity, which also happens to be the DES block size. Four encryption operations take place in an average RPC transaction: the client encrypts the request timestamp, the server decrypts it, the server encrypts the reply timestamp, and the client decrypts it. The average time it takes to encrypt one block is about 1.2 milliseconds. So, the extra time added to the round trip time is about 5 milliseconds. The round trip time for the average NFS request is about 20 milliseconds, resulting in a performance hit of 25 percent if one has encryption hardware. Remember that this is the impact on network performance. The fact is that not all file operations go over the wire, so the impact on total system performance will actually be lower than this. It is also important to remember that security is optional, so environments that require higher performance can turn it off.

Problems with booting and setuid programs

Consider the problem of a workstation rebooting, say after a power failure at some strange hour when nobody is around. All of the secret keys that were stored get wiped out, and now no process will be able to access secure network services, such as accessing an NFS filesystem. The important processes at this time are usually root processes, which would work if root's secret key were stored away, but nobody is around to type the password that decrypts it. The solution to this problem is to store root's decrypted secret key in a file (normally */etc/.rootkey*), which the keyserver can read. The use of this file is described in the section entitled *Administering secure NFS* on page 269.

This solution works well for disc-based workstations that can store the secret key on a physically secure local disc, but not so well for discless workstations, whose secret key must be stored across the network. If you tap the net when a discless workstation is booting, you will find the decrypted key. This is not very easy to accomplish, though.

Another booting problem is that discless workstation booting is not totally secure. It is possible for somebody to impersonate the boot-server, and boot a devious kernel that, for example, makes a record of your secret key on a remote workstation. The problem is that the security system is set up to provide protection only after the kernel and the keyserver are running. Before that, there is no way to authenticate the replies given by the boot server. This is not a serious problem, because it is highly unlikely that somebody would be able to write such a kernel without source code. Also, the crime is not without evidence. If you polled the net for boot-servers, you would discover the devious boot-server's location.

Not all *setuid* programs will behave as they should. For example, if a *setuid* program is owned by *dave* who has not logged into the workstation since it booted, then the program will not be able to access any secure network services as *dave*. The good news is that most *setuid* programs are owned by root, and since root's secret key is always stored at boot time, these programs will behave as they always have.

Conclusion

The goal for network security was to build a system as secure as a time-shared system. This goal has been met with DES authentication. The way you are authenticated in a time-sharing system is by knowing your password. With DES authentication, the same is true. In time-sharing the only person trusted is the system administrator, who has an ethical obligation not to change any users password in order to impersonate that user. In the DES authentication system, the network administrator is trusted not to alter any users entry in the public key database. In one sense, this system is even more secure than time-sharing, because it is useless to place a tap on the network in hopes

of catching a password or encryption key, since these are encrypted. Most time-sharing environments do not encrypt data emanating from the terminal; users must trust that nobody is tapping their terminal lines.

DES authentication is perhaps not the ultimate authentication system. In the future it is likely there will be sufficient advances in algorithms and hardware to render the public key system as currently defined, useless. But at least DES authentication offers a smooth migration path for the future. Syntactically speaking, nothing in the protocol requires the encryption of the conversation key to be Diffie-Hellman, or even public key encryption in general. To make the authentication stronger in the future, all that needs to be done is to strengthen the way the conversation key is encrypted. Semantically, this will be a different protocol, but the beauty of RPC is that it can be plugged in and live peacefully with any authentication system.

For the present at least, DES authentication should satisfy the requirements of most organisations requiring a secure networking environment. The system should be secure enough for use in unfriendly networks, such as a student-run university workstation environment. The price for this security is not high. Nobody has to carry around a magnetic card or remember any hundred digit numbers. Users use their login password to authenticate themselves, just as before. There is a small impact on performance, but if this worries you and you have a friendly net, you can turn authentication off.

Using the RISCiXFS module

Introduction

You can perform some simple System Administration duties from RISC OS using the RISC iX filing system module (called the RISCiXFS module for short). In RISC OS, software modules are the standard method of adding applications programs or extensions to the operating system.

The RISCiXFS module provides you with a sub-set of the standard RISC iX system calls and allows you to:

- alter the boot procedure
- check the state of the RISC iX filing system prior to booting – using *fsck*
- make a new filing system – using *mkfs*

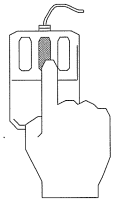
All the above operations can be performed from either the RISC OS Desktop by using the maintenance menu from the !RISCiX application, or from the RISC OS Supervisor command line.

This chapter describes the use of the RISCiXFS module from the RISC OS Desktop and from the Supervisor command line. It also includes a full list of the * commands supported by the RISCiXFS module and also details the RISC iX system calls that are supported from RISC OS.

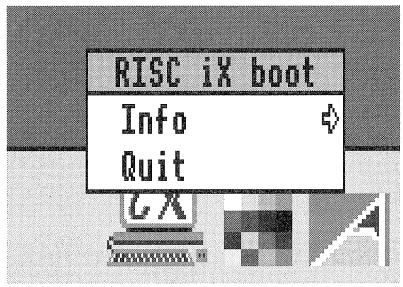
For information about bringing the system from RISC iX to RISC OS, refer to the section entitled *Selecting RISC OS from RISC iX* on page 13.

For more general information about RISC OS * commands and modules, refer to the *RISC OS User Guide*.

Options available from the icon bar

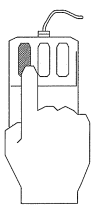


There are two menu options available from the icon bar. To view the options, firstly position the pointer over the RISC iX '*iX*' icon on the icon bar and press the middle mouse button. The following menu is displayed:



- **Info** – This gives you standard information about the RISC iX bootstrap from RISC OS.
- **Quit** – Clicking on this menu item exits the RISC iX start-up application and removes it from memory.

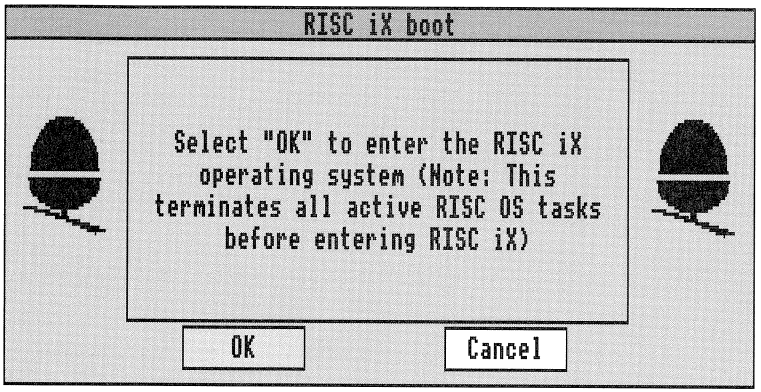
Selecting the maintenance menu

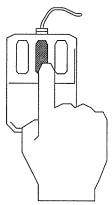


This section describes the *maintenance menu* which provides a set of options for carrying out many different system administration duties. Each of the options available from this menu will be described in this section

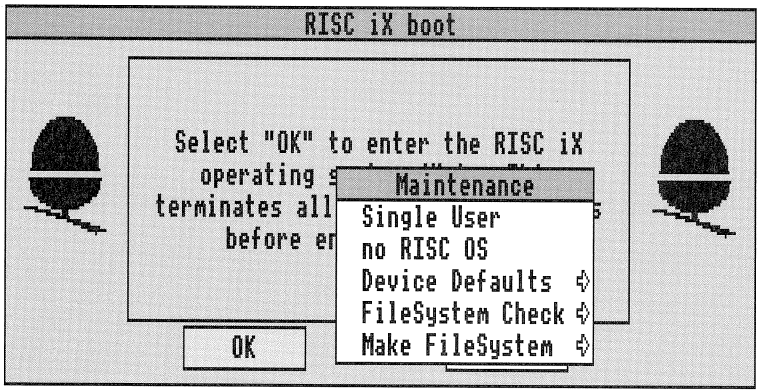
To select the maintenance menu, follow these steps:

1 Click the left mouse button on the RISC iX icon on the icon bar. This displays the RISC iX boot dialogue box:





2 Position the mouse pointer over the middle of this dialogue box and click the middle mouse button. The maintenance menu will be displayed:



The entries displayed on the menu have the following meaning:

- **Single User** – Clicking on this starts RISC iX. The start-up procedure stops when 'single-user' mode is achieved.
- **no RISC OS** – When this is selected (a tick mark is shown), RISC OS is bypassed on start-up. The default RISC iX start-up procedure starts as normal.

- **Device Defaults** – This option allows you to alter the device from which the system is booted
- **FileSystem Check** – This option allows you to check the consistency of the filesystem.
- **Make FileSystem** – This option allows you to construct a RISC iX filesystem on a device.

