Kamal Saini

OPERATING SYSTEM

PAPER CODE: PCC-CSE-206-G

4th Semester

PREVIOUS YEAR'S SOLVED QUESTION PAPERS

For

B.Tech. STUDENTS

Solved By

A TEAM OF EXPERIENCED FACULTY

RG PUBLICATION

Website : www.rgpublication.in
Email : rgpublishinghouse@gmail.com

SYLLABUS PCC-CSE-206 G

OPERATING SYSTEM

Class Work : 25 Marks
Exam : 75 Marks
Total : 100 Marks
Duration of Exam. : 3 Hrs.

SECTION-A

Introduction: Concept of Operating Systems, Generations of Operating systems, Types ofOperating Systems, OS Services.

Processes: Definition, Process Relationship, Different states of a Process, Process State transitions, Process Control Block (PCB), Context switching. Thread: Definition, Various states, Benefits of threads, Types of threads, Multithreading.

Process Scheduling: Foundation and Scheduling objectives, Types of Schedulers, Scheduling criteria: CPU utilization, Throughput, Turnaround Time, Waiting Time, Response Time: Scheduling algorithms: Pre-emptive and Non-pre-emptive, FCFS, SJF, SRTF, RR Scheduling.

SECTION-B

Inter-process Communication: Critical Section, Race Conditions, Mutual Exclusion, The Producer\Consumer Problem, Semaphores, Event Counters, Monitors, Message Passing, Classical IPC Problems: Reader's & Writer Problem, Dinning Philosopher Problem etc.

Deadlocks: Definition, Necessary and sufficient conditions for Deadlock, Deadlock Prevention, and Deadlock Avoidance: Banker's algorithm, Deadlock detection and Recovery.

SECTION-C

Memory Management: Basic concept, Logical and Physical address map, Memory allocation: Contiguous Memory allocation – Fixed and variable partition–Internal and Externalfragmentation and Compaction; Paging: Principle of operation – Page allocation – Hardware support for paging, Protection and sharing, Disadvantages of paging.

Virtual Memory: Basics of Virtual Memory – Hardware and control structures –Locality of reference, Page fault, Working Set, Dirty page/Dirty bit – Demand paging, Page Replacement algorithms: Optimal, First in First Out (FIFO), Optimal Page Replacement and Least Recently used (LRU).

SECTION-D

File Management: Concept of File, Access methods, File types, File operation, Directory structure, File System structure, Allocation methods (contiguous, linked, indexed), efficiency and performance.

Disk Management: Disk structure, Disk scheduling - FCFS, SSTF, SCAN, C-SCAN, Disk reliability, Disk formatting, Boot-block, Bad blocks, Case study on UNIX and WINDOWS Operating System.

NOTE : Examiner will set nine questions in total. Question one will be compulsory. Question one will have 6 parts of 2.5 marks each from all units and remaining eight questions of 15 marks each to be set by taking twoquestions from each unit. The students have to attempt five questions in total, first being compulsory and selecting one from each unit.

PRINCIPLES OF OPERATING SYSTEM

Dec 2013

Paper Code: CSE-301-F

Figures in the bracket indicate full marks.

Note: ***Attempt five questions in total selecting one question from each of the four Sections. Question No.1 is compulsory.***

Q.1 Explain the following briefly :

(a) TLB

(b) Thrashing

(c) Belady's Anamoly

(d) Batch OS. **(20)**

Ans. (a) TLB : A translation lookaside buffer (TLB) is a cache that memory management hardware uses to improve virtual address translation speed. The majority of desktop, laptop, and server processors includes one or more TLBs in the memory management hardware, and it is nearly always present in any hardware that utilizes paged or segmented virtual memory.

The TLB is sometimes implemented as content-addressable memory (CAM). The CAM search key is the virtual address and the search result is a physical address. If the requested address is present in the TLB, the CAM search yields a match quickly and the retrieved physical address can be used to access memory. This is called a TLB hit. If the requested address is not in the TLB, it is a miss, and the translation proceeds by looking up the page table in a process called a page walk. The page walk requires a lot of time when compared to the processor speed, as it involves reading the contents of multiple memory locations and using them to compute the physical address. After the physical address is determined by the page walk, the virtual address to physical address mapping is entered into the TLB. The operation of more common TLB implementations is using hash tables,[citation needed] similarly to the CAM implementations.

Ans. (b) Thrashing : If the process does not have number of frames it needs to support pages in active use, it will quickly page-fault. The high paging activity is called thrashing.

In other words we can say that when page fault ratio decreases below level, it is called thrashing.

Causes of thrashing : It result in several performance problems.

(i) If CPU utilization is too low then we increase the degree of multiprogramming by introducing a new process to the system. A global page replacement algorithm is used. The CPU scheduler sees the decreasing CPU utilization and increases the degree of multiprogramming.

(ii) CPU utilization is plotted against the degree of multiprogramming.

(iii) As the degree of multiprogramming increases, CPU utilization also increases.

(iv) If the degree of multiprogramming is increased further, thrashing sets in and CPU utilization drops sharply.

(v) So, at this point, to increase CPU utilization and to stop thrashing, we must decrease the degree of multiprogramming.

To limit the effect of thrashing : To limit the effect of thrashing we can use *local replacement algorithm*. With Local replacement algorithm, if the process starts thrashing, it

cannot steal frames from another process and cause the latter to thrash as well. The problem is not entirely solved.

Thus the effective access time will increase even for the process that is not thrashing.

How to prevent Thrashing : We must provide a process with as many frames as it needs. Several techniques are used.

(i) *The Working of Set Model* (*Strategy*), It starts by looking at how many frames a process is actually using. This defines the locality model.

(ii) *Locality Model,* It states that as a process executes, it moves from locality to locality.

(iii) A *locality* is a set of pages that are actively used together.

Ans. (c) Belady's Anamoly : Belady's Anamoly is the name given to the phenomenon where increasing the number of page frames results in an increase in the number of page faults for a given memory access pattern. This phenoomenon is commonly experienced when using the First in First Out (FIFO) page replacement algorithm. Laszlo Belady demonstrated this in 1969.

In common computer memory management, information is loaded in specific sized chunks. Each chunk is referred to as a page. The central processor can only hold a limited number of pages at a time. It requires a frame for each page it can load. A page fault occurs when a page is not found, and might need to be loaded from disk into memory.

When a page fault occurs and all frames are in use, one must be cleared to make room for the new page. A simple algorithm is FIFO. Whichever page has been in the frames the longest is the one that is cleared. Until Beladys anomaly was demonstrated, it was believed that an increase in the number of page frames would always result in the same number of fewer page faults.

