UNIT 4 SYSTEM ADMINISTRATION

Structure

4.0 INTRODUCTION

While you might not need to know much about system administration if you are working at a large installation where those tasks are taken care of by a specialist, it is quite likely that you have access to a small UNIX computer, where you will have to be the user and system administrator put together. We will look at some of the common tasks a system administrator is called upon to perform, and also look at some concepts which we have put off for this unit.

4.1 OBJECTIVES

This is the last unit of this block and by now you know about many facilities available in UNIX. At the end of this unit you should be :

- Knowing how a UNIX computer starts up and is shut down
- Aware of how to set the date and maintain user accounts
- Familiar with device files and the UNIX file system structure
- Well versed with backup and restoration procedures

4.2 SYSTEM ADMINISTRATION - A DEFINITION

Throughout this block on UNIX, we have been talking of the System Administrator. Let us now see what system administration means and what it entails. We will then see how to perform some of those tasks.

Our discussion of system administration will be somewhat less specific than the other commands we have talked of earlier, because the exact methods of doing some of the tasks are heavily dependent on the computer you have. System administration refers to all the administrative tasks involved in keeping a computer running. These involve things like creating new users, deleting user accounts for persons who are no longer authorised to login, starting up the system, making sure that there is enough space to work on the disks, taking backups, shutting down the system whenever needed, helping ordinary users who are in trouble (like if somebody has forgotten his password) and many other such things.

Now in the older days when computers were large and expensive machines a computer installation was a large, organised setup, whether at a university or at a commercial site. This meant that there would be at least one person wholly responsible for system administration. So a novice or an ordinary user had only to obtain an account and login and work. He did not have to worry about things like keeping backups or making sure that the file system was consistent or that the system was shut down properly. But with the advent of the personal computer, the situation has changed completely. Many computers are now owned by small businesses or organisations and even by individuals. These computers are now powerful enough to be running UNIX and many of them do actually use it.

This has meant that at many installations there might not be a dedicated system administrator. For example if you have UNIX running on your own machine you will have to

be the system administrator too. Thus it is necessary for you to have some idea of what the job involves and how to do it. In this you will have to be guided largely by the documentation that comes with the UNIX you purchased. This section will serve to give you a preliminary idea of the subject and enable you to use the documentation. While we have always been saying this throughout this block, in this matter this statement has much more importance. Differences among different versions, implementations and vendors, as well as because of different hardware are far too pronounced in this area, much more than in the rest of the UNIX commands.

System administration needs you to use some utilities and commands that other users never need. Such commands are often put in the /etc directory so that ordinary users do not invoke them. In any case almost all administrative tasks have to be done by the super user. As we said in the first unit, the super user is a special kind of user who can do any operation on any file irrespective of its permission modes. He can login to any account and kill any process. Thus none of the usual protection schemes apply to him. Therefore in this section we will assume that you are logged in as the super user. The default super user prompt is a #, which is why the examples here use that prompt instead of the usual shell prompts.

Because a super user has such sweeping powers over the machine, many people are tempted to always work as super user, or root, when they control the installation. Thus if you are working at home on your own computer, you might feel like working as root. This feeling usually comes in because of the power you then enjoy over the installation, and also for convenience because as an ordinary user you might not be able to do all the tasks you want to. At a larger installation you would not be able to do so because the root password would be zealously protected and ordinary users would probably not know it. Certainly no novice would be given the super user password.

It is a good idea to create an ordinary account for yourself and work there even when you control an installation. This is because as super user you have a lot of power. A small slip could damage the installation irretrievably and take a lot of effort to repair. No protection applies to the super user and many times the system does not ask for confirmation when the super user invokes a command, while it might ask an ordinary user to do so. Remember that the super in super user refers to powers on the installation, not that that person is super human. UNIX commands are written with the assumption that the super user knows what he is doing and will not make any mistakes, but this hypothesis is not really true in practice. A super user can also make mistakes. So you should assume those powers only when you need them.

Also as super user if you create files or directories, you might not be able to use them as an ordinary user. So it is best to be an ordinary user for normal tasks and become root only when needed.

