LRU Scheme

7-f
5-f
2-f
1-f
7-f
5-f
4-f
5-h
1-f
2-f
5-h
7-f

Page Fault Ratio = 10/12

Q.20 ***What is virtual memory? Explain the use of virtual memory using a suitable example.*** ***[R.T.U. 2014, 2013]***

Ans. Virtual Memory : Virtual memory, as its name suggests, doesn't physically exist on a memory chip.

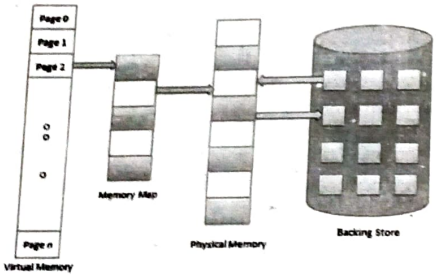


Fig. : Diagram showing virtual memory that is larger than physical memory

It is an optimization technique and is implemented by the operating system in order to give an application program the impression that it has more memory than actually exists. Virtual memory is implemented by various operating systems such as Windows, Mac OS X, and Linux. Virtual memory is a technique that allows the execution of processes which are not completely available in memory. The main visible advantage of this scheme is that programs can be larger than physical memory. Virtual memory is the separation of user logical memory from physical memory.

This separation allows an extremely large virtual memory to be provided for programmers when only a smaller physical memory is available. Following are the situations, when entire program is not required to be loaded fully in main memory.

Uses of Virtual Memory : Virtual memory is an old concept. Before computers had cache, they had virtual memory. For a long time, virtual memory only appeared on mainframes. Personal computers in the 1980s did not use virtual memory. Initially, virtual memory mean the idea of using disk to extend RAM. Programs wouldn't have to care whether the memory was "real" memory (i.e., RAM) or disk. The operating system and hardware would figure that out.

Later on, virtual memory was used as a means of memory protection. Every program uses a range of addressed called the address space.

The assumption of operating systems developers is that any user program can not be trusted. User programs will try to destroy themselves, other user programs, and the operating system itself. That seems like such a negative view, however, it's how operating systems are designed. It's not necessary that programs have to be deliberately malicious. Programs can be accidentally malicious (modify the data of a pointer pointing to garbage memory).

Virtual memory can help there too. It can help prevent programs from interfering with other programs. Occasionally, you want programs to cooperate, and share memory. Virtual memory can also help in that respect.

Let's say that an operating system needs 120 MB of memory in order to hold all the running programs, but there's currently only 50 MB of available physical memory stored on the RAM chips. The operating system will then set up 120 MB of virtual memory, and will use a program called the virtual memory manager (VMM) to manage that 120 MB. The VMM will create a file on the hard disk that is 70 MB (120 - 50) in size to account for the extra memory that's needed. The OS will now proceed to address memory as if there were actually 120 MB of real memory stored on the RAM, even though there's really only 50MB. So, to the OS, it now appears as if the full 120 MB actually exists. It is the responsibility of the VMM to deal with the fact that there is only 50 MB of real memory.

For example, if you load the operating system, an e-mail program, a Web browser and word processor into RAM simultaneously, 32 megabytes is not enough to hold it all. If there were no such thing as virtual memory, then once you filled up the available RAM your computer would have to say, "Sorry, you can not load any more applications. Please close another application to load a new one." With virtual memory, what the computer can do is look at RAM for areas that have not been used recently and copy them onto the hard disk. This frees up space in RAM to load the new application. Because this copying happens automatically, you don't even know it is happening, and it makes your computer feel like it has unlimited RAM space even though it only has 32 megabytes installed. Because hard disk space is so much cheaper than RAM chips, it also has a nice economic benefit. □□□

DEADLOCK AND DEVICE MANAGEMENT 3

PREVIOUS YEARS QUESTIONS

PART-A

Q.1 ***Suppose the head of moving head disk is currently servicing a request at track 60. If the queue of request is kept in FIFO order, what is the total head movement to satisfy these requests for the following disk scheduling algorithm :***
(i) FSFS
(ii) SSFT

Request Sequence	*Track Number*
1	*56*
2	*170*
3	*35*
4	*120*
5	*10*
6	*140*

[R.T.U. 2016]

Ans.(i) FSFS

Head Movement :

$60 \rightarrow 56 \rightarrow 170 \rightarrow 135 \rightarrow 120 \rightarrow 10 \rightarrow 140$

Total Head Movement :

$4 + 114 + 135 + 85 + 110 + 130 = 578$

(ii) SSFT

Head Movement :

$60 \rightarrow 56 \rightarrow 35 \rightarrow 10 \rightarrow 120 \rightarrow 140 \rightarrow 170$

Total Head Movement :

$4 + 21 + 25 + 110 + 20 + 30 = 210$

Q.2 ***Compare FCFS and SSTF disk scheduling algorithms. Initially the Read/Write Head is at 50. The requests are 63, 52, 01, 93, 72, 13, 81, 54 (8 requests). Compute total movement of R/W Head.*** ***[R.T.U. 2015]***

Ans. FCFS Scheduling : The movement of the head will be as follows:

50 – 63 – 52 – 01 – 93 – 72 – 13 – 81 – 54

Total movement = 13 + 11 + 51 + 92 + 21 + 59 + 68 + 27
= 342

SSTF Scheduling

The movement of the head will be as follows:

50 – 52 – 54 – 63 – 72 – 81 – 93 – 13 – 01

Total movement = 2 + 2 + 9 + 9 + 9 + 12 + 80 + 12
= 135

Q.3 ***Compare SCAN and C-SCAN disk scheduling algorithms. Read write Head is at 45. The requests are 63, 52, 01, 93, 72, 13, 81 and 54 (8 requests). Compute total movement of R/W Head.*** ***[R.T.U. 2015]***

Ans. SCAN Scheduling

The Head would move as follows:

45 – 52 – 54 – 63 – 72 – 81 – 93 – 13 – 01

Total movement = 7 + 2 + 9 + 9 + 9 + 12 + 80 + 12
= 140

C-SCAN Scheduling

The Head would move as follows:

45 – 52 – 54 – 63 – 72 – 81 – 93 – (00) – 01 – 1

Total movement = 7 + 2 + 9 + 9 + 9 + 12 + 01 + 1
= 61

Q.4 ***Define disk scheduling.***

Ans. Disk Scheduling : Disk scheduling is done by operating systems to schedule I/O requests arriving for disk. Disk scheduling is also known as I/O scheduling.

Q.5 ***What do you mean by FCFS scheduling.***

Ans. First Come First Serve (FCFS) : It is the simplest form of scheduling operations performed in order requested. No recording of request queue. No starvations i.e. every process is serviced.

PART-B

Q.6 ***Suppose that a disk drive has 200 cylinders, numbered 0 to 199. The drive is initially at cylinder 53. The queue with request from I/O to blocks in cylinders 98 183 37 122 14 124 65 67 Count the total head movement of cylinders in SCAN and C-SCAN scheduling.***

[R.T.U. 2018, 2014, 2013]

Ans. (i) SCAN Scheduling : The drive is initially at cylinder 53.

queue = 98, 183, 37, 122, 14, 124, 65, 67

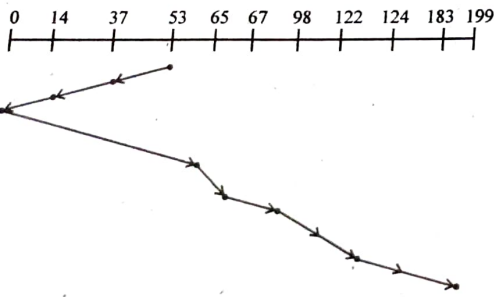


Fig.1

In SCAN scheduling the direction of head ement in addition to the head's current position 53. If disk arm is moving toward 0, the head will service 37 then 14. At cylinder 0, the arm will reverse and will e toward the other end of disk. (fig.(1))

Total head movement of cylinders in SCAN uling

= 16+23+14+65+2+31+24+2+59+16

= 252 tracks

(ii) C-SCAN Scheduling : The drive is initially at cylinder 53

queue = 98, 183, 37, 122, 14, 124, 65, 67

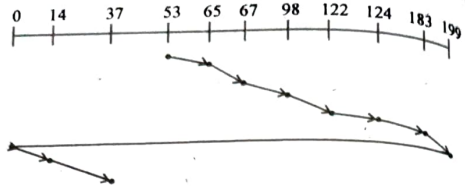


Fig.2

In C-SCAN scheduling movement the head from one end of the disk to the other, servicing request along the way. When the head reaches the other end, it immediately returns to the disk, without servicing any request on the return trip. (fig.(2))

Total head movement of cylinders in C-SCAN scheduling

= 12+2+31+24+2+59+16+199+14+23

= 382 tracks

Q.7 ***Explain the concept of spooling with all its types and its advantage and disadvantages.***

[R.T.U. 2018, 2016]

Ans. Spooling : Spooling is an acronym for simultaneous peripheral operation On line. When the job requests the printer to output a line, that line is copied into a system buffer and is written to the disk. When the job is completed, the output is actually printed. This form of processing is called spooling.

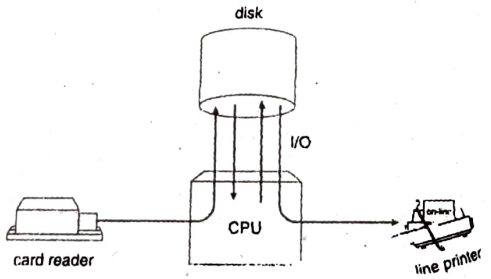


Fig.

Spooling, in essence, uses the disk as a huge buffer for reading as far ahead as possible on input devices and for storing output files until the output devices are able to accept them.

Spooling is also used for processing data at remote sites. The CPU sends the data via communication paths to a remote printer (or accepts an entire input job from a remote card reader). The remote processing is done at its own speed, with no CPU intervention. The CPU just needs to be notified when the processing is completed, so that it can spool the next thatch of data.

Spooling overlaps the I/O of one job with the computation of other jobs. Even in simple system, the spooler may be reading the input of one job while printing the output of a different job. During this time, still another job (or other jobs) may be executed, reading its "cards" from disk and "printing" its output lines onto the disk.

Spooling has a direct beneficial effect on the performance of the system. For the cost of some disk space and a few tables, the computation of one job can overlap with the I/O of other jobs. Thus, spooling can keep both the CPU and the I/O devices working at much higher rates. Spooling leads naturally to multiprogramming which is the foundation of all modern operating systems

Types of Spoolers

1. Print Spooler

A software program is responsible for managing all the current printing jobs which are sent to the computer printer or print server. The print spooler program may allow a user to delete a print job being processed or otherwise manage the print jobs currently waiting to be printed.

2. The System V-style Spooler

The system V-style spooler of printing subsystem uses system V-Release 4 commands, queues, and files and is administered the same way. Typical user commands available to the system V-style spooler are:

- **lp :** the user command to print a document
- **lpstat :** shows the current print queue
- **cancel :** deletes a job from the print queue
- **lpadmin :** a system administration command that configures the print system
- **lpmove :** a system administration command that moves jobs between print queues

3. The Berkeley-style Spooler

The Berkeley-style spoolers one of several standard architectures for printing on the Unix platform. It originated in 2.10BSD, and is used in BSD derivatives such as FreeBSD, NetBSD, OpenBSD, and DragonFly BSD. A system running this print architecture could traditionally be identified by the use of the user command lpr as the primary interface to the print system, as opposed to the system V-style lp command. Typical user commands available to the Berkeley-style spooler are:

- **lpr :** the user command to print
- **lpq :** shows the current print queue
- **lprm :** deletes a job from the print queue

4. CUPS-based Spooler

CUPS (Common UNIX Printing System) is a new type of spooler. It was designed to work across most UNIX and Linux-based system. It is also standards based. It enables printing through RFC1179 (lpr), IPP, CIFS/SMB, Raw socket (JetDirect), and through local printing. CUPS uses network printer browsing and Postscript Printer Description (PPD) files to ease the common tasks of printing.

Advantages of Spooling

The advantages of spooling are as follows:

(i) The spooling operation uses a disk as a very large buffer.

(ii) Spooling is capable of overlapping I/O operation for one job with processor operations for another job.

(iii) Processes are not suspended for a long time.

(iv) It can produce multiple copies of the output without running the process again.

Disadvantages of Spooling

The disadvantages of spooling are as follows:

(i) Need large amounts of disk space.

(ii) Increase disk traffic.

(iii) Not practical for real-time environment, because results are produce at a later time.

Q.8 ***Explain global versus local allocation.***

[R.T.U. 2018, Dec. 2013]

Ans. Global Versus Local Allocation :An importar factor in the way frames are allocated to the variou processes is page replacement. With multiple process competing for frames, we can classify page-replacem algorithms into two broad categories: **global replacem and local replacement.** Global replacement allow process to select a replacement frame from the set o frames, even if that frame is currently allocated to s other process; that is, one process can take a frame another. Local replacement requires that each pr select from only its own set of allocated frames.

For example, consider an allocation scheme where we allow high - priority processes to select frames from low priority processes for replacement. A process can select a replacement from among its own frames or the frames of any ower-priority process. This approach allows a high priority process to increase its frame allocation at the expense of a low priority process.

With a local replacemcent strategy, the number of frames allocated to a process does not change. With global replacemcent, a process may happen to select only frames allocated to other processes, thus increasing the number of frames allocated to it (assuming that other processes do not choose its frames for replacement).

One problem with a global replacement algorithm is that a process cannot control its own page-fault rate. The set of pages in memory for a process depends not only on the paging behavior of that process but also on the paging behavior of other processes. Therefore, the same process may perform quite differently (for example, taking 0.5 seconds for one execution and 10.3 seconds for the next execution) taking 0.5 seconds for one execution and 10.3 seconds for the next execution) because of totally external circumstances. Such is not the case with a local replacement algorithm. Under local replacement, the set of pages in memory for a process is affected by the paging behavior of only that process. Local replacement might hinder a process, however, by not making available to it other, less used pages of memory. Thus global replacement generally results in greater system throughput and is therefore the more common method.

Q.9 ***Explain various disk scheduling algorithms in brief.*** ***[R.T.U. 2017, Dec. 2013]***

OR

Discuss the following disk scheduling algorithms:

(i) Shortest Seek Time First

(ii) First Come First Served

(iii) SCAN

(iv) C-Look ***[R.T.U. 2012]***

Ans. Hard Disk : A hard disk drive is a collection of plates called platters. The surface of hard disk is made of concentric circles called tracks. Further more, each track is divided into smaller pieces called sectors.

For each I/O output request first head is selected. It is then moved over the destination track. The disk is then rotated to position the desired sector under the head and finally the read/write operation is performed.

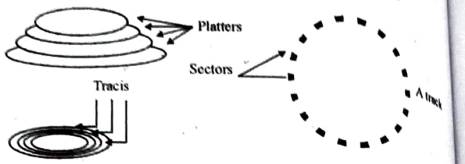


Fig.

The main responsibility of the operating system is to use hardware efficiently. For the disk gives the main responsibility are :

1. **Maximize the Throughput:** The average number of requests satisfied per unit time.
2. **Minimize the Response Time:** The average time that a request must wait before it is satisfied.

The access time consist of two major components.

1. **Seek Time:** Seek time is the time for the disk arm to move the heads to the cylinder containing the desired sector.
2. **Rotational Latency:** Rotational latency is the additional time for the disk to rotate the desired sector to the disk head.

Disk Bandwidth is the total number of bytes transferred divided by the total time between first request for service and the completion of the last transfer. There are several algorithm exist to schedule the servicing of disk I/O requests illustrate them with a request queue (0 - 199) suppose disk head initially at 100th position.

(i) FCFS Scheduling
(ii) SSTF Scheduling
(iii) SCAN Scheduling
(iv) C-SCAN Scheduling
(v) LOOK Scheduling and C-LOOK Scheduling

(i) First Come First Serve (FCFS) : It is the simplest form of scheduling operations performed in order requested. No recording of request queue. No starvations i.e. every process is serviced.

Poor Performance total time estimated by total arm motion $|100-23|+|23-80|+|80-132|+|132-44|+|44-188| = 77+57+52+88+144 = 418$ Cylinder.

Disk head initially at cylinder 100. It will move from 100 to 23, then to 80, 132, 44 and finally to 188.

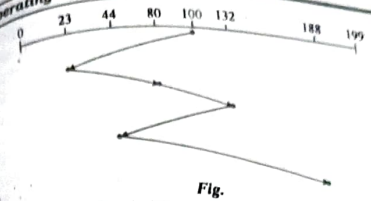


Fig.

(ii) Shortest-Seek-Time-First (SSTF) : Selects the request with minimum seek time from the current position. SSTF makes easy to service all the requests close to the current head position before moving the head for a way to service other requests. Reduce total seek time as compared to FCFS. Starvation is possible.

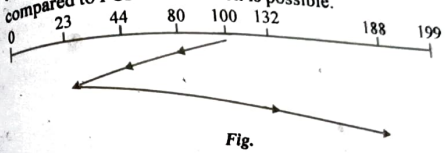


Fig.

The closest request to the initial head position (100) is at cylinder 80. Once we are at cylinder 80, the next closest requests is at 44 from there, the request at cylinder 23 is closer than the one at 132 so then 132 and it finally service 188. Total head movement = $|100-80|+|80-44|+|44-23|+23-132|+|132-188| = 142$ Cylinders.

(iii) SCAN Scheduling : The disk arm starts at one end of the disk and moves towards the other end, servicing a requests until is gets to the other end of the disk. When it reaches at the end, the direction of head is reserved and servicing continues. This is also known as Elevator algorithm because the head continuously scans back and forth across the disk. Reduces variance compared to SSTF.

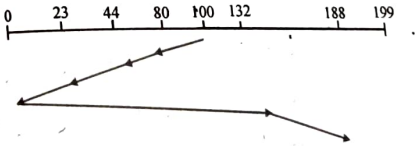


Fig.

If the disk arm is moving towards 0, the head will service 80 and then 44, 23. A cylinder 0, the arm will move towards the other ends of the disk servicing the requests at 132, 188. Total head movement $|100-80|+|80-44|+|44-23|+|23-0|+|0-32|+|132-188| = 88$ Cylinders.

(iv) C - SCAN Scheduling: Circular Scan Scheduling is a variant of SCAN, to provide information time. Like SCAN it servicing request along the way until the head reaches the end 0 or 199. It immediately moves in reverse direction, without servicing any request.

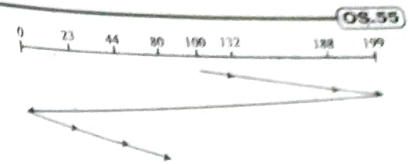
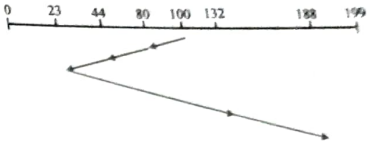
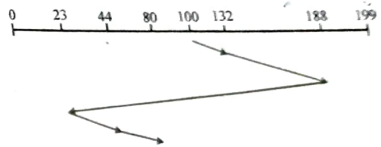


Fig.