Example of Belady's anomaly : Using three page frames, 9 page faults occur. Increasing to four page frames causes 10 page faults to occur. Page faults are in red

Page Requests	3	2	1	0	3	2	4	3	2	1	0	4
Newest Page	3	2	1	0	3	2	4	4	4	1	0	0
		3	2	1	0	3	2	2	2	4	1	1
Oldest Page			3	2	1	0	3	3	3	2	4	4

Page Requests	3	2	1	0	3	2	4	3	2	1	0	4
Newest Page	3	2	1	0	0	0	4	3	2	1	0	4
		3	2	1	1	1	0	4	3	2	1	0
			3	2	2	2	1	0	4	3	2	1
Oldest Page				3	3	3	2	1	0	4	3	2

Ans. (d) Batch O.S. : In this operating system user do not allow to interact the computer directly. Each user prepares his job on an off-line device like punch cards and submits it to the computer operator. The operator sorts jobs into batches with similar requirements and, as the computer become available runs each batch. When the job is complete its output is sent back to the appropriate user. In batch system, major task of operation system is to transfer control automatically from one job to the next.

Disadvantages : (i) Lack of interaction between the user and the job while job is executing.

(ii) Turn around time is high. The delay between job submission and job completion is called turnaround time.

(iii) Utilisation CPP is very poor. The CPU is often idle.

(iv) Dificult to provide the desired priority.

Section-A

Q.2.(a) Differentiate between multiprogramming and multitasking.

Ans. Multi programming : Multiprogramming is the technique of running several programs at a time using timesharing. It allows a computer to do several things at the same time. Multiprogramming creates logical parallelism. The concept of multiprogramming is that the operating system keeps several jobs in memory simultaneously. The operating system selects a job from the job pool and starts executing a job, when that job needs to wait for any i/o operations the CPU is switched to another job. So the main idea here is that the CPU is never idle.

Multi tasking : Multitasking is the logical extension of multi-programming .The concept of multitasking is quite similar to multiprogramming but difference is that the switching between jobs occurs so frequently that the users can interact with each program while it is running. This concept is also known as time-sharing systems. A time-shared operating system uses CPU scheduling and multiprogramming to provide each user with a small portion of time-shared system. For example, let us say you are printing a documet of 100 pages. While your computer is performing that, you still can do other jobs like typing a new document. So, more than one task is performed.

One of the main differences between multiprogramming and multitasking is, "In multiprogramming, a user cannot interact (everything is decided by OS, like picking the next program and sharing on time basis, etc...) where as in multitasking, a user can interact with the system (you can type a letter, while the other task of printing is going on)"

Q.2.(b) Discuss the various operating system programs. **(8)**

Ans. Various operating system programs are as follows :

(i) File management : These programs create, delete, copy, rename, print, dump, list, and generally manipulates files and directories.

(ii) Status information : Some programs simply ask the operating system for date, time, amount of available memory or disk space, number of users or similar status information.

(iii) File modification : Several text editors may be available to create or modify the content of files stored on disk.

(iv) Programming-language support : Compilers, assemblers, and interpreters for common programming languages (such as C, C++, Java, Visual Basic, PERL) are after provided to the user with the operating system.

(v) Program loading and execution : When a program is assembled or complied it must be loaded into memory for its execution. The system may provide absolute loaders, relocatable loaders, linkage editors, and overlay loaders. Debugging systems for either high level language or machine language are also needed.

(vi) Communication : These programs provide the mechanism for creating the virtual connections among processes, users and different computer systems. They allow users to send messages through web browsers, E-mail services from one machine to another.

(vii) System Utilities : Most operating system are supplied with programs that solve common problems, or perform common operations such as web browsers, word processors, statistical - analysis packages, games, spread sheets, database systems, etc.

Q.2.(c) Discuss the architecture of operating system.

Ans. Architecture of operating system : An operating system manages four major resources of the computer system : memory, processor, Input/output devices and files. So, an operating system provides four operations :

(i) *Memory management function* finds free space in memory and allocate it to different processes.

(ii) *Processor management function* allocates the processor to execute a chosen process.

(iii) I/O *device management function* allocates a device to a process.

(iv) *File management function* keeps track of all information about files that how they are opened or closed.