Apart from the large number of ordinary users that usually exist on a system there are also some system users that UNIX needs. Some of these are bin, uucp and sysadm. Not all system accounts are super user accounts and actually most of them are not super user. System accounts usually need to have user ids below 100. Any account with a user id of 0 is a super user and has all the privileges of a super user. The user with user id 0 and group id also 0 is traditionally called root.

4.3 BOOTING THE SYSTEM

As you all know, booting is the process of starting up a computer. It is so called because the job involves something akin to lifting oneself up by one's bootstraps, that is, from a system which is off you need to have a system running a complex operating system like UNIX. This task is done in stages, with a very simple program in the computer's ROM which runs and loads a program on say, the hard disk, which is then able to load the whole of the UNIX kernel.

Of course the UNIX kernel and other utilities cannot reach the hard disk by themselves. A system administrator would do well to know how to set up his system from scratch. On a small system this is usually possible from floppies, while on a larger computer, this task could be done from a cartridge tape or a CD- ROM or a digital audio tape, which is what the vendor supplies it on.

But let us first see what happens when you start up the computer. We will not be able to be

very specific here because the exact steps to follow depend very heavily on the make of the hardware and the vendor from whom you purchased your UNIX. So we can only talk of the actions in general. The boot program is usually located in the block zero of the boot device. This should not be confused with the absolute block zero of the hard disk. This program is loaded and run by a program in the ROM of the computer and gives a message akin to the following

Enter kernel to boot

Here the boot program is asking you which file on the disk contains the UNIX kernel program you want to run. Usually this file is called UNIX and is located in the root device. But you can have any other filename which you want to use to start up. Typically this is done because you have configured a somewhat different kernel tuned to be useful in different situations. Then you can specify that filename. Sometimes the kernel is assumed to be in UNIX and you have to take special action if you want to boot from a different kernel.

You do not always need to be starting up UNIX. There will be ways in which you can start up some special program, for example, a diagnostic utility to check out the hardware. You will need to refer to your system's documentation for learning how to do this. In fact, on many systems, you will have to be specifying where the kernel program resides, apart from giving its name. This allows you to boot from a different physical disk if the usual one has malfunctioned. These matters again are highly system dependent.

After telling the system which kernel is to be loaded, the machine starts running that kernel and first brings you to a system state known as the single user state. In this state only one user can be working on the machine and only the console is active. No multi user capability will exist at this time. The person at the console will be super user and will have all the powers of one. This mode is also called system maintenance mode because it is only in this state that some system actions can be performed.

It is usually in this state that the system administrator sets the system date. Usually the date is maintained with the help of a real time clock which derives its power from a battery so that it does not lose its information even if the power is switched off. Thus the date information should normally be correct even if you are starting up the system after a prolonged shutdown. However, if there is some error in the date, it can be set now. It is a good idea not to set the system date when it is in multi-user mode because many UNIX utilities can get confused if the date changes midway. In the single user state you are sure that nobody else is working. You have already seen that the date command without any arguments displays the current system date. To set the date, say

date 950328104533

where the first two digits are the year, the next two are the month, the next two the day in the month, followed by the time in hours, minutes and seconds. There are some variations to the format in which the date can be set, so please do refer to your documentation.

You are now ready to enter multi-user state. The transition from one state to another is achieved by running a program called init. This provides for 7 run levels from 0 to 6, and an s or S state for single user mode. The main multi-user mode is level 2, and so when the system asks you

Enter new run level

you should enter 2 to reach normal multi-user mode. To see what the other levels mean, you can refer to the documentation. Run level 0 is the down state and readies the machine for powering off. The who command with the -r option tells you the current run level of your system, as you have seen in the second unit.

The devices and users available in each run level are controlled by a file called /etc/inittab and it also specifies what action is to be taken when a run level is entered. For example if you want to enable a new terminal, say tty06 (assuming the hardware is available), you have to make an entry in the inittab file. Similarly you can disable a terminal serial port even if it is physically connected, by making a change in the inittab file. On some versions of UNIX there will be front end utilities to enable you to do these things easily.

Once you reach run level 2, all the terminals which are switched on and are connected to the computer will display the usual login prompt. It is now possible for users to login and work on these terminals. But the system does several things before it shows this prompt. Let us

have a brief look at the different processes involved in reaching multi-user state.