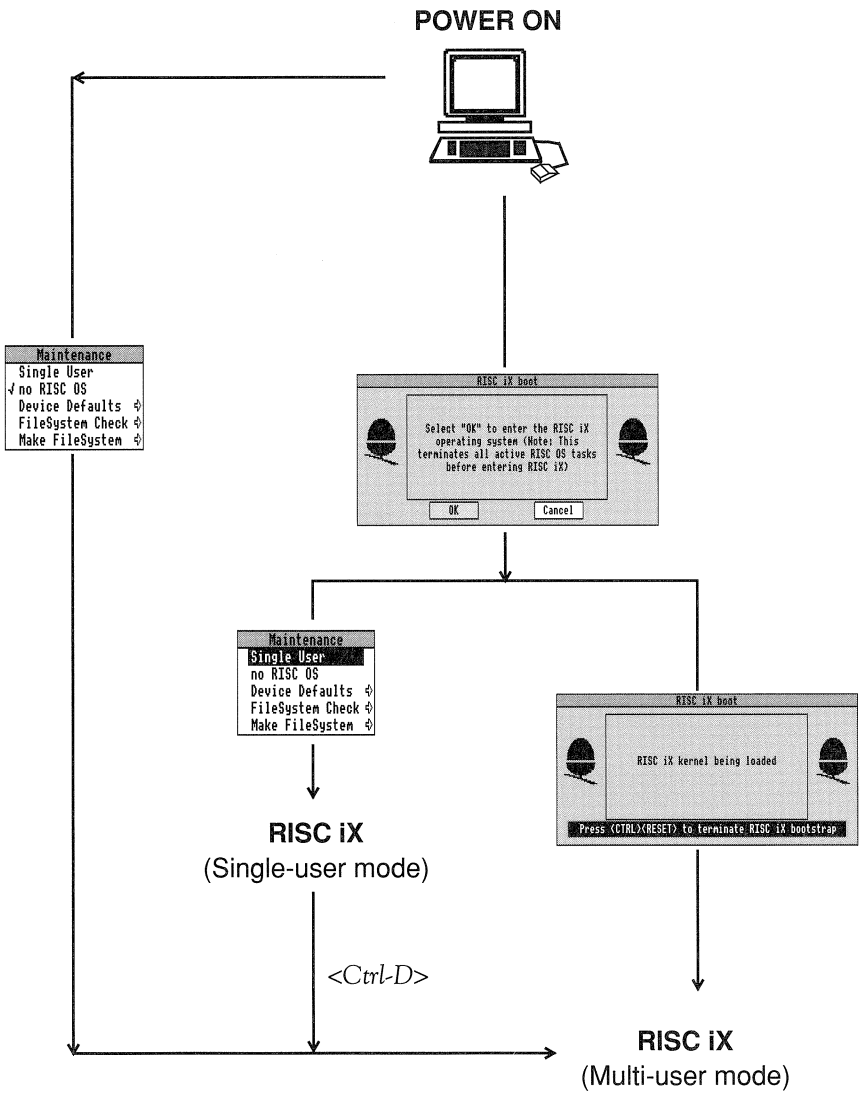
These use of these options are more fully described in the next few sections.

Altering the boot procedure

As normally supplied, the system boots up automatically into RISC iX and enters multi-user mode when switched on.

The first two options in the maintenance menu allow you to change this boot procedure in two ways. Firstly, by starting up the machine in RISC OS and secondly, by booting the machine into single-user mode.

The following diagram shows a schematic of the different boot options available:



Starting the machine up in RISC OS

To change the normal configuration of your machine so that it starts up in RISC OS instead of directly entering RISC iX, click the left mouse button over the *noRISCOS* menu option, removing the tick alongside the option.

The workstation will now start up in RISC OS following a power on.

Booting from RISC OS into RISC iX single-user mode

To boot RISC iX into single-user mode from RISC OS, bring up the maintenance menu as previously described. Move the pointer to *Single User* and click the left mouse button. The RISC iX kernel is bootstrapped and the system is started up in single-user mode.

Before booting RISC iX the system will check to see if there are any active RISC OS tasks running, such as a file being edited that has not yet been saved. If there are any active tasks, you will be warned. You can choose to ignore the warning and continue with the boot request or you can cancel the boot request and save the open file.

If no active RISC OS tasks are found, RISC iX will be booted into single-user mode from where you can perform many system maintenance tasks. After you have performed any maintenance tasks, you can bring the system up into multi-user mode from single-user mode by pressing <Ctrl-D>.

Note that when the system is in single-user mode, you will be logged in as *root*. As it is possible to bring the system down to single-user mode without having to give the *root* password, an important security issue is raised. This security aspect of the system is discussed in more detail in the section entitled *Security on the system* on page 290.

Booting from RISC OS into RISC iX multi-user mode

To boot RISC iX into multi-user mode from RISC OS, bring up the RISC iX boot dialogue box and click the left mouse button over the *OK* option in the box. All active RISC OS tasks are terminated and RISC iX is booted into multi-user mode.

Changing the boot device

The boot device can be changed from the maintenance menu. To change the name of the boot device, move to the *Device Defaults* submenu of the maintenance menu, move the cursor to the right to bring up the window and, using the left mouse button, change the name of the device that you wish to boot RISC iX from.

For example, to boot from a floppy disc that has been initialised as a UNIX filesystem, change the device name to *fd* with major number 0, unit number 0 and partition number 0, ie *fd0(0,0)*.

After you have set the new boot device, just click the left mouse button, to go back to the normal boot window.

Running fsck

The filesystem consistency check program, *fsck*, can be run from the maintenance menu and is primarily used:

- as a preliminary check on the state of a newly initialised filesystem, or
- as a diagnostic tool, if RISC iX fails to boot successfully.

To run *fsck*, click the left mouse button on *FileSystem Check* from the maintenance menu. A filesystem check will be done, by default, on the internal hard disc device (*/dev/st0a*).

To change the name of the device, move to the *FileSystem Check* submenu of the maintenance menu and change the name of the device that you wish to run *fsck* on. For example, to run *fsck* on a floppy disc that has been initialised as a UNIX filesystem, change the device name to */dev/fdf1024*.

For a complete description of the *fsck* program, refer to the manual page for *fsck*(8).

Running mkfs

A new UNIX filesystem can be constructed on a named device, using *mkfs* from the maintenance menu.

To run *mkfs*, click the left mouse button on *Make FileSystem* from the maintenance menu. A new filesystem will be constructed, by default, on the device (*/dev/sd0b*).

To change the name of the device, move to the *Make FileSystem* submenu of the maintenance menu and change the name of the device that you wish to run *mkfs* on.

For example, to run *mkfs* on a floppy disc that has been formatted using the command:

```
ffd 1024
```

change the command to:

```
mkfs /dev/fdf1024 1600 10 2 4096 1024
```

For full documentation on the *mkfs* program, refer to the manual page for *mkfs*(8).

Once the filesystem has been initialised, it can be mounted as a RISC iX filesystem using *FMount under RISC OS. For more information about this command and other * Commands supported by the RISCiXFS module, refer to the section entitled * *Commands supported* on page 291.

The disc can also be mounted from RISC iX using the *mount*(8) command. For more information refer to the chapter entitled *Using the floppy disc utilities* on page 115.

Security on the system

To prevent mischievous users from using the RISCiXFS module to bring up the system in single-user mode and wreaking untold havoc on the system, you can load the module *secureboot*.

With *secureboot* loaded, the system will automatically boot into RISC iX in multi-user mode, completely by-passing the RISCiXFS module and all the features that it supports.

Enabling secureboot

To re-configure your workstation so that it uses *secureboot* instead of RISCiXFS, bring the workstation down to RISC OS by typing:

```
halt -RISCOS
```

In RISC OS Supervisor mode (obtained by pressing the function key <F12>), rename the original *!Boot* file that loads the RISCiXFS module to some other name (say *RFSBoot*) and rename *SecureBoot* to *!Boot*:

```
*rename $.!Boot $.RFSBoot
*rename $.SecureBoot $.!Boot
```

Then type the following *Configure commands:

```
*Configure RMAsize 20
*Configure boot
*Configure dir
*Configure drive 4
*dir :4.$
*opt 4 2
```

Press <Ctrl-Break> to register the changes in CMOS RAM. (For a full list of the default CMOS RAM settings for your workstation, refer to the *Installation Guide*.)

After you have pressed <Ctrl-Break>, the workstation will quickly display the following prompt:

```
**|> !Boot (for secure RISC iX bootstrapping)
**rmload Secureboot
```

Once the module is loaded, the system will automatically boot up into RISC iX in multi-user mode.

From now on, whenever the workstation is switched on or rebooted it will always boot up into RISC iX in multi-user mode, providing of course that no system error occurs during the booting procedure.

All commands that previously brought the workstation to RISC OS (for example *halt -RISCOS*) will just cause the workstation to be re-booted.

Disabling secureboot

To disable *secureboot*, bring down the workstation by logging in as *halt*. Then turn the workstation on, while holding down the <Shift> key. This prevents the *!Boot* file from being executed and so the workstation remains in RISC OS.

In RISC OS, you can rename the Boot files back to their original names:

```
*rename $.!Boot $.SBBoot
*rename $.RFSBoot $.!Boot
```

Press <Ctrl-Break> to register the changes in CMOS RAM. The workstation should now load the RISCiXFS module as before.

Other security considerations

As a final security measure, you should attempt to hide or cover the <RESET> switch at the back of the keyboard and the power supply switch at the back of the workstation unit.

If these two switches are inaccessible and if the *secureboot* module is loaded, your system is quite secure and can be made available for public use. However, no matter how security conscious you are, it is almost impossible to prevent malicious users from gaining access to your system.

* Commands supported

This section details the * commands supported by the RISCiXFS module that can be used in Supervisor mode from RISC OS. For more general information about * commands and modules, refer to the *RISC OS User Guide*.

*Boot

Loads a file as a RISC iX kernel and executes it.

Syntax

```
*Boot [<filename [<root> [<swap>]]]
```

Parameters

filename is the name of the file that is to be loaded as a RISC iX kernel. The default filename used is */vmunix*.

root refers to the device partition that will be used for the root filesystem '/'. This parameter conforms to the naming convention *dd*M(*U*,*P*) where:

dd is a two character device driver identifier:

st – st506 hard disc driver

fd – ADFS physical format floppy disc driver

sd – SCSI hard disc device

M – 0 to 7; the major hardware controller number

U – 0 to 63; the unit device number (drive number)

P – 0 to 7; the partition number on the device

swap refers to the device partition that will be used for the swap area. This parameter also conforms to the naming convention *dd*M(*U*,*P*) as described above.

Use

If *Boot is issued with no parameters, the command will execute the default kernel image */vmunix* and automatically enter multi-user mode.

The default device partitions used for the root filesystem and the swap area are defined by the RISC iX kernel. Normally root will be set to the boot device (as controlled by CMOS RAM parameter settings) and the swap area will be set to partition 1 on the root device.

Alternative kernel images and start-up device information can be given. If alternative parameters are used, the kernel will be started in single-user mode.

Example

To boot off a floppy disc, using an internal st506 driver as the root partition and partition 1 of the first SCSI hard disc drive as the swap partition, type:

```
*FMount fd0(0,0) /mnt
*Boot /mnt/vmunix st0(0,0) sd0(0,1)
```

*Configure Device

Sets the device of the filesystem mounted when the RISCiXFS module is loaded.