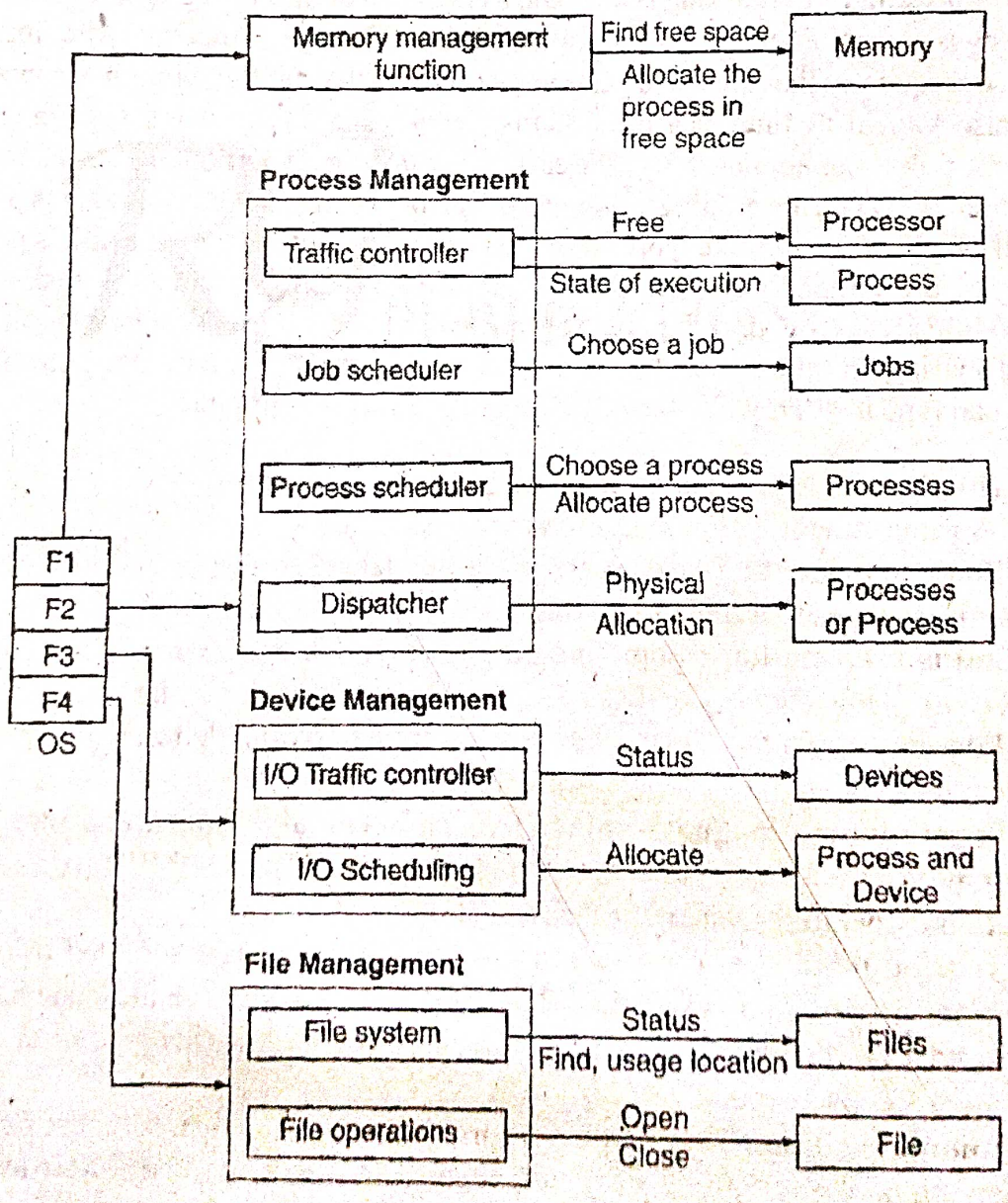


Fig. : Architecture of Operating system

Process management function is performed by traffic controller, job scheduler, process scheduler and dispatcher.

– *Traffic controller* is that part of operating system that constantly checks processor and status of processes.

– *Job scheduler* selects a job from the job queue submitted to the system for execution.

– *Process scheduler* decides when a process is to be executed in the case of multiprogramming.

– *Dispatcher* allocates the processor for a particular process which is chosen by process scheduler.

– *I/O traffic controller* is a program which constantly keeps the track of I/O devices, channels, control processors.

– *I/O Scheduler decides* which device is to be allocated to which process.

– *File system* is a collection of functions which are used to find status, usage and location of a file. From this all informations can be easily find which are stored on a disk in the form of files.

Q.3. Define the following terms :

(a) Dispatcher

(b) Long and short term scheduler

(c) PCB

(d) CPU scheduling criteria. **(20)**

Ans. (a) Dispatcher : The dispatcher is the module that gives control of the CPU to the process selected by the short-term scheduler. This function involves :

(i) Switching context

(ii) Switching to user mode

(iii) Jumping to the proper location in the user program to restart that program.

The dispatcher should be as possible, given that it is invoked during every process switch. The time it takes for the dispatcher to stop one process and start another running is known as the dispatch latency.

Ans. (b) Long term scheduler :

- It is a first level scheduler.
- It can be found in operating system where there is a support of 'batch', like as batch operating system.
- It works with the 'batch queue' and selects the next batch job or process to be executed. Such processes are loaded in main memory and it waits in 'ready' state for CPU to become free.
- It executes much less frequently as compared to other schedulers. The reason is: there may be minutes between the creation of new processes in the system, and this scheduler is required only when new process enters the system.
- It controls the degree of multi programming, i.e. the total number of processes residing in the main memory.

Objective : To provide balanced mix of jobs, such as CPU bound and I/O bound, to the short term scheduler.

Short term scheduler :

- It is a third level scheduler.
- It can be found always in all modern operating system.
- It works with the 'ready queue'. It selects the next process to be executed by CPU whenever more than one process is simultaneously in the 'ready' state.
- It controls the 'ready to running' state transition in process life cycle.
- It executes much frequently compared to other schedulers. And due to this reason, it must to be fast to keep scheduling overhead low.

Objective : To keep CPU always busy, to get better utilization.

Ans. (c) PCB : A process in an operating system is represented by a data structure known as a process control block (PCB) or process descriptor. The PCB contains important information about the specific process including :

– The current state of the process *i.e.*, whether it is ready, running, waiting, or whatever.
– Unique identification of the process in order to track "which is which" information.
– Pointer to parent process.
– Similarly, a pointer to child process (if it exists).
– The priority of process (a part of CPU sheduling information).
– Pointers to locate memory of processes.
– A register save area.
– The processor it is running on.

Ans. (d) CPU scheduling criteria : The criteria includes the following :

(i) CPU utilization : We have to keep the CPU as busy as possible. It may range from 0 to 100%. In a real system it should range from 40-90% for lightly loaded and heavily loaded systems.

(ii) Throughput : It is the measure of work in terms of number of processes completed per unit time. For long processes, this rate may be 1 process per hour; for short transactions, throughput may be 10 process per second.

(iii) Turnaround Time : It is the sum of time-periods spent in waiting to get into mamory, waiting in ready queue, executing on the CPU and doing I/O time.

i.e., turn around time = time of completion of job – time of submission of job.

(iv) Waiting Time : The CPU scheduling algorithms does not affect the amount of time during which a process executes or does I/O because they are fixed for a particular system. It affects only the amount of time that a process spends waiting in a ready queue. Waiting time is the sum of time-periods spent waiting in ready queue.

(v) Response Time : It is the amount of time, it takes to start responding, not the time taken to output that response.

Section-B

Q.4.(a) Discuss the hardware support required to demand paging. **(10)**

Ans. Hardware support required to demand paging : For every memory access operation, the page table needs to be consulted to check whether the corresponding page is resident or not and whether the program has read or write privileges for accessing the page. These checks would have to be performed in hardware. A TLB could serve as a cache and improve the performance of the lookup operation.