When going to multi-user state the system runs a shell script called /etc/rc, and you would do well to examine this file on your computer. This will tell you a lot about what is done in this transition. We will here mention a few things that probably do get done on your machine, but the list is neither exhaustive nor are all actions we mention essential.

The script will probably remove the /etc/mtab file, which is later used by the mount command to mount the various file systems. We will look at file systems and the mount command a bit later. This step is essential, because when you reach the single user state only the root file system is mounted. The script will also empty out the log of users held in /etc/utmp.

Once this is done, the system will mount the various file systems and clear out the /tmp and /usr/tmp directories. Thus you can see that the contents of these directories get erased whenever the system is started up, and so it is not safe to store anything of value here. In your normal working you should put only things which you are prepared to lose without warning in these directories. The system then starts up various daemons (background processes) like cron.

Finally the system displays some message on the console, maybe the date or some other installation specific greeting. At this point the login message appears on the other terminals and the machine is up for use by ordinary users.

4.4 MAINTAINING USER ACCOUNTS

In the second unit itself, you have seen how users can login into the system and how passwords are used to prevent unauthorised access. You also saw how one or more users could be associated with a particular group. When you studied the long listing provided by the -l option to the ls command, you saw how the user name and group were also given for each file in the listing.

You also know that an ordinary user cannot help you if you have forgotten your password, and that only the super-user can set somebody's password without already knowing what it is. One of the duties of the system administrator is in fact maintaining user accounts. This activity encompasses creating new accounts, deleting accounts of users who are no longer permitted access to the site and helping people who might have forgotten their passwords. On many large installations the system administrator will run regular checks on user passwords to make sure that they cannot be guessed easily.

Some UNIX implementations have more sophisticated password mechanisms. Here UNIX itself will impose restrictions on the password chosen by an ordinary user and will disallow choices which do not meet its standards. Some of the checks usually made are that the password must not be a circular shift of the login name, that it must not be shorter than 6 characters and that it must contain at least two special characters. Sometimes a password aging mechanism is also enabled. This imposes a restriction on the minimum and maximum number of weeks that a password for an ordinary user will be valid. If a user does not change his password for the specified duration, he will be told that his password has expired the next time he tries to login. He then has to enter his old password and change it before he is presented with the prompt. This check is based on the premise that if a password is not changed for a long time, it might get compromised. If a user tries to change his password too soon, he is not allowed to do so. This check assumes that an intruder will want to change the password as soon as he gains access (for example, because a careless user has left his terminal unattended).

With all this background, let us see just how user accounts are maintained. The information about a user account is maintained in a file called /etc/passwd. This file looks like this

```
# cat /etc/passwd

root:kWFpmBP9vvKr2:0:0:System Administrator:/:/bin/sh

bin:NOLOGIN:3:3::/:/bin/sh

khanz:Nz1gLj157en8c:200:200:Zafar Khan:/usr/khanz:/bin/csh
```

You will have been able to decipher some of this file. Well, the passwd file contains a one line entry for every account. The different fields in this line are separated by a colon (:). The first field is the login name of the user. This is the name by which the user will be known on the system. The second field is the encrypted password of the user. This is how passwords can be stored on the system and still not be intelligible to anybody. Any user can read the passwd file but this will not help him find anybody's password, at least not easily.

Notice that the encrypted password is always 13 characters long. So if the password is of any other length then it can never be a valid password. This fact is often used in the passwords of system accounts like bin and uucp. A password like NOLOGIN means no ordinary user can login as that account. For the super user, on the other hand, it does not matter because he can login as anybody as the password is not checked in that case.

The third field is the user identification number, or user id, or uid. The system works in terms of the user id rather than name for most purposes. Any user with an id of 0 is a super user, and ids below 100 are usually reserved for the use of UNIX. Ordinary users get ids from 100 onwards. Now we will discover an interesting fact. Suppose you login as an ordinary user, khanz, who has a user id of 200. Create a file /usr/khanz/checking, logout and login as root, and look at its long listing

```
# cd /usr/khanz

# ls -l checking

-rwxr-xr-x  1 khanz  crypt 48  Mar 14 03:32  checking
```

Now edit the passwd file carefully and change khanz in the first field to smith. Look at the listing of the file checking again

```
# ls -l checking

-rwxr-xr-x  1  smith  crypt  48  Mar 14 03:32  checking
```

You will see that ls now reports smith as the owner. This shows that the ls command uses the passwd file to translate the user id to the user name. If you delete the account of khanz by removing his entry in the passwd file, you will see

```
# ls -l checking

-rwxr-xr-x  1  20020048  Mar 14 03:32  checking
```

This is because now ls is not able to find out who the owner is and therefore reports the user id instead. Do put back khanz's account under his own name now!