Syntax

```
*Configure Device <device name><major device>
```

Parameters

device name a two character device driver identifier:

st – st506 hard disc driver

fd – ADFS physical format floppy disc driver

sd – SCSI hard disc device

major device 0 to 7; the major hardware controller number

Use

*Configure Device sets the device of the filesystem that is mounted when the RISCiXFS module is loaded.

Example

```
*Configure Device st0
```

Related commands

*Configure Partition, *Configure Unit.

*Configure noRISCOS

Controls the automatic bootstrapping of the RISC iX kernel.

Syntax

```
*Configure noRISCOS On|Off
```

Arguments

On or *Off* (default *On*)

Use

*Configure noRISCOS controls the automatic bootstrapping of the RISC iX kernel when the RISCiXFS module is loaded.

If this flag is set to *On* the module does NOT stop for confirmation to bootstrap the kernel and will automatically start up RISC iX in multi-user mode – RISC OS is bypassed. If this flag is set to *Off*, the module is loaded, but the *Boot command is not executed.

To bring the machine down directly from RISC iX and into RISC OS, irrespective of the setting of the *Configure noRISCOS flag, type:

```
reboot -RISCOS
```

This command forces the system to ignore the setting of the *Configure noRISCOS setting and bring the system down into RISC OS.

Example

```
*Configure noRISCOS Off
```

*Configure Partition

Sets the partition number of the filesystem mounted when the RISCiXFS module is loaded.

Syntax

```
*Configure Partition <partition number>
```

Parameters

partition number	0 to 7; the partition number on the device

Use

*Configure Partition sets the partition number of the filesystem mounted when the RISCiXFS module is loaded.

Example

`*Configure Partition 0` (the default value)

Related commands

*Configure Device, *Configure Unit.

*Configure Unit

Sets the unit number of the filesystem mounted when the RISCiXFS module is loaded.

Syntax

```
*Configure Unit <unit number>
```

Parameters

unit number	0 to 63; the unit device number

Use

*Configure Unit sets the unit number of the filesystem mounted when the RISCiXFS module is loaded.

Example

`*Configure Unit 0` (the default value)

Related commands

*Configure Device, *Configure Partition.

*Execv fsck

Runs a filesystem consistency check (*fsck*) prior to booting RISC iX.

Syntax

```
*EXECV fsck <options>
```

Parameters

options — for full documentation on the options that can be used with the *fsck* program, refer to the manual page for *fsck*(8). Note that the filename of the object must be given and must be the name of a block special file that exists on the currently mounted filesystem.

Use

*Execv fsck carries out a thorough check of the state of the filesystem and should be used as a diagnostic tool for checking the state of the filesystem if RISC iX fails to boot.

*Execv executes RISC iX objects under the RISC iX system call emulator contained in the RISCiXFS module.

All parameters appended to the command line are passed to the executed object. Objects are searched for along the *RISCiX$Path* variable which uses the default search path '*/,/sbin,/usr/sbin*' and can be modified at any time using the RISC OS *Set command (see the *RISC OS User Guide* for details on how to assign system variables). The ',' character is used to separate individual path entries (conforming to the RISC OS convention).

Example

```
*EXECV fsck /dev/st0a
```

carries out a check of the filesystem on the hard disc device */dev/st0a*.

Related commands

*Execv mkfs.

*Execv mkfs

Makes a new filesystem.

Syntax

```
*EXECV mkfs <options>
```

Parameters

options — for full documentation on the options that can be used with the *mkfs* program, refer to the manual page for *mkfs*(8).

Use

*Execv mkfs makes a new filesystem on the named device by initialising the file structures and the root directory.

*Execv executes RISC iX objects under the RISC iX system call emulator contained in the RISCiXFS module.

All parameters appended to the command line are passed to the executed object. Objects are searched for along the *RISCiX$Path* variable which uses the search path '/,/*sbin*,/*usr*/*sbin*' and can be modified at any time using the RISC OS *Set command (see the *RISC OS User Guide* for details on how to assign system variables). The ',' character is used to separate individual path entries (conforming to the RISC OS convention).

Example

```
*EXECV mkfs /dev/fdf1024 1600 10 2 4096 1024
```

makes a new filesystem on a floppy disc that has been formatted using the command *ffd 1024*.

Related commands

*Execv fsck.

*FMount

Allows other RISC iX filesystems to be attached to the current root directory.

Syntax

```
*FMount <special device> <pathname> [R]
```

Parameters

special device refers to the device partition that *FMount will try to mount as a RISC iX filesystem. This parameter conforms to the naming convention `ddM(U,P)` where:

dd is a two character device driver identifier:

st – st506 hard disc driver

fd – ADFS physical format floppy disc driver

sd – SCSI hard disc device

M – 0 to 7; the major hardware controller number

U – 0 to 63; the unit device number (drive number)

P – 0 to 7; the partition number on the device

pathname a valid pathname that points to a currently unused directory on the filesystem.

R makes the mounted filesystem read-only

Use

*FMount attaches a RISC iX filesystem to a suitable mount point on a currently mounted filesystem. This filesystem then becomes available as a sub-tree from the original mount point.

Example

```
*FMount fd0(0,0) /mnt R
```

mount a read-only filesystem from floppy disc device *fd0(0,0)* in the directory */mnt*.

Related commands

*UMount

*UMount

Releases an attached filesystem from the given mount point.

Syntax

```
*UMount <pathname>
```

Parameters

pathname a valid pathname that points to a currently mounted filesystem.

Use

*UMount releases the filesystem attached to the given mount point by the command *FMount. The mount point reverts to being a normal directory on the parent filesystem.

Example

`*Umount /mnt` releases a previously mounted filesystem from */mnt*.

Related commands

*FMount

System calls supported

The following list details the RISC iX system calls that are supported from RISC OS:

(* denotes limited support)

```
exit
read
write
open
close
creat
mknod
obreak
lseek
getuid      * (always returns 0)
sync
stat
lstat
fstat
gettime     * (only local time)
readv
writev
mkdir
```

```
getdirentries
umount
mount
```

The above system calls are sufficient to support standard UNIX commands such as *ls*, *date* and *cat*. For example, to list the root filesystem from RISC OS, type:

```
execv ls /
```

RISCiXFS module error messages

'This section contains a list of the error messages that you may receive when you are using the RISCiXFS module:

`RISCiXFS module not present or too old'`

The RISCiXFS module must be installed before the application is executed (this is to facilitate the *Configure noRISCOS option). So the above error message is displayed if the module cannot be found or the module loaded cannot support this version of the application.

`'RISCiXFS  Failed to boot last time'`

This message is displayed at initial start-up if, for any reason, RISC iX did not properly start up the last time the RISCiXFS module was loaded. This is not a serious message and should be ignored unless it is continually displayed.

If the system does not boot correctly following this message, restart the system again; the start-up procedure will now proceed normally.

`'RISCiXFS: failed to mount ROOT filesystem'`

This message generally means that one or more of the CMOS RAM parameter settings are wrong. Configure each setting back to its default value using the appropriate *Configure commands.

RISCiXFS module applications error messages

'This section contains a list of the error messages that you may receive when you are using the RISCiXFS module applications:

`'Not enough memory for object'`

This message may occur if you try to load a file as a RISC iX kernel using the *Boot command. The default memory allocation for Tasks in RISC OS is 640K which is usually not enough memory to load any sizable kernel.

To increase the memory allocation for this Task, position the mouse pointer over the Task Manager icon on the extreme right of the icon bar and click the middle mouse button. Select *Task display* from the menu displayed. This produces a window containing details of the use of the computer's memory. Using the left mouse button, alter the size of the *Next* bar in the window to a more usable size (for example, 1024K) and try the *Boot command again.

`'SWI not known'`

This can occur during the use of the *Execv command and is caused by the RISC iX object containing system calls that the RISC iX system call emulator does not understand.

`'The EXECV command requires floating point support'`

This message is displayed when an *Execv command has been issued without the Floating Point Emulator (FPEmulator) loaded.

To load the FPEmulator module, type:

```
*adfs
*RMLoad $.App1.!System.Modules.FPEmulator
```

then try the *Execv command again.

If you are going to use *Execv commands regularly, then it would be a good idea to include the above line in your !Boot file to automatically load the FPEmulator module.

Setting up UUCP

Introduction

Setting up UUCP can be quite a major task, as there are many files and directories to set up correctly, and maintain once set up. However, once it is running, it can be quite a reliable means of file transfer between disparate systems where no other method (eg Ethernet, floppy) exists.

Not only are files transferred, but UUCP is the basis of the electronic mail system provided under UNIX; indeed in most cases the mail system is the only part of UUCP that is used.

The syntax of UUCP is slightly peculiar; remote machines are referred to by means of a 'node name' followed by an exclamation mark (!). This is then followed by the user name on the remote machine (in the case of *mail*), or the path name (in the case of other programs). For example:

```
machine!user
```

and

```
machine!/u/a/b/c.
```

A transfer can be 'multi-hop', ie via one or more intermediate machines, by specifying several node names, thus:

```
machine1!machine2!machine3!...!user.
```

One of the limitations of the UUCP suite of programs is that the route has to be completely known, however some sites overcome this by having large tables of routes and automatically inserting routes to machines known to them.

There are two user-level programs associated with UUCP. These are:

- A program actually called *uucp*, whose function is to copy between systems with command arguments analogous to *cp*.

- `uux`, which is a program which sets up a job to run a command on a remote system, passing specified files as data and recovering the results. In fact this is the command invoked by *mail*, passing the message text as data, and most systems are set up only to permit the passing of mail using *uux*, and perhaps some harmless commands to list files etc.

Likewise commonly-imposed restrictions prevent `uucp` from being used to copy a very limited subset of the files on each machine, and in particular 'multi-hop' transfers are very hard, and sometimes impossible to implement satisfactorily. In most cases files are sent 'multi-hop' as part of a mail message, possibly packed using the utility *tarmail*.

There are usually two background programs started by `uucp`. These are:

- *uucico*, which connects over the serial line to the remote machine and talks to another copy of itself on that machine to actually transfer files. The 'called' machine has a special login which has this program in place of a shell.
- *uuxqt*, which runs after *uucico* has been completed, and processes the files which have arrived.