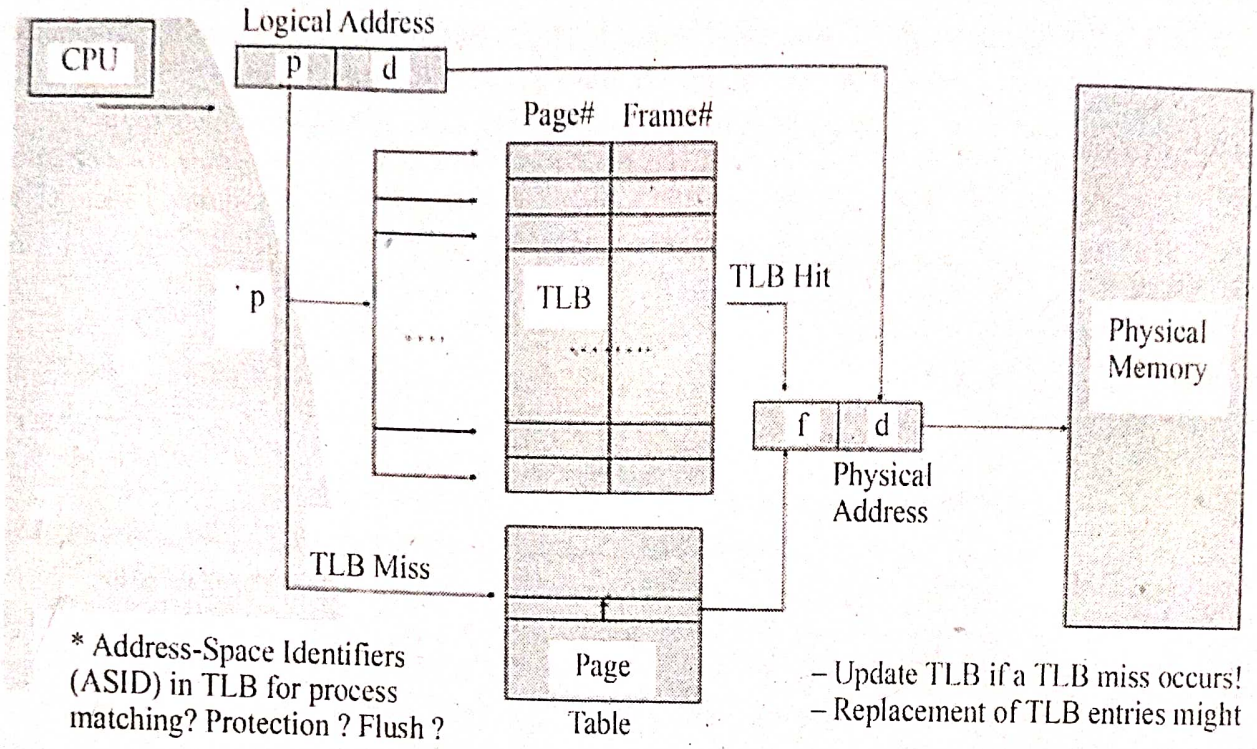


Fig. : Hardware support to demand paging

(i) *Page Table :* Table has the ability to mark an entry invalid through a valid-invalid bit/special value of protection bits.

(ii) *Secondary Memory :* Holds pages that are not in main memory (disk), known as swap device and section of disk used for this purpose is swap space/backing store.

Q.4.(b) Discuss the situations under which the MFU page replacement algorithm generates fewer page faults than LRU algorithm. Also discuss under what circumstances the opposite holds. **(10)**

Ans. Consider the sequence in a system that holds four pages in memory: 1 2 3 4 4 4 5 1. The most frequently used page replacement algorithm evicts page 4 while fetching page 5, while the LRU algorithm evicts page 1. This is unlikely to happen much in practice. For the sequence "1 2 3 4 4 4 5 1," the LRU algorithm makes the right decision.

Q.5(a) Explain the concept of virtual memory and how it is obtained by demand paging and segmentation ? **(12)**

Ans. Virtual memory : Virtual memory is a memory hierarchy consisting of a computer system's memory and a disk, that enables a process to operate with only some portions of its address space in memory.

The Fig.(1) shows simple model of virtual memory system. The logical memory is divided into pages. The physical memory is divided into frames. The mapping is done with the help of page table. The virtual memory system is very similar to paging system except one difference. In virtual memory system paging is done with swapping, is the difference from paging system. The operating system checks whether the required page is available in the main memory, if it is not available then it will swap the required page from the secondary memory. We can understand the functions involved in the virtual memory system, form the fig.(1).

The Virtual memory allows many processes to be executed simultaneously. It increases the CPU utilization. The degree of multi programming is also increased due to virtual memory. This can be implemented using paging, to be more precise, demand paging.

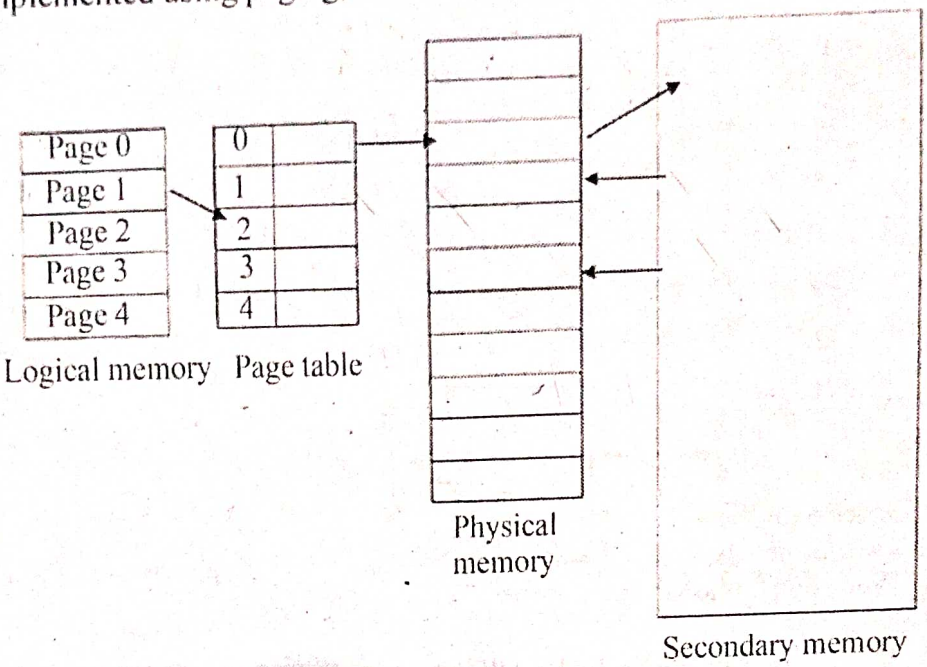


Fig.(1) : Virtual memory system

Demand Paging : The virtual memory can be implemented by demand paging. The demand paging system is used to implement the virtual memory. The demand paging is also a paging system with swapping. The page will be swapped only when there is a demand. This paging is very dynamic in nature. This entire paging activity is done only on demands. All the components discussed in the virtual memory system are also available and functions are also same. The paging is done only on demand. The page table is slightly modified to carry out the demand paging. The page table has one more entity 'present bit' or valid-invalid bit. This is shown in the fig.(2).

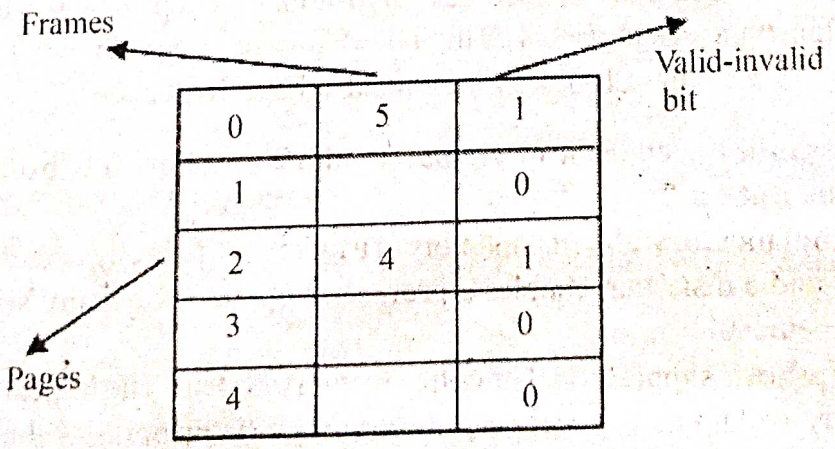


Fig.(2) : Page table in demand paging

The valid-invalid bit is an indicator, which is used to indicate whether the required page is available the physical memory. It can have either '0' or 1 value. If it is '1' means, it is valid and requested page is available in the main memory. If the value is '0' means it is invalid and the

requested page is not available in the main memory. The demand paging can be accomplished using the page table with valid-invalid information.