The fourth field is the group identification number or group id or gid. The file /etc/group contains the group names corresponding to the group ids, and also contains information on which users belong to which group. This file is also used by the ls command to translate group ids to group names, just as it does with the user names.

The fifth field is a comment field containing upto 30 characters, usually the full name and other information about the person. The sixth field gives the home directory of the account and thus determines where he will reach when he logs in. The last field gives the shell he will be presented with when he logs in. This field can be omitted and then the default is the Bourne shell.

Other fields can also be omitted. If the password field is left out, the account has no password and anybody can login by just typing that login id. The system will not ask for a password because there is none. Obviously such a situation is dangerous and in any major installation the system administrator will ensure that all accounts have passwords. By the way an ordinary user can set his password to a carriage return but cannot remove it. The comment field can also be omitted.

You are now ready to learn how to add a new user to the system. Actually most UNIX versions will have a special sysadm shell to perform the common system administration functions. The script to add a new user is called adduser and sometimes there is a special account for sysadm alone. But here we will look at what needs to be done to add a user rather than just use a front end.

Essentially, to add a new user account, you have to create an entry for him in the passwd file,

create his home directory with the appropriate permissions and create an entry for him in the file /etc/group if needed. To add an entry to the passwd file, what is usually done is to edit the file, make a copy of the last line and change the entries as needed. Thus in the passwd file shown earlier, simply duplicate the line containing khanz and change each field as needed. This is easier than trying to add a new line from scratch. For a new user, leave the password field blank so that when he first logs in, he has no password. A user should thus set his password immediately after logging in for the first time. Alternatively, to be safer, create his entry, and set his password for him. Tell him his password and ask him to change it soon. The passwd file will now look like this

```
# cat /etc/passwd
root:kWFpmBP9vvKr2:0:0:System Administrator:/:/bin/sh
bin:NOLOGIN:3:3::/:/bin/sh khanz:Nz1gLj157en8c:200:200:Zafar Khan:/usr/khanz:/bin/csh
kumarr::201:200:Ram Kumar:/usr/kumarr:/bin/csh
```

You also have to create the new user's home directory. So

```
# mkdir /usr/kumarr
# chown kumarr /usr/kumarr
# chgrp crypt /usr/kumarr
```

Remember that the super user can change the owner or group of any file, as well as its permission modes. Here the group crypt presumably exists already (because khanz belongs to it). But if kumarr belongs to some different group, an entry will need to be made in the group file. The group file looks somewhat like the passwd file

```
# cat /etc/group
root::0:root
bin::3:bin
crypt::200:khanz,kumarr
```

This file also consists of colon separated fields. The first field is the group name, the second (usually empty) is for holding the encrypted group password, the third gives the group id and the last field is a comma separated list of all user names who belong to that group.

We will not consider group passwords here as they are rarely used. To add kumarr to the group crypt, just add his name to the tail of the list. You can open up a new group in much the same way as you would add a new user to the passwd file.

In the second unit you have already seen how an ordinary user can change anybody's password if he knows his current password, and also that the super user can change anybody's password because he does not have to enter that user's old password. Thus you can set the password of the new user you just created to anything you like. From this it is easy to see how even an ordinary user can masquerade as another if he knows that user's password. You could always logout and login as somebody else. But that is not usually necessary, because many commands and utilities look at the effective user id and not the real user id when they work. Even if you are logged in as khanz, you can switch to becoming kumarr if you know kumarr's password.

```
% su kumarr
Password: pi,14
```

Now if you create files they will have the owner kumarr. In fact, you may not be able to create files in your home directory because you are unlikely to have write permission there as kumarr. To get back to being your normal self, just type ^D. The super user can change to any other user if he wants, though he does not usually need to, because the su command does not ask him for a password.

