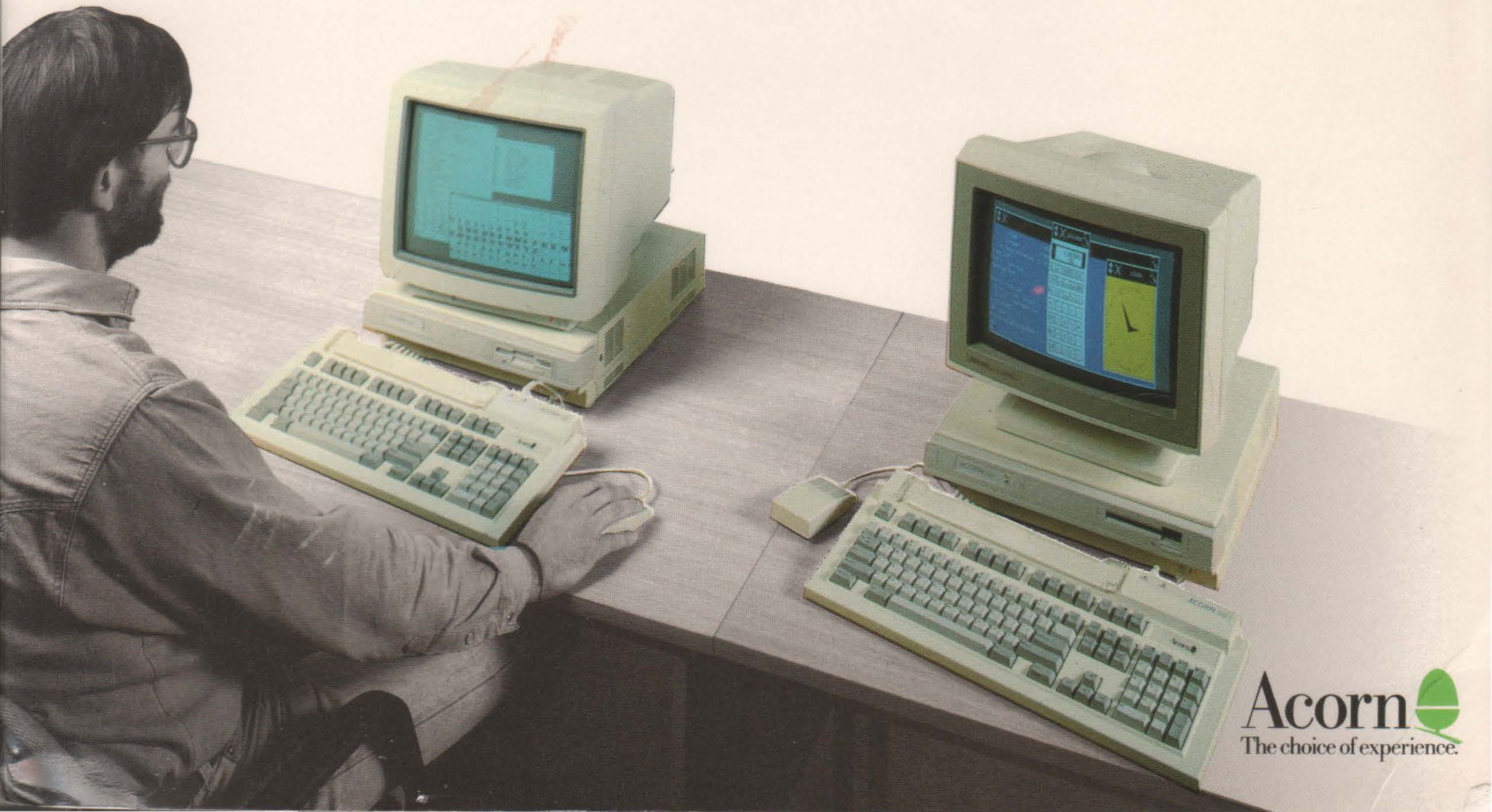
ACORN R140

RISC *i*X SYSTEM ADMINISTRATOR'S GUIDE



Acorn
The choice of experience.

ACORN R140

RISC *iX* SYSTEM ADMINISTRATOR'S GUIDE

Acorn

The choice of experience.

Contents

About this Guide

Readership of this Guide

If you are using your RISC iX workstation as a stand-alone system, this Guide provides you with enough information to start out as a Local System Administrator.

If you are using your workstation on a network, contact your network System Administrator or alternatively refer to the Bibliography at the back of this Guide, which contains a list of books suitable for UNIX System Administration on a network.

This Guide assumes that you have at least read the first few introductory chapters of the *RISC iX User Guide* and that you know how to log in to your system as `root`, as detailed in the *Operations Guide*.

UNIX 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 UNIX environment. So do not be put off by sometimes quite detailed information about the system appearing in this Guide. Many example System Administration programs are available for you to use that will simplify the job of looking after your system.

If you wish to develop your System Administration skills further, you should refer to the *Berkeley 4.3BSD System Manager's Manual* which contains all the reference manual pages and supplementary documents for System Administrators as well as dipping into some of the books referenced in the Bibliography at the back of this Guide.

Overview

Here is a summary of the information that is contained in this Guide:

Chapters

Introduction to System Administration – gives an overview of the role of a System Administrator and explains why this role is so important in a UNIX environment.

Finding out about the filesystem – a general description of the contents and structure of the filesystem, with special emphasis on the parts of it that are of interest to a System Administrator.

Starting up and shutting down the system – describes the different procedures for booting the system up into RISC iX and bringing the system down safely. This chapter also introduces the RISC iX filing system module and shows how it can be used for performing simple System Administration tasks.

Maintaining the filesystem – describes how to keep the filesystem in a healthy state and also introduces some sample system maintenance scripts.

Backing up the filesystem – details the various commands and utilities that you can use to backup your filesystem.

Adding and removing users – introduces the different methods available for adding and removing users from your system.

Attaching peripheral devices – details the types of devices that can be attached to your system and discusses the stages involved in connecting them to your machine.

Using the floppy disc utilities – describes all the floppy disc utilities that are available on your system.

Setting up UUCP – describes how to set up UUCP on your system.

Reference sections

The reference sections at the back of the Guide contain supplementary information:

The RISCiXFS module – gives a full description of the * commands supported by the RISCiXFS module along with a list of likely error messages that may be displayed.

PostScript printer filter – a full listing of the PostScript printer filter that enables you to attach an Apple LaserWriter to your system

Serial port connections – wiring diagrams for attaching specific peripheral devices to the serial port of your system.

Manual pages – contains the manual pages from section 8 of the *4.3BSD System Manager's Manual*, that have been referenced in this Guide.

An extensive bibliography is also included along with an index to help you find your way around the Guide.

Conventions used in this Guide

The following typographical conventions are used throughout this Guide:

Convention	**Meaning**
`<DELETE>`	Press the key indicated.
`<CTRL-D>`	Hold down the first key and press the second.
↵	Press the RETURN key.
`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 the file; for example, 'readme1':

```
$ cat readme1 ↵
```

UNIX 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:

- 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.

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 actions:

- Layout of the filesystem
- Users of the filesystem
- How the filesystem works
- Mounting other UNIX filesystems

Layout of the filesystem

The UNIX system is distributed with utilities and libraries in standard places in the file directory tree. The following list details the standard directories that are provided on your system:

- `/bin` – this is the directory into which standard UNIX programs are placed, without which the system itself cannot run or work normally, such as shells, the basic editor `ed`, `ls`, `cat` etc.

  Never remove or modify anything in this directory. Many other pieces of software assume normal operation of everything here.

- `/dev` – in this directory live a number of so-called **special files**. These are pseudo-files which give direct access to various devices.
- `/etc` – this is where essential system administration files and programs live. Items in here should be changed with extreme care.
- `/lib` – this is where standard libraries and auxiliary programs used by the software in `/bin` live.

You should never remove or modify files in this directory.

- `/tmp` – this is a directory, writable by all users, where various programs create temporary files. Files left in here are likely to be deleted without notice – for example, this directory is cleared out every time the system is restarted. Therefore, files created here should be moved to a more sensible place quickly if they are at all valuable.
- `/usr` – this directory contains a number of sub-directories, which in turn contain standard material not regarded as quite as essential as that in the preceding directories.
- `/usr/acorn` – contains utility programs specific to RISC iX workstations.
- `/usr/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 is contained in `/usr/adm/lastlog`; a record of each time the X Window System server was started is contained in `/usr/adm/X0msgs`.
- `/usr/adm/daily`, `weekly` and `monthly` – a set of three shell scripts for System Administration tasks. These scripts are discussed in the chapter, *Maintaining the filesystem*.
- `/usr/adm/dump` – a directory containing a set of shell scripts for performing incremental backups of your system. These scripts are discussed in the chapter, *Backing up the filesystem*.
- `/usr/bin` – contains UNIX software not as important as the software in `/bin`.
- `/usr/include` – contains standard `#include` files used by the C compiler.
- `/usr/lib` – contains libraries and auxiliary programs not quite as important as the software in `/lib`.
- `/usr/local` – contains programs and files local to the site. Software developed and used on your machine, and bought-in third party software 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 program files.

Administration files

The following lists summarise some of the more important administration files that are provided 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** devices (terminals and physical disc devices).

Care should be taken to preserve the access modes of access that are originally on these files. By widening the access, malicious or careless users can destroy the system completely by randomly writing over disc or memory. On the other hand some minimal access is required; for example, ps requires access to probe kernel memory using the memory access device /dev/kmem.

File	Function
console	Console terminal
eco*	Econet
fb	Frame buffer
fd*	Floppy disc block devices
kbd	Keyboard
kmem	Kernel memory
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)
rf*	Raw floppy disc devices
rst*	Raw hard disc devices (various units and partitions)
sd*	SCSI devices
serial	Serial port
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 generally used everywhere as a character sink, and /dev/tty is translated to mean the 'current terminal'.

Files in /etc

Some programs, such as `fsck`, `mount` etc are held in here, as are the system startup software and *daemon* processes, such as `init` and `cron`.

File	Function
`disktab`	Disc device description file
`dumpdates`	Dates when the `dump` was run on the system
`fstab`	List of filesystems
`gettytab`	Table used by `getty`
`group`	Group file
`motd`	Message of the day
`mtab`	Mount table
`passwd`	Password file
`passwd.dir`	Password file database
`passwd.pag`	Password file database
`printcap`	Printer description file
`rc`	Commands executed at startup
`rc.config`	Commands executed at startup
`rc.local`	Commands executed at startup
`rc.net`	Commands executed at startup for networked machines
`rc.yp`	Commands executed at startup for networked machines running the yellow pages network look-up service
`termcap`	Terminal description file
`ttys`	Terminal login description
`utmp`	List of currently logged-in users

The files `passwd.dir` and `passwd.pag` are created automatically from the password file by the program `/etc/mkpasswd`. This is run automatically by `vipw` and `useradmin`. For more information, refer to the chapter entitled *Adding and removing users*, later on in this Guide.

For more information about the commands executed by the `rc` files, refer to the chapter entitled *Starting up and shutting down the system*, later on in this Guide.

The file `mtab` is updated automatically when discs are mounted and unmounted. The file `utmp` is updated automatically when users log in and out (and is used by the `who` command).

Where groups are used, it is normal to permit other users in the same group to access more files than other users not in the group. You might want to permit other 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 groupids of the user attempting to access the file with these stored ids.

The userids are also used to decide who can send signals to running background processes using the `kill` command. Only if the userids match is this permitted; groupids 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 next section.
- `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, games maintaining 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.
- `operator` – is a system user who owns many of the system files and may have special access to some of them.

  You should not need to log in as this user.

- `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` is run instead. For more information, refer to the chapter *Setting up UUCP* later on in this Guide

The following are the group names installed on the system. You are very unlikely to need to add new names to this list.

- `wheel` – this group contains users who may execute the `su` command – the System Administrator should be added to this group.
- `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.

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 do so your groupid must be `0` (group `wheel`). (This may be the only way to get `root` access from a terminal if not marked secure in the `/etc/ttys` file – see later.)

Whilst running the `su` command you are still logged in as you were, and the terminal 'belongs' to you, and 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 etc, but if you reply the reply may seem to come from you, or the other user, or may get confused, depending on the mailer.

The main function of the `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 you can drop back into your logged-in shell by typing `<CTRL-D>`.

Naturally you will be asked for the password of the user you want to impersonate when you run `su`.

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

The login command

The `login` command enables you to switch completely to the other 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.

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

Set-user and set-group programs

UNIX has a feature enabling utilities to be developed with special privileges normally associated in a general way 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 does not normally have access.

The feature is the ability to make program files **set-user** and/or **set-group**. Programs such as these are called set-userid 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 set-user file is `root`. Thus in the case of `su` the unlimited privileges, once the password is checked, of `root` can be deployed to reset the effective and real users as required.

However other effective users do arise; for example `uucp` is set-user to user `uucp`, which 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 `ls -l`, 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.

How the filesystem works

The filesystem structure of UNIX, with its hierarchy of directories, is one of its key features. It is very important to keep the disc structures intact, as the kernel 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, and how to recover if anything goes wrong.

How files are made up

A file consists of two parts:

- The actual blocks of data of the file.
- The **inode**, which is a block of information containing the whereabouts of the blocks of 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 common for several names to refer to the same inode number and hence the same file.

How files are stored

A directory is itself stored as a file containing this information. 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 level down directory until the target file is found.

For example, to look up the name `/usr/users/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 `usr`. This has a corresponding inode number, and the inode is then looked at to find the blocks of the `/usr` directory, which is then scanned for the name `users` and so on.

The inode has room to store the whereabouts of only the first few blocks of each file. Accordingly extra blocks are used to remember where subsequent blocks of the file are held, which means that there is an escalating requirement 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 of this 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 save text image after execution). This is a special bit that can be set to increase the speed of execution of a regularly-run program, such as a screen editor. When a command is issued, all 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, the text parts of the program files are sequentially read 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. 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 UNIX*, 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 permission.