A process is requesting a page in the physical memory. The operating system checks the page table, if it valid then it is available, the process can proceed further. If it is invalid means the page is not available in the physical memory. This is called page fault. When there is a page fault, then operating system trap occurs. Then the process is terminated. The operating system brings the required page form the secondary memory to the physical memory. Then it sets the valid-invalid bit to '1' in the page table. Once again the process is resumed by restarting the same instruction. These are the functions involved in the demand paging. In order to run a process, the entire memory is not needed in the physical memory. This is due to the demand paging. The demand paging increases the CPU utilization. It also increases the degree of mutli programming *i.e.*, it allows execution more number of processes. The performance of demand paging is tolerable as long as the page fault rate is low.

Segmentation : It is a memory-management scheme that supports this user view of memory. A logical address space is a collection of segments. Each segment has a name and a length. The addresses specify both the segment name and the offset within the segment. The user therefore specifies each address by two quantities : a segment name and an offset.

For simplicity of implementation, segments are numbered and are referred to by a segment number, rather than by a segment name. Thus, a logical address conists of a two tuple :

< segment-number, offset>

Normally, the user program is compiled, and the compiler automatically constructs segments reflecting the input program.

A *C* complier might create separate segments for the following:

– The code
– Global variables
– The heap, from which memory is allocated
– The stacks used by each thread
– The standard *C* library

Libraries that are linked in during compile time might be assigned separate segments. The loader would take all these segments and assign them segment numbers.

Q.5.(b) Explain the process of handling page faults. (8)

Ans. Process of handling page faults goes through following steps:

(i) It checks an internal table for this program to determine whether the reference was a valid or invalid memory access.

(ii) If the reference is invalid, it terminates the process. If it is valid, but the referenced page is not available in the physical memory, trap is activated.

(iii) It finds a free frame.

(iv) It schedules a disk operation to read the desired page into the newly allocated frame.

(v) When the disk read is complete, it modifies the internal table kept with the program and the page table to indicate that the page is now in memory.

(vi) It restarts the instruction that was interrupted by the illegal address trap. The program can now access the page as though it had always been in memory.

Section-C

Q.6.(a) Define semaphore. How can semaphore be used to enforce mutual exclusion ? Explain with exmple. (10)

Ans. Semaphore : An integer value used for signaling among processes. Only three operations may be performed on a semaphore, all of which are atomic : initialize, decrement, and increment. The decrement operation may result in the blocking of a process, and the increment operation may result in the unblocking of a process. Also known as a counting semaphore or a general semaphore.

Mutual exclusions with semaphores :

– A semaphore is a non-negative integer value

– Two atomic operations are supported on a semaphore

– *down* = decrements the value, since the value cannot be negative, the process blocks if the value is zero

– *up* = increments the value; if there are any processes waiting to perform a down, then they are unblocked

– Initializing a semaphore to 1 creates a mutual exclusion (mutex) semaphore

– Initializing a semaphore to 0 creates a signaling semaphore

– Initializing a semaphore to other values can be used for resource allocation and synchronization.

Example :

```
/* program mutualexclusion */
const int n = /* number of processes */;
semaphore s = 1; /* Initialize s to 1 */
void P (int i)
{
        while (true) {
                semWait (s);
                /* critical section */;
                /* remainder */;
        }
}
void main ( )
{
        parbegin (P(1), P(2), ...., P(n));
}
```

Q.6(b) What do you mean by deadlock ? How can we handle a deadlock if it occurs ? (10)

Ans. Deadlock : A deadlock is a situation wherein two transactions wait for each other to give up their respective locks.

Handling deadlocks : Table describes three fundamental approaches used in deadlock handling. Each approach has different implications for user processes and for the operating system.

Table : Deadlock Handling Approaches

Approach	Description
Deadlock Prevention	The kernel uses a resource allocation policy that ensures that the four conditions for resource deadlock (Mutual Exclusion, Hold-and-wait, No preemption and circular-wait) do not arise simultaneously. This approach makes deadlock impossible.
Deadlock avoidance	The kernel analyzes the allocation state to determine whether granting a resource request might lead to a deadlock later on. Only requests that cannot lead to a deadlock are granted, others are kept pending untill they can be granted.
Deadlock detection and resolution.	The kernel analyzes the resource state to check whether a deadlock exists. If so, it aborts some process (es) and allocates the resources held by them to other processes so that the deadlock classes to exist.

To ensure that deadlocks never occur, the system can use either a *deadlock prevention* or *deadlock avoidance scheme*.

Deadlock avoidance requires that the operating system be given in advance additional information concerning which resources, a process will request and use during its lifetime. With this additional knowledge, we can decide for each request whether or not the process should wait. To decide whether the current request can be satisfied or delayed, the system must consider the resources currently available, the resources currently allocated to each process, and the future request and releases of each process.

If a system does not employ either a *deadlock prevention* or a *deadlock avoidance* algorithm, then a deadlock situation may occur. In this environment, the sysem can provide an algorithm that examines the state of the system to determine whether a deadlock has occurred, and and algorithm to resolve or recover from the deadlock.

If a system does not ensure that a deadlock will never occur and also does not provide a mechanism for *deadlock detection* and *resolution*, then we may arrive at a situation where the system is in a deadlock state yet has no way of recognizing what has happened. In this case, the undetected deadlock will result in the deterioration of the system performance, because resources are being allocated by the processes that cannot run and because more and more processes, as they make requests for resources, enter a deadlock state. Eventually, the system will stop functioning and will need to be restarted manually.

Q.7.(a) Discuss the various features of distributed file system in detail. (10)

Ans. Various features of distributed file system are as follows :

(i) Transparency : It have following types :

– *Structure transparency :* A DFS normally uses mutliple file servers. In multiple file servers, the multiplicity of file servers should be transparent to the clients of a DFS. In particular client should not know the number or locations of the File Servers and the storage devices.

– *Access transparency :* File system information should not distinguish between local and remote files.

– *Naming transparency :* The name of a file should give no hint as to where the file is located or a file should be allowed to move from one place to another.

(ii) Replication : Existence of replicas and their locations should be hidden from the user.

(iii) User Mobility :

Performance : It is usually measured as the average amount of time needed to satisfy client requests.

In CFS → Time for accessing storage device + CPU

In DFS → Time for accessing + CPU + communication O/H.

(iv) Simplicity and easy of use : The semantics of the DFS should be easy to under stand *i.e.*, user I/F to the File System must be simple and the number of commands should be as small as possible.

(v) Scalability : A good DFS should be designed to easily cope with the growth of nodes and users in the system *i.e.*, this growth should not cause significant loss of performance to users.