UUCP may be connected over the telephone line using modems, or directly using serial lines. Connection may be two-way, or one system may always call the other (this is usually much easier). Normally when a job is submitted using *uucp* or *uux*, *uucico* is started in the background to place a call to the remote machine and start file transfer.

Once file transfer has started, everything which is pending between the machines will be sent each way. If modems and telephone lines are involved, it is highly undesirable if a request is queued and a copy of *uucico* started up during the peak telephone charging rates. Accordingly what usually happens is that one system is designated a 'slave' system and never places calls, and the other system is a 'master' system and calls the slave system regardless of whether it has work, at fixed times set up using *cron*. Such calling is known as 'polling'.

Checking the serial line

It is assumed that you have successfully connected a modem to the serial port of your workstation as detailed in the chapter entitled *Setting up peripheral devices* on page 93. When you have connected the modem up you can use the program *tip*(1C) to access the modem and check or set its internal settings, prior to commissioning a UUCP link. Use one of the *unixnnnn* entries in */etc/remote* to test the line; if there isn't one for the speed you want, then make one. You can go up to 19200 baud.

The serial line is accessed via */dev/serial*. *tip* and *uucico* are both setuid *uucp*. After you have successfully connected your modem, you should type the following line to avoid any trouble with access permissions:

```
chown uucp /dev/serial
```

Setting up a UUCP name for your machine

Each UUCP machine needs a UUCP name. Normally this is the hostname of the machine, often truncated to six or seven characters depending on the version of UUCP that you have.

If your hostname is still `unix` then you should change it to something more original. If you are linking into the world-wide UUCP network then your UUCP name should ideally be unique in the first six characters, though you may also be able to register a separate domain-based electronic mail name.

Setting up a UUCP login name

Your system should already have an entry in the password file (*/etc/passwd*) for a user called *uucp*:

```
uucp::66:1:UNIX-to-UNIX Copy:/var/spool/uucppublic:/usr/lib/uucp/uucico
```

It is helpful to have a UUCP System Administrator user, for example *uucpsu*, with the same uid and gid but with a normal shell and a secure password. This avoids having to set up *uucp* as a superuser and encountering problems with files being unwriteable by the UUCP system.

Setting up the UUCP files and directories

In order to set up *uucp*, various files in the directory */usr/lib/uucp* need to be carefully edited. The rest of this chapter takes you through the stages involved in setting up UUCP on your system.

/var/spool/uucp

Login as *uucpsu* and change directory to */var/spool/uucp*. Create the directories *D.<nnn>* and *D.<nnn>X*, where *nnn* is the hostname of your system, truncated if necessary to seven characters.

For example, if the hostname of your machine is *stardust*, you would type:

```
mkdir D.stardus
mkdir D.stardusX
```

You will also need to create the directory *XTMP* to enable remote `uux` commands to send and deliver mail:

```
mkdir XTMP
```

List out the contents of the directory */var/spool/uucp*, using the command *ls -lg*. The files for the machine *stardust* along with the correct ownership and access permissions, would be as follows:

```
drwxrwxr-x 2 uucp daemon    512 Dec 11 17:46 AUDIT
drwxr-xr-x 2 uucp daemon    512 Apr  6 12:59 C.
drwxr-xr-x 2 uucp daemon    512 Dec 11 17:46 D.
drwxr-xr-x 2 uucp daemon    512 Apr  5 14:31 D.stardus
drwxr-xr-x 2 uucp daemon    512 Apr  5 14:32 D.stardusX
drwxr-xr-x 2 uucp daemon    512 Apr  6 13:09 STST
drwxr-xr-x 2 uucp daemon    512 Apr  6 13:09 TM.
drwxr-xr-x 2 uucp daemon    512 Dec 11 17:46 X.
drwxr-xr-x 2 uucp daemon    512 Dec 11 17:46 XTMP
drwxrwxr-x 2 uucp daemon    512 Apr  6 11:28 uucplog.archives
```

The following log files; *LOGFILE*, *ERRLOG* or *SYSLOG* may also be contained in this directory.

/usr/lib/uucp

You will need to create the following configuration files in */usr/lib/uucp*:

- *L-devices*
- *L.cmds*
- *L.sys*
- *USERFILE*

Here are some examples, see the appropriate manual pages for more details:

L-devices

```
# Caller Device Call_Unit Class Dialer [Expect Send]...
#
ACU serial unused 2400 hayes2400pulse
DIR serial unused 9600
PAD serial unused 9600
```

L.cmds

```
# An optional PATH=/dir[:/dir]... may be given, followed by a list of
# acceptable commands, one per line.
PATH=/usr/ucb:/bin:/usr/bin
rmail
```

L.sys

```
# System Times Caller Class Device/Phone_Number [Expect Send]...
#
acorn Any DIR 9600 serial "" "" ogin: uucp sword: choker
```

```
testacu Any DIR 9600 123456789 "" send1 expect2 send2 expect3
testdir Any DIR 9600 serial "" send1 expect2 send2 expect3
testpad Any PAD 9600 serial "" send1 expect2 send2 expect3
```

USERFILE

```
# [loginname],[system] /pathname [/pathname]...
#
, /var/spool/uucppublic
```

Much fuss is made over the ACU caller type and the diallers it supports. Not all the diallers listed in the *L-devices* man page are supported and in any case there are so many different modems on the market, each needing its own special settings, that this compiled-in approach is doomed unless you have the sources.

Instead you should use the DIR caller type and put all the chit-chat with the modem in *L.sys* as part of the expect-send sequence. For example:

```
expect nothing, send ATZ, expect OK, send ATDP628847, etc.
```

The DIR caller uses the 'g' protocol, ideal for use with normal non error-correcting modems. If you have a 'Trailblazer' modem at each end of the line you should also use 'g' protocol as they have special handling for it. If you have some other error correcting modems either end of the line, you might get a faster throughput by using the 'f' protocol which you select by saying the caller type is a PAD. Unlike the 'g' protocol which sends short check-summed packets with acknowledgement handshaking, the 'f' protocol sends the whole file with a single checksum at the end. It is used under very good line conditions when the checksum is only rarely expected to be wrong. The 'f' protocol uses only seven bits so any eight bit data gets encoded/decoded on the fly.

Please note that although the caller type is PAD it won't work properly with JNT PADs as used in the UK because <Ctrl-P>'s are used in the initial handshaking.

Testing the UUCP link

Once you have edited the above files, you can test the UUCP link by running *uucico* with debugging on to see what is happening. This should show up any errors in the expect/send sequence. Put a sample file in */var/spool/uucppublic*, give it public read, and queue it for transfer. For example:

```
% uucp -r /var/spool/uucppublic/test remhost\!/var/spool/uucppublic/infile
% /usr/lib/uucp/uucico -r1 -x99 -sremhost
```

Incoming UUCP

The setup for incoming UUCP is similar to that for outgoing UUCP, except that you don't need an expect-send sequence. You should however put the UUCP name in */usr/lib/uucp/L.sys* with a *'Never times'* entry. The site should also appear in

/usr/lib/USERFILE unless covered by any default. For security you should put a '*' in the password field for the generic *uucp* user and give each site that calls in its own login name and password. A good scheme is to append a 'U' to the *uucp* name to make a user id. In each case the uid, gid, home directory and shell remain the same as for the generic `uucp` user. Don't worry if UUCP files seem to change ownership to one of these users, the uid is still 66.

You will need a line like this in */etc/ttys* to enable logins on the serial line:

```
serial  "/etc/getty std.1200"  unknown        on
```

Tidying up UUCP directories

UUCP writes log files in */var/spool/uucp* and may also accumulate abandoned files in its spool directories, so it needs a regular tidy. A shell script for performing this task, called *clean.daily*, is contained in */usr/lib/uucp*.

clean.daily will remove any abandoned files contained in the spool directories and also perform a seven-day rotation of the UUCP log files, keeping them for a seven-day period before deleting them. By saving these files each day an accurate record is available of all the UUCP messages that have passed through the system in the last seven days. This is very useful for debugging purposes when problems occur with UUCP:

The only editing *clean.daily* may need is if your hostname is longer than seven characters. In this case you will need to replace references to *uuname -l* with the hostname (truncated to seven characters) because *uuname* gets it wrong. For example, if the hostname of your machine is *ratatouille* you would need to change the following lines in *clean.daily*:

```
./uuclean -d/var/spool/uucp/D.`uuname -l` -pD. -n240
./uuclean -d/var/spool/uucp/D.`uuname -l`X -pD. -n240
```

to read:

```
./uuclean -d/var/spool/uucp/D.`ratatou` -pD. -n240
./uuclean -d/var/spool/uucp/D.`ratatou`X -pD. -n240
```

Once you have edited *clean.daily* you should edit crontab file for *uucp* in the directory */var/spool/cron/crontabs* and uncomment the line in the file to run *clean.daily* every day at midnight. For example, change:

```
# Clean up the uucp directories at midnight -
# Un-comment the next line if you have set up uucp
#0 0 * * * uucp /usr/bin/csh /usr/lib/uucp/clean.daily 2>&1 | /usr/ucb/mail -s "uucp
daily clean log" root
#
```

to:

```
# Clean up the uucp directories at midnight -
# Un-comment the next line if you have set up uucp
0 0 * * * uucp /usr/bin/csh /usr/lib/uucp/clean.daily 2>&1 | /usr/ucb/mail -s "uucp
daily clean log" root
#
```

In order for *uucp* to run *cron* jobs you will also need to create the file */var/spool/cron/cron.allow* containing the line:

```
uucp
```

cron.allow is a simple text file which contains a list of users who are allowed to run *cron*. Therefore, entering the above line enables the user *uucp* to run *cron* and so run *clean.daily*.

For more information about *cron*, refer to *cron*(8). The other lines in the *crontab* file are discussed in the chapter entitled *Maintaining the filesystem* on page 51.

Locked serial lines

Whilst in use the serial line is locked by UUCP. This prevents another user attempting to access the serial line at the same time.

If the *uucico* process is killed off for some reason, ie the machine crashes, then you will have to manually remove the lock file. Type the following:

```
rm /var/spool/uucp/LCK..serial
```

Further information

For more information about setting up UUCP, refer to the following manual pages *L-devices*(5), *L-dialcodes*(5), *L.aliases*(5), *L.cmds*(5), *L.sys*(5) and *USERFILE*(5).