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 append only 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 can be strung together using commas, for example:

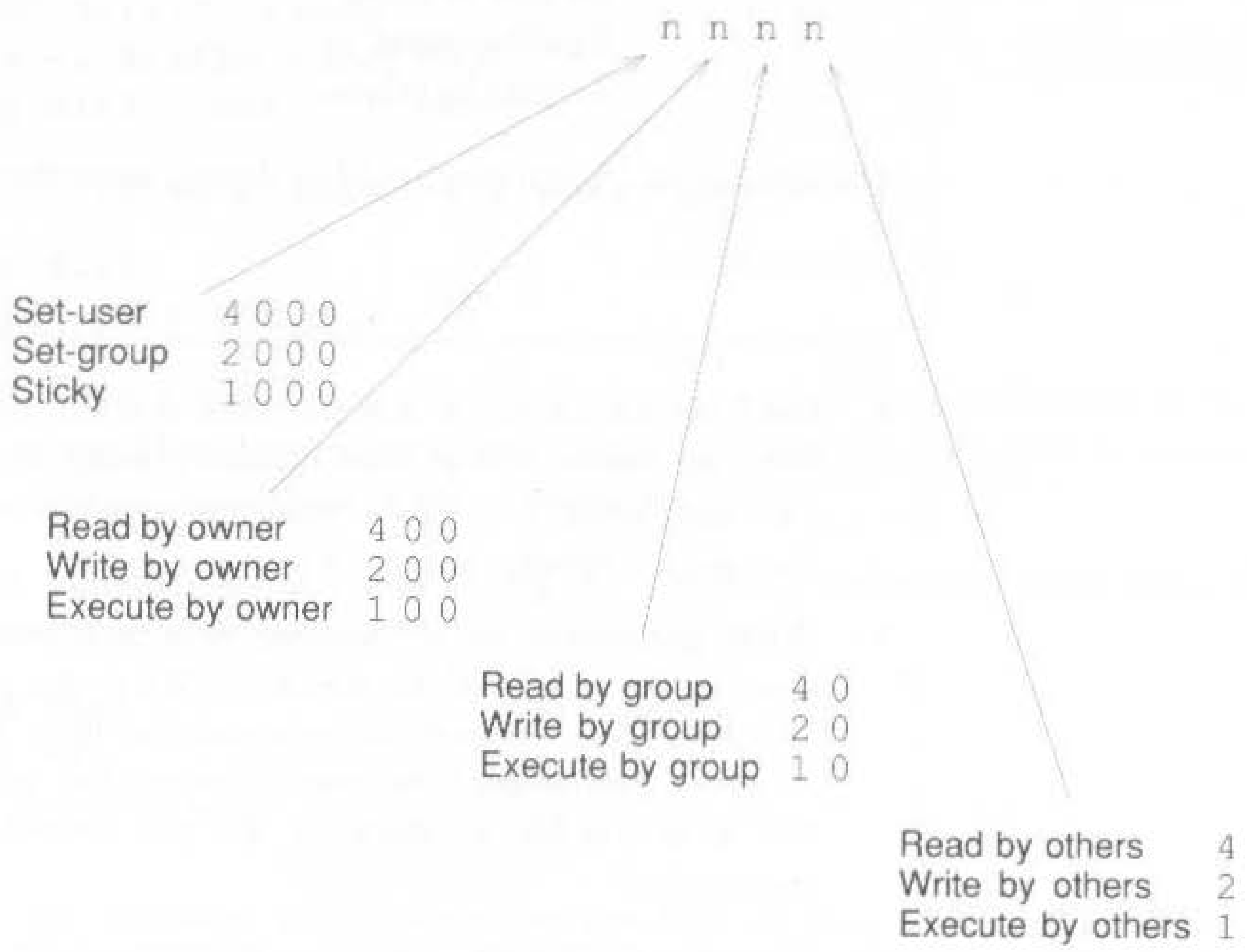
```
chmod u+w-s,o-r file4
```

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.

For more information, refer to `chmod`(1).

You will probably find it easier to think of modes as four octal digits, once you have started to understand them, as the octal notation is used extensively in the reference documentation.

Each digit denotes the set of permissions thus:



`chmod` can take modes in numeric form, 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. Only the last three digits need be specified if the set-user, set-group and sticky bits are all to be zero. Thus:

`chmod 4755` *`filename`*

indicates the following permissions for *`filename`* as follows:

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 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 anything about the type or contents of each file.
- **Write** permission on a directory does not permit even `root` to directly write to the blocks of the directory. What this means is that entries can be added (ie files created) or removed (ie files deleted) from the directory. However, this doesn't necessarily mean that you can read or write to the file; it is possible to delete a file you cannot read (however `rm` usually queries this).
- **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, but `/a/b/c/d` must, as appropriate, have read or write permission to access this file.

For example a directory marked as:

```
drwx--x--x ... dirl
```

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, but not to list the 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. However once you have done so, you will not be able to successfully execute an `ls` command without arguments.

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 we saw 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 this varies 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. 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 the compilers and the link editor `ld` commonly create files with execute permission, with mode `777`.

Note that many items of UNIX 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 `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 `ls -lu`.

  Use of the file as a program for execution 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:

```
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 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. This login name removes the need to log in first to execute this command; you can just type `sync` to the login: prompt instead. 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.

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, which enables processes to interface to devices, 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 devices 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 file and it becomes necessary to output a message onto the user's terminal. Quoting /dev/tty saves you from having to work out which one of 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:

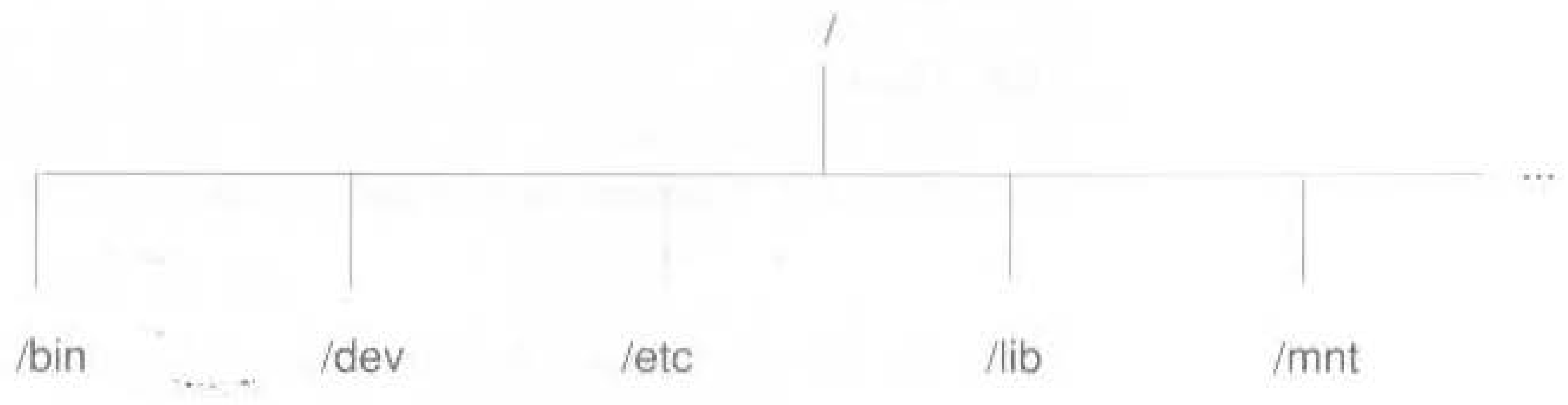
  ```
  cat > /dev/tty
  hello
  hello
  there
  there
  <CTRL-D>
  ```

For information about the other devices referred to in the /dev directory, refer to the chapter entitled *Attaching peripheral devices*, later on in this Guide.

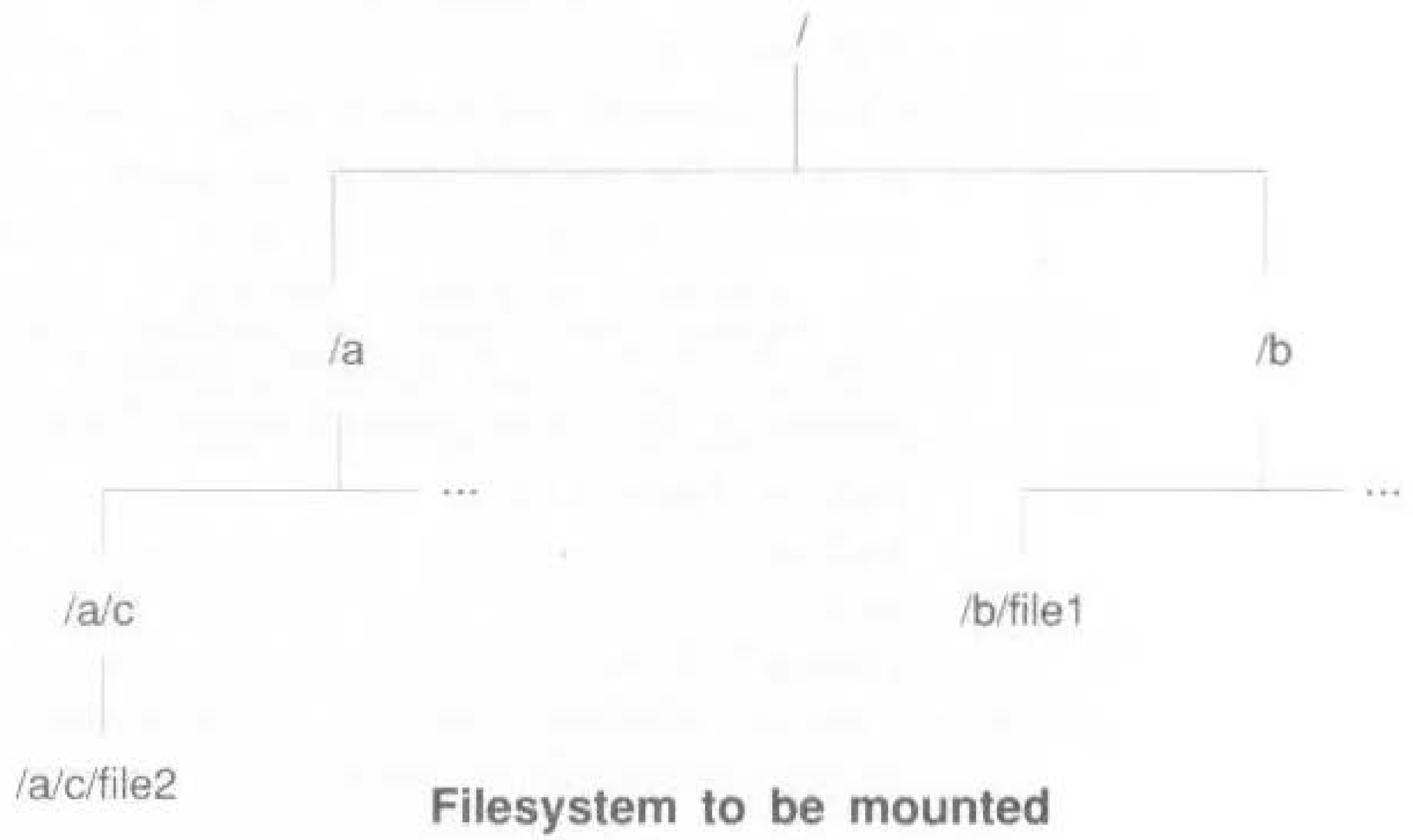
Mounting other UNIX filesystems

UNIX provides facilities for adding to the existing filesystem by enabling you to access external discs that also contain UNIX filesystems and making this filesystem appear to become part of the current filesystem. This is known as **mounting** discs.

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:



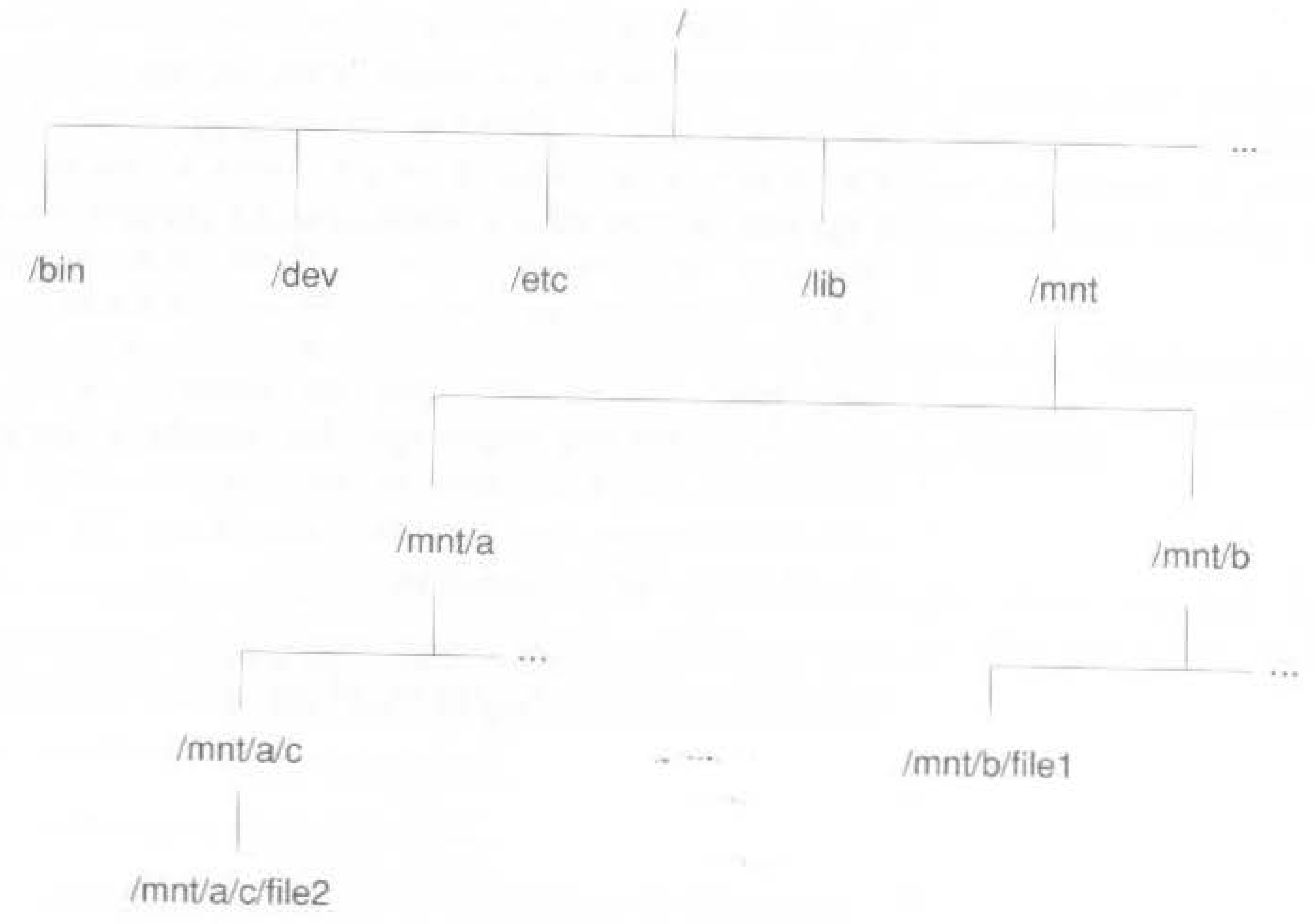
Existing filesystem



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 the `mount` command. 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 disc, 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).

In the case of floppy discs, you can chose whether to create a UNIX 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 UNIX filesystem is created on them, but for reasons of portability, non-UNIX 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 `/u`, `/mnt` as mount points. The files contained therein are invisible once items are mounted, so these directories should not be used to store files.

For more information about mounting floppy discs as UNIX filesystems, refer to the chapter *Using the floppy disc utilities*. Also, refer to `mount`(8).

Starting up and shutting down the system

Introduction

You should already be familiar with the procedures for turning your machine on and shutting it down safely, as detailed in the *Operations Guide*. This chapter moves on from this and explains each of these procedures in more detail and discusses alternative ways of bringing up the system and shutting it down.

This chapter also introduces the RISC iX filing system module, which enables you to configure the start up procedure and also to perform some simple System Administration tasks from RISC OS prior to entering RISC iX.

Starting up the system

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
- system processes are started
- consistency checks are performed on the filesystem
- 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. Some of these alternative procedures are discussed later on in this chapter.

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 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 directory of the boot device which also has a default location (`/dev/st0a` – the internal hard disc). Once the kernel is loaded by the boot program, the kernel image is then executed and by default defines the root filesystem to be the boot device and the swap area partition 1 on the root device.

For information about how to change the CMOS RAM parameter settings that the boot program uses, refer to the section *Using the RISCiXFS module* later in this chapter.

For information about how to change the batteries in the system unit, refer to the *Operations Guide*.

System processes

Once `/vmunix` is loaded 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` creates a simple 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 machine.

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

Multi-user mode

After `fsck` has been completed successfully, the main system processes are started up by `init` as indicated by messages appearing on the screen. For example, daemons are started for electronic mail, temporary files are removed etc. Eventually the following prompt is displayed on the console terminal screen:

```
RISC iX release 1.1
9:50am on Mon, 27 Feb 1989 on console of unix
unix login:
```

This indicates 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 multi-user mode, the main role of `init` is to create a terminal port on which a user may log into and execute commands and create new shells. 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 four terminal ports that can be used. The first three ports have a `getty` invoked on them and the fourth entry invokes `Xarm`, the X server for the X Window System. This last entry is initially commented out. For more information about `Xarm` and how to start the X Window System, refer to the section *Adjusting your system* later on in this chapter.

In the early stages of starting up, only the console terminal is active, and the discs are checked out before multi-user mode is entered. At this point, 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 logins ready to be accepted on each terminal, and various disc partitions are activated. This is said to be multi-user mode.

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.

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.

For more information about bringing your system up into single-user mode, refer to the section *Using the RISCiXFS module* later on in this chapter.

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 machine 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 machine are put into `/etc/rc.local`, those relating to network communications in `/etc/rc.net` and those relating to yellow pages in `/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 section *Adjusting your system* later on in this chapter.

Shutting down the system

It is particularly important to remember that on any UNIX system 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 many different commands you can use to shut down the system:

- `shutdown`
- `halt` and `fasthalt`

Each of the above commands are discussed in the following sections.

Shutting down into single-user mode

To bring the system down into single-user mode, use the `shutdown` command. For example:

```
shutdown +5
```

A message is displayed on the screen, 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.

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

Shutting down the system completely

To shut down your local RISC iX system as cleanly as possible, use the `halt` command. For example, at the normal shell prompt for `root` (`#`), type:

```
halt
```

This command gently brings the system down, by 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
```

You are then free to turn the power 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.

Note that the CMOS RAM settings in your machine may have been altered so that the machine returns automatically to RISC OS following a `halt` command. See the next section for more details.

There is a further invocation of `halt`, called `fasthalt`, that you can use to bring the system down less gracefully and without checking the filesystem.

For more information, refer to `halt`(8), `fasthalt`(8) and `sync`(8).

Selecting RISC OS from RISC iX

If your machine 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 back to RISC OS.

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

`halt -RISCOS`

This command halts RISC iX and brings the machine 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 peform some simple system administration tasks. For more information, refer to *Using the RISCiXFS module* later on in this chapter.

Adjusting your system

As a System Administrator, you may wish to configure parts of your system to suit your needs and the needs of other users. This section shows how you can make adjustments, both in the nature and order of the tasks performed during the start up and shut down procedures.

Setting the date

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.

To reset the date, log in as `root` and use the `date` command, with the time and date encoded in a string of digits *`yymmddhhmm`* for year/month/day/hour/minute (each 2 digits). For example, typing the following command:

```
date 8811121034
```

would set the time and date on the system to 10:34 on 12th November 1988 respectively.

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 name of the machine

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.

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 1.1 made Fri Jan 13 15:19:07 1989
```

The text of this message resides in the file `/etc/motd` (short for *message-of-the-day*) and is created 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.

Starting the X Window System

The X Window System can be started manually from any of the virtual terminals, using the command `xinit`, as detailed in the *RISC iX User Guide*.

However, you can also start-up the X Window System automatically, following a reboot, by editing the file /etc/ttys. This file contains two lines that start Xarm, the server for the X Window System. One line switches on the X server for a colour monitor system and the other switches on the X server for a monochrome monitor system.

For example:

```
# Uncomment only one of the lines "ttyvf" below to switch on the X server
# The following line when uncommented will switch on a monochrome X server
#ttyvf  "/usr/bin/X11/xterm -L -display unix:0 -geometry 96x32+1-1"     network on
secure window="/usr/bin/X11/Xarm -bw2"
# The following line when uncommented will switch on a colour X server
#ttyvf  "/usr/bin/X11/xterm -L -display unix:0 -geometry 80x24+1-1"     network on
secure window="/usr/bin/X11/Xarm -c4"
```

If you have a monochrome monitor, edit the file to remove the # symbol from the third line. The next time the machine is rebooted, the X Window System will be started up in the frame-buffer device fb+kbd (short for *frame buffer and keyboard*).

Alternatively, if this is all that you have changed, it is not worth rebooting the machine. To get init to read the /etc/ttys file again you can send it a signal (remember that it is process 1), type:

```
kill -HUP 1
```

The X Window System will be invoked. For more information, refer to init(8).

To disable the X Window System, just insert the # symbol back into the line that you edited in /etc/ttys and reboot the machine or use the kill command.

Starting the machine on a network

Your system is initially set up to be used as a stand-alone machine, 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
YELLOWPAGES=FALSE
...
```

The lines to change depend on the type of network environment that you are connecting your system to.

If you are connecting your system to a network that runs Sun Microsystem's NFS, edit `/etc/rc.config` to change the settings to:

```
...
STANDALONE=FALSE
FULLNETWORK=TRUE
YELLOWPAGES=FALSE
...
```

If your network is also running yellow pages, the distributed network database, change the settings to:

```
...
STANDALONE=FALSE
FULLNETWORK=TRUE
YELLOWPAGES=TRUE
...
```

You will then have to edit other files on your system that give a correct profile of the other machines on the network. After you have edited these files, to get the changes to take effect on your system, you will need to completely reboot the machine.

Refer to the Bibliography at the back of this Guide for manuals containing information about setting your system up on a network and network System Administration practises.

Using the RISCiXFS module

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`

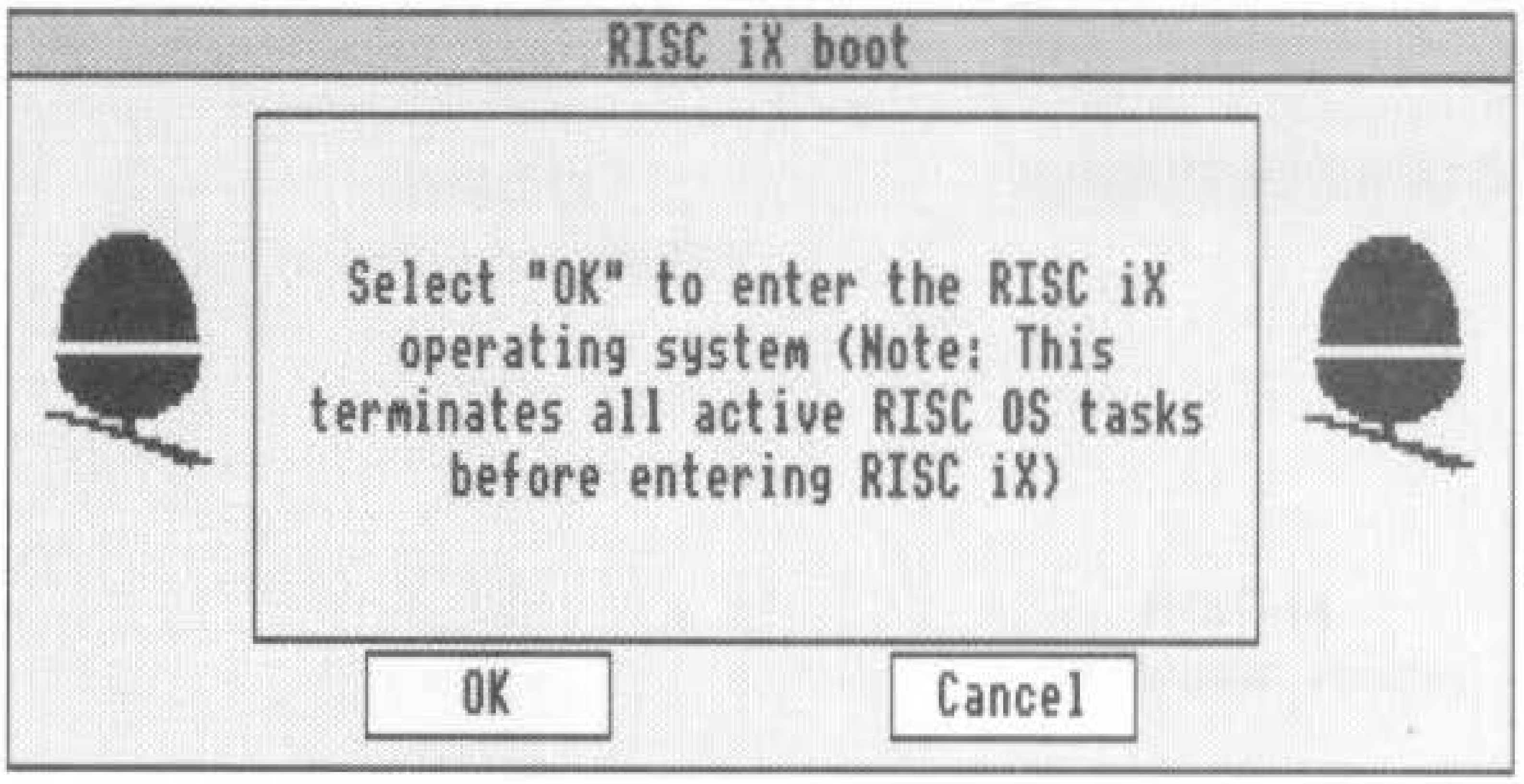
All the above operations can be performed from either the RISC OS Desktop by using the maintenance menu from the !RISCiX application, or from the RISC OS Supervisor command line.

This section describes the use of the RISCiXFS module from the RISC OS Desktop. For information about using the module from the Supervisor command line, refer to *Reference Section A – The RISCiXFS module*, which contains 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 more general information about RISC OS * commands and modules, refer to the *Archimedes User Guide*.

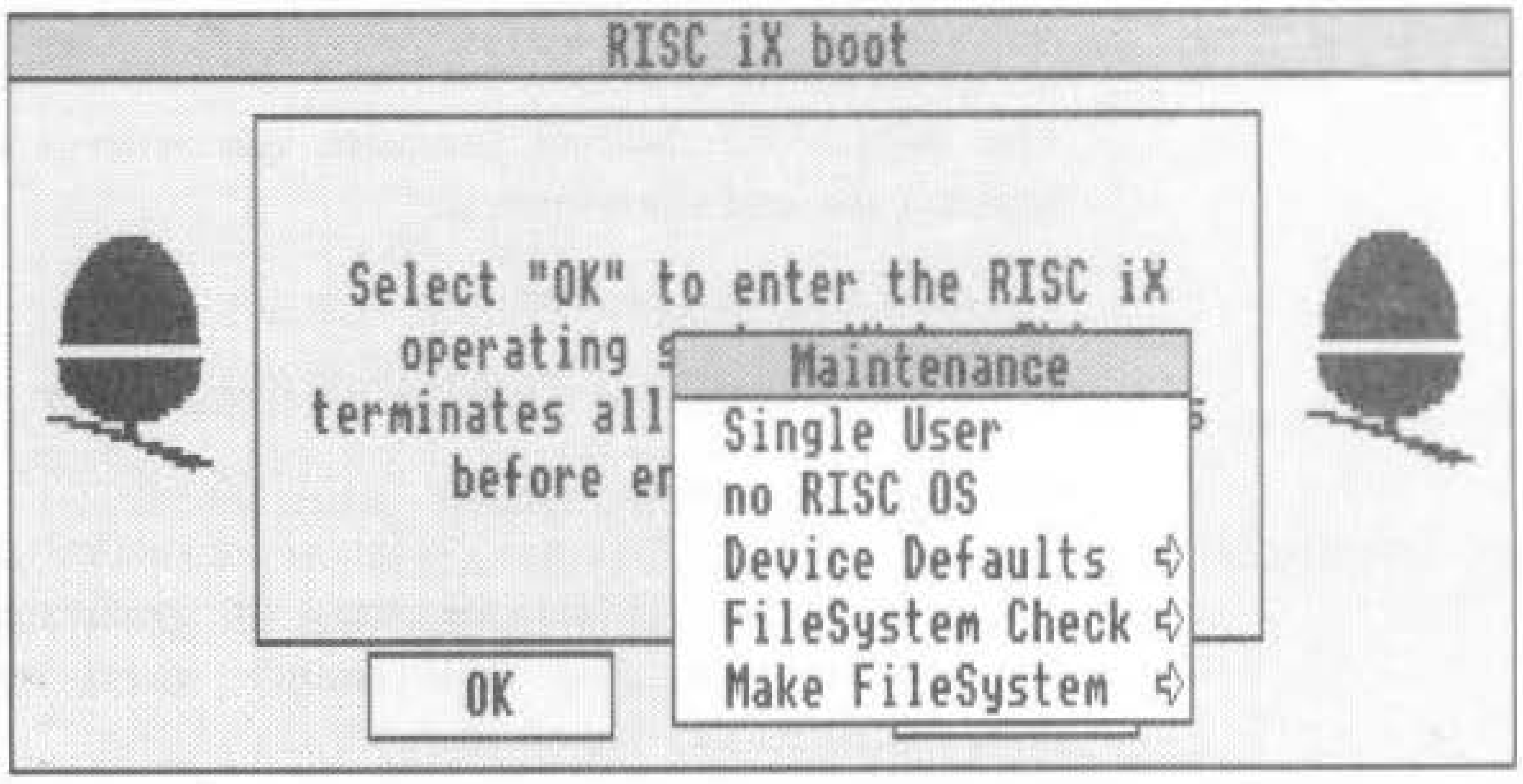
Selecting the maintenance menu

To bring up the maintenance menu, click Select (the left mouse button) on the RISC iX icon. This displays the RISC iX boot dialogue box:


RISC iX boot

Select "OK" to enter the RISC iX operating system (Note: This terminates all active RISC OS tasks before entering RISC iX)

Position the mouse pointer over the middle of the box and click Menu (the middle mouse button). The maintenance menu will be displayed:


RISC iX boot

Select "OK" to enter the RISC iX operating s terminates all before er

OK

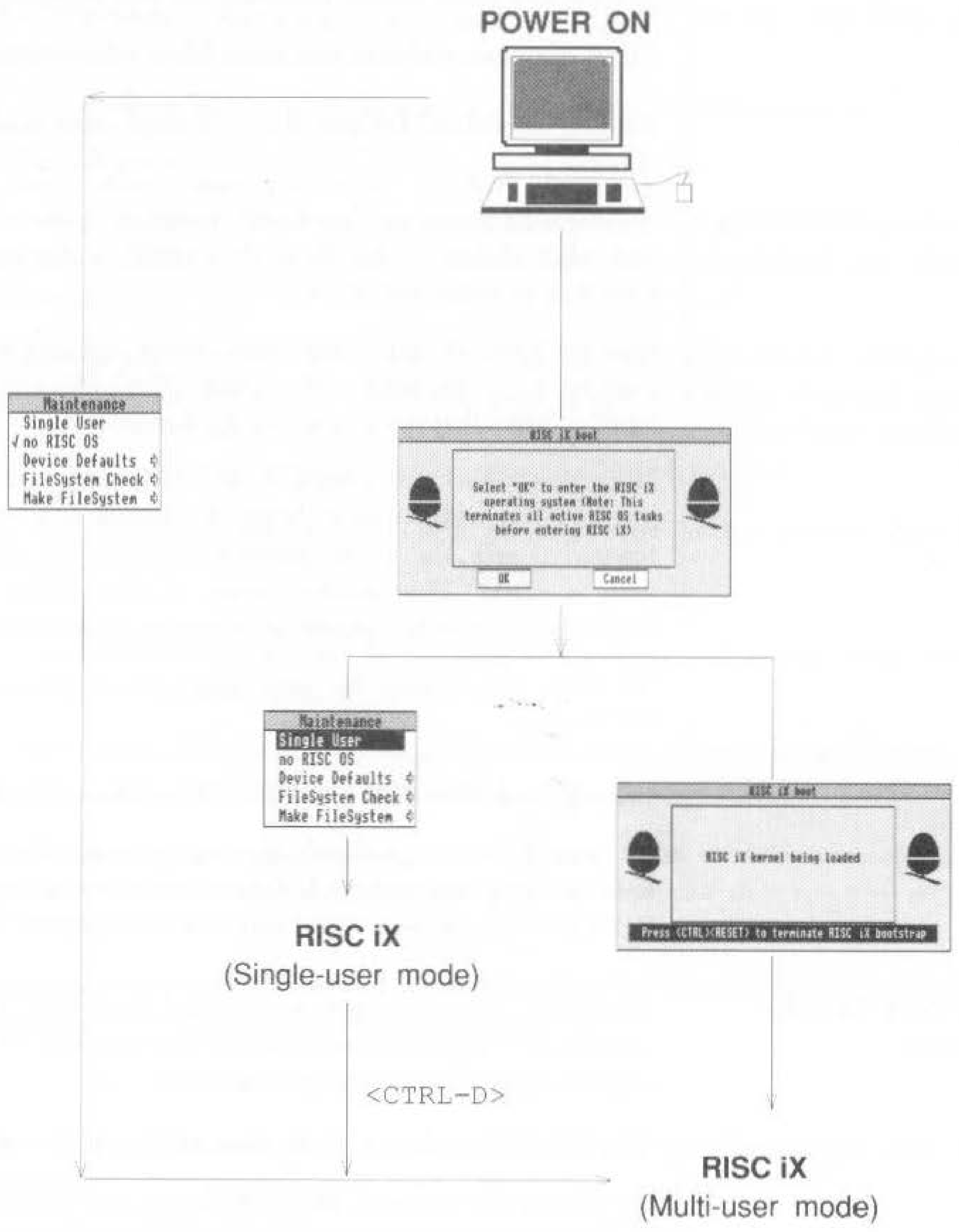
The options listed in this menu are described in the following sections.

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, booting the machine into single-user mode.

The following diagram shows a schematic of the different boot options available:



Select "OK" to enter the RISC iX operating system (Note: This terminates all active RISC OS tasks before entering RISC iX)

RISC iX kernel being loaded

Press <CTRL><RESET> to terminate RISC iX bootstrap

Starting up the machine 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 Select 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 Select. The RISC iX kernel is bootstrapped and the system is started up in single-user mode.

Booting RISC iX into single-user mode terminates **all** active RISC OS tasks completely – so before you enter RISC iX make sure that there are no RISC OS files that need to be saved, otherwise you will lose this work.

Note that when the system is in single-user mode, the user is logged in as `root`. As it is possible to bring the system down to single-user mode without having to give the `root` password (as detailed above), 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*, later on in this chapter.

To bring the system up into multi-user mode from single-user mode, press `<CTRL-D>`.

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 Select 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 default boot device

By default, when the system is booted from RISC OS it boots from the device `st0(0,0)` which corresponds to the internal ST506 hard disc driver, with major number 0, unit 0 and partition 0.

This default 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 and using Select, 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)`.

Running fsck

The filesystem consistency check program, `fsck`, can be run from the maintenence menu and will primarily be 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 Select 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 full documentation on 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 Select on Make FileSystem from the maintenance menu. A new filesystem will be constructed, by default, on the device (`/dev/st0b`).

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 *Reference Section A – The RISCiXFS module*, at the back of this Guide.

The disc can also be mounted from RISC iX by using the `mount` command. For more information refer to the chapter *Using the floppy disc utilities*, later on in this guide.

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 machine so that it uses `secureboot` instead of RISCiXFS, bring the machine down to RISC OS by typing:

```
halt -RISCOS
```

In RISC OS Supervisor mode, rename the original !Boot file that loads the RISCiXFS module to some other name (say RFSBoot) and rename SBBoot to !Boot:

```
*rename $.!Boot $.RFSBoot
*rename $.SBBoot $.!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. The machine should display the following prompt:

```
**|> !Boot (for secure RISC iX bootstrapping)
**rmload Secureboot
```

Once the module is loaded, the system will then automatically boot up into RISC iX in multi-user mode.

From now on, whenever the machine 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 machine to RISC OS (for example `halt -RISCOS`) will have no effect and will just cause the machine to be re-booted.

Disabling secureboot

To disable `secureboot`, reboot the machine and press `<ESC>` when the `secureboot` module is being loaded, as indicated by the display below:

```
**|> !Boot (for secure RISC iX bootstrapping)
**rmload Secureboot <ESC>
*
```

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 machine should now load the RISCiXFS module as before.

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*, later on in this Guide.

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)`.

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 `/tmp` directory that is usually cleared out during the start up procedure, may need to be deleted occassionally. Other temporary directories (`/usr/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 /usr/tmp -ctime +2 -exec rm -rf {} \;
```

This command will delete all the files and sub-directories found in `/tmp` or `/usr/tmp` that have 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 concerning files.

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 /usr/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 * * * /etc/clean-disc
```

In this case the commands would be placed in a (Bourne) shell script called `/etc/clean-disc`.

For more information, refer to `cron`(8).

Filesystem maintenance scripts

To help you to keep your system healthy, there are three scripts on your system in the directory `/usr/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 above 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 six different 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         -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 /usr/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 /usr/tmp; find . ! -name . ! -name lost+found \
                    -mtime +7 -exec rm -f {} \;
```

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

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

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

```
find /usr/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`. For example:

```
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 {} \;
```

The next part of the script, runs the command:

```
msgs -c
```

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

If you have system accounting set up on your machine, 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:"
#/etc/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 `/usr/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 commands that are to be run at certain times in a similar manner to entries in the crontab file. 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 /usr/spool/debugging as determined by the following line in the system configuration file /etc/syslog.conf:

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

This is initially commented out, but debugging can be started by removing the hash symbol from the line and rebooting the machine. All system debugging messages issued by the kernel will then be logged in the file /usr/spool/debugging.

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

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

This is initially set to be running on your system. For more information, refer to syslogd(8). The file syslog in /usr/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 /usr/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 daily script runs the shell script `/usr/lib/uucp/clean.daily` that rotates the UUCP log files in a similar fashion to the mail system log files, keeping them for a seven-day period before deleting them. This will only be relevant if UUCP is running on your system:

```
echo ""
echo "Cleaning up UUCP:"
su uucp << EOF
     /usr/lib/uucp/clean.daily
EOF
```

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:

```
echo ""
echo "Checking filesystems:"
sync
/etc/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. Followed by the command `mailq` 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 weekly script

The first part of the `weekly` shell script searches through all the user directories on the system in `/usr/users` for all the '`.o`' files that are generated by the C compiler, `cc`. If any of these files have not been accessed for more than 21 days, they are removed:

```
# Comment out the next three lines if you dont want user .o files purged
echo ""
echo "Removing old .o files:"
find /usr/users -name '*.o' -atime +21 -print -a -exec rm -f {} \;
```

As indicated by the comment included above the command, if you do not wish to remove these files comment out the command completely, by inserting '#' symbols at the beginning of each line.

The next part of the script performs a four-weekly rotation of the system messages file `/usr/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    /usr/adm/messages
```

This is initially set to be running on your system. For more information, refer to syslogd(8). The file messages in /usr/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 /usr/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 /usr/adm/wtmp:

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

If this file exists, then the script runs the command ac with a 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 `/usr/adm/wtmp` file:

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

Running the scripts using cron

The `crontab` file located in `/usr/lib/crontab` on your system already contains entries for the 'housekeeping' scripts just described:

```
...
#
# Run the daily script at 1 am every day
#
0 1 * * *       root    /bin/sh /usr/adm/daily 2>&1 | mail root
#
# Run the weekly script at 2am on Saturday
#
0 2 * * 6       root    /bin/sh /usr/adm/weekly 2>&1 | mail root
#
# Run the monthly script at 3am on the first of the month
#
0 3 1 * *       root    /bin/sh /usr/adm/monthly 2>&1 | mail root
...
```

As indicated by the comments in the `crontab` file, the `daily` script is run at 01.00 am every day, the `weekly` script is run at 02.00 am every Saturday and the `monthly` script is run at 03.00 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 `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:

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

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.

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 onto floppy disc. For example, to write the contents of the directories dir1 and dir2 to floppy disc in tar format, 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 verbose account of it's 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 another 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 and 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 the 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.

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 about five hours, requires constant attention and uses approximately 45 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 higher-numbered (and thereby lower priority) 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 accidently 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/rst0a
```

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 /usr/adm/dump called zerodump which you can use to perform level 0 backups of your system. For more information, refer to the section entitled *Using the backup scripts* later on in this chapter.

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
```

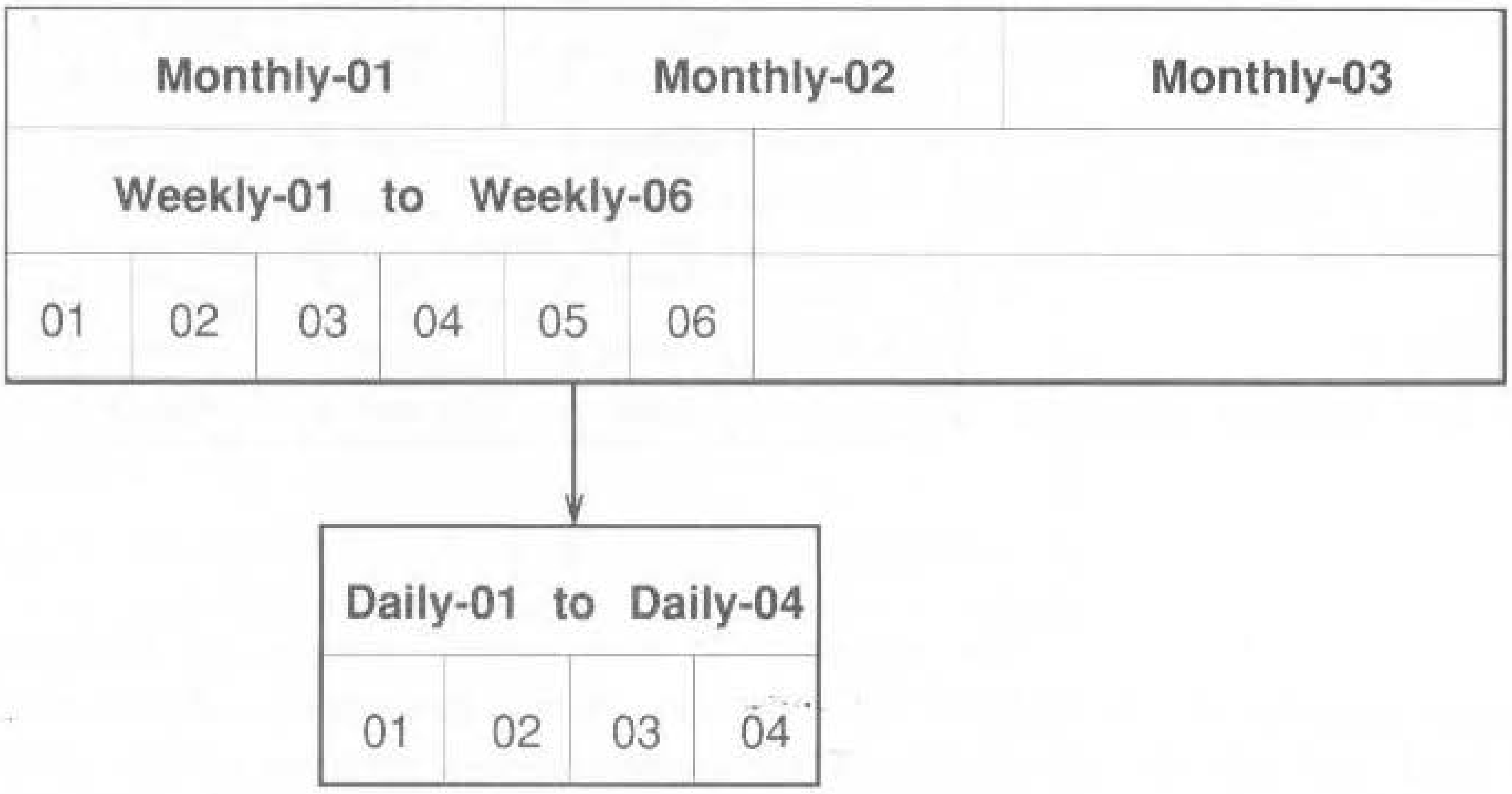
If the backup was taken using compress and dsplit, type:

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

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-01` 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 ensures that you always have two recent backup copies of a file that is changing daily.

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

Monthly	Daily				Weekly
	Monday	Tuesday	Wednesday	Thursday	Friday
Level 0	Level 3	Level 2	Level 5	Level 4	Level 1
	Level 3	Level 2	Level 5	Level 4	Level 1
	Level 3	Level 2	Level 5	Level 4	Level 1
	Level 3	Level 2	Level 5	Level 4	Level 1
Level 0	Level 3	Level 2	Level 5	Level 4	Level 1
	Level 3	Level 2	Level 5	Level 4	Level 1

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:

```
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:

```
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 moveable 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 attatch 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 directories, 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 on your system. A typical entry appears below:

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

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.

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 doing this.

- `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 standard UNIX utility for editing the password file. This is more suited to experienced system administrators and should be used with care.

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, at your normal shell prompt type:

```
useradmin
```

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

To add a new user, type `u`, the cursor drops to the bottom of the screen and you are prompted to type in a username for the new user:

```
User name:
```

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

With a valid username 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.

With a userid entered, you will then be prompted to insert a group name:

```
Group:
```

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.

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 `/usr/users/`*`username`*.

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

```
Shell:
```

The choices for this field are either `/bin/sh` or `/bin/csh`. Pressing the space-bar selects the default shell `/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 will not be deleted.

Removing an existing user

An existing user can 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 (userid 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, namely:

- 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 "/bin/shell":
```

Type in the new shell (either /bin/sh or /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 username 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 groupids 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 groupname:

```
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 groupname that already exists, the original prompt is displayed again.

With a valid groupname entered, you will then be prompted to insert a group id 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 groupid that already exists, the original prompt is displayed again.

With a valid groupid 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).

For example:

```
psmith::101:100:Paul Smith:/usr/users/paul:/bin/csh
```

The fields in these lines have the following meanings:

- `User name` – 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. This is normally up to seven 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 assists 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 similar other variation.

- `password` – is either empty, to indicate that the user does not have a password (this is not recommended), or a string of characters inserted automatically by the `passwd(1)` program. The characters displayed represent an encoded version of the password.

- `Userid` – is the numeric userid of the user. The userid 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).

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

- `Groupid` – is the groupid of which the user is by default a member. This should normally correspond to a groupid in the `/etc/group` file.

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

- `Full user name` – this 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 path name of the login shell. Standard path names are `/bin/sh` and `/bin/csh` for the Bourne shell and C shell respectively. If left blank, then the Bourne Shell is assumed.

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 `/etc/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
/etc/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:

```
users::100:joe,bill
```

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

To add any members to this file just edit it and save the changes, using a normal editor.

Creating the home directory

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

```
mkdir /usr/users/paul
```

As the directory `paul` is contained in a directory owned by you (`/usr/users`), 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 /usr/users/paul
```

To change the group ownership of the directory, type:

```
chgrp users /usr/users/paul
```

Following this, any files created within this home directory will be owned by the user but will still have the groupid of the directory and not the groupid 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 `passwd` command. For example:

```
passwd paul
New passwd:
```

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

```
Reetype new password:
```

Type the password in again. A check is run to see if the two passwords comply. If they do, 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 created 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.

Attaching peripheral devices

Introduction

There are a number of peripheral devices that you can attach to your system to expand its capabilities. For example:

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

This chapter offers some general guidelines for setting up each of the above devices on your system. Where possible, examples are given for attaching 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`
- Parallel port – device name `/dev/lp`
- SCSI port (only available if your system has a SCSI expansion card installed)

The characteristics of each of the above ports are described in the next few sections along with information about the types of devices that can be connected to them.

Serial port

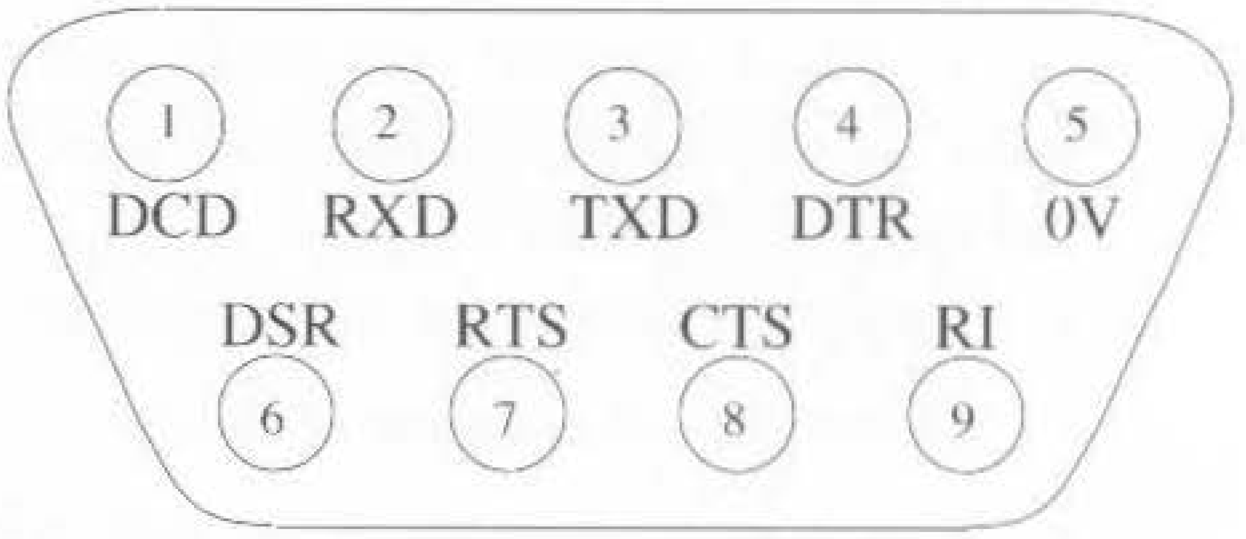
The serial port is by far the most versatile and therefore most troublesome port to use. The port supports two different types of peripheral or equipment: Data Terminal Equipment (or DTE – for example printers and computers), and Data Communication Equipment (or DCE – for example modems). It follows that you will need different cables, depending upon which type of peripheral (DTE or DCE) you are connecting to your machine.

In UNIX 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.

Serial port pin assignment

The following diagram shows the assignment of the pins on the serial port of the device that is to be connected to a RISC iX workstation, viewed from the side that is to be soldered:



This view also corresponds to the view of the serial port socket from the rear of the RISC iX workstation. The pin assignment of 9-pin serial ports on other hardware is often the same as this.

When you connect peripherals to the serial port:

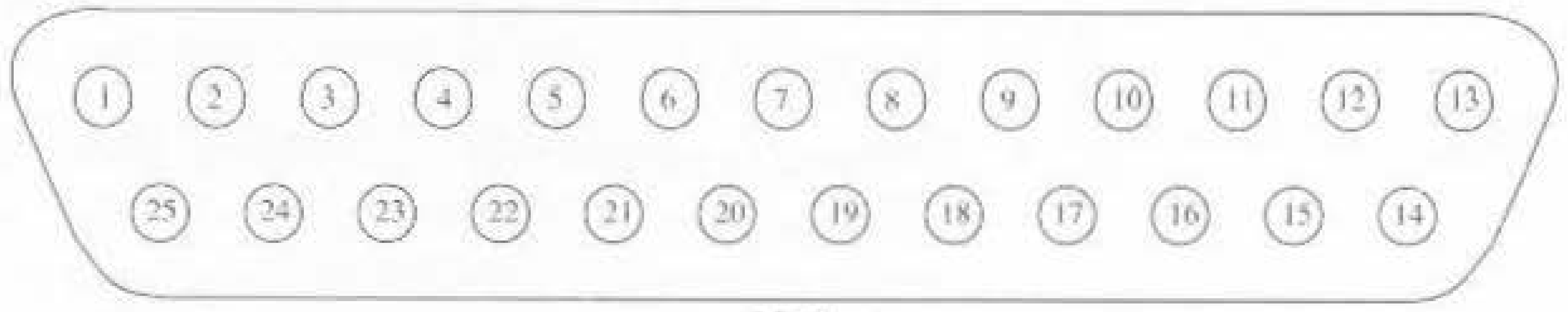
- ensure that screened cabling is used to connect up the peripheral.
- consult the peripheral manufacturer's instructions for pin connections but note that at the RISC iX workstation end of the cable, connections to the CTS signal should instead be made to the DSR signal. Examples of signal connections to specific peripheral devices are given in *Reference Section C – Serial port connections.*

Parallel port

The parallel port sends data in a pre-determined manner and so is less susceptible to error. Parallel ports are more commonly used to connect to printers and send data to a device using eight data pins in parallel making a byte of character data.

Parallel port pin assignment

The following diagram shows the assignment of the pins on the parallel port:



PIN	Name
1	-STB
2	D0
3	D1
4	D2
5	D3
6	D4
7	D5
8	D6
9	D7
10	-ACK
11	-BSY
12-16	Not connected
17-25	0V

SCSI port

The SCSI port provides an industry standard interface that enables a whole host of SCSI-compatible devices to be connected with little or no confusion over signal connections.

The SCSI port is a high-speed interface that conforms to the ANSI standard SCSI specification and allows any ANSI compatible SCSI peripherals such as tape streamers and hard discs to be connected to it.

For more information about the characteristics of the SCSI hardware interface, refer to the *SCSI User Guide* accompanying the SCSI expansion card.

Stages involved in connecting a device

The remainder of this chapter describes how to attach various peripheral devices to your system using the ports just described.

There are 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.

The procedures outlined in the next few sections for connecting each type of peripheral device are split up into these two stages.

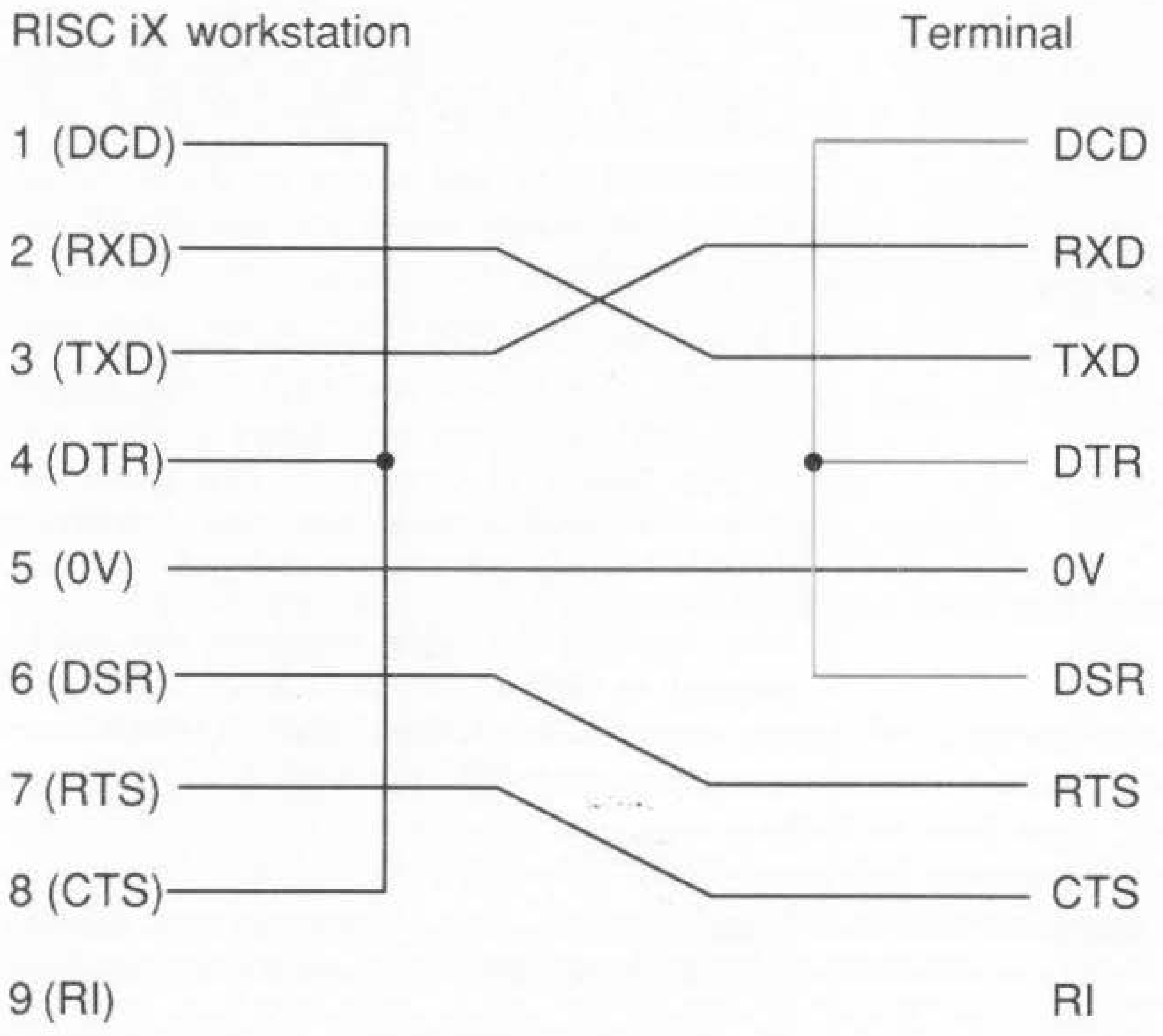
Note that the signal connection diagrams that are given for each hardware connection act as guidelines which will work with most hardware. Some or all of the connections shown with dotted lines may be unnecessary, and you must first check the manuals accompanying the hardware that you are connecting.