When you change yourself to being another user, the new user's id is called your effective user id. If you have switched to super user, your effective user id is root. The su command has options, one of which allows you to run that user's login scripts as well, so that after the switch you are in his home directory rather than yours.

We will now learn about three special file modes. These are modes which can be set only by

the super user and they apply only to executable files. The first is called the "set user id" mode, and means that when even an ordinary user executes the file, the user id of the file becomes that of the owner of the file while it executes. The second mode is called the "set group id" mode which means that the group id of the file becomes that of the actual file whenever any user executes it. These situations are in contrast to the normal case where the user id and group id of an executable file are the same as those of the user (rather than that of the file owner) while the file is being executed.

The set user id bit is set using a fourth octal bit in the permission modes. Thus

```
# chmod 04000 /etc/passwd
```

will set the user id bit on the passwd file. Similarly the group id bit is set by setting the permission mode to 02000. In a directory listing, if the user id bit is set, the permission modes appear as -rwsr-xr-x. If the owner does not have execute permission on the file, the s is replaced by an S. The same letters appear in the group permission bits if the group id bit is set. So if both are set and the owner has execute permission on the file but the group does not, the permission modes will appear as -rwsr-Sr-x. By default neither get set because in a normal chmod command you do not specify the fourth octal bit at all.

Apart from this there is a mode which can again be set only by the super user

```
# chmod 01000 popular
```

This will make the swap space of the file popular reserved and will not reuse it even if nobody is running the program. This bit is therefore called the sticky bit. This mode is used for programs which are heavily used for execution as it then improves system performance.

You will be wondering why one might want to set the user id bit. Take a program like passwd, which allows anybody to change the password. The /etc/passwd file is read only for users other than root, so no ordinary user can write to it. How then can an ordinary user change somebody's or even his own password, considering that this means changing an entry in the passwd file? The answer is that the /bin/passwd program has the user id bit set and so whenever any user executes the passwd command, it runs with root privileges so that the user is able to alter the passwd file.

Before closing this section we will look at how important it is from a security point of view for system files to have the correct ownerships and permissions. If the passwd file could be written to by anybody other than root, any ordinary user could edit the file and blank out anybody's password. Thus there would be no security worth the name, as anybody can see. But what is sometimes missed out is the fact that the same situation will prevail if the /etc directory (owned by root) has write permission for others. Believe it or not, this author has seen such a thing in more than one installation from a reputed vendor. That was sometime ago and it is hoped that this has now been rectified. The point is that if as an ordinary user you can write to the /etc directory, then you can become super user anytime as follows

```
% cp /etc/passwd ~
```

Now edit the file passwd in your home directory and make any change you want to, like blanking out the root password or opening another account with super user privileges. Then

```
% mv passwd /etc
```

which will work because you can write to the /etc/ directory and can thus create files there. Now you have your own doctored passwd file on the system and can login as super user whenever you wish. Try it!

4.5 FILE SYSTEMS AND SPECIAL FILES

We have looked at two kinds of files so far, files and directories. In the second unit we had mentioned that UNIX considers everything to be a file. Thus hardware devices like terminals, tape drives, floppy drives and printers are all considered to be files in UNIX. What this means is that there is a filename associated with each device and you can read from or write to these files just as you would to an ordinary file. There is a device driver in the kernel for every device supported and the name of the file for the device points to it within the system. Thus when you perform an operation on such a device special file the device driver

takes over.

There are two kinds of special files, character and block special, which are associated with character oriented devices and block oriented devices respectively. A tape is a character oriented device while a hard disk can be both character or device oriented. Thus you will find that both kinds of special files exist for your hard disk on the system, but that there are only character special files for any tape drives. Actually there are also pipe special files but we will not consider them here.

The bridge between the physical device name (the filename) and the driver for the device is located in the /dev directory. Look at a long listing of this directory

```
# cd /dev; ls -l

crw—w—w- 1 root  root  0, 0  May 1 10:44 console

crw—w—w- 1 root  root  5, 0  Mar 3 15:39 rhd0

brw—w—w- 1 root  root  5, 0  Mar 3 15:45 hd0
```

This is a small portion of this listing, and of course since it is hardware and implementation dependent your listing could look somewhat different. We will look at some of the main features of this directory listing.