(v) LOOK and C - LOOK Scheduling : The arm goes only as far as the final request in each direction. Then it reverse its direction. Without going all the way to travel of the disk. These versions of SCAN and C- SCAN all known as LOOK and C -look. They look for a request before continuing to more in a given direction.



LOOK



C - LOOK

Fig.

Q.10 ***Apply deadlock detection algorithm to the following data and show the results :***

Available $= (2, 1, 0, 0)$

$$\textit{Request} = \begin{pmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{pmatrix}$$

$$\textit{Allocation} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{pmatrix}$$ ***[R.T.U.***

Ans.

1. First process 3 is satisfied so it can be con and resources freed.

Available :

(2 2 2 0)

Request :

$$\begin{pmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Allocation :

$$\begin{pmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

2. Now process 2 can be satisfied and its resources freed.

Available :

(4 2 2 1)

Request :

$$\begin{pmatrix} 2 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Allocation :

$$\begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

3. Finally process 1 can be satisfied and its resources freed

Available :

(4 2 3 1)

Request :

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Allocation :

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

As all processes can be completed in an order that does not create a deadlock, hence the deadlock detection algorithm returns the result that no deadlock will be faced in the given scenario.

Q.11 *What are safe and unsafe states?* ***[R.T.U. 2013]***

Ans. A state is safe if the system can allocate resources to each process (up to its maximum) in some order and still avoid a deadlock. More formally, a system is in a safe state only if there exists a safe sequence. A sequence of processes $< P_1, P_2, \ldots P_n >$ is a safe sequence for the current allocation state if, for each P_i, the resources that P_i can still request can be satisfied by the currently available resources plus the resources held by all the P_j with $j < i$. In this situation, if the resources that process P_i needs are not immediately available, then P_i can wait until all P_j have finished. When they have finished, P_i can obtain all of its needed resources, complete its designated task, return its allocated resources, and terminate. When P_i terminates, P_{i+1} can obtain its needed resources, and so on. If no such sequence exists, then the system state is said to be unsafe.

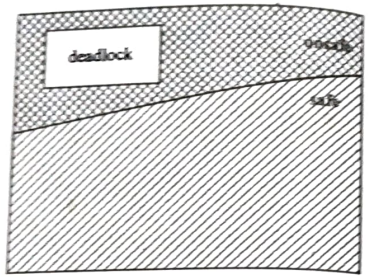


Fig. : Safe, unsafe, and deadlock state spaces

A safe state is not a deadlock state. Conversely, a deadlock state is an unsafe state. Not all unsafe state are deadlock, however (see figure). An unsafe state may lead to a deadlock. As long as the state is safe, the operating system can avoid unsafe (and deadlock) states. In an unsafe state, the operating system cannot prevent processes from requesting resources such that a deadlock occurs: The behavior of the processes controls unsafe states.

Q.12 *Write short note on Device Drives.* ***[R.T.U. 2011]***

Sol. Device Drivers

Device driver is software via which the operating system communicates with the device controllers. Each device has its own device driver and a device controller which is specific to the device. The device drivers hide the differences among the different device controller and present a uniform interface to the operating system. The device driver provides correct commands to the controller, interprets the controller register and transfers data to and from device controller register as required for the correct device operation.

In the presence of the I/O controller, the job of the operating system becomes simpler. A part of the OS, called the device driver, interacts with the I/O controllers. The job of the device driver is to accept the I/O request from the device-independent software and execute it. The I/O request corresponds to reading or writing in certain logical blocks of the file system. The driver converts the logical location into physical location in terms of the drive, track, sector etc. Device driver initializes the I/O controller with few parameters in the control register. For example, in the case of video controller, the parameter may be (1) the number of characters per line and (2) number of lines per screen. It also provides the detail of the data transfer. The I/O controller then takes care of the actual transfer. The I/O controller may also perform serial to parallel conversion and do error correction if necessary. Till the I/O operation is complete, the process which has called for the I/O transfer will typically be in the blocked state.

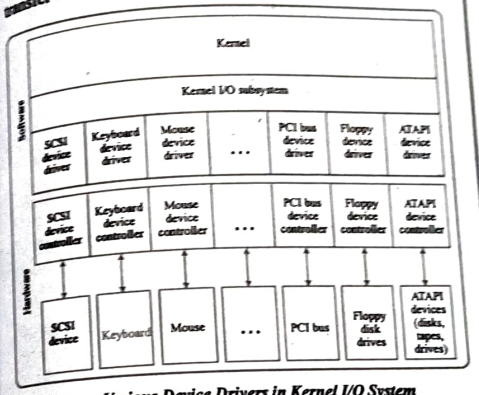


Fig. : Various Device Drivers in Kernel I/O System

The windows XP I/O manger supports four different types of drivers. All hardware devices (scanner, modem, printer and so on) require hardware device drivers that manipulate the hardware to retrieve input from or write output to the physical device or network. File system drivers are device drivers that accept file-oriented I/O request and translate them to the appropriate primitive I/O commands for the physical device. A network driver is a file system driver that redirects the I/O request to the appropriate machine on the network and receives data from the remote machine. Windows XP device driver meets a new standard called the windows driver model (WDM) that enables windows XP to share drivers with other windows operating systems. WDM also incorporates features that enhance real time streaming media by processing the data in the kernel mode rather than user mode.

PART-C

Q.13 *What is deadlock? What are necessary conditions for deadlock to occur?* ***[R.T.U. 2018, 2013]***

OR

What is deadlock? Explain the conditions and prevention of deadlock? ***[R.T.U. 2017]***

OR

What are the different deadlock prevention schemes? Explain. ***[R.T.U. 2015]***

OR

What is deadlock? What are the necessary conditions to occur the deadlock? What are the various methods to recover from the deadlock? ***[R.T.U. 2014]***

Ans. Introduction to Deadlocks : A set of processes is deadlocked if each process in the set is waiting for an event that can cause only by another process in the set.

Because all the processes are waiting, none of them will, even cause any of the events that could wake up any of the other members of the set and all the processes continue to wait forever. For this model, we assume that processes have only a single thread and that there are no interrupts possible to wake up a blocked process. The no-interrupts condition is needed to prevent an otherwise deadlocked process from being awakened by, say, an alarm and then causing events that release other processes in the set.

Conditions for Deadlock

Four essential conditions for deadlock to occur : Coffman et al. (1971) showed that four conditions must hold for these to be a deadlock :

1. Mutual exclusion condition : Each resource is either currently assigned to exactly one process or is available.

2. Hold and wait condition : Processes currently holding resource granted earlier can request new resources.

3. No preemption condition : Resources previously granted cannot be forcibly taken away from a process. They must be explicitly released by the process holding them.

4. Circular wait condition : There must be a circular chain of two or more processes, each of which is waiting for a resource held by the next member of chain.

All four of these conditions must be present for a deadlock to occur. If one of them is absent, no deadlock is possible.

Deadlock Prevention : Deadlock avoidance is essentially impossible because it requires information about future requests. Different deadlock prevention schemes are as follows :

(i) Attacking the mutual exclusion condition : If no resource were ever assigned exclusively to a single process, we would never have deadlocks. However it is equally clear that allowing two processes to write on the printer at the same time will lead to chaos.

By spooling printer output, several processes can generate output at the same time. In this model, the only process that actually requests the physical printer is the printer daemon. Since the daemon never requests any other resources, we can eliminate deadlock for the printer.

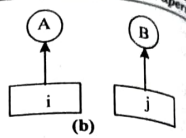
(ii) Attacking the hold and wait condition : The second conditions stated by Coffman et al looks slightly more promising. If we can prevent processes that hold resources from waiting for more resources, we can eliminate deadlocks. One way to achieve this goal is to require all processes to request all their resources before starting execution. If every thing is available, the process will be allocated whatever it needs and can run to completion. If one or more resources are busy, nothing will be allocated and the process would just wait.

An immediate problem with this approach is that many processes do not know how many resources they will need until they started running. In fact, if they knew, the Banker's algorithm could be used. Another problem is that the resources will not be used optimally with this approach. Nevertheless, some mainframe batch system requires the user to list all the resources on the first line of each job. The system then acquires all resources immediately and keeps them until the job finishes. While this method puts a burden on the programmer and wastes resources, it does prevent deadlocks.

(iii) Attacking the no preemption condition : Attacking the third condition (no preemption) is even less promising than attacking the second one. If a process has been assigned the printer and is in the middle of printing its output, forcibly taking away the printer because a needed plotter is not available is tricky at best and impossible at worst.

***(iv)* Attacking the circular wait condition :** The circular wait can be eliminated in several ways. One way is simply to have a rule saying that a process is entitled only to a single resource at any moment. If it needs a second one, it must release the first one for a process that needs to copy a huge file from a tape to a printer, this restriction is unacceptable.

1. Image setter
2. Scanner
3. Plotter
4. Tape drive
5. CD Rom drive



(a) (b)

Fig. : (a) Numerically ordered resources (b) A resource graph

Another way to avoid the circular wait is to provide a global numbering of all the resources, as shown in Fig. Now the rule is this : processes can request resources whenever they want to, but all requests must be made in numerical order. A process may request first a printer and then a tape drive, but it may not request first a plotter and then printer.

The various approaches to deadlock prevention are :

Condition	Approach
Mutual exclusion	Spool everything
Hold and wait	Request all resources initially
No preemption	Take resource away
Circular wait	Order resources numerically

Recovery from deadlock

There are two options for breaking a deadlocks as follows :

(1) To abort one or more processes to break the circular wait i.e. **Process Termination.**

(2) To preempt some resources from one or more of the deadlock processes i.e. **Resource Preemption.**

1. Process Termination : To eliminate deadlocks by aborting process, we use one of two methods:

(i) Abort all deadlocked processes : This method clearly break the deadlock cycle but it is a very extensive method that the deadlocked processes may have computed for a long time, and the results of these partial computations must be discarded and probably will have to be recomputed later.

(ii) Abort one process at a time until the deadlock cycle is eliminated

The method incurs considerable overhead, since after each process is aborted, a deadlock-detection algorithm must be invoked to determine whether any processes are still deadlocked.

We should abort those processes whose termination will incur the minimum cost. Many factors may affect which process is chosen, including.

- How many processes will need to terminated.
- How long the process has computed.
- How much longer the process will compute.
- How many and what type of resources the process has used.
- How many more resources the process needs.
- Whether the process is interactive or batch.

2. Resource Preemption : To eliminate deadlocks using resource preemption, be successively preempt some resources from processes and give those resources to other processes until the deadlock cycle is broken, need to be addressed.

(i) Selecting a victim : Which resources and which processes are to be preempted?

(ii) Rollback : We must roll back the process to some safe state and restart it from the state. It is more effective to roll back the process only as far as necessary to break the deadlock rather then total rollback.

Starvation : How can we guarantee that resources will not always be preempted from the same process? It may happen that the same process is always picked as a victim. As a result, this process never completes its designated task. We must ensure that a process can be picked as a victim only a (small) finite number of times.

Q.14 *Write and explain Bankers algorithm for deadlock avoidance.* ***[R.T.U. 2018, 2015]***

OR

Explain Banker's Algorithm for dead lock avoidance with an example.

[R.T.U. 2016, 2012, 2011, Raj. Univ. 2007, 2005, 2003]

Ans. The Banker's Algorithm for a Single Resource : A scheduling algorithm that can avoid deadlocks is due to Dijkstra and is known as the banker's algorithm and is an extension of the deadlock detection algorithm. It is modeled on the way a small-town banker might deal with a group of customers to whom he has granted lines of credit. What the algorithm does is check to see if granting the request leads to an unsafe state. If it does, the request is denied, If granting the request leads to a safe state, it is carried out.

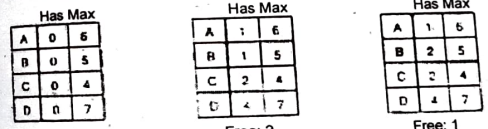


Fig. 1 : Three resource allocation states : (a) Unsafe (b) Safe, (c) Safe

In fig.1(a) we see four customers, A, B, C and D, each of them has been granted a certain number of credit units (e.g., 1 unit is 1k Dollars). The banker knows that all customers will not need their maximum credit immediately, so he has reserved only 10 units rather than 22 to service them. (In this analogy, customers are processes, units are tape drives and the banker is the operating system).

The customers go about their respective business making loan requests from time to time (i.e > asking for resource). At a certain moment the situation is as shown in fig.1(b). This state is safe because with two units left the banker can delay any request except thus letting C finish and release all four of his resource. With four units in hand the banker can let either D or B have the necessary units and so on.

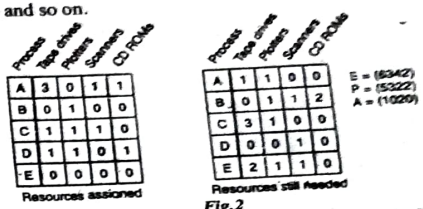


Fig.2

Consider what happen if a request from B for one more unit were granted in fig.1. We would have situation fig.1(a) which is unsafe. If all the customers suddenly asked for their maximum loans the banker could not satisfy any of them and we would have a deadlock. An unsafe state does not have to lead to deadlock since a customer might not need the entire credit line available but the safe state. If it does the request is granted otherwise it is postponed.

The Banker's Algorithm for Multiple Resource

The banker's algorithm can be generalized to handle multiple resources.

In fig.2 we see two matrices. The one on the left shows how many resources each process. The matrix on the right shows how resources each process still needs in order to complete. These matrices are just C and R. As in the single resources case processes must state their total resource needs before executing so that the system can compute the right hand matrix at each instant.

Data Structure by Banker's Algorithm : Several data structure must be maintained to implement the banker's algorithm. These data structures encode the state of the resource-allocation system. Let n be the number of processes in the system and m be the number of resources types. We need the following data structures :

Available : A vector of length m indicates the number of available resources of each type. If available[j] equals k, there are k instances of resource type R_j available.

Max : An m×n matrix defines the maximum demand of each processes. If Max[i][j] equals k, then process P may request at most k instasnces of resource type R_j.

Allocation : An m×n matrix defines the number o resources of each type currently allocated to eac

process. If allocation[i][j] equals k, then process P_i is currently allocated k instances of resources type R_j.

Need : An m×n matrix indicates the remaining resources need of each process. If Need[i][j] equals k, then process P_i may need k more instances of resource type R_j to complete its task. Note that [i][j] equals Max[i][j] – Allocation[i][j].

Q.15 ***What is deadlock avoidance? Explain banker's algorithm with following SNAPSHOT of a system? Resource A = 3, B = 14, C = 12 and D = 12 instances. If P1 request 1021 resource instance, it can granted or not?***

	Allocation				Maximum				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P0	0	0	1	2	0	0	1	2	1	5	2	0
P1	1	0	0	0	1	7	5	0				
P2	1	3	5	4	2	3	5	6				
P3	0	6	3	2	0	6	5	2				
P4	0	0	1	4	0	6	5	6				

[R.T.U. 2017]

Ans. Deadlock Avoidance : Deadlock is a state in which a process is waiting for the resource that is already used by another process and that another process is waiting for another resource. Deadlock Avoidance is an approach to the deadlock problem that anticipates deadlock before it actually occurs. This approach employs an algorithm to access the possibility that deadlock could occur and acting accordingly. This method differs from deadlock prevention, which guarantees that deadlock cannot occur by denying one of the necessary conditions of deadlock.

If the necessary conditions for a deadlock are in place, it is still possible to avoid deadlock by being careful when resources are allocated. The most famous deadlock avoidance algorithm, given by Dijkstra [1965], is the Banker's algorithm.

Banker's Algorithm

The banker's algorithm is a resource allocation and deadlock avoidance algorithm that tests for safety by simulating the allocation for predetermined maximum possible amounts of all resources, then makes an "s-state" check to test for possible activities, before deciding whether allocation should be allowed to continue.

Following **Data structures** are used to implement the Banker's Algorithm:

Let **'n'** be the number of processes in the system and **'m'** be the number of resources types.

Available :

- It is a 1-d array of size **'m'** indicating the number of available resources of each type.
- Available[j] = k means there are **'k'** instances of resource type R_j

Max :

- It is a 2-d array of size **'n*m'** that defines the maximum demand of each process in a system.
- Max[i, j] = k means process P_i may request at most **'k'** instances of resource type R_j.

Allocation :

- It is a 2-d array of size **'n*m'** that defines the number of resources of each type currently allocated to each process.
- Allocation[i, j] = k means process P_i is currently allocated **'k'** instances of resource type R_j

Need :

- It is a 2-d array of size **'n*m'** that indicates the remaining resource need of each process.
- Need [i, j] = k means process P_i currently allocated **'k'** instances of resource type R_j
- Need [i, j] = Max [i, j] – Allocation [i, j]

$Allocation_i$ specifies the resources currently allocated to process P_i and $Need_i$ specifies the additional resources that process P_i may still request to complete its task.

Banker's algorithm consist of Safety algorithm and Resource request algorithm

Safety Algorithm

The algorithm for finding out whether or not a system is in a safe state can be described as follows:

1. Let Work and Finish be vectors of length 'm' and 'n' respectively.
   Initialize: Work = Available
   Finish[i] = false; for i=1, 2, 3, 4….n
2. Find an i such that both
   a. Finish[i] = false
   b. $Need_i$ <= Work if no such i exists goto step (4)
3. Work = Work + Allocation
   Finish[i] = true
   goto step (2)
4. if finish [i] = true for all i
   then the system is in a safe state

Resource-Request Algorithm

Let $Request_i$ be the request array for process P_i. $Request_i[j] = k$ means process P_i wants k instances of resource type R_j. When a request for resources is made by process P_i, the following actions are taken:

1. If $Request_i$ <= $Need_i$
   Goto step (2) ; otherwise, raise an error condition, since the process has exceeded its maximum claim.