Terminals

Attaching an extra terminal to your system is a relatively painless task that greatly enhances the capability of your machine, as it enables an extra person to use the system at any one time.

Hardware connection

A wide variety of different terminals can be connected to your system using the serial line port (`/dev/serial`). To physically connect a terminal to your system, follow the instructions given in the documentation provided with the terminal. Here is a general guide to the signals required on the cable:



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 hangup 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, then certain connections need to be transposed in the cable, thus both send data on the line 'Transmitted Data' and receive 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' (although UNIX does not support this properly), 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 monitor, 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 banner and 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 login, are listed in the file /etc/ttys. Lines in this file take the following form:

```
console   "/etc/getty std.9600"   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.9600) 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, with a speed of 9600 baud. This is usually the command invoked, however sometimes special commands relating to window servers are also used. For example, the command xterm (the terminal emulator for the X Window System) is invoked on the line ttyvf.

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 safe to log in as root. To disable this feature, delete the word secure from the line.

Using this information, a suitable entry for a new vt100 terminal, connected to the serial port of your system and running at 9600 baud, would be:

```
serial        "/etc/getty std.9600"  vt100
```

For more information about seting 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

Printers on the serial port

If a 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 etc 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:

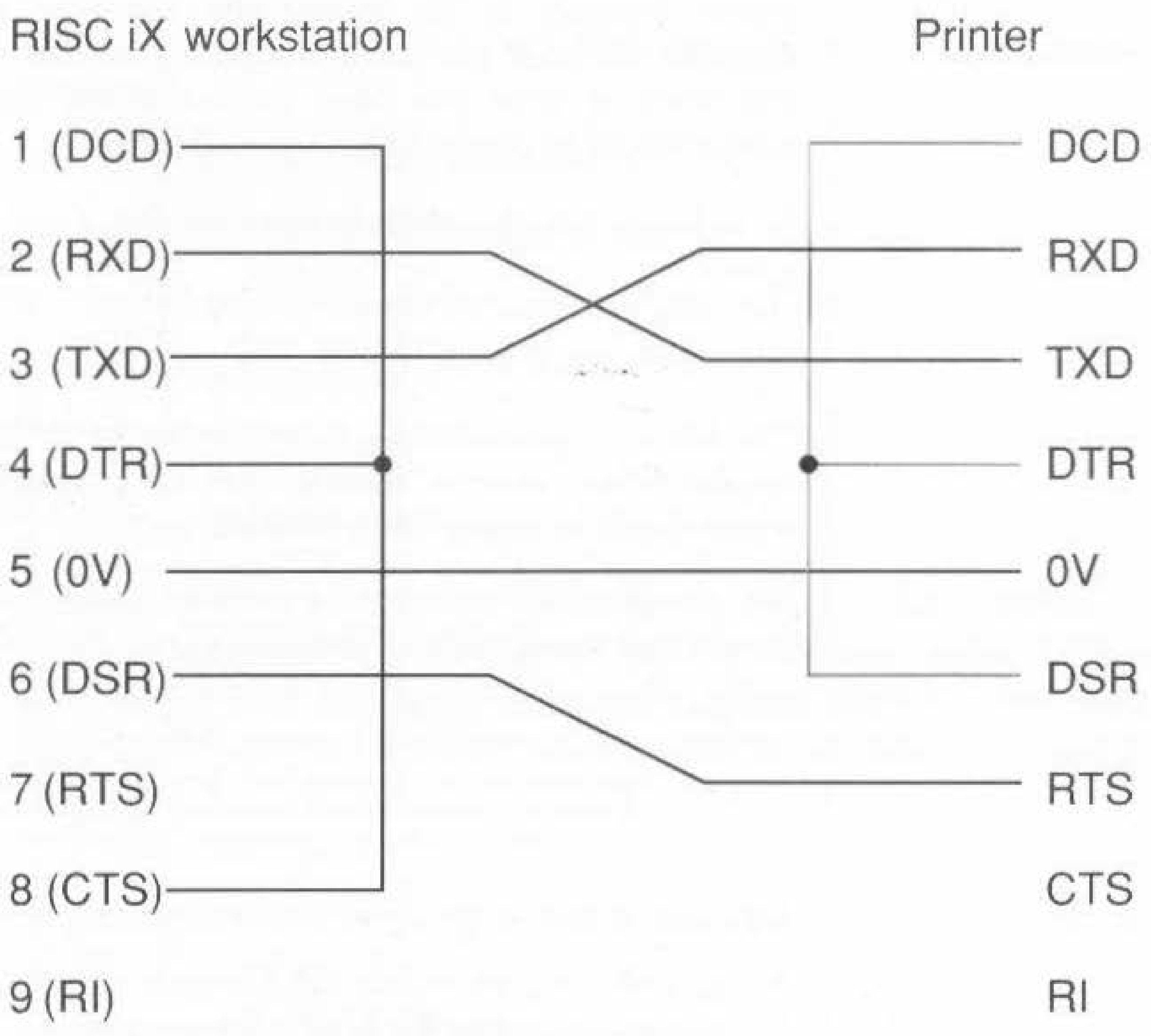
```
stty 1200 </dev/serial
```

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.

You should check carefully what options for flow control your printer uses, and if possible set it to XOFF/XON flow control. Some printers have alternative flow control mechanisms involving ETX/ACK (different control characters from XOFF/XON transmitted under slightly different circumstances), or RTS/CTS (signals on additional wires on the interface cable). UNIX does not have options to handle these methods of flow control. Here is a general guide to the signals required on the cable:



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 process for doing this for printers connected to the parallel port or serial port is outlined below.

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 computer that the computer 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 misfeeds. 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.

For example, here is a typical printcap entry that is suitable for connecting a LaserWriter to your RISC iX workstation:

```
# 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=/bin/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 which all data will be sent to the printer ie the serial port. If the printer is connected to the parallel port this would be `/dev/parallel`.
- `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=/bin/psfilter` – sets the name of the printer text filter to do accounting. An example of a suitable PostScript filter is listed in *Reference Section B – PostScript printer 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 full 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.

Modems

A modem is usually connected up differently from a terminal. This is because a modem is a DCE (Data Communications Equipment) device, whereas terminals and workstations are DTE (Data Terminal Equipment) devices. The main difference is that the connections are 'straight through', in that the modem expects to *receive* data on the 'transmitted data' line, and vice versa, whereas when connecting terminal to computer or computer to computer the connections have to be crossed over.

Hardware connection

You may need to make an adapter cable to connect a modem or other standard RS232 DCE device. Here is the wiring guide for connecting to a standard RS232 25-way female D-type socket:

RISC iX workstation (9-way)	Modem (25-way)
Female D-plug	Male D-plug
1 (DCD)	8 (DCD)
2 (RXD)	3 (RXD)
3 (TXD)	2 (TXD)
4 (DTR)	20 (DTR)
5 (0V)	7 (0V)
6 (DSR)	6 (DSR)
7 (RTS)	4 (RTS)
8 (CTS)	5 (CTS)
9 (RI)	22 (RI)

In practise you may leave out RI and DSR as they are not used. If the modem doesn't provide a CTS signal, leave out this wire and instead strap it to RTS at the RISC iX workstation end.

If you are connecting a different type of modem than the one illustrated the differences will be in which signals the modem looks at and can supply. More modern or expensive modems can be configured to use or ignore the various control signals, by means of DIP switch settings or by sending commands.

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 comunicate, 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 `gettytab`.

For more information about setting up UUCP, refer to the chapter *Setting up UUCP*, later on in this Guide.

SCSI peripherals

There are a number of SCSI peripherals that can be attached to your system. These include:

- hard discs
- tape streamers

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 system aware that a new device is attached to your system.

For information about connecting a SCSI device to your system, refer to the *SCSI User Guide*, accompanying the SCSI expansion card.

Software connection

This section provides a lot of general information about how to set up a SCSI device on your system. A RODIME RO3000S hard disc drive will be used as an example SCSI device, to show the commands that are needed for a particular device.

The SCSI 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, refers 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 *Boot in *Reference Section A – The RISCiXFS module*.

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:

- Create suitable entries in the `/dev` directory for the device
- Format and partition the device
- Add a suitable entry in the `/etc/disktab` file to enable a filesystem to be created on the device
- Add a suitable entry in the `/etc/fstab` file for mounting the new device onto your system

Creating suitable entries in the /dev directory

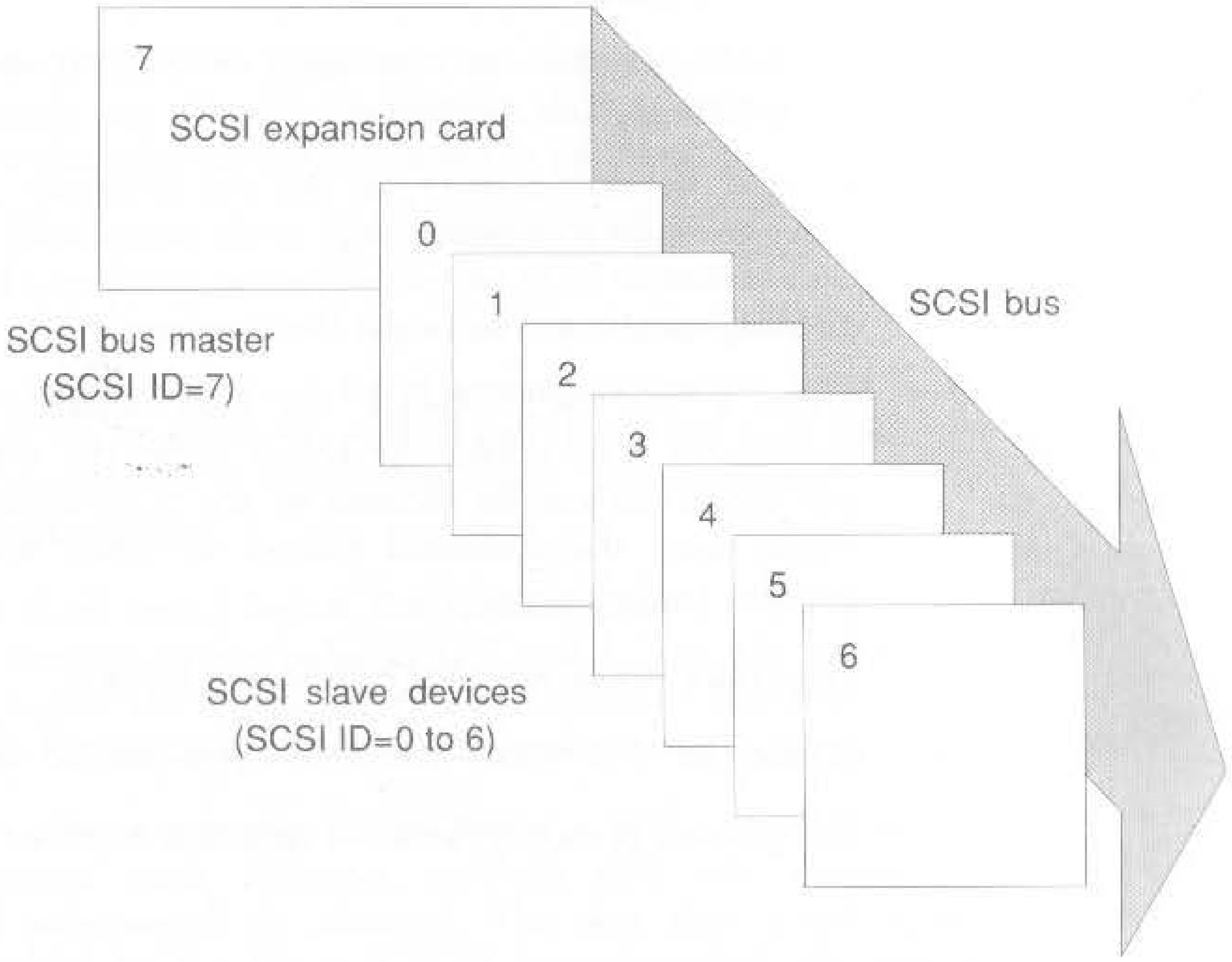
When a new SCSI device has been attached to your system the first requirement is to inform the kernel where the new device is to be found, so that the device can be accessed by the SCSI device driver in the kernel. This entails using the command `mknod` to create suitable entries in the `/dev` directory 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 options 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 the number 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.

Therefore, if you are attaching a SCSI hard disc device and this is the first SCSI device you have attached, then you should choose the device name `/dev/sd0`.

To confuse matters slightly, each of the seven devices (0-6) may internally support a further eight SCSI devices. So in theory, there are potentially 56 devices that can be attached to the SCSI bus.

- `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 and tapes that write data as a string of bytes. Character devices include printers and terminals 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 your hard disc: 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** and 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:

MSB → LSB

L	L	L	E	D	X	X	X

`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`=partition number

For a sequential access device, the first two most significant bits are ignored. The third least significant 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 (`sd0a` to `sd0h`)

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 (RODIME RO3000S). The device is assumed to be connected to a SCSI expansion card located in the first SCSI slot:

```
mknod /dev/sd0a b 12 0
mknod /dev/rsd0a c 21 0
mknod /dev/sd0b b 12 1
mknod /dev/rsd0b c 21 1
mknod /dev/sd0h b 12 7
mknod /dev/rsd0h c 21 7
```

The above commands all create suitable entries in the `/dev` directory for the device named `/dev/sd0` ie SCSI ID=0.

The first and second `mknod` commands specify a block device and character device entry for the SCSI device partition 0, with major number 12 and minor number 0 (block device) and major number 21 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 2, with major number 12 and minor number 1 (block device) and major number 21 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 7, with major number 12 and minor number 7 (block device) and major number 21 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 an `n` is added. For example:

```
mknod /dev/ct0 b 12 1
mknod /dev/rct0 c 21 1
mknod /dev/ct0n b 12 0
mknod /dev/rct0n c 21 0
```

The first two `mknod` commands, specify a block device and character device entry for the cartridge tape drive `/dev/ct0` with major number 12 and minor number 1 (block device) and major number 21 and minor number 1 (character device).

The last two `mknod` commands create similar block device and character device entries, but specify them for the cartridge tape drive with no rewind on close.

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 `mknods` 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, 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 format and initialise the disc for use as a UNIX filesystem.

The sequence of activities for initialising a SCSI hard disc device, using `scsidm` is:

- format the disc
- verify the disc

- partition the disc
- generate an example entry for the disc, to be included in `/etc/disktab`

`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.