Setting up an Ethernet/Econet network

Introduction

This chapter describes how to set up two or more RISC iX workstations together via Econet to a gateway RISC iX machine which is fitted with both Econet and Ethernet.

This chapter assumes that you have already set up an Ethernet network and wish to extend the network further by adding an Econet network. For details about setting up an Ethernet local area network, refer to the chapter entitled *Setting up a local Ethernet network* on page 125 of this Guide.

Econet is Acorn's proprietary networking system used to network Acorn machines. Although the transfer rate is relatively slow (50-300 Kbps) it is a cheap way of connecting together Acorn machines.

Econet supports the standard Internet Protocols (as described in the chapter entitled *Setting up a local Ethernet network* on page 125) and so enables RISC iX machines connected in this way to use the Berkeley networking commands, described in the chapter entitled *Networking and NFS* in the *RISC iX User Guide*.

There are five main stages in setting up an Econet/Ethernet network:

- Installing an Econet network
- Setting the station number for each machine
- Configuring the gateway machine
- Configuring the Econet machines on the Econet network
- Booting the machines.

Installing an Econet network

Follow the instructions in the manuals supplied with your Econet products to connect up the Econet side of the network. The following list details what you will need in order to set up an Econet network for your RISC iX workstations:

- four-core screened cable with which to lay the network
- a five-pin DIN socket for each computer
- a clock box to feed a clock signal onto the network

- two terminator boxes to correctly terminate either end of the network.

For each RISC iX workstation you wish to connect to the network you will need:

- an Econet upgrade kit, which includes:
  - the Econet module (Master type ADF-10)
  - the *Econet Installation Guide* with instructions on installing the interface and information on the Econet network components, hardware configuration and testing.
- a five-pin DIN to five-pin DIN lead, to connect to the socket.

You might want to add to the network:

- a file server
- a print server.

If you want to connect together two Econet networks you will also need:

- an Econet bridge.

There are two manuals that you will find useful:

- the *Econet Advanced User Guide*
- the *Econet Print Server Manager's Guide*.

All the above are available from your Acorn supplier, who can advise you which configuration of Econet will best suit your needs. You may find the Econet Starter Kit a particularly good way to get started.

Remember that each RISC iX workstation requires an Econet module to be installed. The machines can be connected to either a terminator or a socket box using a cable fitted with 5-pin DIN plugs.

Setting the station number in CMOS RAM

Each RISC iX workstation on the Econet network requires a unique station number to be placed in its CMOS RAM, to identify it to the rest of the machines on the network.

A station number is analogous to an Ethernet address for machines on an Ethernet network. Station numbers are arbitrary numbers between 1 and 254 (excluding 235 which is used by Acorn for the print server).

The first byte of the CMOS RAM is used to store the station number of a machine, so it is simply a case of writing the value to */dev/cmos* using the following command:

```
echo 'ASCII character' | dd of=/dev/cmos bs=1 count=1
```

where *ASCII character* is a character that refers to the station number you wish to set the machine to. For example, to set the station number to 35, you would type:

```
echo '#' | dd of=/dev/cmos bs=1 count=1
```

A list of the ASCII character set used by RISC iX workstations is contained in the *RISC OS User Guide*, supplied with your machine.

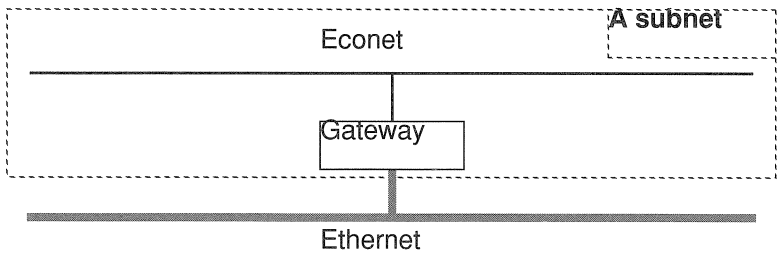
To check the station number of a machine, type:

```
dd if=/dev/cmos bs=1 count=1 | od -d
1+0 records in
1+0 records out
0000000  00035
0000001
```

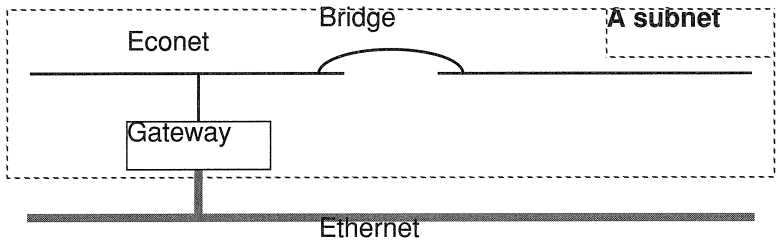
For details about allocating Econet station numbers, refer to the *Econet Installation Guide*. This Guide also contains a set of system record sheets that you should use to record all station numbers that have been allocated.

Configuring the gateway machine

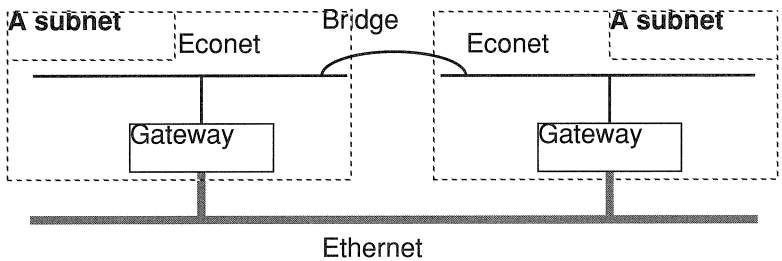
You will need to fit an Ethernet card to at least one of your RISC iX workstations on the Econet network to act as a gateway between the two networks. The Econet network is also a TCP/IP subnet. For example:



You can use Econet bridges to link together two or more Econet networks. These can share a single gateway, and so form part of the same TCP/IP subnet:



If you put a gateway on the second Econet above, it then becomes a separate TCP/IP subnet because it's got its own gateway:



Ethernet cards are available from your Acorn supplier, and come complete with fitting instructions.

Bottlenecks

One potential problem in designing networks like this is *bottlenecks*. When a network packet goes through an Econet bridge or a gateway there are inevitable delays as it's merged with the existing traffic on the other side. Consequently these can cause bottlenecks in the system. In general:

- The fewer Econet bridges a network packet has to cross before reaching the Ethernet, the quicker it will reach a machine on the Ethernet. (This is, of course, also true of packets going in the reverse direction.)
- If too many network packets are using the same gateway to cross from the Ethernet to Econet, you can get a bottleneck there too. This is because the Ethernet is a faster network, and can potentially deliver packets to the gateway more quickly than they can be placed on the Econet. The more gateways you

provide, the better performance will be; although there comes a point where you'll have enough gateways that they're almost always working at peak efficiency anyway.

It's hard to give more precise guidelines, because of the wide range in how network-intensive computer usage is at different sites. If you're already running a network, you'll have a good feel for how things work out at your site.

Software configuration

Once you have installed the card, you need to configure the gateway machine to make it known to both networks. This involves editing the following files on the machine:

- */etc/rc.config*
- */etc/rc.net*
- */etc/hosts*

Editing /etc/rc.config

RISC iX workstations are initially set up to be used as standalone machines, not attached to a network. This configuration can be altered by changing suitable lines in the file */etc/rc.config*. The lines to be changed and their initial settings are as follows:

```
...
STANDALONE=TRUE
FULLNETWORK=FALSE
NIS=FALSE
...
```

The lines to change depend on the type of network environment that you are connecting your system to.

If you are connecting your machine to an Ethernet network, edit */etc/rc.config* to change the settings to:

```
...
STANDALONE=FALSE
FULLNETWORK=TRUE
NIS=FALSE
...
```

If your Ethernet network is also running the NIS distributed network database, change the settings to:

```
...
STANDALONE=FALSE
FULLNETWORK=TRUE
NIS=TRUE
...
```

Editing /etc/rc.net

Two changes need to be made to this file:

- Setting the hostname of the machine (this may already have been done)
- Initialising the Econet interface.

For more information on suitable practices to adopt for assigning host and interface names refer to the section entitled *Assigning host and interface names* on page 135.

Setting the hostname of the machine

To set the hostname of the gateway machine, edit the following line in the file:

```
HOSTNAME=
```

The example below sets the hostname of the machine to *gate*:

```
HOSTNAME=gate
```

Initialising the Econet interface

To initialise the Econet interface, you need to invoke the *econetup*(8) program in this file. To do this, insert the following lines at the end of the file:

```
# Bring Econet up
/etc/econetup 40 20 >/dev/console 2>&1
if ifconfig ec0 inet arp -trailers ${NETMASK} ${HOSTNAME}_e ${BROADCAST}
>/dev/console 2>&1
then
 echo "Econet configured as address "`hostname`"_e" >/dev/console 2>&1
else
 echo "***** problem with Econet ifconfig in /etc/rc.net *****" >/dev/console 2>&1
 exit 2
fi
```

econetup accepts two parameters, which refer to the maximum and minimum size allocation of the Econet reception buffers. The first parameter defines the number of small size buffers that can be allocated and the second parameter defines the number of large size buffers that can be allocated. If no parameters are specified, or zero is given for either parameter, default values of 20 and 5 respectively are used.

Editing /etc/hosts

This file contains information regarding known hosts on the network. The Econet address adopts the convention of using 88 as the first two digits of the address, while the convention for Internet addresses is 89.

Note that the Internet address of 89 should only be used if you are not planning to connect to a public network. If you are, then you must register for an official Internet network number and use this number to assign Internet addresses for all the machines on the network, including the machines on the Econet network. For more information see the section entitled *If you plan to connect to other sites...* on page 137.

The file needs to be edited to contain the Internet and Econet addresses of all the machines on both networks. In an example three machine network, the file would read:

```
#
89.0.0.7     gate
#
# Econet network
88.0.0.34    gate_e
88.0.0.35    econet1
88.0.0.36    econet2
```

Notice that the gateway machine has an Internet address *and* an Econet address. This is because it has to be identifiable to machines on both networks. To avoid confusion, the Econet name of the machine is specified as *gate_e* and the Internet name is specified as *gate*.