2. If $Request_i$ <= Available Goto step (3); otherwise, P_i must wait, since the resources are not available.
3. Have the system pretend to have allocated the requested resources to process Pi by modifying the state as follows:

$Available = Available - Request_i$
$Allocation_i = Allocation_i + Request_i$
$Need_i = Need_i - Request_i$

	Allocation				Maximum				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P0	0	0	1	2	0	0	1	2	1	5	2	0
P1	1	0	0	0	1	7	5	0				
P2	1	3	5	4	2	3	5	6				
P3	0	6	3	2	0	6	5	2				
P4	0	0	1	4	0	6	5	6				

Step 1 - Calculate Need Matrix-
Need = Max – Allocation

	A	B	C	D
P_0	0	0	0	0
P_1	0	7	5	0
P_2	1	0	0	2
P_3	0	0	2	0
P_4	0	6	4	2

Step 2
Predicting Safe Sequence-

(i) Work = Available
= 1, 5, 2, 0

Finish = | F | F | F | F | F |
P0 P1 P2 P3 P4

F = False, T = True

(ii) For i = 0
Need P0 = 0, 0, 0, 0
No allocation needed P0 in kept in safe sequence.
Work = Work + Allocation P0
= 1, 5, 2, 0 + 0, 0, 1, 2
= 1, 5, 3, 2
Finish [P0] = T

(iii) For i = 1
Need P1 = 0, 7, 5, 0
Need P1 > Work
Need P1 = > 1, 5, 3, 2
∴ P1 = must wait

for i = 2
Need P2 = 1, 0, 0, 2
Need P2 < work
1, 0, 0, 2 ≤ 1, 5, 3, 2
P2 is kept in safe sequence
work = work + Allocation
= 1, 5, 3, 2 + 1, 3, 5, 4
= 2, 8, 8, 6
Finish [P2] = T

(iv) For i = 3
Need P3 < Work
P3 is kept in safe sequence
work = 2, 8, 8, 6 + 0, 6, 3, 2
= 2, 14, 11, 8
Finish [P3] = T

(v) for i = 4
Nead P4 < work
P4 is kept in safe sequance
work = 2, 14, 11, 8 + 0, 0, 1, 4
= 2, 14, 12, 12
Finish [P4] = T

(vi) For i = 5
Nead P1 < work
P1 is kept in safe sequance
work = 2, 14, 12, 12 + 1, 0, 0, 0
= 3, 14, 12, 12
Finish [P1] = T

Finish = | T | T | T | T | T |
P0 P1 P2 P3 P4

Hence, the system is in safe state and safe sequance = P0, P2, P3, P4, P1,

(b) P1 request resource as
A B C D
1, 0, 2, 1
Using Resource Request Algo-
Request P1 < Need P1
1, 0, 2, 1 < 0, 7, 5, 0
This statement in false
Hence resource request will not be ful filled.

Q.16 ***What do you mean by disk scheduling? Suppose the head of moving head disk is currently servicing s request at track 60. If the queue of request is kept in FIFO order. What is the total head movement to satisfy these requests for the following disk scheduling algorithm:***

(i) FCFS
(ii) SCAN
(iii) C-SCAN

Request Sequence	Track Number
1	55
2	175
3	30
4	125
5	10
6	140

[R.T.U. 2017]

Ans. Disk scheduling is done by operating systems to schedule I/O requests arriving for disk. Disk scheduling is also known as I/O scheduling.

Disk scheduling is important because:

- Multiple I/O requests may arrive by different processes and only one I/O request can be served at a time by disk controller. Thus other I/O requests need to wait in waiting queue and need to be scheduled.
- Two or more request may be far from each other so can result in greater disk arm movement.
- Hard drives are one of the slowest parts of computer system and thus need to be accessed in an efficient manner.

Request Sequence	Track Number
1	55
2	175
3	30
4	125
5	10
6	140

(i) FCFS

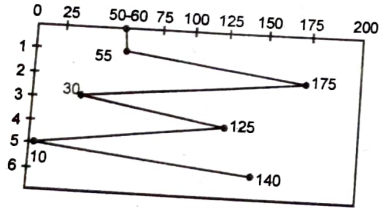


Fig.

Total Head Movement = 5 + 120 + 145 + 95 + 115 + 130
= 610 tracks

(ii) SCAN

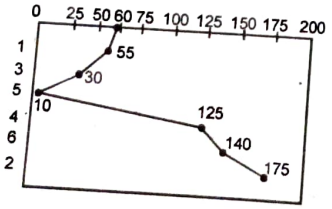
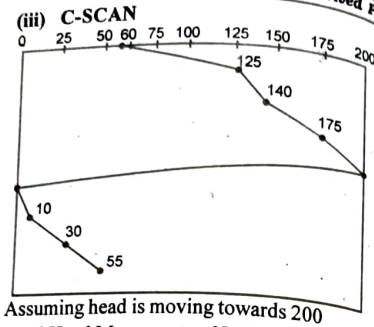


Fig.

Assuming head is moving towards 0 at start point
Total Head Movement = 5 + 25 + 20 + 115 + 15 + 35
= 215 tracks

(iii) C-SCAN



Assuming head is moving towards 200
Total Head Movement = 65 + 15 + 35 + 25 + 200 + 10 + 20 + 25
= 395 tracks

Q.17 ***Explain the following :***
(i) Resource allocation graph
(ii) Recovery from deadlock. *[R.T.U. Dec. 2013]*

Ans. (i) Deadlock modeling

Holt (1972) showed how these four conditions can be modeled using directed graphs. The graphs have two kinds of nodes: **processes**, shown as circles and **resources**, shown as squares. An arc from a resource node (square) to a process node (circle) means that the resource has previously been requested by, granted to and is currently held by that process. In Fig. 1(a) resource R is currently assigned to process A.

An arc from a process to a resource means that the process is currently blocked waiting for that resource. In fig.1(b) process B is waiting for resource S. In fig.1(c) we see a deadlock: process C is waiting for resource T, which is currently held by process D. Process D is not about to release resource T because it is waiting for resource U, held by C. Both processes will wait forever. A cycle-in the graph means that there is a deadlock involving the processes and resources: in the cycle (assuming that there is one resource of each kind). In this example, the cycle is C-T-D-U-C.

Now let us look at an example of how resource graphs can be used. Imagine that we have three processes A, B and C and three resources R, S and T.

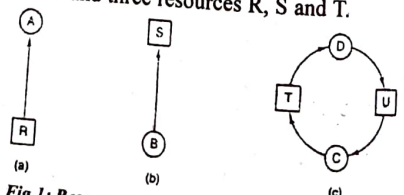


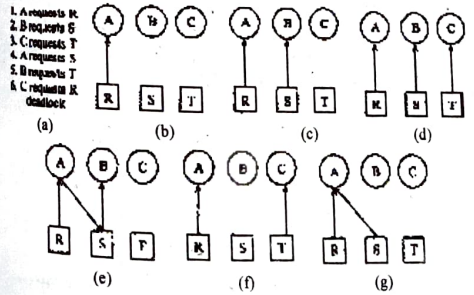
Fig.1: Resource allocation graphs, (a) Holding a resource, (b) Requesting a resource, (c) Deadlock

The operating system is free to run any unblocked process at any instant, so it could decide to run A until A finished all its work, then run B to completion and finally run C.

This ordering does not lead to any deadlocks (because there is no competition for resources) but it also has no parallelism at all. In addition to requesting and releasing resources, processes compute and do I/O. When the processes run sequentially, there is no possibility that while one process is waiting for I/O, another can use the CPU. Thus running the processes strictly sequential may not be optimal. On the other hand, if none of the processes do any I/O at all, shortest job first is better than round robin, so under some circumstances running all process sequentially may be the best way.

Let us now suppose that the processes do both I/O and computing, so that round robin is a reasonable scheduling algorithm. The resource requests might occur in the order of fig. 2(a). If these six requests are carried out in that order, the six resulting resource graphs are shown in fig. 2(b)-(g). After request 4 has been made, a block waiting for S, as shown in fig.2(e). In the next two steps B and C also block, ultimately leading to a cycle and the deadlock fig.2(g).

However, the operating system is not required to run the processes in any special order. In particular, if granting a particular request might lead to deadlock, the operating system can simply suspend the process without granting the request (i.e., just not schedule the process) until it is safe in fig.2. if the operating system knew about the impending deadlock, it could suspend B instead of granting it S. By running only A and C, we would get the requests and releases of fig.2 (h) instead of fig. 2 (a). This sequence leads to the resource graphs of fig.2 (*i*)-(h) which do not lead to deadlock.



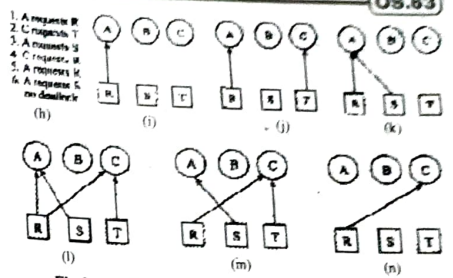


Fig.2: Examples of how deadlock occurs and how it can be avoided.

After step (h) process B can be granted S because A is finished and C has everything it needs. Even if B should eventually block when requesting T, no deadlock can occur. B will just wait until C is finished.

For the moment, the point to understand is that resource graphs are a tool that let us see if a given request release sequence leads to deadlock. We just carry out the requests and releases step by step and after evey step check the graph to see if it contains any cycles. If so, we have a deadlock: if not, there is no deadlock. Although our treatment of resource graphs has been for the case of a single resource of each type, resource graphs can also be generalized to handle multiple resources of the same type.

In general, four strategies are used for dealing with deadlocks.

1. Just ignore the problem altogether. May be if we ignore it, it will ignore us.
2. Detection and recovery. Let deadlocks occur, detect them and take action.
3. Dynamic avoidance by careful resource allocation.
4. Prevention, by structurally negating one of the four conditions necessary to cause a deadlock.

A B C
requests R requests S requests T
requests S requests T requests R
requests R requests S requests T
requests S requests T requests R

Ans. (ii) Recovery from Deadlock : *Refer to Q.12.*

Q.18 ***Define the device characteristics and its hardware considerations.***

Ans. Device Characteristics : Almost everything imaginable can be, and probably has been, used as a peripheral device to a computer, ranging from steel mills

to laser beams, from radar to mechanical potato pickers, from thermometers to space ships. Fortunately, most computer installations utilize only a relatively small set of peripheral devices. Peripheral devices can be generally categorized into two major groups: (1) input or output devices (2) storage devices.

1. Input or Output Devices : An input device is one by which the computer "senses" or "feels" the outside world. These devices may be mechanisms such as thermometers or radar devices but, more conventionally, they are devices to read punched cards, punched paper tape or messages typed on typewriter like terminals. An output device is one by which the computer "affects" or "controls" the outside world. It may be a mechanism such as a temperature control knob or a radar direction control, but more commonly it is a device to punch holes in cards or paper tape, print letters and numbers on paper, or control the typing of typewriter like terminals.

2. Storage Devices : A storage device is mechanism by which the computer may store information (a procedure commonly called writing) in such a way that this information may be retrieved a later time (reading). Conceptually, storage devices are analogous to human storage mechanisms that use pieces of paper, pencils, and erasers.

It is helpful to differentiate among three types of storage devices. Our differentiation is based on the variation of access times (T_{ij}) where :

T_{ij} = time to access item j, given last access was to item i (or current position is item i)

We differentiate the following types of storage devices :

1. Serial Access where T_{ij} has a large variance (e.g., tape)
2. Completely Direct Access where T_{ij} is constant (e.g., core)
3. Direct Access where T_{ij} has only a small variance (e.g., drum)

1. Serial Access Devices : A serial access storage device can be characterized as one that relies on strictly physical sequential positioning and accessing of information. Access to an arbitrary, stored item requires a "linear search" so that an average access takes the time required to read half the information stored. A Magnetic Tape Unit (MTU) is the most common example of a serial access storage device. The MTU is based upon the same principles as the audio tape deck or tape cassette, but instead of music or voices, binary information is stored. A typical serial access device is depicted in fig. 1.

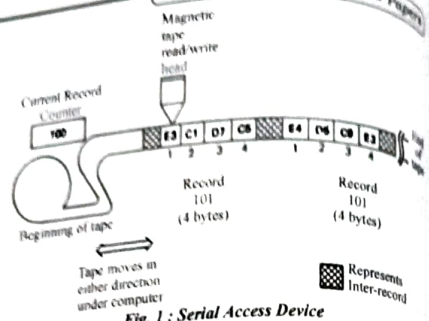


Fig. 1 : Serial Access Device

Information is usually stored as groups of bytes, called records, rather than single bytes. In general, a record can be of arbitrary length, e.g., from 1 to 32,768 bytes long. In figure 1 all the records are four bytes long. Records may be viewed as being analogous to spoken words on a standard audio tape deck. Each record can be identified by its physical position on the tape; the first record is 1, the second, 2, and so on. The current record counter of figure 1 serves the same function as the "time" or "number-of-feet" meter of audio tape decks. If the tape is positioned at its beginning (i.e., completely rewound), and we wish to read record number 101, it is necessary to skip over all interviewing records (e.g., 1, 2, ..., 99, 100) in order to reach record number 101.

In addition to obvious I/O commands, such as Read the Next Record or Write the Next Record, a tape unit usually has commands such as Read the Last Record (read backward), Rewind to Beginning of Tape at High Speed, Skip Forward (or Backward) One Record without Actually Reading or Writing Data. In order to help establish groupings of records on a tape, there is a special record type called a tape mark. This record type can be written only by using the Write Tape Mark command. If the tape unit encounters a tape mark while reading or skipping, a special interrupt is generated. Tape marks are somewhat analogous to putting bookmarks in a book to help find your place.

Serial access storage devices are normally used for applications that require mainly sequential accessing, such as the copies of punched card decks, or the intermediate file copies produced by pass 1 of an assembler or loader. In current data processing, a large percentage of the applications are primarily sequential.

The Inter-record Gap (IRG) shown in figure 1 is necessary due to the physical limitations of starting and stopping a tape. In order to read the next record in a stopped tape, the tape drive takes a finite amount of time to get up to speed. Similarly, after a record has been read, some tape goes by before it an be stopped.

To minimize gap waste, we frequently block records. Blocking is placing multiple logical records into one physical record (a block). Figure 2 depicts four 80-byte logical records (cards) in one 320-byte physical record (block).

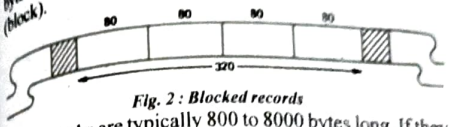


Fig. 2 : Blocked records

Blocks are typically 800 to 8000 bytes long. If they were any longer, there would be reliability problems and when read in, they would take up too much memory.

(ii) Completely Direct Access Devices : A completely direct access device is one in which the access time T_{ij} is a constant. Magnetic core memory, semiconductor memory, read-only wired memories, and diode matrix memories are all examples of completely direct access memories.

Figure 3 depicts the working of a core memory.

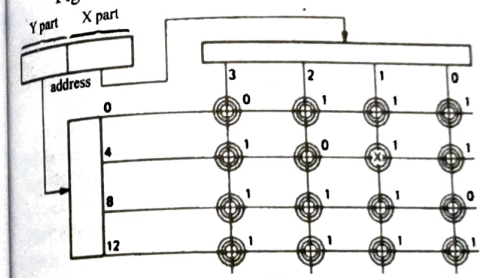


Fig. 3 : Core Memory

In figure 3 the 16 circles represent magnetic ferrite cores. All cores are connected by wires, each core having two selection wires through it. When the hardware receives a read request for some address, the address is decomposed into an x and y portion. The hardware then selects the appropriate core. This is done by passing current through the two selection wires. Only the core at the specified (x,y) coordinate received a "double dose" of current; all other cores receive half the amount of current or no current at all. The double dose of current is sufficient to "flip" the magnetic field from 1 to 0.

One property of a magnetic core is that when it changes magnetic state it induces a current. A third sensing wire is passed through all the cores. If the appropriate core was 1 and it was switched to 0, a current would be induced in this wire. If it was already 0, there would be no induced current.

For example, in figure 3, if a request to read bit 5 (binary 0101) were received, the hardware would decompose the address :

$$\underset{y}{01}\ \underset{x}{01} \quad \begin{matrix} y = 4 \\ \text{or} \\ x = 1 \end{matrix} \quad \text{and}$$

(iii) Direct Access Storage Devices (DASD) : A direct access device is one that is characterized by small variances in the access time T_{ij}. These have been called Direct Access Storage Devices. Although they do not offer completely direct access, the name persists in the field.

In this section we give two examples of DASD, fixed-head and moving-head devices.

(a) Fixed-head Drums and Disks : Figure 4 depicts a DASD storage device similar to a magnetic drum.

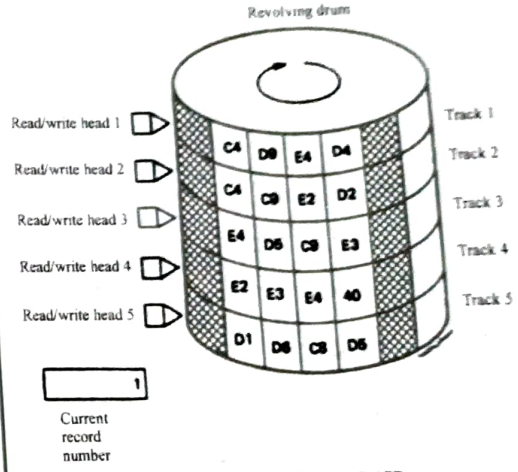


Fig. 4 : Fixed-head Drum DASD

A magnetic drum can be simplistically viewed as several adjacent strips of magnetic tape wrapped around a drum so that the ends of each tape strip join. Each tape strip, called a track, has a separate read/write head. The drum continuously revolves at high speed so that the records repeatedly pass under the read/write heads (e.g., record 1, record 2, ..., then record 1 again, record 2, ..., etc.). Each individual record is identified by a track number and then a record number. For example, record (2, 1) is X'C4C9E2D2', and record (5, 1) is X'D1D6C8D5' in figure 4. Typical magnetic drums spin very fast and have several hundred read/write heads; thus a random access to read or write can be accomplished in 5 or 10 ms in contrast to the minute required for random access to a record on a full magnetic tape.

Unlike the MTU, which could directly access only the preceding or following record, a DASD can access any record stored on the device. Thus it is necessary to specify the DASD address in the I/O channel commands. Logically, the I/O command would be :

READ
(1) DASD record (2,1) into
(2) memory address (14680)
(3) length (80)

(b) Moving-head Disks and Drums : The magnetic disk is similar to the magnetic drum but is based upon the use of one or more flat disks, each with a series of concentric circles, one per read/write head. This is analogous to a phonograph disk. Since read/write heads are expensive, some devices do not have a unique head for each data track. Instead, the heads are physically moved from track to track; such units are called moving-arm or moving-head DASDs. Each arm position is called a cylinder.

In order to identify a particular record stored on the moving-head DASD shown in figure 5, it is necessary to specify the arm position, track number, and record number. Notice that arm position is based upon radial movement whereas record number is based upon circumferential movement. Thus, to access a record, it is necessary to move to the correct radial position (if not already there) and then to wait for the correct record to rotate under the read/write head.