(vi) High availability : A DFS should continue to function even when partial failures occur due to the failure of one or more components.

(vii) High reliability : In a good DFS, the probability of loss of stored data should be minimized as far as practicable.

(viii) Data integrity : A file is often shared by multiple users. For a shared file, the file system must guarantee the integrity of data stored in it. *i.e.*, some form of concurrency of their data.

(ix) Security : A DFS should be secure so that its users can be confident of the privacy of their data.

(x) Heterogeneity : A DFS should be designed to allow the integration a new type of workstations or storage media.

Q.7.(b) Explain the concept of booting from disk and bad block recovery in disk management. **(10)**

Ans. The disk management consists if disk formatting, block and bad blocks.

Boot blocks : Whenever a computer is switched on or power is switched on the operating system boots up. This booting system requires some of the programs to initially run on it. Hence the system initializes all aspects of system from CPU registers to device controllers and content of main memory and then start the operating system. To do its jobs, boot strap program finds operating system kernel an disk, loads that kernel into memory and jumps to an initial address to begin operating system execution. Boot straps are stored on ROM (Read Only Memory), because ROM needs no initialization and is at fixed location that the processor can start executing when powered up or reset. Hence ROM cannot get infected by any virus as it is read only file. So no virus can write on it.

The code in boot ROM instruct the disk controller to read boot block into memory and then start executing that code. While disk controller reads boot block, no device are loaded into memory at this pointer of time. The full boot strap program is more sophisticated than boot strap loader in boot ROM and is able to load entire operating system from non-fixed location on disk

and to start operating systems to run. For *e.g.* MS DOS uses 512 bytes block for its boot programs.

Bad blocks : The disk having moving parts are prone to failure. Hence the disk needs to be replaced when failure has occurred and the contents are restored from backup media to new disk. Hence due to this one or more sector of disk becomes defective. Most disk even come from the factory with bad blocks. Hence these blocks are handled by controller in different ways.

Disk with IDE controller handle bad blocks manually. MS-DOS has format command which does a logical format of disk. It scans the disk and finds bad block. If bad block is found then, a special value is written into corresponding FAT entry to tell the allocation routines not to use that block. If block goes bad during operation a special program (chkdsk) must be run manually to search for bad block and lock them. Data residing in bad blocks are lost.

In SCSI disk, whenever bad blocks are found, the controller were adviced to change the bad block sectors with the spare sectors. This method is called sector sparsing or forwarding.

The typical bad sector transactions are as follows :

– The operating system tries to read logical block called logical block 87.

– The controller calculates ECC and finds which sector is bed. It reports the finding to the operating system.

– The next time that the system is re booted, a special command runs to tell SCSI controller to replace bad sector with spare sector.

After that whenever the system request for logical block 87, the request in translated into replacement sectors addressed by controller.

Section-D

Q 8.(a) Explain the life cycle of transferring I/O requests to hardware operations. **(12)**

Ans. Typical life cycle of a blocking read request, as depicted in fig. The fig. suggests that an I/O operation requires a great many steps that together consume a tremendous number of CPU cycles.

(i) A process issues a blocking read ( ) system call to a file descriptor of a file that has been opened previously.

(ii) The system-call code in the kernel checks the parameters for correctness. In the case of input, if the data are already available in the buffer cache, the data are returned to the process, and the I/O request is completed.

(iii) Otherwise, a physical I/O must be performed. The process is removed from the run queue and is placed on the wait queue for the device, and the I/O request is scheduled. Eventually, the I/O subsystem sends the request to the device driver. Depending on the operating system, the request is sent via a subroutine call or an in-kernel message.

(iv) The device driver allocates kernel space to receive the data and schedules the I/O. Eventually, the driver sends commands to the device controller by writing into the device-control registers.

(v) The device controller operates the device hardware to perform the data transfer.

(vi) The driver may poll for status and data, or it may have set up a DMA transfer into kernel memory. We assume that the transfer is managed by a DMA controller, which generates an interrupt when the transfer completes.

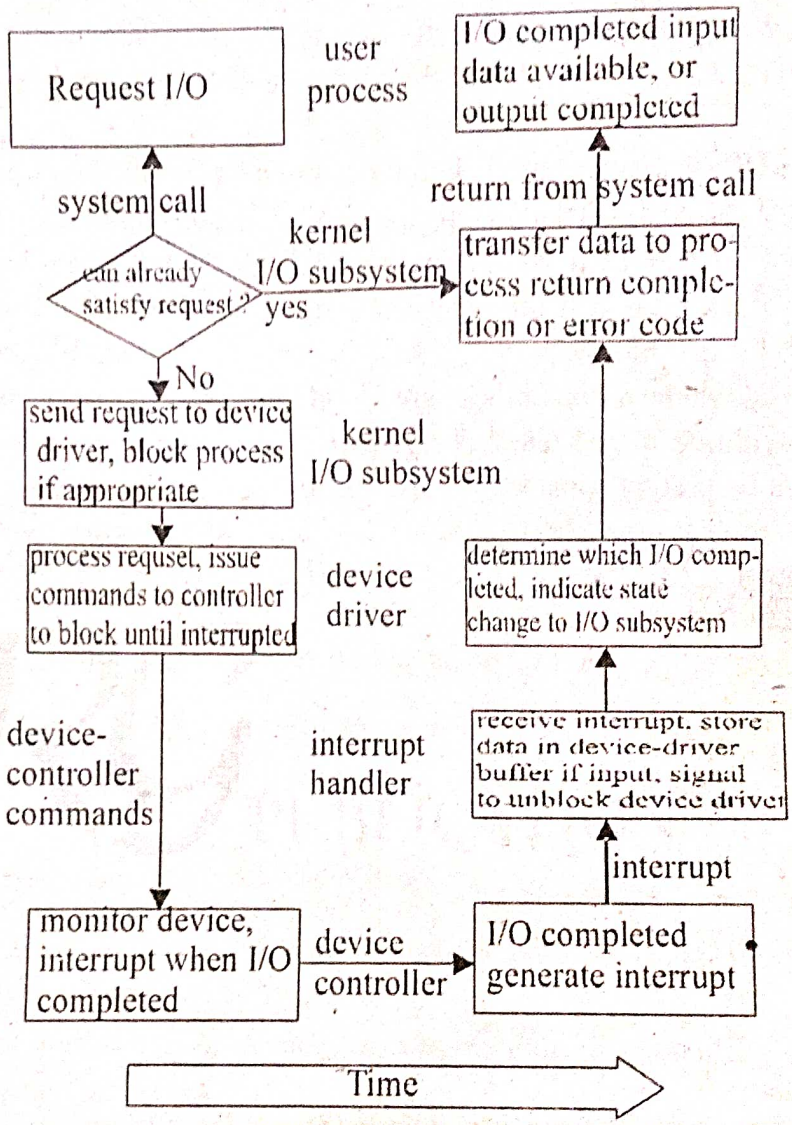


Fig. : The life cycle of an I/O request

(vii) The correct interrupt handler receives the interrupt via the interrupt vector table, stores any necessary data, signals the device drive, and returns from the interrupt.