If NIS is running then the new machines should be added to the */etc/hosts* file on the NIS master and then the NIS *hosts.byaddr* map updated. For more information on how to update the NIS database, refer to the chapter entitled *The Network Information Service* on page 219.

Configuring the Econet RISC iX machines

Once you have configured the gateway machine, you need to set up all the RISC iX machines that are on the Econet network. This involves editing the following files on each machine:

- */etc/rc.config*
- */etc/rc.net*

Editing /etc/rc.config

RISC iX workstations are initially set up to be used as stand-alone machines, not attached to a network. This configuration can be altered by changing suitable lines in the file */etc/rc.config*. The lines to be changed and their initial settings are as follows:

```
...
STANDALONE=TRUE
FULLNETWORK=FALSE
NIS=FALSE
...
```

This should be changed to:

```
...
STANDALONE=FALSE
FULLNETWORK=TRUE
NIS=FALSE
...
```

Editing /etc/rc.net

Two changes need to be made to this file:

- Setting the hostname of the machine (this may already have been done)
- Initialising the Econet interface.

Setting the hostname of the machine

To set the hostname of the machine, edit the following line in the file:

```
HOSTNAME=
```

The example below sets the hostname for the machine to *econet1*:

```
HOSTNAME=econet1
```

Initialising the Econet interface

To initialise the Econet interface, you need to invoke the *econetup*(8) program in this file. To do this, insert the following lines at the end of the file and comment out all references to the Ethernet interface:

```
hostname ${HOSTNAME} >/dev/console2>&1
# if ifconfig et0 inet ${NETMASK} ${HOSTNAME} ${BROADCAST} >/dev/console 2>&1
# then
#       echo "ethernet configured as address "`hostname` >/dev/console 2>&1
# else
#       echo "***** problem with ethernet ifconfig in /etc/rc.net *****"
>/dev/console
#       exit 2
# fi
#
# Bring Econet up
/etc/econetup 40 20 >/dev/console 2>&1
if ifconfig ec0 inet arp -trailers ${NETMASK} ${HOSTNAME}_e ${BROADCAST}
```

```
>/dev/console 2>&1
then
        echo "Econet configured as address "`hostname`"_e" >/dev/console 2>&1
else
        echo "***** problem with econet ifconfig in /etc/rc.net *****" >/dev/console
2>&1
        exit 2
fi
```

econetup accepts two parameters, which refer to the maximum and minimum size allocation of the Econet reception buffers. The first parameter defines the smallest size buffer that can be allocated and the second parameter defines the largest buffer that can be allocated. If no parameters are specified, or zero is given for either parameter, default values of 20 and 5 respectively are used.

Booting the machines

Once you have configured the machines, you need to reboot all the machines.

Firstly, reboot the gateway machine. You should see messages similar to those given below appear during its boot sequence:

```
ethernet configured as address gate
Econet: init stn 59.34, 40 srxb, 20lrxb
Econet configured as address gate_e
```

The first message confirms that the gateway machine has been successfully configured on to the Ethernet network.

The second message is generated by *econetup* when initialising the machine. In the above example, the machine is configured as station 34 on network 59, with 40 small and 20 large reception buffers.

The third message is produced by the successful configuration of the Econet interface by *ifconfig*. If *ifconfig* fails, an appropriate error message will be displayed.

Now reboot the RISC iX machines on the Econet network. You should see the same two Econet messages illustrated above, appear during their boot sequences. Although naturally with different station numbers:

```
Econet: init stn 59.35, 40 srxb, 20lrxb
Econet configured as address econet1
```

The machines are now ready to be used.

Troubleshooting

For further information about any errors that may occur either during the use of Econet or during the initialisation of the Econet device driver, refer to *econetup*(8), *ifconfig*(8C) and *eco*(4).

Using RISC OS machines on the network

The *TCP/IP Protocol Suite* is a RISC OS application which enables RISC OS machines to be connected to either an Econet or Ethernet network and use Internet protocols to communicate with other machines on the network.

The applications provided include:

- *NFS Filer* – an application that gives RISC OS machines NFS-like access to the files stored by other machines on the local network.
- *NFS MailMan* – an application which provides an enhanced version of the RISC OS Mailman program, with extensions to allow RISC OS machines to send and receive mail over NFS.
- *VT220* – an application which provides a full emulation of a DEC VT220 terminal, running in a window within the RISC OS desktop. It will also emulate the earlier VT52 and VT100 terminals.
- *Telnet* – an application that converts between the Telnet protocol and Acorn TIP (the standard protocol used by RISC OS terminal emulators). Hence this enables RISC OS machines to connect to any terminal emulator that uses the Acorn TIP (such as VT220) to any remote computer that supports Telnet.

For more information about the *TCP/IP Protocol Suite*, contact your supplier.

Creating and Removing RISC iX Application Packages

Introduction

To assist you in your System Administration of the machine, the RISC iX operating system can be thought of as existing as a series of discrete software packages that can be removed from and added to the system. Each package is self-contained so that you only need to load those packages you require, and you can remove packages that are not wanted. This way you can conserve disc space for your own files or applications.

A RISC iX application package is a group of related files and programs that together perform a set of similar or related functions. For example, all of the files and programs that relate to the NIS networking protocol have been grouped into a single package.

A utility called *packageadmin*(8) controls the addition and removal of application packages. This chapter describes how to use *packageadmin* to add and remove these application packages and also how to create your own packages.

Initial set-up

The RISC iX operating system is installed in two stages; first the core operating system and core application packages are installed (they are usually pre-loaded onto the hard disc by your supplier), and then any additional application packages can be installed to suit your specific system requirements.

Some packages are pre-installed onto the hard disc, other packages are stored on floppy disc and need to be installed on the hard disc before they can be used.

The core application packages include:

- Berkeley networking – *bnet*
- Sun Open Network Computing Environment – *onc*
- On–line manual pages – *man*
- Interactive UNIX lessons – *learn*
- X11 core programs – *X11core*

The additional application packages include:

- Network Information Service – *nis*

- Program Development – *progdev*
- Fortran77 – *F77*
- Pascal – *pascal*
- Games – *games*
- X11 programming – *X11prog*
- X11 demonstrations – *X11demos*

Note: There may be other packages provided with the system, in addition to the packages listed above. A complete list of all the packages that are provided together with the amount of disc space they occupy, is detailed in the *RISC iX Release Notice* provided with your system.

An overview of the packages

This section gives a general overview of the contents and uses of the packages listed in the previous section.

Core application packages

A summary of the RISC iX core application packages is given below:

- **Berkeley networking** (*bnet*) – provides the commands that allow interactive communication over the network between your workstation and other UNIX workstations.
- The **Sun Open Network Computing Environment** (*onc*) – contains all the files needed to support the Sun Network File System (NFS). To use this package you must also have the Berkeley networking application package (*bnet*) installed.
- The **On-line manual pages** (*man*) – provides an on-line version of the UNIX reference manual pages. The pages contain detailed descriptions about how to use individual commands.
- **Interactive UNIX lessons** (*learn*) – is a set of interactive lessons to help you learn about the UNIX operating system. There are lessons about files, the editors *vi* and *ed*, the *-ms* macro language, and an introduction to C programming.
- **X11 Core Programs** (*X11core*) – provides support for the industry standard X11 Window System and its Window Managers. This also includes the Motif user interface (*mwm*) and the X.desktop user interface. This package must be installed before either the X11 programming (*X11prog*) and X11 demonstration packages (*X11demos*) can be used.

Additional application packages

A summary of the RISC iX additional application packages is given below:

- **Network Information Service** (*nis*) – provides the further files and commands needed to implement the NIS networking protocol. To use this package you must have the Berkeley networking (*bnet*) and Sun Open Network Computing Environment (*onc*) packages installed.
- **Program Development** (*progdev*) – offers the complete environment needed for programming using either the C, Fortran or Pascal programming language.
- **Fortran 77** (*f77*) – allows programs to be written in the programming language FORTRAN 77. This package requires the Program Development package (*progdev*) to be installed.
- **Pascal** – allows programs to be written in the programming language Pascal. This package requires the Program Development package (*progdev*) to be installed.
- **Games** (*games*) – provides over twenty games such as Backgammon, Boggle and Cribbage, for your relaxation and diversion.
- **X11 Programming** (*X11prog*) – provides facilities to enable a programmer to develop X11 applications, including Motif based applications. To develop programs with this package, the X11 core programs package (*X11core*) must also be installed.
- **X11 Demonstrations** (*X11demos*) – contains some simple demonstration programs that use the features of the X Window system. To use this package, the X11 core programs package (*X11core*) must also be installed.

The packageadmin utility

The handling of packages is controlled by a simple menu driven program called *packageadmin* that can transfer or remove the packages onto your hard disc. Once the software is on your hard disc you can access it by typing in the relevant commands.

Starting packageadmin

To start the utility, at your normal *root* command prompt, type:

```
packageadmin
```

You will see the following display:

```
      PACKAGE ADMINISTRATION - Version 1.11

Main menu
=========
```

```
*help
*exit
*list packages installed
*install a package
*remove a package

enter selection (e/h/i/l/r) [h]>>
```

Listing the packages installed

If you are unsure as to which packages are already installed you can find out by selecting the *list packages installed* option. Type the letter *l* (for *list*) after the enter selection prompt:

```
enter selection (e/h/i/l/r) [h]>>l

***List

The following packages are installed:

X11core -- X11 core programs Version 1.1
      / 0 kbytes   /usr 4133 kbytes

bnet --Berkeley Networking Version 1.1
      / 1342 kbytes /usr 600 kbytes

learn -- Interactive Unix lesson Version 1.1
      / 0 kbytes   /usr 821 kbytes

man -- On-line manual information Version 1.1
      / 48 kbytes  /usr 2175 kbytes

onc -- Sun Open Network Computing environment Version 3.2
      / 113 kbytes /usr 584 kbytes

/usr is part of the root partition

Free space: / 4789 kbytes
Hit any key to continue>>
```

The list shows the packages that have already been installed on your system. You may find that most of the core and application packages have been installed on your system. This is done because most users will need to make use of these packages. However, if the needs of your users are different, you can use *packageadmin* to remove these packages from your hard disc. This is described in the next section.