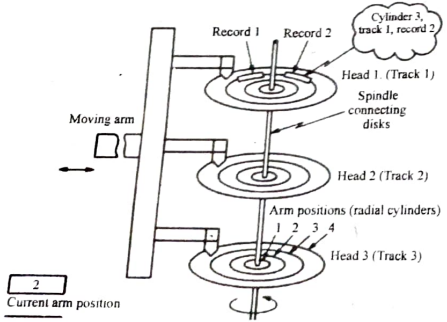


Fig. 5 : Moving-head Drum

Q.19 ***Define device handling by device handler in detail.***

Ans. Device Handlers : Device handler is a process which is responsible for servicing the requests on a device request queue and for notifying the originating process when the service has been completed. There is a separate handler for each device, but since all handlers operate in a similar way they can make use of shareable programs. Any differences in behaviour between handlers are derived from the characteristics stored in the descriptors of the particular devices.

A device handler operates in a continuous cycle during which it removes an IORB from the request queue, initiates the corresponding I/O operation, waits for the operation to be completed, and notifies the originating process. The complete cycle for an input operation is

```
repeat indefinitely
begin wait (request pending);
    pick an IORB from request queue;
    extract details of request;
    initiate I/O operation;
    wait (operation complete);
    if error then plant error information;
    translate character(s) if necessary;
    transfer data to destination;
    signal (request serviced);
    delete IORB
end;
```

The following notes, which should be read, may clarify the details.

1. The semaphore request pending, contained in the device descriptor, is signalled by the I/O procedure each time it places an IORB on the request queue for this device. If the queue is empty the semaphore is zero and the device handler is suspended.
2. The device handler may pick an IORB from the queue according to any desired priority algorithm. A first come, first served algorithm is usually adequate, but the handler could be influenced by priority information placed in the IORB by the I/O procedure. In the case of a disk the device handler may service requests in an order which minimizes head movements.
3. The instruction to initiate the I/O operation can be extracted from the device characteristics held in the device descriptor.
4. The semaphore operation complete is signalled by the interrupt routine after an interrupt is generated for this device. The outline program of the interrupt routine, which is entered from the first level interrupt handler, is

```
locate device descriptor;
signal(operation complete);
```

The semaphore operation complete is held in the device descriptor. Note that the interrupt routine is very short; all housekeeping is performed by the

device handler, which operates in user mode as a normal process.

5. The error check is made by interrogating the device status when the operation is complete. If there has been an error, such as hardware detected data error, printer out of paper, or even end of file, information about is it deposited in an error location whose address is included by the I/O procedure in the IORB.
6. Character translation is made according to the mode specified in the IORB and the tables pointed to by the device descriptor.
7. The cycle shown above is for an input operation. In the case of an output operation, the extraction of data from its source and any character translation occur before the operation rather than after.
8. The address of the semaphore request services is passed to the device handler as a component of the IORB. It is supplied by the process requesting I/O, as a parameter of the I/O procedure.

The synchronization and flow of control between the process requesting I/O, the I/O procedure, and the appropriate device handler and interrupt routine are summarized in figure. Solid arrows represent transfers of control; broken arrows represent synchronization by means of semaphores. It is important to note that in the scheme illustrated the requesting process proceeds asynchronously with the device handler so that it may perform other computations or make other I/O requests while the original request is being services. The requesting process need be held up only on occasions when the I/O operation is incomplete when it executes wait (request serviced).

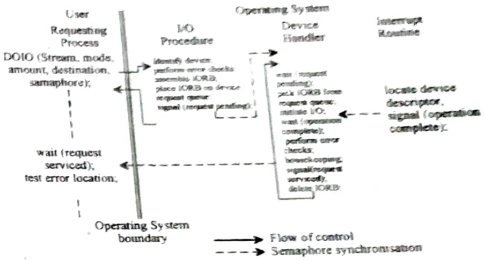


Fig. : Sketch of the I/O System

File Management 4

Previous Years Questions

Part-A

Q.1 ***Define sequential access files.***

Ans. Sequential files are those which can be read sequentially (one record, then another and so on) in an order, starting at the beginning to the end of file.

Q.2 ***What do you mean by file system.***

Ans. It is a method of organizing files on physical media such as hard disk, CD's and flash drives.

Q.3 ***Write the basic operations on files.***

Ans. Basic Operations on Files :

(i) Creating a file
(ii) Writing a file
(iii) Reading a file
(iv) Deleting a file
(v) Truncating a file
(vi) Repositioning within a file

Q.4 ***What do you mean by truncating a file operation.***

Ans. Truncating a File : User may want to erase contents of file but keep its attributes. Rather than forcing the user to delete a file and then recreate it, truncation function allows all attributes to remain unchanged except for file length.

Q.5 ***Write advantage of contiguous allocations in a file system.***

Ans. Advantages of Contiguous Allocations

(i) It is simple to implement.
(ii) Due to contiguous (or continuous) allocations, the performance is good.

Part-B

Q.6 ***What are the various access methods for file system?*** *[R.T.U. 2017, Dec. 2013]*

Ans. File Access Methods

Files can be categorized broadly into two types depending upon their access methods.

1. Sequential Access Files : Sequential files are those, which are read sequentially (one record, then another and so on) in an order, starting at the beginning to the end of the file. Sequential files can be rewound so that they can be read as often as needed. However, records in such files cannot be read out of order e.g. reading of 34th record followed by 5th record and then 1st record is not possible with sequential files. Sequential files are convenient when the storage media is serial access e.g. magnetic tape rather than direct-access e.g. magnetic disk.

2. Random Access Files : Files whose records can be read in any order are called random access files. Random access files are basically a collection of records stored on a disk and these records are numbered 1, 2, 3 and so on and therefore, can be referred to by their numbers instead of their position. Such files should be necessarily be stored on direct access media like disk.

Random access files are essential for many applications, for example, database systems.

In a banking application, a customer may want to look up his current balance. This can be easily done by locating this customer's record using his account number as a key, rather than sequentially reading the records for thousands of other customers before this customer's record is located and read.

It is worth mentioning here that not all operating systems support both sequential and random access for files. Some systems allow only sequential file access, others allow only random access. Some systems require that a file should be defined as sequential or random when it is created, so that it can be accessed in the way it has been declared.

Q.7 ***Define file system? Explain file operations in detail?*** *[R.T.U. 2016]*

Ans. File System : A file system is a method of organizing files on physical media, such as hard disks, CD's, and flash drives. In the Microsoft Windows family of operating systems, users are presented with several different choices of file systems when formatting such media. These choices depend on the type of media involved and the situations in which the media is being formatted. Common file systems in Windows are as follows :

- NTFS
- FAT
- exFAT
- HFS +
- EXT

File Operations :

A file is an abstract data type. Operation on file, operating system provides system calls for creating, deleting, read etc. Basic operations on files are :

1. Create a file
2. Writing a file
3. Reading a file
4. Delete a file
5. Truncating a file
6. Repositioning within a file

1. Create a file : For creating a file, address space in the file system is required. After creating a file, entry of the file is made in the directory. The directory entry records the name of the file and the location in the file system.

2. Writing a file : System call is used for writing into file. It is required to specify the name of the file and information to be written to the file. According to the file name, system will search the name in the directory to find the location of the file.

3. Reading a file : To read a file, system call is used. It requires the name of file and memory address. Again the directory is searched for the associated directory entry and the system needs to keep a read pointer to the location in the file where the next read is to take place.

4. Delete a file : System will search the directory, which file to be deleted. If directory entry is found, it releases all file space. That free space can be reused by another (user) files.

5. Truncating a file : *Refer to Q.4.*

6. Repositioning within a file : The directory is searched for the appropriate entry, and the current file position is set to a given value. Repositioning within a file does not need to involve any actual I/O. This file operation is also known as file seek.

Q.8 ***Explain various features of file system of linux.*** *[R.T.U. 2015]*

Ans. In Linux, everything is configured as a file. This includes not only text files, images and compiled programs (also referred to as executables), but also directories, partitions and hardware device drivers.

Each filesystem contains a control block, which holds information about that filesystem. The other blocks in the filesystem are inodes, which contain information about individual files and data blocks, which contain the information stored in the individual files.

There is a substantial difference between the way the user sees the Linux filesystem and the way the kernel (the core of a Linux system) actually stores the files. To the user, the filesystem appears as a hierarchical arrangement of directories that contain files and other directories (i.e., subdirectories). Directories and files are identified by their names. This hierarchy starts from a single directory called root, which is represented by a "/" (forward slash).

The Filesystem Hierarchy Standard (FHS) defines the main directories and their contents in Linux and other Unix-like operating systems. All files and directories appear under the root directory, even if they are stored on different physical devices (e.g., on different disks or on different computers). A few of the directories defined by the FHS are /bin (command binaries for all users), /boot (boot loader files such as the kernel), /home (users home directories), /mnt (for mounting a CD-ROM or floppy disk)

/root (home directory for the root user), /sbin (executables used only by the root user) and /usr (where most application programs get installed).

To the Linux kernel, however, the filesystem is flat. That is, it does not

(1) have a hierarchical structure
(2) differentiate between directories, files or programs
(3) identify files by names.

Instead, the kernel uses inodes to represent each file.

An inode is actually an entry in a list of inodes referred to as the inode list. Each inode contains information about a file including:

(1) its inode number (a unique identification number)
(2) the owner and group associated with the file
(3) the file type (for example, whether it is a regular file or a directory)
(4) the file's permission list
(5) the file creation, access and modification times
(6) the size of the file
(7) the disk address (i.e., the location on the disk where the file is physically stored)

Q.9 ***Write short note on File naming.*** *[R.T.U. 2014]*

Ans. File Naming : A file is the smallest allotment of secondary storage device e.g. disks. Logically a file is sequence of logical records *i.e.* a sequence of bits and bytes. A file has various attributes like name, type, location, size, protection, time and data of creation, user information etc. various attributes from the user's point of view.

The exact rules for file naming vary somewhat from system to system, but all operating systems allow strings of one to eight letters as legal file names. Frequently digits and few special characters are also permitted therefore, a file names like 'operatingsystem1', 'Sys_Prog' are also permitted. Similarly, some file system distinguish between upper case and lower case letters, whereas other do not.

Almost every operating system supports two parts of a file name, where the two parts are separated by a period, as in 'os.c'. The part following the period is .c called the file extension, and usually indicates something about the file. The size of this extension can be upto 3 characters is MS-DOS operating system whereas, UNIX permits that a file may have two or more extensions like 'os.c.z'., where '.z' indicates that a file has been compressed or zipped. Table shows some typical file extensions.

Table : Some Typical File Extensions

EXTENSION	MEANING
file.bak	Backup file
file.bas	BASIC source program
file.bin	Executable binary program
file.c	C source program
file.dat	Data file
file.doc	Documentation file
file.ftn	FORTRAN source program
file.hlp	Text for HELP command
file.lib	Library of object files used by the linker
file.man	Online manual page
file.obj	Object file (compiler output, not yet linked)
file. Pas	Pascal source program
file.txt	General text file

File extension are important to know because they indicate a particular software needed to open a particular file e.g. a C complier will compile only '.c' files and not any other file. Looking at the file "os.c", the user can identify that this is a program file in 'C' language and can be opened and complied using a 'C' complier.

Q.10 ***Write short note on Directory structure in Linux.*** *[R.T.U. 2014]*

Ans. Directory Structure in Linux : The files are represented by entries in a device directory or volume table or contents. The device directory records information, such as name, location, size, and type, for all files on that device. A device directory may be sufficient for a single user system with limited storage space. As the amount of storage and the number of user increases, it becomes increasingly difficult for the users to organize and keep track of all of the files. The solution of this problem is the imposition of a directory structure on the file system. A directory structure provides a mechanism for organizing the many files in the file system.

Many system actually have two separate directory structures, the device directory and the file directories. The device directory is stored on each physical device and describes all files on that device. The device directory entry mainly concentrates on describing the physical properties of each file, where it is, how long it is, how it is allocated, and so on. The file directories are a logical organization of the files on all devices. The file directory entry concentrates on logical properties of each file: name, file type, owing users, accounting information, protection access codes, and so on. Operation which are to be performed on a directory as follows:

Different Directory Structures

1. Single-level Directory : The simplest directory structure is the single-level directory. The device directory is an example of a single-level directory. All files are contained in the same directory, which is very easy to support and understand.

A single-level directory has significant limitations, however, when the number of files increases or when there is more than one user. Since all files are in the same directory, they must have unique names. If we have two users who call their test data file TEST, then the unique name rule is violated.

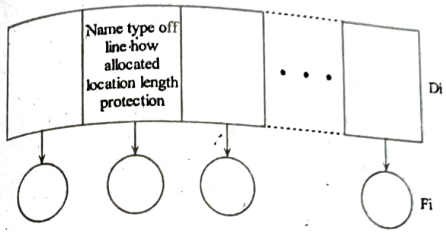


Fig. : Single level directory

Even with a single user, as the number of files increases, it becomes difficult to remember the names of all the files in order to create only files with unique names.

2. Two-Level Directory : In the two-level directory structure, each user has his own **user file directory (UFD).** The UFDs have similar structures, but each lists only the files of a single user. When a job starts or a user logs in, the system's **master file directory (MFD)** is searched. The MFD is indexed by user name and account number and each entry points to the UFD for that user.

When a user refers to a particular file, only his own UFD is searched. Thus, different users may have files with the same name, as long as all the file names within each UFD are unique. To create a file for a user, the operating system searches only that user's UFD to as certain whether another file of that name exists. To delete a file, the operating system confines its search to the local UFD; thus, it cannot accidentally delete another user's file that has the same name.

The user directories themselves must be created and deleted as necessary. A special system program is run with the appropriate user name and account information. The program creates a new UFD and adds an entry for it to the MFD. The execution of this program might be restricted to system administrations.

Although the two - level directory structure solves the name - collision problem, it still has disadvantages.

This structure effectively isolates one user from another. Isolation is an advantage when the users are completely independent but is a disadvantage when the users want to cooperate on some task and to access one another's files. Some systems simply do not allow local files to be accessed by another users.

If access is to be permitted; one user must have the ability to name a file in another user's directory. To name a particular file uniquely in a two - level directory, we must give both the user name and the file name.

A two - level directory can be thought of as a tree or an inverted tree, of height 2. The root of the tree is the MFD. Its direct descendants are the UFDs. The descendants of the UFDs are the files themselves. The files are the leaves of the tree. Specifying a user name and a file name defines a path in the tree from the root (the MFD) to a leaf (the specified file). Thus, a user name and a file name define a path name. Every file in the system has a path name. To name a file uniquely, a user must know the path name of the file desired.

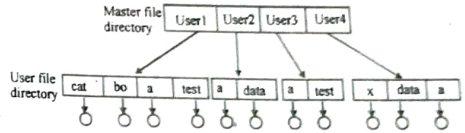


Fig. : Two level directory structure

For example, if user A wishes to access her own test file named test, she can simply refer to test. To excess the file named test of user B (with directory - entry name user B), however, she might have to refer to user B/ test. Every system has its own syntax for naming files in directories other than the user's own.

Q.11 ***Write short note on File organization.*** *[R.T.U. 2014]*

Ans. File Organization : A file consists, generally speaking, of a collection of records, a key element in file management is the way in which the records themselves are organized inside the file, since this heavily affects system performances and far as record finding and access. Note carefully that by "organization" we refer here to the logical arrangement of the records in the file (their ordering or, more generally, the presence of "closeness" relations between them based on their content), and not instead to the physical layout of the file as stored on a storage media, To prevent confusion, the latter is referred to by the expression "record blocking", and will be treated later on.

Choosing a file organization is a design decision, hence it must be done having in mind the achievement of good performance with respect to the most likely usage of the file. The criteria usually considered important are:

1. Fast access to single record or collection of related records.
2. Easy record adding/update/removal, without disrupting.
3. Storage efficiency.
4. Redundance as a warranty against data corruption.

Logical data organization is indeed the subject of whole shelves of books, in the "Database" section of your library. Here we'll briefly address some of the simpler used techniques, mainly because of their relevance to data management from the lower-level (with respect to a database's) point of view of an OS. Five organization models will be considered:

- Pile.
- Sequential.
- Indexed-sequential.
- Indexed.
- Hashed.

Q.12 ***Explain the file system mounting.***

[R.T.U. Dec. 2013]

Ans. File System Mounting: As we know that a file must be opened before it is used similiarly, a file system must be mounted before it can be available to processes on the system. The directory structure can be bilt out of multiple volumes which must be mounted to make them available within the file system name space.

The mount procedure is as follows :

• The OS is given the name of the device and the mount point (the location within the file structure where the file system is to be attached.)

For instance, on a UNIX system, a file system containing a user's home directories might be mounted as home; then to access the directory structure within that file system, we could precede the directory names with / home; as in / home / dheer.

Mounting that file system under / users would result in the path name / users / dheer, which we could use to reach the same directory.

• Next the OS verifies that the device contains a valid file system. It does so by asking the device driver to read the device directly and verifying that the directory has the expanded format.

• Finally, the OS notes in its directory structure that a file system is mounted at the specified mount point.

To illustrate file mounting, consider the file system depicted in fig. .

The Microsoft Windows family of operating systems (95,98,NT, small 2000, XP) maintains an extended two-level directory structure, with devices and volumes assigned drive letters. Volumes have a general graph directory structure associated with the drive letter. The path to a specific file takes the form of *drive - letter: / path/ to/file*. The more recent versions of Windows allow a file system to be mounted anywhere in the directory tree, just as UNIX does. Windows operating systems automatically discover all devices and mount all located file systems at boot time. In some systems, like UNIX, the mount commands are explicit. A system configuration file contains a list of devices and mount points for automatic mounting at boot time, but other mount may be executed manually.

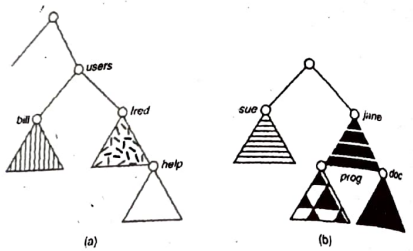


Fig. : File system (a) Existing system. (b) Unmounted volume

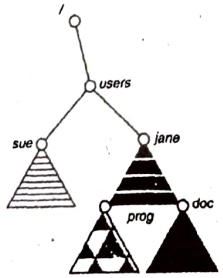


Fig. : Mount point

Q.13 ***Write short note on attributes of files.***

[R.T.U. 2013]

Ans. File Attributes: A file's attribute may vary from one operating system to another, but typically consists of:

Table : Some possible file Attributes

Field	Meaning
Protection	Who can access the file and in what way
Password	Password needed to access the file
Creator	ID of person who created the file
Owner	Current owner
Read only flag	0 for read/ write, 1 for read only
Hidden flag	0 for normal, 1 for do not display in listings
System flag	0 for normal file, 1 for system file
Archive flag	0 has been backed up, 1 for need to be backed up
ASCII/binary flag	0 for ASCII file, 1 for binary file

Field	Meaning
Random access flag	0 for sequential access only, 1 for random access
Record length	Number of bytes in a record
Key position	Offset of the key within each record
Key length	Number of bytes in the key field
Creation time	Data and time file was created
Time of last access	Data and time file was last accessed
Time of last change	Data and time file was last changed
Current size	Number of bytes in the file
Maximum size	Maximum size file may grow to

1. **Name :** A file is named, for the convenience of the user in human readable form.
2. **Identifier :** This is a unique tag usually a number, which identifies the file within the file system. It is non-human-readable.
3. **Size :** The current or the maximum allowed size of the file is included in this attribute.
4. **Location :** This is a pointer to the device and to the location of the file on that device.
5. **Protection :** File must be readable or writable to some specific users only. Access control information determines who can do read write or execute a file. This is normally done by giving file permissions and protection bits.
6. **Time, date and user identification :** This information contains creation, last modification and last use of the file.
7. **Type :** Different systems support different formats of files. For such purpose this attribute is used.