Format and verify the disc

1 Start the formatter program by typing `scsidm`. Once the program starts, you will see the prompt:

```
scscidm>
```

2 Define the device name of your SCSI disc by typing:

```
scsidm>device device-name
```

For example:

```
scsidm>device sd0h
Current device is /dev/rsd0h,type 0(direct access)
Device identifies itself as a RODIME  RO3000S 2.2
```

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 typing `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.

4 Once the disc is formatted, it should be verified by typing:

```
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.

5 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
```

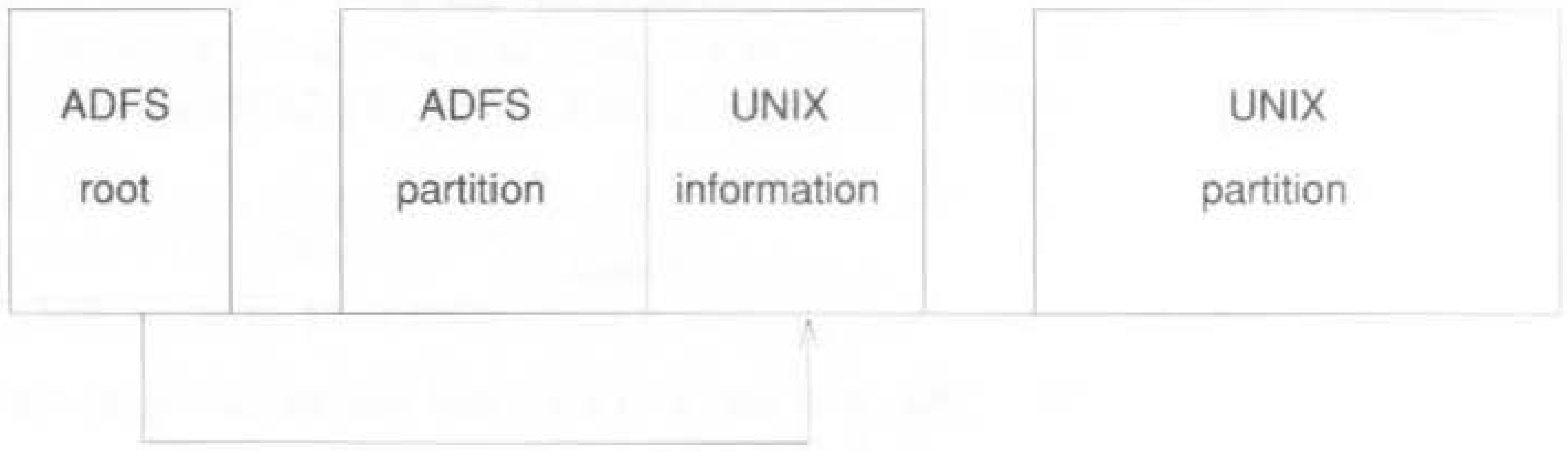
The disc can now be partitioned for use with RISC iX. There are further commands available that allow you to remap out and redefine any subsequent bad blocks that may occur on your disc.

A full explanation of all of the formatter commands is given in the manual page `scsidm`(8).

Partition the disc

Partitioning the hard disc means dividing the total disc area into several logical partitions that can be used for various purposes. For example, one partition can be used for storing ADFS files and other areas can be used to contain various parts of the UNIX filesystem in separate partitions.

The overall layout of the disc after it has been formatted is as follows:



Information about any partitions that you create on the disc are held in the form of a partition table in the 'UNIX information' area of the disc. This partition table is also replicated between disc partitions that are created on 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 disc so that most of the area is given over to UNIX and the minimum amount is given to ADFS, type:

```
scsidm>pa
partition
Include UNIX partitions ? (yes) >yes
Device /dev/sd0 has 18836 logical blocks, each 512 bytes long

ADFS partition size (min 200 blocks)> ↵
Rounded no. of blocks in ADFS partition up to 208
Writing ADFS partition ... done
Define which partitions? >
```

The above sequence of commands firstly invokes the `partition` option from `scsidm` and asks if you wish to include UNIX partitions. Typing `yes` causes `scsidm` to partition the disc according to the previous diagram. Thus a minimum ADFS partition is created and the remainder of the disc area is given over for use with UNIX.

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 6 as partition 7 is used to refer to the entire disc area given over for UNIX and is automatically written to the partition table.

The following example shows how to divide the disc up into two partitions, 0 and 2, corresponding to the sample `mknod` entries that were created for the device:

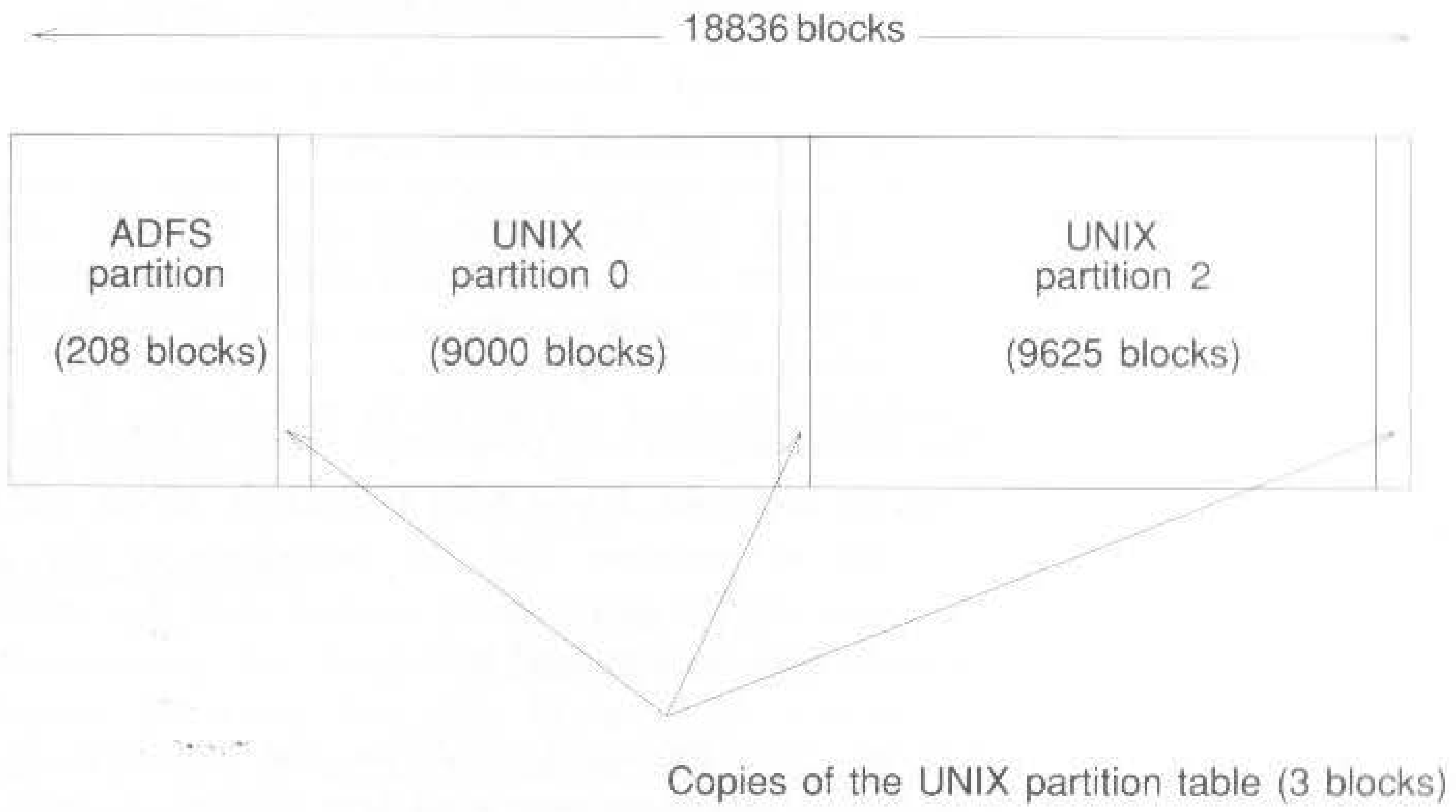
```
Define which partitions? >0,2
UNIX starts at block 209, and is 18626 blocks long

Starting block for partition 0, (209) ? > ↵
The maximum allowable length of this partition is 18625 blocks
Length of partition 0 ? >9000

Starting block for partition 2, (9210) ? > ↵
The maximum allowable length of this partition is 9625 blocks
Length of partition 2, (9625) ? >9625
Writing UNIX partition table ... done
```

The previous sequence of commands specifies two partitions to be placed on the disc area given over to UNIX. Note that the total length of the disc has decreased by three blocks. One block is used at the start of the UNIX partitions to hold the partition table and an extra block is used at the end of each partition specified to contain another copy of the partition table.

For example:


18836 blocks
ADFS partition (208 blocks)
UNIX partition 0 (9000 blocks)
UNIX partition 2 (9625 blocks)
Copies of the UNIX partition table (3 blocks)

If more partitions were specified, the end of each partition would contain another copy of the partition table.

After completing the partitioning of the disc, the partition table and a simple diagram showing the partitioning of the disc is displayed. Use the form supplied in the *SCSI User Guide* to copy the partition table details and keep them as a record.

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.

Also, check if there are any other entries for the device in this file. If there are then these entries should be deleted.

At this stage it is possible to use `mkfs` or `newfs` on the individual disc partitions. These will create UNIX filesystems on the partitions, which are then suitable for mounting.

Note, never create a UNIX filesystems on the 'all' partition (`sd?h`) as this will destroy all the UNIX partition information as well as the ADFS partition on the disc. For more information, refer to `mkfs`(8) and `newfs`(8).

If the SCSI disc partition is 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 Microsystems 386i machines, Xenix systems etc.
  - Acorn ADFS compatible machines
  - MS-DOS compatible machines
- Expanding the storage capacity of your system by using floppy discs as mountable UNIX filesystems.

Some of the utilities described in this chapter are also described in the chapter entitled *Communicating with other systems and users* in the *RISC iX User Guide*. So for a more gentle introduction to using floppy discs, refer to this Guide first.

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	Master Compact ADFS format
`/dev/rfdf512`	512K bytes/sector	MS-DOS 720K format
`/dev/rfdf1024`	1024K bytes/sector	Archimedes 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 Although there are some exceptions, notably Apple 3.5" discs.

However, most of the time you will probably only use the *Archimedes* workstation ADFS 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* 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`(1) to reset the parameters for these variable-specification formats. For more information, refer to `flpop`(1).

Formatting discs

Brand new discs, or discs which have been written on 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 workstaions as they do not expect to find any information about file structure on the disc.

To format a disc in the standard ADFS-style D format, insert the disc into the drive (having made sure that the write-protect slide tab is covering the hole), and type:

```
ffd 1024↵
```

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 machines

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 machine and another different type of machine. You can only transfer binary files betweeen two RISC iX workstations.

Transferring information to and from other UNIX machines

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 previously described. 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 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 onto 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 *Archimedes* machine, 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 an *Archimedes* machine.

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 from an *Archimedes* machine 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 UNIX filesystems

To use floppy discs as mountable UNIX filesystems you need to do three things:

- format the disc
- create a UNIX filesystem on the formatted floppy disc by initialising the file structures and the root directory on the floppy disc
- mount the disc in an appropriate directory on the filesystem.

Format the disc in ADFS-style D format using the command `ffd` as previously described.

Creating a UNIX 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 reccommended that you use `newfs` for creating a UNIX 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 block device, `/dev/rfdf1024` is used because the raw device limits transfers to multiples of 1024 bytes and `mkfs` uses smaller transfer sizes. 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 ommitted 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 UNIX 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/st0a      34983   32010    2623    92%    /
/dev/rfdf1024  663        10     586     2%    /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 UNIX 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 firstly `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).

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 previous chapter. 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:/usr/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.

/usr/spool/uucp

Login as `uucpsu` and change directory to `/usr/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 `/usr/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 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]...
#
, /usr/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 happenning. This should show up any errors in the expect/send sequence. Put a sample file in `/usr/spool/uucppublic`, give it public read, and queue it for transfer. For example:

```
% uucp -r /usr/spool/uucppublic/test remhost\!/usr/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 userid. 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
```

Miscellany

UUCP writes logfiles in `/usr/spool/uucp` and may accumulate abandoned files in its spool directories, so it needs a regular tidy. There is a script for doing this in `/usr/lib/uucp/clean.daily`, called every night by `cron`.

If your hostname is longer than seven characters you will need to replace references to `uuname -l` with the seven character truncation because `uuname` gets it wrong.

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).

Bibliography

Here is a list of books that you should refer to for further information about System Administration on UNIX systems:

1 *UNIX System Administration* – David Fielder and Bruce H. Hunter, Hayden books (1986)

2 *UNIX System Security* – Patrick H. Wood and Stephen G. Kochan, Hayden books (1986)

3 Berkeley 4.3 BSD System Manual Set, which comprises seven volumes:

- *User's Reference Manual* (URM)
- *User's Supplementary Documents* (USD)
- *Programmer's Reference Manual* (PRM)
- *Programmer's Supplementary Documents*, Volume 1 (PS1)
- *Programmer's Supplementary Documents*, Volume 2 (PS2)
- *System Manager's Manual* (SMM)
- *UNIX User's Manual Master Index*

The above manual set is available from the European UNIX User Group (EUUG), Owles Hall, Buntingford, Herts, SG9 9PL.

Reference Section A: The RISCiXFS module

Introduction

This reference section details the * commands supported by the RISCiXFS module that can be used in Supervisor mode from RISC OS.

All the above operations can be performed from the RISC OS Supervisor command line, or alternatively from the RISC OS Desktop, by using the maintenance menu from the !RISCiX application. This section describes the use of the module from the Supervisor command line. For information on using the module from the RISC OS Desktop, refer to the chapter entitled *Starting up and shutting down the system*, at the start of this manual.

For more general information about * commands and modules, refer to the *Archimedes 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 `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

`<swap>`	refers to the device partition that will be used for the swap area. This parameter also conforms to the naming convention `ddM(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 startup 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 the 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 st0` (the default value)

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 set the flag to 'Off' from RISC iX use the command:

```
reboot -RISCOS
```

This will shutdown RISC iX and bring the machine into RISC OS, irrespective of the *Configure noRISCOS setting.