Removing a package

One of the main reasons for using archived packages is that by removing the parts of the system that are not used, you can save substantial disc space. The packages that do not get used will vary from system to system, but here are some general guidelines about groups of packages that could possibly be deleted:

- If you are not using the X11 Window system, you do not need the *X11core*, *X11prog* or *X11demos* packages installed.
- If your workstation is used standalone (not networked) you do not need the *bnet*, *onc* or *nis* packages installed.
- If you are not developing programs you do not need the *progdev*, *f77* or *pascal* packages installed.
- You can also save substantial disc space by not having the packages *man*, *learn* or *games* installed. For example, in a network environment it is normal to have packages such as these installed on one machine which is then mounted by all the other machines on the network, thereby enabling you to remove these packages from the other machines.

To remove a package from your hard disc and archive it onto floppy disc, you use the *remove a package* function from the *packageadmin* menu. Just type r (for remove) at the Main menu prompt:

```
enter selection (e/h/i/l/r) [h]>>r
```

The utility then displays a list of the packages you can remove. For example, if you wish to remove the *man* package from your hard disc type *man* at the Enter choice prompt:

```
*** Removal

Choose a package from:
X11core  bnet  learn  man  onc  ...

Enter choice >>man
```

You are then asked if you wish to archive this package onto floppy disc:

```
Do you wish to archive package man? (y/n) [y] >> y

Place the first disc for the archive into the floppy drive

Press <return> to continue>>
```

To begin the archive, type y (for yes), insert a floppy disc into the disc drive and press ↵.

If your floppy disc is unformatted it will be automatically formatted. If the floppy disc contains data, you will be prompted to confirm that you wish to overwrite this data and reformat the disc.

When the disc is formatted, the archive begins. If the archive does not fit on a single disc you will be prompted to insert another disc. For example:

```
Please load disc, to be number 2 of 'Package: man ROOTNEEDED=48 USRNEEDED=2175'
Type RETURN when ready:
```

Remove the first disc and label it, then insert the next archive disc. A check is made on each new disc inserted to see if it is formatted and is not already an archive disc, before the archive proceeds.

When the package has been successfully archived, you are prompted to confirm that you want to remove the package from your system. If you merely wish to create a backup copy of the package, type *n* and the package will not be deleted from your hard disc.

Otherwise, type *y* to confirm the removal of the package from your hard disc:

```
Do you really want to delete package man? (y/n) [n]>>y

Removing package man

The following files are being deleted

...
list of files deleted
...

Removing the installation control files

man removed
```

Once the original package has been removed, as indicated by the final message above, eject the final archive floppy disc, label it along with the other archive discs and keep them with the other package discs. Should you wish to re-install the manual pages package, you can do so using these archive discs.

Installing a package

To load a package from a floppy disc, you use the *install a package* function from the *packageadmin* menu. For example, if you have previously removed the package called *games*, which you now wish to replace back onto your hard disc, type *i* (for install) at the enter selection prompt. You will see the installation menu:

```
Installation menu
=================

*exit
*install
```

```
enter selection (e/i) [i]>>i
```

Press *i* (for install) to carry on with the installation, you then see the prompt:

```
Place the first package disc in the drive

Press <return> when ready >>
```

Locate the *games* discs (there should be two) and put the first into the floppy disc drive, then press ↵ to continue the installation.

You are now asked to confirm that you wish to continue with the installation:

```
Do you really want to install package games? (y/n) [n]>> y
```

Type `y` (for yes) and the package will install automatically.

If the package extends over more than one disc, you will be prompted to insert the next disc in the sequence with a title comprising the name of the package, followed by a breakdown of the amount of space it takes up on the disc. For example:

```
Load disc 2 of 'Package: games ROOTNEEDED=0 USRNEEDED=1893'
Type RETURN when ready:
```

Press ↵ to continue the installation. When the installation has finished the Installation menu is redisplayed. You can now either carry on and install another package (by typing *i*) or exit the Installation menu (by typing *e*) to return back to the Main menu.

Customising packages

As well as the pre-defined set of packages you received with the RISC iX operating system, you can also define *any* set of files you choose to be a *package*. You can then use the *packageadmin* utility to archive the package and then remove it from the hard disc if you wish.

For example, if you never use the UNIX spelling checker you could group together all the files that make up the spelling checker into a package, archive the package, and then remove the original files from your hard disc. This will save around 420 KB of disc space.

For more information about creating your own sets of packages, refer to the *packageadmin*(8) manual page.

Alternative system commands

Introduction

The UNIX operating system has been developed over the course of many years by a variety of different companies and institutions. As a result, there currently exists a tremendous diversity of 'flavours' of UNIX, each one slightly different from the other.

This variety of flavours of UNIX presents problems for software developers who are writing applications that will run on UNIX systems *per se*. Due to these subtle differences between how manufacturers have implemented UNIX on their systems, an application program that is written to run under one version of UNIX may not be easily portable to another version of UNIX.

In an attempt to reduce such problems of incompatibility, a whole host of standards authorities have been created to specify how certain parts of the UNIX operating system should behave, including the utilities and tools that it should provide. Likewise, other standards authorities have been set up to offer guidelines to application developers for producing easily portable programs, at the source code level.

Despite the publicity from organisations such as the Open Software Foundation (OSF) and UNIX International (UI), Acorn have developed their version of UNIX in line with vendor neutral standards (TCP/IP, X/Open etc) to provide as much software compatibility as possible, across *all* ranges of UNIX workstations.

Ultimately, it is expected that a complete specification of the UNIX operating system will be produced and that this generic UNIX system will be implemented by all manufacturers of UNIX machines, thereby providing complete software compatibility for all UNIX-based applications across all UNIX systems.

RISC iX has incorporated many of the most important standards that have so far been defined. The standards that RISC iX conforms to are listed below:

- X/Open, XPG3 base level
- System V Interface Definition (SVID) conformance, Issue 3

There are a few areas where the X/Open XPG3 standard supersedes System V. In places where this occurs, Acorn have chosen to follow the X/Open standard. For more information about the above standards and what their implications are for writing software applications for RISC iX, refer to the chapter entitled *Conformance to standards* in the *RISC iX Programmer's Reference Manual, Volume 1 – Kernel and Languages*.

The implications of these standards for you as a System Administrator will be the facility it gives you to offer users of your system different operating environments in which to work, that conform to one or the other of the above standards.

This will be particularly important for users who are only familiar with working in one type of UNIX environment and the use of commands within this environment, or alternatively for users who wish to run an application that has been written to run under a specific version of UNIX that conforms to a particular standard.

The following directories in */usr* contain different sets of system binary commands that are included for compatibility with other versions of the UNIX operating system and for conformance with UNIX operating system standards. The directories are:

- */usr/5bin* – provides compliance with System V Interface Definition (SVID) conformance, Issue 3 and X/Open, XPG3 base level.
- */usr/ucb* – provides compatibility with the University of California's 4.3 Berkeley Software Distribution, usually referred to as 4.3BSD UNIX.
- */usr/bin* – provides the binaries which are not included in any of the above directories.

The contents of these *binary* directories are very similar – commonly used commands are contained in each directory. However, there are subtle differences in usage between similarly named commands contained in any of the binary directories, as each directory conforms to a particular standard.

Therefore, the order in which these directories appear in a user's PATH will determine which environment they are using. The list below sets out the most likely ordering of these directories, together with an explanation of the type of environment they define for a user:

- *PATH=.:/usr/5bin:/usr/bin:/usr/ucb:* – defines an operating environment that gives full System V compliance and conforms to the requirements of X/Open XPG3 base level. It also provides access to 4.3BSD functions.
- *PATH=.:/usr/ucb:/usr/bin:/usr/5bin:* – defines an operating environment that is 4.3BSD compliant and also meets some of the requirements of X/Open XPG3 base level and SVID, Issue 3.

You can also choose a subset of these directories to initially prevent a novice user from accessing certain system commands. For example, if you want to provide a user with an environment that is compliant with System V and meets some of the requirements of X/Open, but provides no access to any 4.3BSD functions, you would set their PATH to:

```
PATH=.:/usr/5bin:/usr/bin
```

For more information about creating new users, refer to the chapter entitled *Adding and removing users* on page 81.

Alternative operating systems

Introduction

Although your workstation is intended for use with the RISC iX operating system, you can also use other operating systems.

The RISC OS operating system

RISC OS is Acorn Computers' proprietary operating system used on their latest range of workstations including A3000s, A400s and A500s. RISC OS provides a single-user multi-tasking desktop environment, together with the advanced disc filing system ADFS.

RISC OS is also compatible with earlier Acorn computers via software emulators for the BBC model B computer and 6502 second processor.

Notes:

- Your supplier may have configured your workstation to run only specific software and you may not be able to use RISC OS without modifications from your supplier.
- There is a small RISC OS partition on the hard disc, mostly filled with application suite programs. Due to disc space limitations, large RISC OS applications and data files may have to be stored and run from floppy disc. However, if you obtain a second hard disc for your workstation you can allocate a much larger RISC OS partition.

An introduction to RISC OS is contained in *Installation Guide*. For more information about RISC OS, read the *RISC OS User Guide* supplied with your workstation.

The MS-DOS operating system

Your workstation can also run the MS-DOS operating system via an emulator.

To run MS-DOS you must first start up your workstation running RISC OS, then load the emulator which starts-up the MS-DOS operating system.

The emulator is available from your supplier.

Note: Do not try to create an MS-DOS partition on your hard disc without first contacting your supplier for advice.

Appendix A: Internet form

Introduction

The following reference section contains the Internet form that you should complete to apply for an official Internet number. The use of Internet numbers in network environments is discussed in the chapter entitled *Setting up a local Ethernet network* on page 129.