PART-C

Q.14 ***Explain the classification of Allocation Methods?***

[R.T.U. 2018, 2016]

Ans. Classification of Allocation Methods : There are two major methods of disk space allocation which are widely in use:

1. Contiguous Allocation

In this method, the files are assigned to contiguous areas of secondary storage. The file size area of the file to be created in specified by the user in advance. If the desired amount of contiguous space is not available then the file cannot be created. In this method, all successive records of a file are normally physically adjacent to each other. This increases the accessing speed of records. So, the scattered records (throughout the disk) are accessed slowly. This scheme of contiguous allocation is simple. For sequential access, the file system remembers the disk address of the last block and when access reads the next block. Say, for direct access to block, b, of a file that starts at location, C, we can immediately access block (C + b). Thus, contiguous allocation supports both sequential and direct accessing.

The file directories in contiguous allocation systems are easy to implement. Please note that for each file, it is necessary to retain the starting address of the file and the files size. Also note that if the size of the file is N blocks and starts at location, L, then it occupies blocks L, L+1, . . ., L+N – 1. As shown in fig.(1), the directory entry for each file indicates the address of the starting blocks and the size of the length for this file.

File	Start	Size
F1	1	3
F2	10	2
F3	7	1
F4	5	2

Directory

Fig. 1 : Directory

For this directory, the block allocation is done as follows :

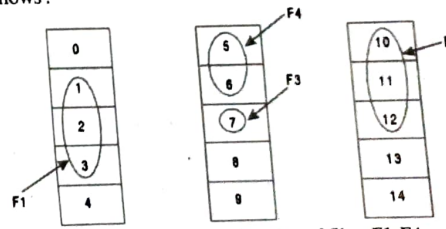


Fig. 2 : Contiguous allocation of files, F1-F4

The problem here, is that how to find the space for a new file if the implementation of a free space list is done through a bit-map method. It is so because then for the creation of a n-bytes long file, we need to find n, 0-bits a row.

Advantages of Contiguous Allocations : *Refer to Q.5.*

Disadvantages of Contiguous Allocations

(i) It is not feasible unless the maximum file size is known at the time of file creation.

(ii) Disc fragmentation may occur here. Compaction cannot be done as Disk's compaction is quite costlier.

(iii) It suffers from dynamic storage allocation problem *i.e.*, how to satisfy a request of size, N, from a list of free holes (or unallocated, free segment).

(iv) Major problem is how to determine the space needed for a file. This is not a problem during file-copy task but for other applications, this is quite difficult and unreliable.

We may have to make an initial guess also, many a times. If the file turns out to be larger than the guessed size then its extension can be placed in some other disk area. This is called as file overflow.

2. Non-contiguous Allocation

As we know that files tend to either grow or shrink overtime. So, contiguous storage allocations are being replaced by more dynamic non-contiguous storage allocation systems.

(i) Linked allocation

(ii) Indexed allocation

(i) Linked Allocation : This method of allocation is a disk-based version of the linked list. Here, each file is the linked list of disk blocks. Please note that these disk blocks may be scattered throughout the disk. A few bytes of each disk-block contains the address of the next block. Also note that the directory contains a pointer to the first and last blocks of the file.

Consider an example to understand this. Say, that directory is :

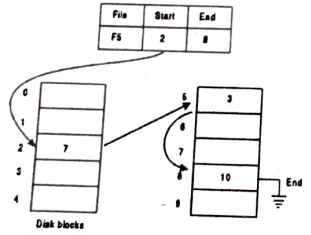


Fig. 3 : Linked list allocation of disk blocks

As shown in fig. 3 there is a file of 2 disk blocks, which starts at block-2 and ends at block-8.

Advantages of Linked Allocation

(a) Blocks are completely utilize here. So, no disk fragmentations.

(b) The disk address of first block can be used to locate the rest of the blocks.

Disadvantages of Linked Allocation

(a) It is not an efficient scheme because the list traversal needs to read each block which is quite time consuming.

(b) Pointers also consume more memory.

(c) No support for direct accessing as blocks are scattered all over the disk.

(ii) Indexed Allocation : In this scheme, each file is provided with its own index block, which is an array of disk block pointers i.e., addresses. The N^{th} entry in the index block points to the N^{th} disk block of the file. The directory contains the address of the index block. Please note that to read the N^{th} disk block, the pointer in the N^{th} index block entry is used to find the desired block and then read. Also note that these index blocks are just like the page map tables in our paged memory systems.

For example, Say disk blocks are of size 512 bytes, pointers take 4 bytes, then the index block can contain upto 128 pointers to disk blocks. This means that the largest file in such a system cannot exceed 128 blocks of 512 bytes each.

Please note that if file size exceeds 64 KB then we may use two-level indexing as shown in fig. 4 :

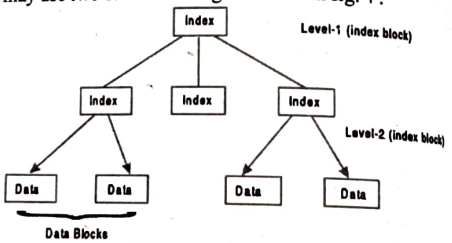


Fig. 4 : Two-level indexing

If we want to search for a disk block in this scheme, the O.S. uses the first level index to find the second level index which in turn contains pointers to the disk blocks. This can continue till level-3 or level-4 also but usually two-levels are sufficient. This increase our addressing capacity to almost double, thereby supporting file sizes upto 8 MB and 1 GB of file spaces.

Advantages of Indexed/Grouped Allocation

The following advantages of indexed allocation are:

(a) It supports both sequential and direct access. So, they are very popular.

(b) These block indexes are not physically stored as the part of FAT (File Allocation Table).

(c) No external fragmentation.

(d) Indexing of free space can be done by means of the bit map.

(e) Entire block is available for data as no space is occupied by pointers.

Disadvantages of Indexed/Grouped Allocation

The following disadvantages of indexed allocation are :

(a) Larger number of disk accesses is necessary to retrieve the address of the target block on disk. So, we have to keep some index blocks in memory which is dangerous if disk is large.

(b) It requires lot of space for keeping pointers. Here, pointer overhead is more as compared to the pointer overhead of linked-list allocation.

(c) Most files are generally small. If we use linked allocation for these files of one or two blocks then we only need a space for one pointer per block whereas in an indexed allocation, an entire index block must be allocated even if one or more pointers are null.

(d) Indexed allocation is more complex and time consuming.

Q.15 ***Explain various file system features of windows operating system.*** *[R.T.U. 2015]*

Ans. File system : *Refer to Q.7.*

The NTFS file system

NTFS (New Technology File System) is a modern, well-formed file system that is most commonly used by-Windows Vista, 7 and 8. It has feature-rich, yet simple organization that allows it to be used on very large volumes.

NTFS has the following Properties :

- NTFS partitions can extend up to 16EB (about 16 million TB).
- Files stored to NTFS partitions can be as large as the partition.
- NTFS partitions occasionally become fragmented and should be defragmented every one to two months.
- NTFS partitions can be read from and written by Windows and Linux systems and, can only be read from by Mac OS X systems (by default). Mac OS X, with the assistance of the NTFS-3G driver, can write to NTFS partitions. Installation instructions for the NTFS-3G driver can be found here: Mac OS X - Writing to NTFS drives.

It is recommended that NTFS be used on all media which use is primarily with modern Windows systems. It should not be used for devices which need to be written to by Mac OS X systems or on media that is used in devices which are not compatible with NTFS.

The FAT file system

The FAT (File Allocation Table) file system is a general purpose file system that is compatible with all major operating systems (Windows, Mac OS X, and Linux/Unix). It has relatively simple technical underpinnings, and was the default file system for all Windows operating systems prior to Windows 2000. Because of its overly simplistic structure, FAT suffers from issues such as over-fragmentation, file corruption, and limits to file names and size.

The FAT file system has the following properties :

- FAT partitions cannot extend beyond 2TB.
  NOTE : Windows cannot format a disc larger than 32 GB to FAT32, but Mac OS X can.
- Files stored to a FAT partition cannot exceed 4GB.
- FAT partitions need to be defragmented often to maintain reasonable performance.
- FAT partitions larger than 32GB are generally not recommended as that amount of space starts to overwhelm FAT's overly simplistic organization structure.

FAT is generally used only for devices with small capacity where portability between operating system is paramount. When choosing a file system for a hard disk, FAT is not recommend unless we are using an older version of Windows.

NOTE: This section refers to the FAT32 file system. Some early versions of Windows 95 used the FAT16 file system, which had even more technical issues and stricter limitations. It is recommended that FAT16 is never used on any modern media.

The exFAT file system

The exFAT (Extended File Allocation Table) is a Microsoft file system that is compatible with Windows and Mac OS 10.6+. It is also compatible with many media devices such as TVs and portable media players.

exFAT has the following properties:

- exFAT partitions can extend up extremely large disc sizes. 512 TB is the recommended maximum.
- Files up to 16 EB can be stored on an exFAT partition.
- exFAT is not compatible with linux/Unix.
- exFAT partitions should be defragmented often.
- exFAT cannot pre-allocate disk space.

The HFS + file system

HFS + (Hierarchical File System) is a file system developed by Apple for Mac OS X. It is also referred to as Mac OS Extended.

HFS + has the following properties:

- Maximum volume is 8 EB (about 8 million TB).
- Files stored to HFS+ partitions can be as large as the partition.
- Windows users can read HFS+ but not write.
- Drivers are available that allow Linux users to read and write to HFS+ volumes.

The EXT file system

The extended file system was created to be used with the Linux kernel. EXT4 is the most recent version of EXT.

EXT4 has the following Properties:

- EXT4 can support volumes up to 1 EB.
- 16 TB maximum file size.
- Red Hat recommends using XFS (not EKT4) for volumes over 100 TB.
- EXT4 is backwards compatible with EXT2 and EXT3.
- EXT4 can pre-allocate disk space.
- By default, Windows and Mac OS cannot read EXT file systems.

Q.16 Define security and user authentication in file system.

Ans. Security and user Authentication in File System

Cryptography as a security tool

- Cryptography can be utilised as a tool for computer security.
- In a networked system, an operating system can never be absolutely sure about the identity of its communication partner.
- Cryptography can help here to remove the necessity to trust the network.
- Constraints the number of potential senders/ receivers of a message.
- Typically based on keys that are selectively distributed to computers in a network.

Encryption :

- Encryption constrains the possible receivers of a message.

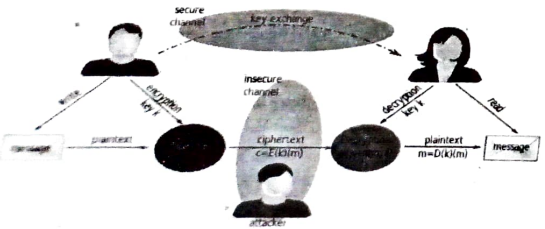


Fig.

Symmetric Encryption :

- The same key is used to encrypt and to decrypt.
- The secrecy of E(k) must be protected as well as D(k).
- If the same key is utilised for an extended amount of data, it becomes vulnerable to an attack.
- **Examples:** AES (advanced encryption standard), DES (data encryption standard).

Asymmetric Encryption :

- Different encryption and decryption keys are used.
- We distinguish between public and private keys.
- Uses one-way function for which the inverse operation is much harder to execute (e.g. factorisation).
- Exkurs: RSA Cryptosystems

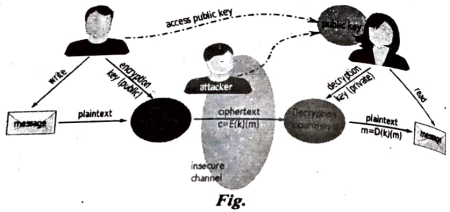


Fig.

Authentication

- Encryption constrains the amount of potential senders of a message.
- Constraining the amount of possible receivers of a message is called authentication.
- Authentication is complementary to encryption.
- Observe, that an encrypted message can also prove the identity of the sender.
- Authentication is also useful for proving that a message has not been modified.

Key Distribution

- With symmetric cryptography, the transmission of the key becomes a great challenge and possible vulnerability out-of-band transmission paper document, conversation, untypical transmission technique/band.
- Since this might be insecure and is possible also very elaborate, public key cryptosystems are utilised.
- Since public keys need not be secured, a key distribution center can be utilised.
- Vulnerable to man-in-the-middle-attacks.
- With trusted authorities the authentication of individual communication partners can then be verified.
- The system has also to authenticate the user.

- Typically based on one or more of the following.
  - The possession of something (key or card)
  - Knowledge of something (identifier and password)
  - Attribute of the user (fingerprint, retina pattern, signature)

Issues with Passwords

- How to keep the password stored and save in the system.
- In UNIX systems, encryption is used to secure the password.
- However, with brute force attacks, these passwords may be decoded in acceptable time off-line
- Therefore, the file containing the passwords is often visible to a superuser only.
- Programs that use the password run setuid to root in order to be allowed to read the file.
- Typically based on one or more of the following :
  - The possession of something (key or card)
  - Knowledge of something (identifier and password)
  - Attribute of the user (fingerprint, retina pattern, signature)

❑❑❑

UNIX AND LINUX OPERATING SYSTEM 5

PREVIOUS YEARS QUESTIONS

PART-A

Q.1 ***Describe the function of process scheduler in Linux OS.***

Ans. Process Scheduler (SCHED) is responsible for controlling process access to the CPU. The scheduler enforces a policy that ensures that processes will have fair access to the CPU, while ensuring that necessary hardware actions are performed by the kernel on time.

Q.2 ***Why NFS protocol is used?***

Ans. The NFS protocol provides a set of RPCs for remote file operations. The protocol support the following operations :

- Searching for a file within a directory
- Reading a set of directory entries
- Manipulating links and directories
- Accessing file attributes
- Reading and writing files

Q.3 ***Write any two difference between Unix and Linux.***

Ans. Difference between Unix and Linux:

	Unix	Linux
1.	UNIX is propriety system.	Linux is an Open Source system.
2.	Development is targeted toward specific audience and platform.	Linux development is diverse.

Q.4 ***Define C-shell.***

Ans. C Shell : The C shell, as its name might imply, was designed to allow users to write shell script programs using a syntax very similar to that of the C programming language. It is known as *csh*.

Q.5 ***What do you mean by palm OS.***

Ans. Palm OS : Palm OS is designed for ease of use with a touchscreen-based graphical user interface.

Q.6 ***Describe the usage and functionality of the command "rm -r *" in UNIX?***

Ans. The command "rm –r *" is a single line command to erase all files in a directory with its subdirectories.

- "rm" - Is for deleting files.
- "–r" - Is to delete directories and subdirectories with files within.
- "*" - Is indicate all entries.

Q.7 ***Describe fork ( ) system call.***

Ans. The command use to create a new process from an existing process is called fork( ). The main process is called parent process and new process id called child process. The parent gets the child process id returned and the child gets 0. The returned values are used to check which process which code executed. The returned values are used to check which process which code executed.

Q.8 ***What is UNIX?***

Ans. It is a portable operating system that is designed for both efficient multi-tasking and multi-user functions. Its portability allows it to run on different hardware platforms. It was written is C and lets users do processing and control under a shell.

PART-B

Q.9 ***WSN on process scheduling in LINUX operating system.*** *[R.T.U. 2016]*

Ans. Process Scheduling

Linux considers processes as belonging to either of two main categories: realtime and others. The terms real-timer is a bit of misnomer, since it does not necessarily have to do anything with real-time activities. Instead, this term refers to more important or more urgent processes. Based on this idea, Linux process scheduling can be classified into three categories, as shown in fig. 1.

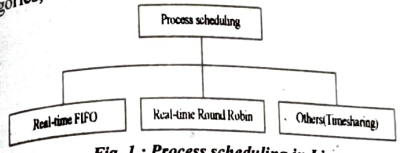


Fig. 1 : Process scheduling in Linux

The broad-level characteristics of the three scheduling types are as follows :

Real-time FIFO : This is the category which contains processes with the highest priority. These processes cannot be preempted. The only ways to preempt a processes in this category is via a new real-time FIFO process.

Real-time Round Robin: These processes are quite similar to the real-time FIFO processes, except that the CPU clock can preempt them.

Others (Timesharing): These are ordinary processes, which do not have any urgency and are scheduled by using the default timesharing algorithms.

The mechanisms used by Linux to schedule processes is quite interesting. Each process has a scheduling priority. The default value of this field is 20. This value for a process can be altered by using the "nice" system call. The syntax for "nice" is quite simple, as follows:

nice (value);

When a call to the "nice" function is made, the new scheduling priority of the thread becomes equal to:

Old scheduling priority-Value

The scheduling priority can be between 1 and 40, whereas the value can be between-20 and +19. The higher the value of the scheduling priority, the higher is the attention provided by the operating system to the process (*i.e.* more CPU time, faster response time, etc).

Each process also has another value, called quantum, associated with it. This is equal to the number of CPU clock ticks. The default clock assumed by Linux is of 100 Hz. Therefore, one tick = 100 milliseconds (ms). Each tick is called as jiffy in Linux terminology.

Based on all these concepts, the scheduler calculates the value of the goodness of a process as illustrated in fig.2. Linux always selects the process to be scheduled next, based on the value of the goodness value is scheduled.

With every clock tick, Linux reduces the value of the quantum for that process by 1. A process stops executing, if one of the following happens:

(a) The value of the quantum becomes 0, *i.e.* the time slice is over.

(b) The process needs some I/O and therefore, cannot continue.

(c) A previously blocked process with a goodness value greater than the currently executing process becomes ready.

The scheduler penodically resets the quantum of all the processes to a value based on the following formula:

Quantum (Quantum/2) + Scheduling priority

Process that have been blocked because of I/O would usually have some quantum left. Whereas the processes that have exhausted their full quantum (i.e. the ones, which are more CPU-hungry) will have the value of quantum closer to 0. In order to ensure that the CPU-hungry processes do not continue grabbing the CPU more and instead, the processes that were blocked because of I/O but are now ready, get a higher priority, this formula is devised. Let us understand how it works.