The first thing you must have noticed is that the first character of the permission modes is no longer a hyphen (-), but is c or b. This indicates whether the device file is character or block special. The other permission modes have their usual meanings, except that you cannot execute a device no matter how disgusting its behaviour might be. So execute permission has no significance here.

Instead of the file size you find two numbers separated by a comma. These are called the major and minor device numbers. A major number stands for a class of device like a terminal or a printer, while the minor number gives the number of that kind of device. Thus different terminals might have minor device numbers 0 (for tty00), 1 (for tty01) and so on.

An entry for a device can be added by the mknod command. Suppose your terminal major device number is 0 and you have 5 entries in the /dev directory from 0 to 4. You will not be able to activate another terminal even if the driver can support it unless the device entry is made

```
# cd /dev

# /etc/mknod tty05 c 0 5
```

The arguments to the command are the device name by which it will be known, the type (c for character and b for block) followed by the major and minor numbers. A device entry can be removed as usual with the rm command.

Your version of UNIX might have the /dev directory itself organised into several directories like /dev/rdsk for the hard disk as a character device, /dev/dsk for the hard disk as a block device and so on. This is for convenience because of the large number of devices supported, whereupon the number of entries in the /dev directory would become too large.

You must remember that merely making a device entry is not enough. The UNIX kernel must have support for that device. If you purchase a new device like a CD-ROM, it will come with a driver which you can install on your system. This will rebuild your UNIX code and put the device driver for your CD-ROM in your kernel. You will then be able to use your device. Thus the device entry is a link to the device driver.

We will now look at hard disks and see how they are utilised in a UNIX system. A file system is the complete hierarchy of directories and files starting from its root. A small device like a floppy cannot hold many files but a large (2 GB) hard disk can easily hold several file systems. It is common to partition a large physical hard disk into several logical disks, on each of which a file system can then be created. This is done by the mkfs command

```
# /etc/mkfs /dev/rdsk/0s2 200000
```

which here makes a file system on the disk of size 200000 1K blocks, or close to 200 MB. This command assumes that the logical partitioning of the disk is such that the device here

does contain at least that much space. If it contains more, then the extra space cannot be used by the file system and is wasted. The device is named as a character device before the file system is made. You can still access the disk as a character device, but then it would be very difficult for you to interpret the data stored on it, unless you know intimately the structure imposed by the file system. Creating a file system is naturally done only in system maintenance mode.

What happens when a file system is created on a disk? The disk could previously be accessed only as a character device, but now a structure is imposed on this. You can now take advantage of the fact that you can access blocks on the disk randomly without having to go through all previous blocks. So you can now look at the disk quickly. The file system structure consists of the boot block, the super block, the i-nodes and the data blocks.

The zeroeth block of every file system is reserved for storing booting information. It does not have any significance as far as the file system itself is concerned. It is the first block, called the super block, which contains information about the file system. Some of these items are the size of the file system, its name, the number of blocks kept apart for i-nodes, the list of i-nodes and the list of free blocks.

The index node or i-node blocks vary in number depending on the size of the file system and can actually be specified by the user while creating the file system. This number is the same as given in the super block. There is one i-node for every file or directory in the file system and contains information about the file. You cannot have more files and directories in the file system than the number of i-nodes. The remaining blocks in the file system contain the actual data in the files.

Each i-node contains information about the file like the permission modes and the number of links. Then there is the list of block numbers occupied by the file. The first ten such numbers are called direct blocks because they directly give the numbers of the data blocks on the disk. The eleventh number contains a block address, and that block holds the addresses of the actual blocks. These are thus indirect blocks. The twelveth number points to a block address, and that block contains the addresses of certain blocks. Each of these latter blocks holds the addresses of the actual data blocks. Thus the twelveth number points to the data blocks through two levels of indirection. Finally the the thirteenth number in the i-node contains the triple indirect blocks. Thus you can see that you can hold fairly large files in a UNIX file system.