(viii) The device driver receives the signal, determines which I/O request has completed, determines the request's status, and signals the kernel I/O subsystem that the request has been completed.

(ix) The kernel transfers data or return codes to the address space of the requesting process and moves the process from the wait queue back to the ready queue.

(x) Moving the process to the ready queue unblocks the process. When the scheduler assigns the process to the CPU, the process resumes execution at the completion of the system call.

Q.8.(b) Define the terms - Buffering and Polling.

Ans. Buffering : If a block is being transferred from a user process area directly to an input/output module, then the process is blocked during the transfer and the process may not be swapped out. To avoid these overheads and in efficiencies, it is convenient to perform input transfers in advance of request being made and to perform output transfer sometimes later, after the request has been made and to perform output transfer sometimes later, after the request has been made. This technique is known as buffering. Buffering have following types :

(i) *Single buffering* **:** The simplest type of buffering supported by an operating system is called single buffering (Fig.1). When a user process issues an input/output request, the operating system assigns a buffer in the system that is a portion of main memory to the operation.

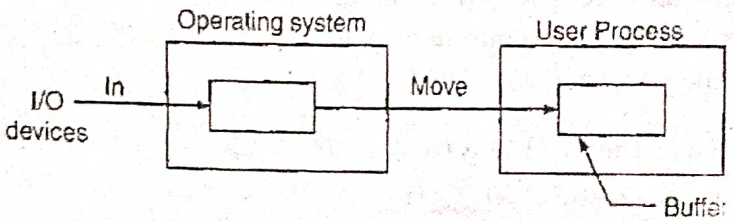


Fig.(1) : Single buffering

(ii) *Double buffering :* An improvement over single buffering can be made by assigning two system buffers to the input/output operations. A process new transfers data to (or from) one buffer while the operating system empties (or files) the other . This technique is known as double buffering or buffer swapping (fig.2).

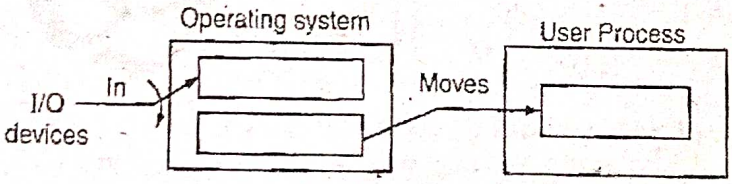


Fig.(2) : Double buffering

(iii) *Circular buffering :* When more than two buffers are used, the collection of buffers is itself referred as a circular buffer (fig.3), with each individual buffer being one unit in the circular buffer. This is simply the producer-consumer problem with bounded buffer.

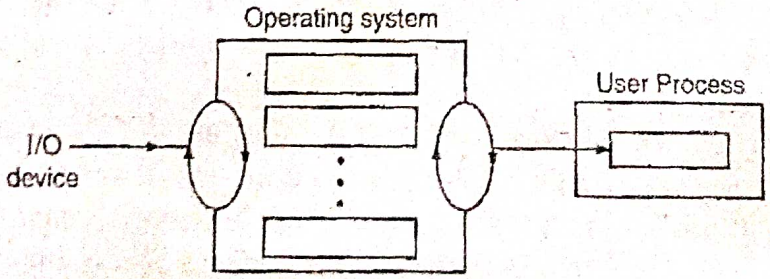


Fig.(3) : Circular buffering

Polling : Polling, or polled operation, in computer science, refers to actively sampling the status of an external device by a client program as a synchronous activity. Polling is most often used in terms of input/output (I/O), and is also referred to as polled I/O or software-driven I/O.

Polling has the disadvantage that if there are too many devices to check, the time required to poll them can exceed the time available to service the I/O device. Polling can be described in following steps :

– The host repeatedly reads the busy bit of the controller until it becomes clear.

– When clear, the host writes in the command register and writes a byte into the data-out register.

– The host sets the command-ready bit (set to 1).

– When the controller senses command-ready bit is set, it sets busy bit.

– The controller reads the command register and since write bit is set, it performs necessary I/O operations on the device. If the read bit is set to one instead of write bit, data from device is loaded into data-in register, which is further read by the host.

– The controller clears the command-ready bit once everything is over, it clears error bit to show successful operation and reset busy bit (0).

Q.9.(a) Discuss the UNIX system calls for process management. **(10)**

Ans. System calls provide an interface between the process and the operating system. System calls allow user-level processes to request some services from the operating system which process itself is not allowed to do. In handling the trap or system call, the operating system will enter in the kernel mode, where it has access to privileged instructions and can perform the desired service on the behalf of user-level process. It is because of the critical nature of operations that the operating system itself does them every time they are needed. For example, for I/O a process involves a system call telling the operating system to read or write particular area and this request is satisfied by the operating system.

The UNIX system - comprising the kernel, shell and applications - is written in C. Thought there are over a thousands commands in the system, they all use a handful of functions, called system calls, to communicate with the kernel. All UNIX flavors have one thing in common : They use the same system calls. These calls are described in the POSIX specification. If an operating system uses different system calls, then it won't be UNIX. The reason why Linux can't replace UNIX is that Linux uses the same system calls ; Linus is UNIX.

A typical UNIX command writes a file with the write system call without going into the innards that actually achieve the write operation. Often the same system call can access both a file and a device; the open system call opens both. These system calls are built into the kernel, and interaction through them represents an efficient means of communication with the system. This also means that once software has been developed on one UNIX system, it can easily be ported to another UNIX machine.

C programmers on a Window system use the standard library functions for everything. You can't use the write system call on a Windows system; you'll need to use a library function like fprintf for the purpose. In contrast, the C programmer in the UNIX environment has complete access to the entire system call library as well as the standard library functions.

Q.9.(b) Explain the following : **(10)**

(i) Shell interpreter

(ii) DMA

Ans. (i) Shell Interpreter : The shell or the command interpreter is the routine, which communicates with the user in a console based operating system. The shell displays a prompt gets user input analyses it and performs the neccessary action. Usually there are commands that

the shell understands itself and provides the user with the necessary result these commands are called internal commands. Sometimes the shell doesn't understand the command and usually in this situationit tries to find a file with the command name and loads it for execution, if it is executable.

The shell is the interface between the operating system and the user and hence it is very important. The design of a shell takes into consideration many factors :

(i) Should the shell be part of the kernel or should it be another application that the kernel loads automatically? This is an important consideration as shell being part of the kernel will usually retain that space in memory throughout, resulting in reduced memory availability for user applications. The other option is better as the shell can be overwritten by the user application and re-loaded when the user application terminates. We will use the former approach as it is easy to code and we are aiming to provide only minimal functionality with the shell, which wouldn't take much space.