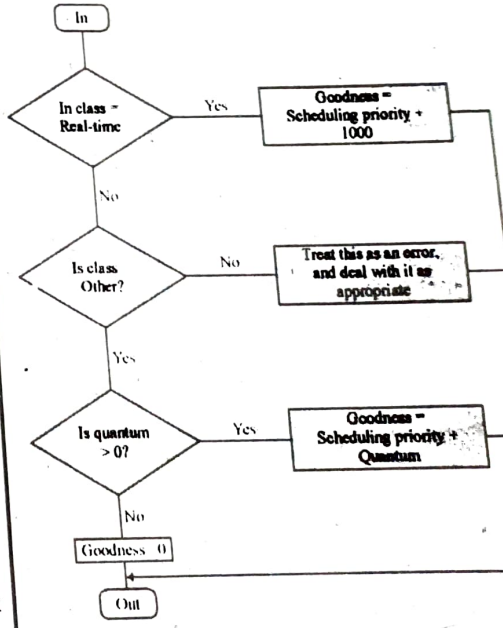


Fig.2 : Calculation of Goodness

As we have noted, for a CPU-hungry process, quantum will be closer to 0. Therefore, based on the above formula, the new quantum will be closer to its scheduling priority. On the other hand, for processes that were blocked due to I/O, but are now ready, the new quantum will also consider their old quantum

Note that by using such a mechanism, Linux automatically makes sure that the processes that are using the CPU extensively would now get a lower share of the CPU.

For inter-process synchronization, Linux implements wait queues and semaphores. Wait queues are circular linked list, which describe the process descriptors. In Linux, a semaphore consists of three fields: semaphore counter, number of waiting processes and list of processes waiting for the semaphore.

Q.10 ***Explain the network file system of LINUX in detail.*** *[R.T.U. 2016]*

Ans. The network file system of Linux : NFS views a set of interconnected workstations as a set of independent machines with independent file systems. The goal is to allow some degree of sharing among these file systems (on explicit request) in a transparent manner. Sharing is based on a client-server relationship. A machine may be, and often is, both a client and a server. Sharing is allowed between any pair of machines. To ensure machine independence, sharing of a remote file system affects only the client machine and no other machine.

One of the design goals of NFS was to operate in a heterogeneous environment of different machines, operating systems, and network architectures. The NFS specification is independent of these media and thus encourages other implementations. This independence is achieved through the use of RPC primitives built on top of an external data representation (XDR) protocol used between two implementation-independent interfaces. Hence, if the system consists of heterogeneous machines and file systems that are properly interfaced to NFS, file systems of different types can be mounted both locally and remotely.

The NFS specification distinguishes between the services provided by a mount mechanism and the actual remote-file-access services. Accordingly, two separate protocols are specified for these services a mount protocol and a protocol for remote file accesses, the NFS protocol. The protocols are specified as sets of RPCs. These RPCs are the building blocks used to implement transparent remote file access.

The NFS Protocol : *Refer to Q.2.*

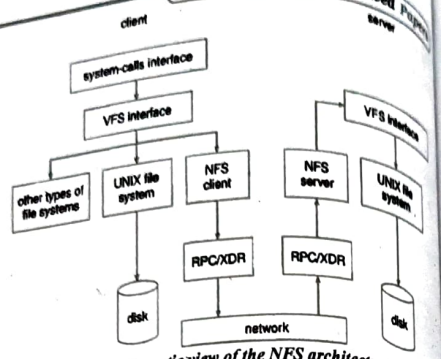


Fig. : Schematic view of the NFS architecture

Q.11 ***Write short note on security system of LINUX OS.*** *[R.T.U. 2016]*

Ans. Security System of LINUX OS : Linux's security model is closely related to typical UNIX security mechanisms. The security concerns can be classified in two groups:

1. Authentication : Making sure that nobody can access the system without first proving that she has entry rights

2. Access control : Providing a mechanism for checking whether a user has the right to access a certain object and preventing access to objects as required.

1. Authentication

A new security mechanism has been developed by UNIX vendors to address authentication problems. The pluggable authentication module (PAM) system is based on a shared library that can be used by any system component that needs to authenticate users. An implementation of this system is available under Linux. PAM allows authentication modules to be loaded on demand as specified in a system-wide configuration file. If a new authentication mechanism is added at a later date, it can be added to the configuration file, and all system components will immediately be able to take advantage of it. PAM modules can specify authentication methods, account restrictions, session setup functions, and password-changing functions (so that, when users change their passwords, all the necessary authentication mechanisms can be updated at once).

2. Access Control

Access control under UNIX systems, including Linux, is performed through the use of unique numeric identifiers. A user identifier (UID) identifies a single user or a single set of access rights. A group identifier (GID) is an extra identifier that can be used to identify tights belonging to more than one user.

Access control is applied to various objects in the system. Every file available in the system is protected by the standard access-control mechanism. In addition, other shared

objects, such as shared-memory sections and semaphores, employ the same access system.

Linux performs access control by assigning objects a protection mask that specifies which access modes-read, write, or execute-are to be granted to processes with owner, group, or world access. Thus, the owner of an object might have full read write, and execute access to a file; other users in a certain group might be given read access but denied write access; and everybody else might be given no access at all.

Q.12 ***What is mobile operating system? Explain Symbian & palm OS.*** *[R.T.U. 2015]*

Ans. Mobile Operating System : Mobile OS is an operating system for mobile devices. It is the software platform on top of which other programs, called *application programs*, can run on mobile devices such as mobile phones, smart phones, PDAs and handheld computers. Abbreviated as *mobile OS*.

Windows CE : A version of the Windows operating system designed for small devices such as personal digital assistants (PDAs) (or *Handheld PCs* in the Microsoft vernacular). The Windows CE graphical user interface (GUI) is very similar to Windows 95 so devices running Windows CE should be easy to operate for anyone familiar with Windows 95.

Symbian OS : Symbian is an operating system designed specifically for advanced mobile phones running on ARM processor. Symbian OS combines the power of an integrated applications environment with mobile telephony, bringing advanced data services to the mass market. Symbian OS includes a multi-tasking kernel, integrated telecommunication support (such as GPRS, Bluetooth, SyncML and ultimately 3G), communications protocols, data management, graphics support, a low-level graphical user interface framework and a variety of application engines.

Symbian is a Single User, preemptive multi-tasking Operating System. The multi-tasking in Symbian is priority-based with priority inheritance. It is real-time operating system designed especially for advanced mobile phones running on ARM processor. It has integrated telecommunication support, data management, graphics support and framework for variety of application engines. It integrates the power of computing in the mobile telephony. Symbian OS is not only an operating system but a full software and communications platform. This means more applications are already available for developers. Several application platforms were defined and their usages depend on the mobile phone capability integrating this OS. Its application platform are series 60, series 80 and series 90 UI of mobile devices. Technically and financially, it is very successful operating system which is dominant in today's mobile market and more than 52 mobile companies have used this Operating System.

Palm OS : Palm OS (also known as **Garnet OS**) is a mobile operating system initially developed by Palm, Inc. for personal digital assistants (PDAs) in 1996. Palm OS is designed for ease of use with a touchscreen-based graphical user interface. It is provided with a suite of basic applications for personal information management. Later versions of the OS have been extended to support smart phones. Several other licensees have manufactured devices powered by Palm OS.

Palm OS is the computer operating system that provides a software platform for the Palm series of handheld personal digital assistants (PDAs) made by Palm Inc.

Palm OS uses multitasking, but only one task is for applications. The user uses one application at a time, one application program must finish before the next can be selected. This constraint allows the operating system to devote full attention to the application that is open. The space needed by the system for any application that is running is kept in dynamic, reusable random access memory (RAM). The application and its related database are kept in what is called permanent storage, but here the permanent storage is RAM (rather than a hard disk) that cannot be reused as the dynamic RAM can. Palm OS divides an application into runnable code and different types of data elements, such as user interface elements and icons. The data elements can be easily changed without necessarily having to rewrite code.

Palm OS comes with these applications built-in: Dates, Address Book, To Do Lis, Memo Pad, Calculator, and Password Protection. New applications can be written and added using several facilities that accelerate development.

The Palm user interface is emulated within a window in the desktop environment, encouraging rapid application development. Simpler applications can be developed using Palm's forms interface.

Palm OS comes with communication interfaces to infrared transmission devices, TCP/IP (for Web connection through wireless or wireline devices), and, optionally, barcode recognition scanners.

Q.13 ***Describe the basic design principles used in Linux operating system.*** *[R.T.U. 2013]*

Ans. Design Principles : Linux is designed to be compliant with the relevant POSIX documents; some Linux distributions have achieved official POSIX certification. Linux is a

Multiuser Multitasking system with a full set of UNIX-compatible tools. Its file system adheres to traditional UNIX semantics and it fully implements the standard UNIX networking model. Main design goals are speed, efficiency, flexibility and standardization.

Table : Linux Design Principles

Design Principle	Purpose
Multiple processes & Multiple processors	Linux can run +1 program at a time using one or several processors.
Multiple platforms	Operate on Intel's x86 processors and other platforms including Alpha, Sparc, certain 6800 platforms, certain MIPS machines and Power PCs.
Multiple users	Allow several users to work on the same machine at same time.
Inter-process communications	Linux supports pipes, IPC, sockets, etc.
Terminal management	Its terminal management conforms to POSIX standards and it also supports pseudo-terminals as well as process control systems.
Peripheral devices	Supports a wide range of devices including sound cards, graphics interfaces, networks, SCSI, etc.
Buffer cache	Supports a memory area reserved to buffer the input and output from different processes.
Demand paging memory management	Loads pages into memory only when they are needed.
Dynamic and shared libraries	Dynamic libraries are loaded only when they are needed and their code is shared if several applications are using them.
Disk partitions	Linux allows file partitions used by file systems such as Ext2 and partitions having other formats (MS-DOS, ISO 9660, etc.)
Network protocol	It supports TCP/IP and other network protocols.

Q.14 ***Write short notes on File System Structure.***

[R.T.U. 2013]

Ans. File System Structure : File systems in Linux are organized in a hierarchy, beginning from root (/) and continuing downward in a structure of directories and subdirectories. As an administrator Linux system, it is your duty to make sure that all the disk drives that represent your file system are available to the users of the computer. It is also your job to make sure that there is enough disk space in the right places in the file system for users to store the information they need. Linux uses the frontslash / instead of the backslash \ as in Windows, it is because it is simply following the UNIX tradition. Linux, like Unix also chooses to be case sensitive.

File systems are organized differently in Linux than they are in MS Windows operating systems. Instead of drive letters (e.g., A:, B:, C:) for each local disk, network file system, CD-ROM, or other type of storage medium, everything fits neatly into the directory structure. Under Windows, the various partitions are detected at boot and assigned a drive letter. Under Linux, unless you mount a partition or a device, the system does not know of the existence of that partition or device. This might not seem to be the easiest way to provide access to your partitions or devices but it offers great flexibility.

This kind of layout, known as the *Unified Filesystem*, does offer several advantages over the approach that Windows uses. Let us take the example of the /usr directory. This sub-directory of the root directory contains most of the system executables. With the Linux file system, you can choose to mount it off another partition or even off another machine over the network using an innumerable set of protocols such as NFS (Sun), Coda (CMU) or AFS (IBM). The underlying system will not and need not know the difference. The presence of the /usr directory is completely transparent. It appears to be a local directory that is part of the local directory structure.

Table : Directories in Linux System

/bin	Essential command binaries
/boot	Static files of the boot loader
/dev	Device files
/etc	Host specific system configuration
/lib	Essential shared libraries and kernel modules
/media	Mount point for removeable media
/mnt	Mount point for mounting a filesystem temporarily
/opt	Add on application software packages
/sbin	Essential system binaries
/srv	Data for services provided by this system
/tmp	Temporary files
/usr	Secondary hierarchy
/var	Variable data

Another feature of the Unified Filesystem is that Linux caches a lot of disk accesses using system memory while it

is running to accelerate these processes. It is therefore vitally important that these buffers are flushed (get their content written to disk), before the system closes down. Otherwise files are left in an undetermined state which is of course a very bad thing. Flushing is achieved by *unmounting* the partitions during proper system shutdown. In other words, don't switch your system off while it is running. You may get away with it quite often, since the Linux file system is very robust, but you may also wreak havoc upon important files. Just hit ctrl-alt-del or use the proper commands (e.g. shutdown, poweroff, init 0). This will shut down the system in a decent way which will thus, guarantee the integrity of your files.

The Linux File System Structure is a document. Often the group, which creates this document or the document itself, is referred to as the *FileSystem Standard (FSSTND)*. FSSTND for Root Directory is given below. To comply with the FSSTND the following directories, or symbolic links to directories, are required in /.

Q.15 ***Explain process management for Linux operating system. How Linux is secure compared to other operating system.***

[R.T.U. 2013]

OR

Write short note on Process Management in Linux.

[R.T.U. 2011]

Ans. Process Management

UNIX process management separates the creation of processes and the running of a new program into two distinct operations. The fork system call creates a new process. A new program is run after a call to exec(). Under UNIX, a process encompasses all the information that the operating system must maintain to track the context of a single execution of a single program. Under Linux process properties fall into three groups: the process's identity, environment and context.

Linux operating system is secure compared to other operating system in many ways:

1. Linux is multitasking operating system.
2. Linux won't affect by virus, because it won't run .exe files. So disallowing the virus to run in system.
3. Most Linux operators are more familiar with the inner working of their system than most windows users, so they are usually capable of setting up their machine securely.
4. Many windows "feature" requires execute and other priviledges on the machine. This means that by default windows security is set up to be looser. These priviledges can be taken advantage of by people with malicious infectious against users that have met made adequate security adjustments on their machine.
5. Linux has the advantage of the code being not in the public domain. This can be double edge sword, while you can look the code and develops can fix holes rapidly, it also means that hackers can't see the code. And can't able to hack the system.
6. Linux has two types of password
   (i) Administrator password
   (ii) Root password

   Thus creating firewall for anonymous users to get into the hard disk.

Q.16 ***Write short note on Types of Shells in Linux.***

[R.T.U. 2013]

Ans. Shell : A shell is a piece of software that provides an interface for users to access the services of a kernel. In layman's term shell means any program that users use to type commands. It is called a ***shell*** because it hides the details of the underlying operating system behind the shell's interface.

Various types of Shells in Linux

Bourne Shell : The original Bourne shell is named after its developer at Bell Labs, Steve Bourne. It was the first shell used for the Unix operating system and it has been largely surpassed in functionality by many of the more recent shells. However, all Unix and many Linux versions allow users to switch to the original Bourne Shell, known simply as ***sh*** if they choose to forgo features such as file name completion and command histories that later shells have added.

C Shell : *Refer to Q.4.*

TC Shell : TC shell is an expansion upon the C shell. It has all the same features, but adds the ability to use keystrokes from the Emacs word processor program to edit text on the command line. For example, users can press Esc-D to delete the rest of the highlighted word. It is also known as ***tcsh***.

Korn Shell : Korn Shell was also written by a developer at Bell Labs, David Korn. It attempts to merge the features of the C shell, TC shell and Bourne shell under one package. It also includes the ability for developers to create new shell commands as the need arises. It is known as ***ksh***.

Bourne-Again Shell : The Bourne-Again shell is an updated version of the original Bourne shell that was created by the Free Software Foundation for its open source GNU project. For this reason, it is a widely used shell in the open source community. Its syntax is similar to that used by the Bourne shell, however, it incorporates some of the more advanced features found in the C, TC and Korn shells.

Among the added features that Bourne lacked are ability to complete file names by pressing the TAB key, ability to remember a history of recent commands and ability to run multiple programs in the background at once is known as ***bash***.

Q.17 ***What is a UNIX Shell? Explain the steps that a UNIX shell follows while processing a command.***

Ans. The shell is a type of program called an interpreter. An interpreter operates in a simple loop: It accepts a command, interprets the command, executes the command, and then waits for another command. The shell displays a "prompt," to notify you that it is ready to accept your command.

After the command line is terminated by the key, the shell goes ahead with processing the command line in one or more passes. The sequence is well defined and assumes the following order.

Parsing: The shell first breaks up the command line into words, using spaces and the delimiters, unless quoted. All consecutive occurrences of a space or tab are replaced here with a single space.

Variable evaluation: All words preceded by a $ are evaluated as variables, unless quoted or escaped.

Command substitution: Any command surrounded by back quotes is executed by the shell which then replaces the standard output of the command into the command line.

Wild-card interpretation: The shell finally scans the command line for wild-cards (the characters *, ?,[,]). Any word containing a wild-card is replaced by a sorted list of filenames that match the pattern. The list of these filenames then forms the arguments to the command. PATH evaluation: It finally looks for the PATH variable to determine the sequence of directories it has to search in order to hunt for the command.

Q.18 ***What is Shell? What are some common shells and what are their indicators? Also write key features of the Korn Shell and describe the Shell's responsibilities.***

Ans. A shell acts as an interface between the user and the system. As a command interpreter, the shell takes commands and sets them up for execution.

Types of shell and their indicators

- sh - Bourne shell
- csh - C shell
- bash - Bourne again shell
- tcsh - enhanced C shell
- zsh - Z shell
- ksh - Korn Shell

atures of korn shell

- history mechanism with a built-in editor that simulates emacs or vi
- built-in integer arithmetic
- string manipulation capabilities
- command aliasing
- arrays
- job control

shell responsibilities :

- program execution
- variable and file name substitution
- I/O redirection
- pipeline hookup
- environment control
- interpreted programming language

PART-C

Q.19 ***Explain in detail Kernel structure of LINUX operating system.*** ***[R.T.U. 2014]***

Ans. Kernel Structure : The *kernel* is the central component of most Linux operating systems; it acts as a bridge between software applications and the actual data processing done at the hardware level. The kernel's responsibilities include managing the system's resources (the communication between hardware and software components).The Linux kernel is composed of five main subsystems :

1. Process Scheduler (SCHED) : *Refer to Q.1.*

2. Memory Manager (MM) permits multiple process to securely share the machine's main memory system. In addition, the memory manager supports virtual memory that allows Linux to support processes that use more memory than is available in the system. Unused memory is swapped out to persistent storage using the file system then swapped back in when it is needed.

3. Virtual File System (VFS) abstracts the details of the variety of hardware devices by presenting a common file interface to all devices. In addition, the VFS supports several file system formats that are compatible with other operating systems.

4. The Network Interface (NET) provides access to several networking standards and a variety of network hardware.

5. Inter-Process Communication (IPC) subsystem supports several mechanisms for process-to-process communication on a single Linux system.