Now when the system is booted, these file systems are not immediately accessible to users. This is because each file system has to be attached to the main file system. This is done by the mount command.

```
# /etc/mount /dev/dsk/0s2 /usr2
```

The first argument to the command is the device file associated with that file system. The second argument is the name of a directory. The mount command associates the logical device containing the file system with the directory. Now /usr2 becomes the root of the directory hierarchy on the device. So if you had a directory called khanz on /dev/dsk/0s2, you can now access it as /usr2/khanz. The directory /usr2 should be preferably empty before you mount some file system onto it, because while the file system is mounted you cannot access anything that was there on /usr2 previously. To break the link between the file system and the directory say

```
# /etc/umount /dev/dsk/0s2
```

Now you can no longer get at the files in that file system. The various file systems are usually mounted automatically when the system reaches the multi-user state through commands in the /etc/rc shell script. If you issue the mount command by itself, it will list all the file systems currently mounted and the directories to which they are attached.

If somebody is working on a file system, it would be disastrous to umount it because the file system would then become inconsistent when the device suddenly vanished. That is why you cannot umount a file system unless no user is accessing it. Umount will tell you that the device is busy and will not take any action.

In the booting process, a special root file system is mounted on the root directory (/). The major system directories like bin, etc and dev are under this directory. Since this file system is always in use, you cannot umount or mount it yourself. All the other file systems that you mount are below / in the hierarchy, as you know. Sometimes /tmp is mounted as a separate

file system.

You have seen the intricate structure of the file system. This structure is subject to disruption due to many reasons. For example a block on the free list might also appear attached to a file, or a block might appear on neither. An inode might appear duplicated or a file might seem to be under no directory. Such situations potentially mean the loss of data. Fortunately the file system contains a fair amount of redundant information, and this enables a program to check whether it is consistent. If it is not, it is usually possible to repair the damage. The UNIX system comes with a program called fsck which performs a file system consistency check.

This program is run before mounting the various file systems because if the system is inconsistent to begin with then it will get worse as users work on it. The fsck program looks for the file systems to check in a file called /etc/checklist. It always checks the root file system anyway. You can run it on a file system anytime

```
# /etc/fsck /dev/dsk/0s2
```

Every file system should contain a directory called lost+found, and this should be the first operation on it when the file system is made. Whenever fsck discovers a file not linked to any directory, it puts the file in this directory. The program works in 5 phases in each of which some checks are performed. When it discovers a problem, like a bad free list which it wants to salvage, it prompts the user for a y or n response. You can use the -y or -n flag while invoking fsck to ask it to assume that response for every question it asks. In that case it does not wait for user input before it proceeds.

When in operation, the file system is buffered in RAM, meaning that the file system information is maintained in RAM (core storage) and is flushed to the disk from time to time. Thus the file system on disk is not always in step with the information in RAM. Thus abruptly powering off the computer can badly damage the file system, sometimes beyond repair. This is different from the case with a single user system like MS-DOS.

The only safe way to prepare the system for powering off is to reach the down state

```
# init 0
```

This unmounts the file systems and does other tasks like shutting down system daemons. It then announces that the system is down and it is safe to power off. This command starts off immediately, and does not give time to anybody working on the system to make an orderly exit. Therefore there is usually a front end to this, called shutdown.

```
# shutdown -g120
```

This broadcasts a message using the wall command, telling the users that the system will shut down in 120 seconds. If no grace period is specified with the -g option, 60 seconds is assumed. The command then shuts down the system after the grace period is over.

4.6 BACKUPS AND RESTORATION

At a large installation, users do not usually have their own backups and the system administrator is responsible for the integrity of the system. On smaller installations and certainly on your own personal computer, you will be responsible for your own data and program files. So it is necessary that you know how to take backups of your work and how to restore it when needed. As a computer professional you do not need to be told how important backups are.

Backups are commonly taken with the cpio or tar commands, of which we will discuss only the second here. The tar command allows you to take a backup of all or selected files in a directory hierarchy onto tape, floppy disk or the hard disk itself. Tar knows about directories and links, and maintains headers, checksums and file permissions, owners etc. To take a backup of all files under /usr2/khanz

```
% tar cvkbf 1000 20 /dev/flp0 /usr2/khanz
```

will create a tar backup on the device /dev/flp0 (the 0th floppy drive) of everything starting from /usr2/khanz. Let us look at the meanings of the various letters after the tar command.