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 '`/`,`/etc/`,`/bin/`' and can be modified at any time using the RISC OS `*set` command (see the *Archimedes User Guide* for details). 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 internal hard disc (`/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 default search path ',/,/etc/,/bin/' and can be modified at any time using the RISC OS '`*set`' command (see the *Archimedes User Guide* for details). 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:

`ddi`	s 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]`	make 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, with the Floating Point Emulator loaded, type:

```
execv ls /
```

RISCiXFS module error messages

'`RISCiXFS module not present or too old`'

The RISCiXFS module must be installed before the application is executed (this is to facilitate the 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. If the system does not boot correctly following this message, restart the system again.

'`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

'**`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 Menu. Select 'Task display' from the menu displayed. This produces a window containing details of the use of the computer's memory. Using Select, 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 $.Appl.!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.

Reference Section B: PostScript printer filter

Introduction

The following C program is a filter that is used in connecting your RISC iX workstation to a LaserWriter. For more information, refer to the section entitled *Printers* in the chapter *Attaching peripheral devices*, earlier on in this Guide.

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` and is sent directly to the printer without any processing.

```
/* >c.psfilter - expands tabs to spaces */

#define point 10
#define chrwid ((point*3)/5)
#define physhyt (797-point)     /* got from the LaserWriter clip path */
#define physwid (560-20)        /* but it misses right side of A4 paper */
#define MAXLINE 200

#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <signal.h>
init()
{
  printf("%%!\n/Courier findfont %d scalefont setfont", point);
  initpage();
}

windup()
{
  putchar('\004');
  putchar('\n');
}


int tabspace = 8;
int line = 0, page = 0;
int inshow = 0, col = 0;
char tabstops[MAXLINE];
int pwidth = 86;                /* default line length */
int plength = 66;               /* page length */
int indent;                     /* indentation length */
int literal;                    /* print control characters */
char *name;                     /* user's login name */
char *host;                     /* user's machine name */
char *acctfile;                 /* accounting information file */
int wide, length, width, paghyt, pagwid;

initpage()
{
  if (wide) puts(" 20 830 translate -90 rotate");
  else puts(" 20 20 translate");
}

main(argc, argv)
  int argc;
  char **argv;
{
  int c;
  register int i;
  register char *cp;

  while (--argc)
  {
    if (*(cp = *++argv) == '-')
    {
      switch (cp[1])
      {
      case 'n':
        argc--;
        name = *++argv;
        break;

      case 'h':
        argc--;
        host = *++argv;
        break;

                       case 'w':
        if ((i = atoi(&cp[2])) > 0 && i <= MAXLINE)
          pwidth = i;
        break;

      case 'l':
        plength = atoi(&cp[2]);
        break;

      case 'i':
        indent = atoi(&cp[2]);
        break;
        case 'c':           /* Print control chars */
          literal++;
          break;
        }
      }
      else
        acctfile = cp;
    }

    if (pwidth <= 86)
    {
      pagwid = physwid;
      paghyt = physhyt;
      wide = 0;
    }
    else
    {
      pagwid = physhyt;
      paghyt = physwid;
      wide = 1;
    }
    length = (paghyt/point-5);
    width = (pagwid/chrwid);
    settabs(tabstops);
    col = 1;
    line = 1;
    page = 0;
/*
    printf("%%!\n/Courier findfont dup setfont (%s) stringwidth", name);
    printf(" 750 exch sub 2 div 50 exch moveto 500 exch div");
    printf(" scalefont setfont (%s) show showpage\n",name);
*/

    c = getchar();
    if (c == '%')
    {
      c = getchar();
      if (c == '!')
      {
        putchar('%');
        putchar('!');
        while ((c = getchar()) != EOF)
          putchar(c);
        windup('\004');
        exit(0);
      }
      else
      {
        init();
        process('%');
      }
    }
    else
    init();

  while (c != EOF)
  {
    process(c);
    c = getchar();
  }
  if (inshow || line != 1)
    newpage();
  windup('\004');
  exit(0);
}


process(c)
{
  switch (c)
  {

  case '\t':
    do
      translate(' ');
    while (!tabpos(col, tabstops));
    break;
  case '\n':
    newline();
    break;
  case '\f':
    newpage();
    break;
  case '\b':
    endshow();
    if (col)
      col -= 1;
    break;
  default:
    translate(c);
  }
}

translate(c)
  int c;
{
  if (!inshow)
  {
    if (c == ' ')
    {
      ++col;
      return;
    }
    printf(" %d %d moveto\n(", 30 + col * chrwid, paghyt - point * (line + 1));
    inshow = 1;
  }
  if (c == '\\' || c == '(' || c == ')')
    putchar('\\');
  putchar(c);
  ++col;
/*
  if (col == width + 1) newline();
*/
}

endshow()
{
  if (inshow)
    printf(") show\n");
  inshow = 0;
}


newline()
{
  endshow();
  line += 1;
  if (line == length + 1)
    newpage();
  col = 1;
}

newpage()
{
  endshow();
  page += 1;
/*
  printf(" %d %d moveto (Page %d) show showpage",
         pagwid / 2, paghyt - point*(length + 3), page );
*/
  printf("showpage");
  line = 1;
  col = 1;
  initpage();
}

tabpos(col, tabstops)
  int col;
  char tabstops[];
{
  if (col > MAXLINE)
    return 1;
  else
    return tabstops[col];
}

settabs(tabstops)
  char tabstops[];
```

```
{
  int i;

  for (i = 1; i <= MAXLINE; i++)
    tabstops[i] = ((i % tabspace) == 1);
}
```

Reference Section C: Serial port connections

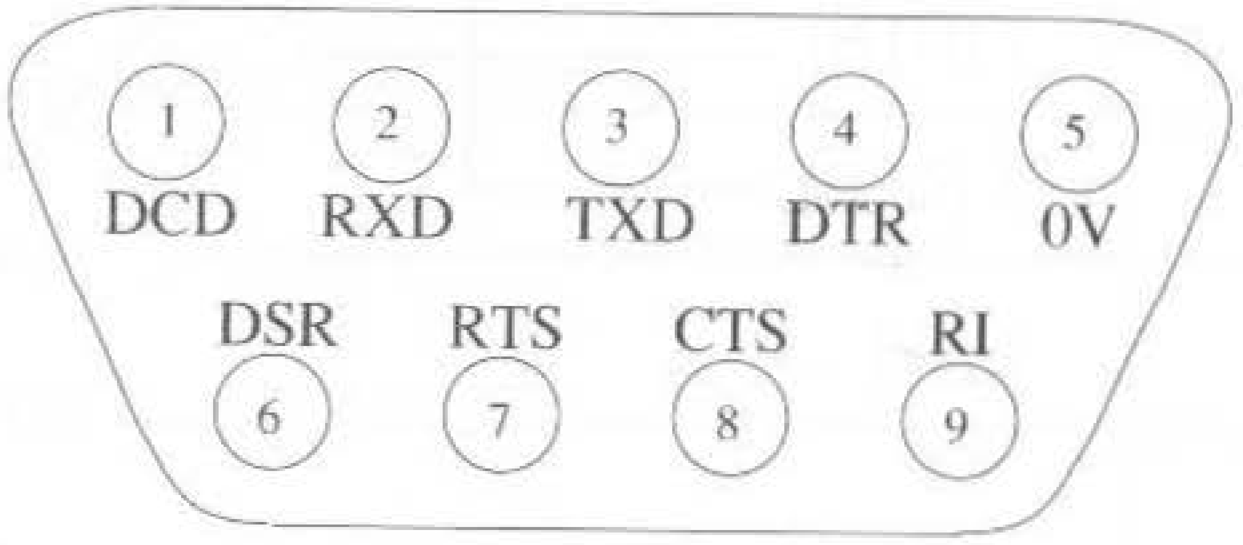
Introduction

This reference section defines the signal port connections for connecting various specific peripheral devices to the serial port of a RISC iX workstation.

If the device that you wish to connect to the serial port is not included then you should refer to the chapter entitled *Attaching peripheral devices*, which contains general guidelines about attaching devices to the serial port.

Serial port pin assignment

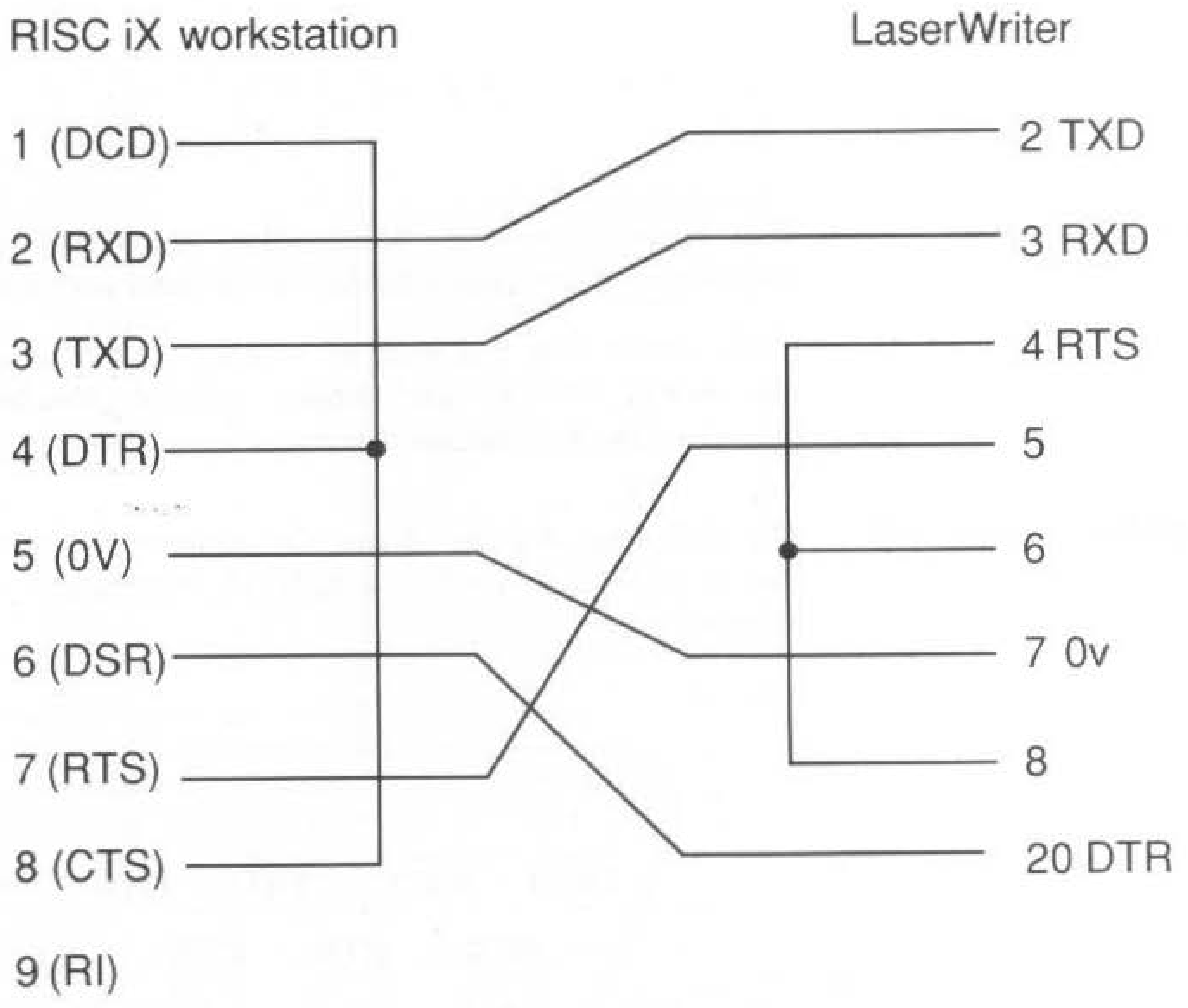
The following diagram shows the assignment of the pins on the serial port that is to be connected to a RISC iX workstation, viewed from the side that is to be soldered:



This view also corresponds to the view of the serial port socket from the rear of the RISC iX workstation. The pin assignment of 9-pin serial ports on other hardware is often the same as this.

LaserWriter signal connections

The following diagram shows the serial port signals that were used to connect a LaserWriter to the serial port of a RISC iX workstation.



The corresponding filter that is also needed is listed out in *Reference Section B – PostScript printer filter*. For more information about connecting other types of printers, refer to the chapter entitled *Attaching peripheral devices*.

25-way modem signal connections

The following diagram shows the serial port signals that used to connect a standard RS232 25-way female D-type socket modem to the serial port of a RISC iX workstation.

RISC iX workstation (9-way)	Modem (25-way)
Female D-plug	Male D-plug
1 (DCD)	8 (DCD)
2 (RXD)	3 (RXD)
3 (TXD)	2 (TXD)
4 (DTR)	20 (DTR)
5 (0V)	7 (0V)
6 (DSR)	6 (DSR)
7 (RTS)	4 (RTS)
8 (CTS)	5 (CTS)
9 (RI)	22 (RI)

For more information about connecting other types of modems, refer to the chapter entitled *Attaching peripheral devices*.

Hazeltine 1500 terminal pin connections

The following diagram shows the serial port pin connections for connecting a Hazeltine 1500 terminal to the serial port of your RISC iX workstation

RISC iX workstation	Hazeltine 1500 terminal (25-way)
1 (DCD)	4
2 (RXD)	2
3 (TXD)	3
4 (DTR)	5 & 6
5 (0V)	7
6 (DSR)	20
7 (RTS)	8
8 (CTS)	20
9 (RI)	Not connected

For more information about connecting other types of terminals, refer to the chapter entitled *Attaching peripheral devices*.

Reference Section D: Manual pages

Introduction

The following reference section contains the manual pages that have been referenced in this Guide and are suitable for System Administrators.

Other manual pages referenced in this Guide are available on-line or in the *Berkeley 4.3BSD System Managers Manual.*

The manual pages iuncluded in this section are:

```
ac
cron
dump
fastboot,fasthalt
ffd
fsck
getty
halt,reboot
init
mkfs
mknod
mount
rc
restore
sa,accton
scsidm
shutdown
sticky
sync
tunefs
useradmin,groupadmin
vipw
```

NAME

fastboot, fasthalt – reboot/halt the system without checking the disks

SYNOPSIS

/etc/fastboot [*boot-options*]
/etc/fasthalt [*halt-options*]

DESCRIPTION

Fastboot and *fasthalt* are shell scripts which reboot and halt the system without checking the file systems. This is done by creating a file */fastboot*, then invoking the *reboot* program. The system startup script, */etc/rc*, looks for this file and, if present, skips the normal invocation of *fsck*(8).

SEE ALSO

halt(8), reboot(8), rc(8)