Here the fig. shows a high-level decomposition of the Linux kernel, where lines are drawn from dependent subsystems to the subsystems they depend on :

This diagram emphasizes that the most central subsystem is the process scheduler: all other subsystems depend on the process scheduler since all subsystems need to suspend and resume processes. Usually a subsystem will suspend a

process that is waiting for a hardware operation to complete and resume the process when the operation is finished. For example, when a process attempts to send a message across the network, the network interface may need to suspend the process until the hardware has completed sending the message successfully. After the message has been sent (or the hardware returns a failure), the network interface then resumes the process with a return code indicating the success or failure of the operation. The other subsystems (memory manager, virtual file system and inter-process communication) all depend on the process scheduler for similar reasons.

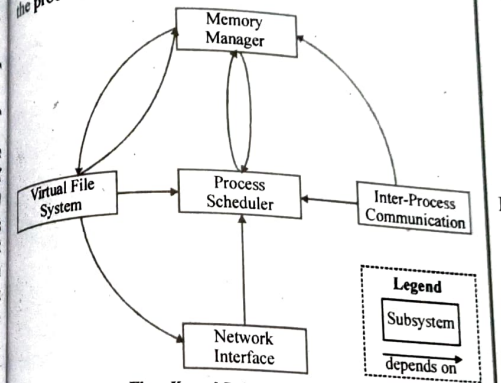


Fig. : Kernel Subsystem Overview

The other dependencies are somewhat less obvious, but equally important :

- The process scheduler subsystem uses the memory manager to adjust the hardware memory map for a specific process when that process is resumed.
- The inter-process communication subsystem depends on the memory manager to support a shared-memory communication mechanism. This mechanism allows two processes to access an area of common memory in addition to their usual private memory.
- The virtual file system uses the network interface to support a network file system (NFS) and also uses the memory manager to provide a ramdisk device.
- The memory manager uses the virtual file system to support swapping; this is the only reason that the memory manager depends on the process scheduler. When a process accesses memory that is currently swapped out, the memory manager makes a request to the file system to fetch the memory from persistent storage and suspends the process.

Q.20 ***(a) What is process management? Explain process scheduling in Linux.***

(b) Explain various differences in Unix and Linux operating system. ***[R.T.U. 2015]***

Ans.(a) Process Management : *Refer to Q.15.*

Process Identity

Process ID is a unique priority number assigned to a currently running or about to run process. On the basis of process id the process executes according to process queues.

- **Process ID (PID) :** The unique identifier for the process; used to specify processes to the operating system when an application makes a system call to signal, modify, or wait for another process.
- **Credentials :** Each process must have an associated user ID and one or more group IDs that determine the process's rights to access system resources and files.
- **Personality :** Not traditionally found on UNIX systems, but under Linux each process has an associated personality identifier that can slightly modify the semantics of certain system calls.Used primarily by emulation libraries to request that system calls be compatible with certain specific flavors of UNIX.

Process Environment

The process's environment is inherited from its parent and is composed of two null terminated vectors:

- The argument vector lists the command-line arguments used to invoke the running program; conventionally starts with the name of the program itself.
- The environment vector is a list of "NAME=VALUE" pairs that associates named environment variables with arbitrary textual values.

Passing environment variables among processes and inheriting variables by a process's children are flexible means of passing information to components of the user-mode system software. The environment-variable mechanism provides a customization of the operating system that can be set on a per-process basis, rather than being configured for the system as a whole.

Process Context : Process context is the state of a running program at any one time, it changes constantly. The scheduling context is the most important part of the process context; it is the information that the scheduler needs to suspend and restart the process. The kernel maintains accounting information about the resources currently being consumed by each process and the total resources consumed by the process in its lifetime so far. The file table is an array of pointers to kernel file structures. When making file I/O system calls, processes refer to files by their index into this table. The current root and default directories to be used for new file searches are stored here. The signal-handler table defines the routine in the process's address space to be called when specific signals arrive. The virtual memory context of a process describes the full contents of its private address space.

Processes and Threads : Operating System Concept Processes and Threads Linux uses the same internal representation for processes and threads; a thread is simply a new process that happens to share the same address space as its parent. Thread creation is done through clone() system call. Clone() allows a child task to share the address space of the parent task(process). The flags provided to clone() help specify the behaviour of the new process and detail what resources the parent and child will share.

Table 1 : Clone() Flags

Flags	Description
CLONE_FILES	Parent and child share open files.
CLONE_FS	Parent and child share file-system information.
CLONE_IDLETASK	Set PID to zero (used only by the idle tasks).
CLONE_NEWNS	Create a new namespace for the child.
CLONE_PTRACE	Continue tracing child.
CLONE_SIGHAND	Parent and child share signal handlers.
CLONE_THREAD	Parent and child are in the same thread group.
CLONE_STOP	Start process in the TASK_STOPPED state.
CLONE_SETTLS	Create a new TLS (thread-local storage) for the child.
CLONE_VM	Parent and child share address space.

System Calls used for Process Management : The system call interface is the boundary with the user and allows higher-level software to gain access to specific kernel functions. The system call is the means by which a process requests a specific kernel service. Some of the system calls for process management in Linux operating system are given below in Table 2.

Table 2 : System Calls for Process Management

System call	Description
Fork ()	Used to create a new process.
Exec()	Execute a new program.
Wait()	Wait until the process finishes execution.
Exit()	Exit from the process.
Getpid()	Get the unique process ID of the process.
Getppid()	Get the parent process unique ID.
Nice()	To bias the existing property of process.

Process Scheduling : *Refer to Q. 9.*

Ans.(b) Unix Vs Linux

Both of the operating systems are differ at the conceptual level, as *process management, memory management,*

Difference between Unix and Linux

	Unix	Linux
1.	UNIX is propriety system.	Linux is an Open Source system.
2.	Development is targeted toward specific audience and platform.	Linux development is diverse.
3.	UNIX maintains consistency between different versions.	Linux have inconsistencies between versions.
4.	In UNIX developers are bounded by standard.	Linux developers are free and have no restriction.
5.	In UNIX commands, tool and utilities etc. are rarely changed over versions.	In Linux commands, tools and utilities may change over time.
6.	UNIX was coded for small handful hardware platform/architecture.	Linux was designed to be as compatible as possible. Runs on dozens of architecture and support numerous I/O devices & other external devices. Supported devices are limitless.
7.	UNIX kernel is not freely available.	Linux kernel is freely available.
8.	UNIX patches available are highly tested.	Linux patches are not highly tested as UNIX patches.
9.	Commercial UNIX is usually custom written for each system, making the original cost quite high.	Linux has base packages.
10.	User has to wait for a while, to get the proper bug fixing patch.	Threat detection and solution is very fast.
11.	Different flavors of Unix have different cost structures.	Linux can be freely distributed.
12.	A rough estimate of Unix viruses is between 85-120 viruses reported till date.	Linux has had about 60-100 viruses listed till date.
13.	Market share of Unix is less than 0.5 percent of the pc market.	The Market share of Linux is about 0.8%.

Q.21 ***(a) Explain network file system and its implementation.***

(b) Explain the following :

(i) Inter-Process communication.

(ii) Booting & login process *[R.T.U. 2015]*

Ans.(a) A Network File System (NFS) , the network file system, is probably the most prominent network services using RPC. It allows to access files on remote hosts in exactly the same way as a user would access any local files. This is made possible by a mixture of kernel functionality on the client side (that uses the remote file system) and an NFS server on the server side (that provides the file data). This file access is completely transparent to the client, and works across a variety of server and host architectures.

NFS offers a number of advantages:

(i) Data accessed by all users can be kept on a central host, with clients mounting this directory at boot time. For example, you can keep all user accounts on one host, and have all hosts on your network mount /home from that host. If installed alongside with NIS, users can then log into any system, and still work on one set of files.

(ii) Data consuming large amounts of disk space may be kept on a single host. For example, all files and programs relating to LaTeX and METAFONT could be kept and maintained in one place.

(iii) Administrative data may be kept on a single host. No need to use rcp anymore to install the same stupid file on 20 different machines.

When someone accesses a file over NFS, the kernel places an RPC call to nfsd (the NFS daemon) on the server machine. This call takes the file handle, the name of the file to be accessed, and the user's user and group id as parameters. These are used in determining access rights to the specified file. In order to prevent unauthorized users from reading or modifying files, user and group ids must be the same on both hosts.

On most implementations, the NFS functionality of both client and server are implemented as kernel-level daemons that are started from user space at system boot. These are the NFS daemon (nfsd) on the server host, and the Block I/O Daemon (biod) running on the client host. To improve throughput, biod performs asynchronous I/O using read-ahead and write-behind; also, several nfsd daemons are usually run concurrently.

The NFS implementation of is a little different in that the client code is tightly integrated in the virtual file system (VFS) layer of the kernel and doesn't require additional control through biod. On the other hand, the server code runs entirely in user space, so that running several copies of the server at the same time is almost impossible because of the synchronization issues this would involve. NFS currently also lacks read-ahead and write-behind, but Rick Sladkey plans to add this someday.

The biggest problem with the NFS code is that the kernel as of version 1.0 is not able to allocate memory in chunks bigger than 4K; as a consequence, the networking code cannot handle data grams bigger than roughly 3500 bytes after subtracting header sizes etc. This means that transfers to and from NFS daemons running on systems that use large UDP data grams by default (e.g. 8K on SunOS) need to be downsized artificially.

Ans.(b) (i) Linux Inter-Process Communication : Inter-Process Communication, which in short is known as IPC, deals mainly with the techniques and mechanisms that facilitate communication between processes. Processes communicate with each other and with the kernel to coordinate their activities. Linux supports a number of Inter-Process Communication (IPC) mechanisms. Signals and pipes are two of them but Linux also supports the System V IPC mechanisms named after the Unix release in which they first appeared.

The types of inter process communication are :

1. **Signals :** Sent by other processes or the kernel to a specific process to indicate various conditions.
2. **Pipes :** Unnamed pipes set up by the shell normally with the "|" character to route output from one program to the input of another.
3. **FIFOs :** Named pipes operating on the basis of first data in, first data out.
4. **Message queues :** Message queues are a mechanism set up to allow one or more processes to write messages that can be read by one or more other processes.
5. **Semaphores :** Counters that are used to control access to shared resources. These counters are used as a locking mechanism to prevent more than one process from using the resource at a time.
6. **Shared memory :** The mapping of a memory area to be shared by multiple processes.

Message queues, semaphores and shared memory can be accessed by the processes if they have access permission to the resource as set up by the object's creator. The process must pass an identifier to the kernel to be able to get the access.

(ii) Booting and Login Process : When a PC is booted, it starts running a BIOS program which is a memory resident program on an EEPROM integrated circuit. The BIOS program will eventually try to read the first sector on a booting media such as a hard or floppy drive. The boot sector contains a small program that the BIOS will load and attempt to pass run control to. This program will attempt to read the operating system from the disk and run it. LILO is the program that Linux systems typically use to give users a choice of operating systems to run. It is usually installed in the boot sector which is also called the master boot record.

Booting

In Linux, the flow of control during a boot is from BIOS, to boot loader, to kernel. The kernel then starts the scheduler (to allow multi-tasking) and runs the first userland (*i.e.* outside

kernel space) program init (which sets up the user environment and allows user interaction and login), at which point the kernel goes idle unless called externally.

In detail:

1. The BIOS performs hardware-platform specific startup tasks.
2. Once the hardware is recognized and started correctly, the BIOS loads and executes the partition boot code from the designated boot device, which contains phase 1 of a Linux boot loader. Phase 1 loads phase 2 (the bulk of the boot loader code). Some loaders may use an intermediate phase (known as phase 1.5) to achieve this since modern large disks may not be fully readable without further code.
3. The boot loader often presents the user with a menu of possible boot options. It then loads the operating system, which decompresses into memory and sets up system functions such as essential hardware and memory paging, before calling start_kernel().
4. Start kernel() then performs the majority of system setup (interrupts, the rest of memory management, device initialization, drivers, etc) before spawning separately, the idle process and scheduler and the Init process (which is executed in user space).
5. The scheduler effectively takes control of the system management, as kernel goes dormant (idle).
6. The Init process executes scripts as needed that set up all non-operating system services and structures in order to allow a user environment to be created and then presents the user with a login screen.

On shutdown, Init is called to close down all user space functionality in controlled manner, again via scripted directions, following which Init term and the Kernel executes its own shutdown.

Login Process

After the system boots, at serial terminals or virtual terminals, the user will a login prompt similar to:

Machine name Login : This prompt is being generated by a program, usually getty or mingetty, is regenerated by the Init process every time a user ends a session on the console. The getty program will call login and login, if successful will call users shell. The steps of the process are :

The init process spawns the getty process.

1. The getty process invokes the login process when the user enters their name and passes the user name to login.
2. The login process prompts the user for a password, checks it, then if there is success, the users shell is started. On failure the program displays an error message, ends and then Init will respawn getty.
3. The user will run their session and eventually logout. On logout, the shell program exits and we return to step 1.

Note : This process is what happens for runlevel 1 but runlevel 5 uses some different programs to perform similar functions. These X programs are called X clients.

Q.22 (a) What is Linux OS? Explain its advantages.
(b) What are caches ? Explain linux pagecache. *[R.T.U. 2012]*

Ans.(a) The Linux system is composed of three main bodies of code, in line with most traditional UNIX implementation:

(1) Kernel: The kernel is responsible for maintaining all the important abstractions of the OS, including such things as virtual memory and processes.

(2) System Libraries: The system libraries define a standard set of functions through which applications can interact with the kernel, and that implement much of the OS functionality that does not need the full privileges of kernel code.

(3) System Utilities: The system utilities are programs that perform individual specialized management tasks. Some system utilities may be invoked just once to initialize and configure. Some aspect of the system; others-known as daemons in UNIX terminology-may run permanently, handling such tasks as responding to incoming network connections, accepting log on request from terminals, or updating log files.

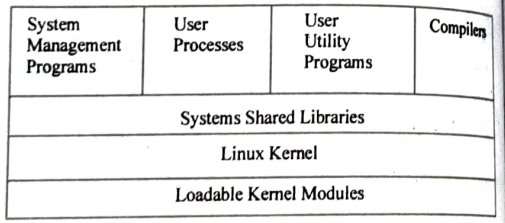


Fig.: Components of the Linux System

The figure illustrates the various components that make up a full Linux system. The most important distinction here is between the kernel and everything else. All the kernel code executes in the processor's privileged mode with full access to all the physical resources of the computer.

Advantage: It is multiuser, multitasking system with a full set of UNIX compatible tools. Linux's file system adheres to traditional UNIX semantics and the standard UNIX networking model is implemented fully. The internal details of Linux's design have been influenced heavily by the history of this OS development.

Ans.(b) Caching is an important principle of computer systems. Information is normally kept in some storage system (such as main memory). As it is used it is copied into a faster storage system- the cache-on a temporary basis. When we need a particular piece of information, we first check whether it is in the cache. If it is, we use the information directly from the cache, if it is not, we use the information from the main storage system, putting a copy in the cache under the assumption that we will need it again soon.

Some systems maintain a separate section of main memory for a disk cache, where blocks are kept under the assumption that they will be used again shortly. Other system cache file data using a pagecache. The page cache uses virtual memory techniques to cache file data as pages rather than as file system-oriented blocks. Caching file data using virtual addresses is for more efficient than caching through physical disk blocks. Several systems, including solaries, some new Linux releases use pagecaching to cache both process pages and file data.

Q.23 Write short notes on the following :
(a) Thread Management and scheduling
(b) Booting and Login process in Linux
(c) Memory Management and I/O Management in Linux *[RTU 2011]*

Ans.(a) Thread Management

Threads operate in the same manner as processes, but can execute more efficiently. Linux includes multi threading capability and some multithreading libraries are available for linux. A thread is an independent flow of control within the process. A traditional unix process has a single thread that has possession of the process memory and other resources. Thread within the same process share global data like global variable, files, etc but each thread has its own stack, local variables and program counter. Threads are referred to as lightweight processes, because their context is smaller than the context of a process.

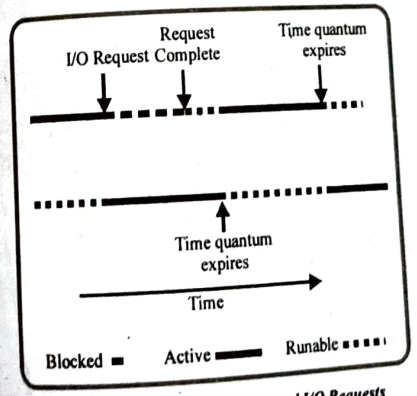


Fig. : Overlapping Processing and I/O Requests

Threads are useful for improving application performance. A program with only one thread of control must wait each time it request a services from the operating system.

Using more than one thread lets a process overlap processing with one or more I/O request. In multiprocessor machines, multiple threads are an efficient way for application developers to utilize the parallelism of the hardware. This feature is especially important for client/server application. Server program in client/server application may get multiple requests from independent clients simultaneously. If the server has only one thread of control, client request must be served in a serial manner. Using multi thread server, when a client attempt to connect to the server, a new thread can be created to manage the request.

There are two traditional models of thread control: *user level thread* and *kernel level thread*.

User level thread packages usually run on top of existing operating system. The thread within the process is invisible to the kernel. Threads are scheduled by a runtime system which is part of the process code. Switching between user-level threads can be done independently of the operating system. User-level threads have a problem, when a thread becomes blocked while making a system call, all the other threads within the process must wait until the system call returns. This restriction limits the ability to use the parallelism provided by multiprocessor platforms.

Kernel level threads are supported by thread. The kernel is aware of each thread as a scheduled entity. In this case, a set of system call similar to these for processes is provided and the threads are scheduled by the kernel. Kernel thread can take advantage of multiple processors, however, switching among thread is more time-consuming because the kernel is involved.

There are also hybrid models, supporting user-level and kernel level threads. This gives the advantages of both models to running process. Solaris offer this kind of model.

POSIX Threads

The portable operating system interface defines an application program interface which is derived from Unix but may as well be provided by any other operating system. This standard includes a set of thread extensions (POSIX 1003.1c). These thread extensions provide the base standard with interfaces and functionality to support multiple flows of control within the process. The facilities provided represent a small set of syntactic and semantic extension to POSIX 1003.1 in order to support a convenient interface for multi threading functions. The specific function areas covered by this standard include thread management, synchronization primitives.

- **Thread Management:** Creation, control and termination of multiple flows of control in the same process under the assumption of a common shared address space.
- **Synchronization Primitives:** Mutual exclusion and condition variables optimized for tightly coupled operation of multiple control flows within a process.

Scheduling Policy

The set of rules used to determine when and how selecting a new process to run is called *scheduling policy*. The scheduling algorithm of traditional Unix operating systems must fulfill several conflicting objectives :

- Fast process response time.
- Avoidance of process starvation.
- Good throughput for background jobs.
- Support for soft real time processes.