The c stands for create, and it causes tar to create the backup. Remember that anything

previously present on the backup device gets erased thereby. The v is the verbose option of tar, and makes it chatter about what it is backing up. The k is the multi volume option (might not be supported on your UNIX) and the corresponding 1000 is the size of the backup device in blocks. You are at liberty to specify a size smaller than the physical space available on the device. When tar comes to the end of a device it will prompt you to mount the next volume until the backup is complete. The b is the blocking factor in hexadecimal blocks. A low blocking factor will mean more space wasted on the device because of the large amount used up in the inter-block gap. The f indicates the filename on which the archive is to be placed.

Here we have an example of a backup taken using an absolute pathname. This is not usually a wise thing to do. Also, if you do not want to backup the whole tree, you can specify the files you want archived

```
% tar cvkbf 1000 20 /dev/flp0 /usr2/khanz/crypt /usr2/khanz/nlp
```

will backup everything under the crypt or nlp directories. If either crypt or nlp are a file, then only that file is backed up. Here we have given only two files to backup but you can give several.

The file on which the archive is created can be a disk file itself. This is useful in backing up a directory hierarchy onto a single disk file. To restore from a backup, use the x (extract) option to tar

```
% tar xvf /dev/flp0
```

While restoring you do not have to specify anything else. The block size and the size of the volume are known to tar from the header. By default it will restore everything on the volume, but if you want only specific files you can mention them. Then only those files will be restored. If the file you specify is a directory then everything under it will also be restored.

The user and group ids of the files are preserved while restoring. So if you backed up something with user id 208 and group id 104, standing for khanz and projects on the source system, the restored files will all have the same ids. This can be a problem if you are trying to transfer files from somebody's account or your own files from a different installation where your uid and gid might be different. To restore archives under the uid and gid of the person doing the restoration, say

```
% tar xvof /dev/flp0
```

The tar command also has a t option which lets you look at the contents of an archive. Thus

```
% tar tvf /dev/flp0
```

will list the files on that volume. If it is a multi-volume backup tar will tell you the number of that volume and the total number of volumes in the archive.

Check Your Progress

1. Turn off the computer abruptly when you and others are working on it. See how the file system consistency check is performed. Is it possible that some data gets lost?

   .....

   .....

   .....

2. Remove the device file for a terminal and see what happens. Also try renaming it.

   .....

   .....

3. What would happen if you tried to make a file system on a device which is (a) larger (b) smaller than the file system size?

   .....

   .....

4. You have backed up a directory containing several files onto say 12 floppies using tar. When you try to restore these after a crash, the 4th floppy is corrupted. How much do you lose?

.....

.....

.....

5. Add a blank line to the beginning of the password file. What happens? What if there is a blank line in the middle?

.....

.....

.....

6. How can you locate a file with more than one link? How can you locate all the links to such a file?

.....

.....

.....

4.7 SUMMARY

With this we come to the end of this unit and this block. While we had to leave out many things for lack of space, you should now be having some idea of how the UNIX system is set up. As always, you are urged to study the documentation which comes with your system and experiment as much as possible.

In this unit, we have had a look at system administration and what it entails. We have seen how a UNIX computer starts up and how it shuts down. We looked at the intricate UNIX file system structure and saw how it is vulnerable to disruption and how it can be repaired. Devices are files in UNIX and we saw how the mechanism is handled. Finally we learned to take backups and restore them if needed.

We hope that you enjoyed reading and working through this block. There are many things we have not even talked of, and you must have seen some of those things from the documentation and books you must have studied to supplement the material given here. We hope you will continue on your voyage of exploration. There are newer versions of UNIX already widely available which offer many more features and take care of several problems that were present in the previous versions. With what you have studied here you should be easily able to teach yourself about them from the documentation. Happy learning!

4.8 MODEL ANSWERS

Check Your Progress

1. Yes, it is possible for serious data loss to occur. You can lose important system files as well.

2. No model answer.

3. You cannot make a file system larger than the physical device. If you make a file system smaller than the size of the device, the extra space is not used and is thus wasted.

4. No model answer.

5. No model answer.

6. The number of links to a file is given in the long listing of a directory

   % ls - l

   To locate the other links, say

   % ls - li

   The -i option gives the inode numbers of all the files. You now need to locate files with the same inode numbers. They are links to the same file.