Internet form

To obtain an official network number, you must answer all the questions listed below. Send your answers, via electronic mail, to HOSTMASTER@SRI-NIC.ARPA. If electronic mail is not available to you, post the answers to:

DDN Network Information Centre
SRI International
Room EJ217
333 Ravenswood Avenue
Menlo Park, CA 94025

1 If the network is going to be connected to the DARPA Internet or the DDN Internet, you must provide the name of the sponsoring organization, and the name, title, mailing address, phone number, net mailbox, and NIC Handle (if any) of the contact person (POC) at that organization who has authorized the network connection. This person will serve as the POC for administrative and policy questions about authorization to be a part of the DARPA Internet or the DDN Internet. Examples of such sponsoring organizations are:

- Defence Communications Agency (DCA)
- Defence Advanced Research Projects Agency (DARPA)
- the National Science Foundation (NSF), or similar military or government sponsors.

NOTE: If the network will NOT be connected to either the DARPA Internet or the DDN Internet, then you do not need to provide this information.

Example:

Sponsor

Organization	DARPA
Name	Bloggs, Joe
Title	Program Manager
Mail Address	DARPA/ISTO Office 1400 Wilson Boulevard Arlington, VA 22209
Phone Number	(012) 345-6789
Net Mailbox	progmgr@VAX.DARPA.MIL
NIC Handle	AA12

2. Provide the name, title, mailing address, phone number, and organization of the administrative POC for the network requesting the number. This is the POC for administrative and policy questions about the network itself. If the network is associated with a research project this POC should be the Principal Investigator of the project.

   The online mailbox and NIC Handle (if any) of this person should also be included.

   Example:

   Administrator

Organization	SRI International Network Information Centre
Name	Bloggs, Joe
Title	Principal Investigator
Mail Address	SRI International 333 Ravenswood Avenue Menlo Park, CA 94025
Phone Number	(012) 345-6789)
Net Mailbox	pi@SRI-NIC.ARPA
NIC Handle	BB34

3. Provide the name, title, mailing address, phone number, and organization of the technical POC. The online mailbox and NIC Handle (if any) of the technical POC should also be included. This is the POC for resolving technical problems associated with the network and for updating information about the network. The technical POC may also be responsible for hosts attached to this network.

   Example:

   Technical POC

Organization	SRI International Network Information Centre
Name	Bloggs, Joe
Title	Computer Scientist
Mail Address	SRI International 333 Ravenswood Avenue Menlo Park, CA 94025
Phone Number	(012) 345-6789
Net Mailbox	cs@SRI-NIC.ARPA
NIC Handle	CC56

4. Supply the short mnemonic name for the network (up to 12 characters). This is the name that will be used as an identifier in Internet name and address tables.

   Example: ALPHA-BETA

5. Supply the descriptive name of the network (up to 20 characters).

   This name might be used to clarify the ownership, location, or purpose of the network.

   Example:

   Greek Alphabet Net

6. Identify the network geographic location.

   Example:

   SRI International
   Network Information Centre
   333 Ravenswood Avenue
   Menlo Park, CA 94025

7. Provide a citation to a document that describes the technical aspects of the network. If the document is online, give a pathname suitable for online retrieval.

   Example:

   The Ethernet, a Local Area Network: Data Link Layer and Physical Layer Specification, X3T51/80-50 Xerox, Stamford Connecticut, October 1980.

8. Gateway information required:

   If the network is to be connected to the DARPA Internet or the DDN Internet, answer questions 8.1 and 8.2.

8.1 Describe the Gateway that connects the new network to the DARPA Internet or the DDN Internet, and the date it will be operational. The gateway must be either a core gateway supplied and operated by BBN, or a gateway of another Autonomous System. If this gateway is not a core gateway, then an identifiable gateway in this gateway's Autonomous System must exchange routing information with a known core gateway via EGP.

A good way to answer this question is to say "Our gateway is supplied by person or company X and does whatever their standard issue gateway does".

Example: Our gateway is the standard issue supplied and operated by BBN, and will be installed and made operational on 1-April-83.

8.2 Describe the gateway machine, including:

- Hardware (LSI-11/23, VAX-11/750, etc. interfaces)
- Addresses (what host on what net for each connected net)
- Software (operating system and programming language)

Example:

- Hardware

  PDP-11/40, ARPANET Interface by ACC, Ethernet Interfaces by 3COM.

- Address

  10.9.0.193 on ARPANET

- Software

  Berkeley Unix 4.2 BSD and C

NOTE: for networks connected to the DARPA Internet or the DDN Internet the gateway must be either a core gateway supplied and operated by BBN, or a gateway of another Autonomous System. If this gateway is not a core gateway, then an identifiable gateway in this gateway's Autonomous System must exchange routing information with a known core gateway via EGP.

9. Estimate the number of hosts that will be on the network:

- Initially,
- Within one year,
- Within two years
- Within five years.

Example:

- initially = 5
- one year = 25
- two years = 50
- five years = 200

10. Unless a strong and convincing reason is presented, the network (if it qualifies at all) will be assigned a class C network number. If a class C network number is not acceptable for your purposes state why. (Note: If there are plans for more than a few local networks, and more than 100 hosts, you are strongly urged to consider subnetting, this is described in section entitled *Setting up subnets* on page 143.)

    Example:

    Class C is fine.

    Networks are characterized as being either Research, Defence, Government - Non Defence, or Commercial, and the network address space is shared between these three areas. Which type is this network?

    Example: Research

11. What is the purpose of the network?

    Example:

    To economically connect computers used in DARPA sponsored research project FROB-BRAF to the DARPA Internet or the DDN Internet to provide communication capability with other similar projects at UNIV-X and CORP-Y.

Further Information

For further information contact the DDN/ARPANET Network Information Centre (NIC):

- Via electronic mail: HOSTMASTER@SRI-NIC.ARPA
- Via telephone: (800) 235-3155
- Via postal mail: SRI International
  DDN Network Information Centre
  333 Ravenswood Avenue
  EJ217
  Menlo Park, CA 94025

Recommended Reading (available from the NIC)

- Braden, R.T.; Postel, J.B. Requirements for Internet gateways. Marina del Rey, CA: University of Southern California, Information Sciences Inst.; 1987 June; RFC 1009. 55 p. (SRI-NIC.ARPA RFC:RFC1009.TXT).
- Mogul, J.; Postel, J.B. Internet standard subnetting procedure. Stanford, CA: Stanford University; 1985 August; RFC 950. 18 p. (SRI-NIC.ARPA RFC:RFC950.TXT).
- Postel, J.B. Internet Control Message Protocol. Marina del Rey, CA: University of Southern California, Information Sciences Inst.; 1981 September; RFC 792. 21 p. (SRI-NIC.ARPA RFC:RFC792.TXT).
- Postel, J.B. Transmission Control Protocol. Marina del Rey, CA: University of Southern California, Information Sciences Inst.; 1981 September; RFC 793. 85 p. (SRI-NIC.ARPA RFC:RFC793.TXT).
- Postel, J.B. Address mappings. Marina del Rey, CA: University of Southern California, Information Sciences Inst.; 1981 September; RFC 796. 7 p. (SRI-NIC.ARPA RFC:RFC796.TXT). Obsoletes: IEN 115 (NACC 0968-79)
- Postel, J.B. User Datagram Protocol. Marina del Rey, CA: University of Southern California, Information Sciences Inst.; 1980 August 28; RFC 768. 3 p. (SRI-NIC.ARPA RFC:RFC768.TXT).
- Postel, J.B. Internet Protocol. Marina del Rey, CA: University of Southern California, Information Sciences Inst.; 1981 September; RFC 791. 45 p. (SRI-NIC.ARPA RFC:RFC791.TXT).
- Reynolds, J.K.; Postel, J.B. Assigned numbers. Marina del Rey, CA: University of Southern California, Information Sciences Inst.; 1987 May; RFC 1010. 44 p. (SRI-NIC.ARPA RFC:RFC1010.TXT). Obsoletes: RFC 990 (NACC 2139-86)
- Reynolds, J.K.; Postel, J.B. Official Internet protocols. Marina del Rey, CA: University of Southern California, Information Sciences Inst.; 1987 May; RFC 1011. 52 p. (SRI-NIC.ARPA RFC:RFC1011.TXT).
- Romano, S.; Stahl, M.K.; Recker, M. Internet numbers. Menlo Park, CA: SRI International, DDN Network Information Centre; 1988 August; RFC 1062. 65 p. (SRI-NIC.ARPA RFC:RFC1062.TXT).

Appendix B: Bibliography

Here is a list of books that you should refer to for further information about System Administration on UNIX systems:

- *UNIX System Administration* – D Fiedler & B H Hunter. Hayden Books, 1986. ISBN 0 905104 21 8.
- *UNIX System Security* – Patrick H. Wood and Stephen G. Kochan, Hayden books 1986. ISBN 0 8104 6267 2.
- *Internetworking with TCP/IP*– Douglas Comer. Prentice Hall, 1988. ISBN 0 13 470188 7.

For more information about security on your system and network, refer to:

- *New Directions in Cryptography* – Diffie and Hellman, *IEEE Transactions on Information Theory IT-22*, November 1976.
- TEMPO: A Network Time Controller for a Distributed Berkeley UNIX System – Gusella & Zatti, *USENIX 1984 Summer Conference Proceedings*, June 1984.
- *Data Encryption Standard* – National Bureau of Standards, *Federal Information Processing Standards Publication 46*, January 15, 1977.
- Using Encryption for Authentication in Large Networks of Computers – Needham & Schroeder, *Xerox Corporation CSL-78-4*, September 1978.

Index

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

V

W

X

Y

Z

Reader's Comment Form

RISC iX System Administrator's Guide

We would greatly appreciate your comments about this Guide, which will be taken into account for the next issue:

Did you find the information you wanted?

Do you like the way the information is presented?

General comments:

If there is not enough room for your comments, please continue overleaf.

How would you classify your experience with computers?

☐ **First-time user** ☐ **Used computers before** ☐ **Experienced user** ☐ **Programmer**

Cut out (or photocopy) and post to:

Dept RC, Technical Publications
Acorn Computers Limited
645 Newmarket Road
Cambridge CB5 8PB.

Your name and address:

This information will only be used to get in touch with you in case we wish to explore your comments further.