Ans.(b) Booting and Login Process : *Refer to Q.21(b)(ii).*

Ans.(c) Memory Management : The memory management subsystem is one of the most important parts of the operating system. Since the early days of computing, there has been a need for more memory that exists physically in a system. Strategies have been developed to overcome this limitation and the most successful of these is virtual memory. Virtual memory makes the system appear to have more memory than it actually has by sharing it between competing processes as they need it.

The memory management subsystem provides :

- **Large Address Spaces :** The operating system makes the system appear as if it has a larger amount of the memory management subsystem provide memory than it actually has. The virtual memory can be many times larger than the physical memory in the system.
- **Protection :** Each process in the system has its own virtual address space. These virtual address spaces are completely separate from each other and so a process running one application cannot affect another. Also, the hardware virtual memory mechanisms allow areas of memory to be protected against writing. This protects code and data from being overwritten by rogue applications.
- **Memory Mapping :** Memory mapping is used to map image and data files into a processes address space. In memory mapping, the contents of a file are linked directly into the virtual address space of a process.
- **Fair Physical Memory Allocation :** The memory management subsystem allows each running process in the system a fair share of the physical memory of the system,
- **Shared Virtual Memory :** Although virtual memory allows processes to have separate (virtual) address spaces, there are times when you need processes to share memory. For example, there could be several processes in the system running the *bash* command shell. Rather than having several copies of bash, one in each processes virtual address space, it is better to have only one copy in physical memory and all of the processes running bash share it. Dynamic libraries are another common example of executing code shared between several processes.

Shared memory can also be used as an Inter Process Communication (IPC) mechanism, with two or more processes exchanging information via memory common to all of them.

Input Output Management

Linux follows the philosophy that everything is a file. For example, a keyboard, Monitor, Mouse, Printer. The I/O system in Linux is like that in any Unix system. Here all device drivers appear as normal files. A user can access a device in the same way as he opens any other file. The administrator can set access permission for each device.

There are two kinds of device files that exist: *block device files* and *character device files*. Block devices transfer data in chunks and character devices (as the name implies) transfer data one character at a time. A third device type, the *network device*, is a special case that exhibits attributes of both block and character devices. However, network devices are not represented by files.

Device Classes

Basically Linux divides the devices into three classes which are given below :

1. Block Devices
2. Character Devices
3. Network Devices

1. Block Devices

It includes all devices such as Hard Disks, Floppy Disks, CD-ROMs and Flash Memory. These devices can be accessed randomly. They read only blocks of data. The block buffer cache serves two main purposes :

- It acts as a pool of buffers for active I/O.
- It serves as a cache for completed I/O.

The request manager manages the reading and writing of buffer contents to and from a block device driver.

Block devices provide the main interface to all disk devices in a system, or block devices allow random access and fixed sized blocks of data, including hard disk, floppy disk, CD-ROMs etc. In the context of block devices, block represents the unit with which the kernel perform I/O. When a block read in to memory, it is stored in a buffer. A separate list of requests is kept for each block device. These request have been scheduled according to a unidirectional elevator algorithm. The request are maintained in sorted order of increasing starting sector number. When a request is accepted for processing by a block device driver, it is not removed from the list .It is removed after only the input output is complete.

In the block devices there are two problems that may occur:

- Starvation
- Deadline

The deadline for read requests is 0.5 sec. & for writing request is 5 sec. block device also maintains three queue

- Sorted queue
- Read queue
- Write queue

These queues are ordered according to deadline.

2. Character Devices: It includes devices such as Mice and Keyboards. These devices are accessed only serially or

sequentially. They read the data character by character. A character device driver must register a set of functions which implement the driver's various file I/O operations. The kernel performs almost no preprocessing of a file read or write request to a character device, but simply passes on the request to the device. The main exception to this rule is the special subset of character device drivers which implement terminal devices, for which the kernel maintains a standard interface.

All character devices deal with data one character at a time and process them sequentially. For example the keyboard strokes, mouse clicks etc. Any character device drivers registered to the Linux kernel must also register a set of functions that implement the file I/O operations that the driver can handle. The kernel performs almost no preprocessing of a file read or write request to the device in question and lets the device deal with request. Printers are character devices and after the kernel sends data to the printer, the responsibility for that data passes to the printer, the kernel cannot back up and reexamine the data. The exception to this rule is the special subset of character device drivers that implement terminal devices. The kernel maintains a standard interface to these drivers by means of a set of *tty_struct* structures. Each of these structures provide buffering and flow control on the data stream from the terminal device and feeds those data to a line discipline. A line discipline is an interpreter for the information from the terminal device. The most common line discipline is the *tty discipline*. *tty discipline* decides which process's data should be attached or detached from the terminal device.

3. Network Devices: These are dealt differently from Block and Character Devices. Users cannot directly transfer data to Network Devices; instead, they must communicate indirectly by opening a connection to the kernel's networking sub system.

Users cannot directly transfer data to network devices. They communicate indirectly by opening a connection to the kernel's networking subsystem. Networking in LINUX kernel is implemented by three layers of software :

- The socket interface
- Protocol drivers
- Network device drivers

(i) The socket interface: User applications perform all networking requests through the socket interface. It looks like BSD socket layer, so that the program designed for BSD socket can easily run on LINUX without any source code changes. BSD socket is sufficient to represent network addresses for networking protocols.

(ii) Protocol drivers: It is the second layer of software. When data arrives to this layer ,it is expected to have been tagged with an identifier specifying which network protocol they contain.

Functions of protocol Layer

- Rewrite packets
- Create new packets
- Split or reassembling packets into fragments
- Discard incoming data

(iii) Network Device Drivers: Communication between the layers of networking stack is performed by passing single skbuff structures. An skbuff is a set of pointers into a single continuous area of memory, representing a buffer inside which network packets can be constructed. The networking code either add or trim data from the end of packet. The most important set of protocol used in LINUX is TCP/IP suite.

IP protocol implements routing between different hosts, anywhere on the network. UDP protocol carries arbitrary individual datagram's between hosts .TCP protocol implements reliable connection between hosts. ICMP protocol is used to carry various errors and status messages between hosts. Incoming IP packets are delivered to the IP driver. This layer performs routing.

After deciding the destination of a packet is forwarded to appropriate internal protocol driver to delivered locally or injects it back into a selected network device driver queue to be forwarded to another host.

The routing can be done by two tables :

- FIB(forwarding information base).
- Cache of recent routing decisions.
- FIB is a set of hash tables indexed by destination address.
- It caches routes only by specific destination. An entry in route cache expires after a fixed period with no hits.

IP software passes its packets to a separate section code for firewall management. It manages separate firewall chains :

- For forwarded packets
- For packets being input to host
- For data generated at the host.

IP driver performs disassembly and reassembly of large packets.

Large outgoing packets are split up into smaller fragments. At the receiving host these fragments are reassembled. Incoming fragments are matched against each known *ipq*. If a match is found fragment is added to it otherwise, new *ipq* is created . Once the final fragment has arrived new skbuff is created, to hold a new packet. This is passed back to IP driver.

Q.24 ***Write detailed note on process management in unix.***

Ans. A major design problem for operating systems is th representation of processes. One substantial differenc between UNIX and many other systems is the ease wi which multiple processes can be created and manipulate These processes are represented in UNIX by various contr

blocks. There are no system control blocks accessible in the virtual address space of a user process; control blocks associated with a process are stored in the kernel. The information in these control blocks is used by the kernel for process control and CPU scheduling.

Process Control Blocks

The most basic data structure associated with processes is the process structure. A process structure contains everything that the system needs to know about a process when the process is swapped out, such as its unique process identifier, scheduling information (such as the priority of the process) and pointers to other control blocks. There is an array of process structures whose length is defined at system linking time. The process structures of ready processes are kept linked together by the scheduler in a doubly linked list (the ready queue) and there are pointers from each process structure to the process parent, to its youngest living child and to various other relatives of interest, such as a list of processes sharing the same program code (text).

The virtual address space of a user process is divided into text (program code), data, and stack segments. The data and stack segments are always in the same address space, but may grow separately and usually in opposite directions: most frequently, the stack grows down as the data grow up toward it. The text segment is sometimes (as on an Intel 8086 with separate instruction and data space) in an address space different from the data and stack and is usually read only. The debugger puts a text segment in read-write mode to be able to allow insertion of breakpoints.

Every process with sharable text (almost all, under FreeBSD) has a pointer from its process structure to a text structure. The text structure records how many processes are using the text segment, including a printer into a list of their process structures and where the page table for the text segment can be found on disk when it is swapped. The text structure itself is always resident in main memory: an array of such structures is allocated at system link time. The text, data and stack segments for the processes may be swapped. When the segments are swapped in, they are paged.

The page tables record information on the mapping from the process virtual memory to physical memory. The process structure contains pointers to the page table, for use when the process is resident in main memory, or the address of the process on the swap device, when the process is swapped. There is no special separate page table for a shared text segment; every process sharing the text segment has entries for its pages in the process page table.

Information about the process that is needed only when the process is resident (that is, not swapped out) is kept in the user structure (or u structure), rather than in the process structure.

Every process has both a user and a system mode. Most ordinary work is done in user mode, but when a system call is made, it is performed in system mode. The system and user phases of a process never execute simultaneously. When a process is executing in system mode, a kernel stack for that process is used, rather than the user stack belonging to that process. The Kernel stack for the process immediately follows the user structures. The kernel stack and the user structure together compose the system data segment for the process. The kernel has its own stack for use when it is not doing work on behalf of a process (for instance, for interrupt handling).

Figure illustrates how the process structure is used to find the various parts of a process.

The fork system call allocates a new process structure (with a new process identifier) for the child process and copies the user structure. There is ordinarily no need for a new text structure, as the processes share their text; the appropriate counters and lists are merely updated. A new page table is constructed and new main memory is allocated for the data and stack segments of the child process. The copying of the user structure preserves open file descriptors, user and group identifiers, signal handling and most similar properties of a process.

The vfork system call does not copy the data and stack to the new process; rather, the new process simply shares the page table of the old one. A new user structure and a new process structure are still created. A common use of this system call is by a shell to execute a command and to wait for its completion. The parent process uses vfork to produce the child process. Because the child process wishes to use an execve immediately to change its virtual address space completely, there is no need for a complete copy of the parent process. Such data structure as are necessary for manipulating pipes may be kept in registers between the vfork and the execve.

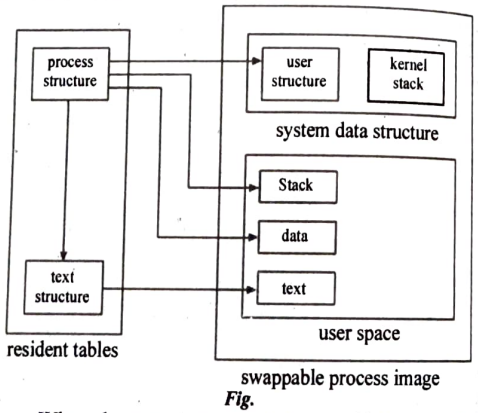


Fig.

When the parent process is large, vfork an produce substantial savings in system CPU time. However, it is a fairly dangerous system call, since any memory changes occurs in both processes until the execve occurs. An alternative is to share all pages by duplicating the page table, but to mark the entries of both page tables as copy-on-write.

An execve system call creates no new process or user structure; rather, the text and data of the process are replaced. Open files are preserved (although there is a way to specify that certain file descriptors are to be closed on an execve).

CPU Scheduling

CPU scheduling in UNIX is designed to benefit interactive processes. Processes are given small CPU time slices by a priority algorithm that reduces to round-robin scheduling for CPU-bound jobs.

Every process has a scheduling priority associated with it; larger numbers indicate lower priority. Processes doing disk I/O or other important tasks have priorities less than "zero" and cannot be killed by signals. Ordinary user processes have positive priorities and thus are all less likely to be run than are any system process, although user processes can set precedence over one another through the nice command.

The more CPU time a process accumulates, the lower (more positive) its priority becomes, and vice versa, so there is negative feedback in CPU scheduling and it is difficult for a single process to take all the CPU time. process aging is employed to prevent starvation.

Older UNIX systems used a 1-second quantum for the round-robin scheduling. FreeBSD reschedules processes every 0.1 second and recomputes priorities every second. The round-robin scheduling is accomplished by the time out mechanism, which tells the clock interrupt driver to call a kernel subroutine after a specified interval; the subroutine that resubmits a timeout for itself to be called in this case causes the rescheduling and then resubmits a time out to call itself again. The priority recomputation is also timed by a subroutine.

There is no preemption of one process by another in the kernel. A process may relinquish the CPU because it is waiting on I/O or because its time slice has expired. When a process chooses to relinquish the CPU, it goes to sleep on an event. The kernel primitive used for this purpose is called sleep (not to be confused with the user-level library routine of the same name). it takes an argument, which is by convention the address of a kernel data structure related to an event that the process wants to occur before that process is awakened. When the event occurs, the system process that knows about it calls wakeup with the address corresponding to the event and all processes that has done a sleep on the same address are put in the ready queue to be run.

For example, a process waiting for disk I/O to complete will sleep on the address of the buffer header corresponding to the data being transferred. When the interrupt routine for the disk driver notes that the transfer is complete, it calls wakeup on the buffer header. The interrupt uses the kernel stack for whatever process happened to be running at the time and the wakeup is done from that system process.

Q.25 *Write detailed note on UNIX Directories.*

Ans. UNIX directories : UNIX directories are similar to regular files; they both have names and both contain information. Directories, however, contain other files and directories. Many of the same rules and commands that apply to files also apply to directories.

All files and directories in the UNIX system are stored in a hierarchical tree structure. Envision it as an upside-down tree, as in the figure below.

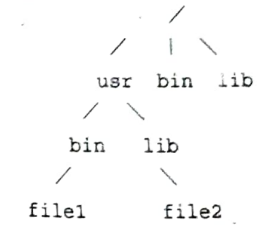


Fig. : UNIX Directory Structure

At the top of the tree is the root directory. Its directory name is simply / (a slash character). Below the root directory is a set of major *subdirectories* that usually include **bin**, **dev**, **etc**, **lib**, **pub**, **tmp** and **usr**. For example, the **/bin** directory is a subdirectory, or "child," of / (the root directory). The root directory, in this case, is also the ***parent*** directory of the **bin** directory. Each path leading down, away from the root, ends in a file or directory. Other paths can branch out from directories, but not from files.

Many directories on a UNIX system have traditional names and traditional contents. For example, directories named **bin** contain binary files, which are the executable command and application files. A **lib** directory contains library files, which are often collections of routines that can be included in programs by a compiler. **dev** contains device files, which are the software components of terminals, printers, disks etc. **tmp** directories are for temporary storage, such as when a program creates a file for something and then deletes it when it is done. The **etc** directory is used for miscellaneous administrative files and commands. **pub** is for public files that anyone can use and **usr** has traditionally been reserved for user directories, but on large systems it usually contains other **bin**, **tmp** and **lib** directories.

Your home directory is the directory that you start out from when you first login. It is the top level directory of your account. Your home directory name is almost always the same as your userid.

Every directory and file on the system has a path by which it is accessed, starting from the root directory. The path to the directory is called its pathname. You can refer to any point in the directory hierarchy in two different ways: using its full (or absolute) pathname or its relative pathname. The full pathname traces the absolute position of a file or directory back to the root directory, using slashes (/) to connect

every point in the path. For example, in the figure above, the full pathname of **file2** would be **/usr/bin/file2**. Relative pathnames begin with the current directory (also called the working directory, the one you are in). If **/usr** were your current directory, then the relative pathname for **file2** would be **bin/file2**.

If you are using C shell, TC shell, or the Bourne Again shell, UNIX provides some abbreviations for a few special directories. The character "~" (tilde) refers to your home directory. The home directory of any user (including you, if you want) can be abbreviated from */parent-directories/userid* to ~userid. Likewise, you can abbreviate */parent-directories/youruserid/file* to ~/file. The current directory has the abbreviation . (period). The parent of the current directory uses .. (two consecutive periods) as its abbreviation.

Displaying Directories

When you initially log in, the UNIX system places you in your home directory. The **pwd** command will display the full pathname of the current directory you are in.

```
pwd
/home/userid
```

By typing the **ls -a** command, you can see every file and directory in the current directory, regardless of whether it is your home directory. To display the contents of your home directory when it is not your current directory, enter the **ls** command followed by the full pathname of your home directory.

```
ls /home/userid
```

If you are using a shell other than the Bourne shell, instead of typing the full pathname for your directory, you can also use the tilde symbol with the **ls** command to display the contents of your home directory.

```
ls ~
```

To help you distinguish between files and directories in a listing, the **ls** command has a **-F** option, which appends a distinguishing mark to the entry name showing the kind of data it contains: no mark for regular files; "/" for directories; "@" for links; "*" for executable programs:

```
ls -F ~
```

Changing Directories

To change your current directory to another directory in the directory tree, use the **cd** command. For example, to move from your home directory to your **projects** directory, type:

cd projects (relative pathname from home directory) or,

cd ~/projects (full pathname using ~)

or,

cd /home/*userid*/projects (full pathname)

Using **pwd** will show you your new current directory.

```
pwd
/home/userid/projects
```

To get back to the parent directory of **projects**, you can use the special ".." directory abbreviation.

```
cd ..
pwd
/home/userid
```

If you get lost, issuing the **cd** command without any arguments will place you in your home directory. It is equivalent to **cd ~**, but also works in the Bourne shell.

Moving Files Between Directories

You can move a file into another directory using the following syntax for the **mv** command:

mv *source-filename destination-directory*

For example,

```
mv sample.txt ~/projects
```

moves the file **sample.txt** into the **projects** directory. Since the **mv** command is capable of overwriting files, it would be prudent to use the **-i** option (confirmation prompt). You can also move a file into a another directory and rename it at the same time by merely specifying the new name after the directory path, as follows:

```
mv sample.txt ~/projects/newsample.txt
```

Copying Files to Other Directories

As with the **mv** command, you can copy files to other directories:

```
cp sample.txt ~/projects
```

As with **mv**, the new file will have the same name as the old one unless you change it while copying it.

```
cp sample.txt ~/projects/newsample.txt
```

Renaming Directories

You can rename an existing directory with the **mv** command:

mv *oldDirectory newDirectory*

The new directory name must not exist before you use the command. The new directory need not be in the current directory. You can move a directory anywhere within a file system.

Removing Directories

To remove a directory, first be sure that you are in the parent of that directory. Then use the command **rmdir** along with the directory's name. You cannot remove a directory with **rmdir** unless all the files and subdirectories contained in it have been erased. This prevents you from accidentally erasing important subdirectories. You could erase all the files in a directory by first going to that directory (use cd) and then using **rm** to remove all the files in that directory. The quickest way to remove a directory and all of its files and subdirectories (and their contents) is to use the **rm -r** (for recursive) command along with the directory's name. For example, to empty and remove your **projects** directory, move to that directory's parent, then type :

rm -r projects (remove the directory and its contents)

□□□