(ii) Length of the commands the shell can receive. We will set this limit as 255 characters. The shell will receive only characters.

(iii) How many operands will a typical command have ? The operands can modify the command and sometimes will be parameters to the command. We will decide that the shell will accept a command with 4 operands and anything more will be discarded. The command and operands will be on a single string of characters and will be separated by spaces. The shell can identify leading/trailing/multiple-intermittent spaces and will remove them while deriving the command and the operands.

(iv) What internal commands will the shell supprot ? We will not define this at this moment. We will add internal commands as they are needed.

Ans. (ii) DMA : When large volumes of data are to be moved, a more efficient technique is required which called direct memory access (DMA). The DMA function can be performed by a separate module on the system bus, or it can be incorporated into an I/O module. In either case, the technique works as follows :

When the processor wishes to read or write a block of data, it issues a command to the DMA module by sending the following information :

– Whether a read or write is requested.
– The address of the I/O devices.
– Starting location in memory to read from or write to
– The number of words to be read or written.

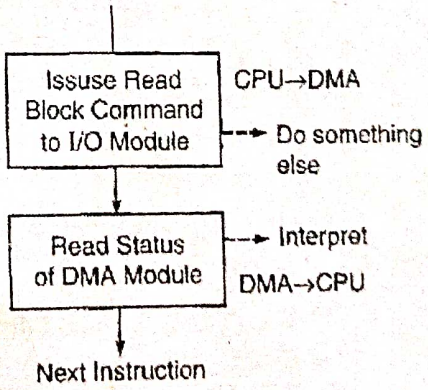


Fig. : DMA

The processor then continues with other work. It has delegated this I/O operation to the DMA module, and that module will take care of it. The DMA module transfers the entire block of data, one word at a time, directly to or from memory, without going through the processor. When the transfer is complete, the DMA module sends an interrupt signal to the processor. Thus, the processor is involved only at the beginning and at the end of transfer.

PRINCIPLES OF OPERATING SYSTEM

Dec - 2014

Paper Code:-CSE-301-F

Note: Attempt five questions in total selecting one question from each of the four Units. Question No.1 is compulsory.

Section-A

Q.1.(a) Differentaite between multitasking and multiprocessing operating systems. (2)

Ans. Multitasking : Multitasking is the ability of an operating system to execute more than one task simultaneously on a single processor machine. Though we say so but in reality no two tasks on a single processor machine can be executed at the same time. Actually CPU switches from one task to the next task so quickly that appears as if all the tasks are executing at the same time. More than one task/program/job/process can reside into the same CPU at one point of time.

Multiprocessing : Multiprocessing is the ability of an operating system to execute more than one process simultaneously on a multi processor machine. In this, a computer uses more than one CPU at a time.

Q.1.(b) Differentiate between paging and segmentation. (2)

Ans. Paging and segmentation are closed relative to each other but they have following difference.

(1) Paging divides the virtual memory in physical memory areas while the segmentation divides it logically.

(2) Paging divides the memory in fixed length memory areas while segmentation divides into variable length memory areas.

(3) In segmentation a full logical portion of the process in loaded, but in paging information about page is loaded.

(4) The segments are large length as compare to page.

(5) In segmentation memory is divided into segments through software and in paging the memory in divided through hardware.

Q.1.(c) Differentiate between a process and a program.

Ans. The differences between a program and a process are -

Program	Process
1. A program consists of a set of instructions in a programming language.	1. A process is a sequence of instruction exection.
2. A program is a static object that exists in a file	2. A process is a dynamic object i.e. a program in execuation.
3. A program is loaded into the secondary storage devices.	3. A process is loaded into the main memory.
4. The time span of a program in unlimited.	4. The time span of a process is limited.
5. A program is a passive entity.	5. A process is an active entity.

Q.1.(d) Why are page size always powers of 2 ? Explain. (2)

Ans. All addresses are binary and are divided into page frame number and offset. By making the page size a power of 2, the page number the page frame number and the offset can be determined by looking at particular bits in the address, no mathematical calculations are required. The physical address can be generated by concatenating the offset to the frame number.

Q.1.(e) Is it possible to have a deadlock involving only one process? Explain your answer. (2)

Ans. No, deadlock is not possible in only single process. This follows directly from the hold and wait condition.

Q.1.(f) Differentiate between short term and long term schedulers.

Ans. Long-Term Scheduling : Long-term scheduling performs a gatekeeping function. It decides whether there's enough memory, or room, to allow new programs or jobs into the system. It limits the degree of multi-tasking to prevent slow performance on currently-running programs. When a job gets past the long-term scheduler, it's sent on to the medium-term scheduler.

Short-Term Scheduling : The short-term scheduler takes jobs from the "ready" line and gives them the green light to run. It decides which of them can have resources and for how long. The short-term scheduler runs the highest-priority jobs first and must make on-the-spot decisions. For example, when a running process is interrupted and may be changed, the short-term scheduler must recalibrate and give the highest-priority job the green light.

Q.1.(g) Consider a system consisting of six drives, with *n* process competing for them. Each process may need two drives. For which values of '*n*' is deadlock free. (2)

Ans. The maximum value of n for which the system is guaranteed to be deadlock free is 2.Two processes can never lead to deadlock as the peak time demand of (3+3 =6) tape drives can be satisfied.

But three processes can lead to deadlock if each process hold 2 drives and then demand one more.

Q.1.(h) Differentiate between virtual memory and main memory.

Ans. Main memory, also called RAM, is the physical memory unit in the computer. Virtual memory also serves as computer memory, but is actually hard drive space acting as temporary storage for computer processes.

Main memory speeds up a computer, while virtual memory tends to transfer data at speeds much slower than physical RAM.

Q.1.(i) What is the cause of thrashing? How does system detect thrashing? (2)

Ans. Thrashing is caused by under allocation of the minimum number of pages required by a process, forcing it to continuously page fault.

The system can detect thrashing by evaluating the level of CPU utilization as compared to the level of multiprogramming.

Q.1.(j) Differentiate between user threads and kernel threads.

Ans. Difference between User Level and Kernel Level Thread (2)

Sr.No.	User level threads	Kernel level threads
1.	User level threads are faster to create and manage.	Kernel level threads slower to create and and manage.
2.	Implemented by a thread library at the user level.	Operating system support directly to kernel threads.
3.	User level thread can run on any operating system.	Kernel level threads are specific to the operating system.
4.	Support provided at the user level called user level thread.	Support may be provided by Kernel is called kernal level thread.
5.	Multithread application cannot take advantage of multiprocessing.	Kernel routines themselves can be multi-threaded.

Q.2. Consider performance of FCFS algorithm for three compute-bound processes. What if have 3 processes P1 (takes 15 seconds), P2 (takes 4 seconds) and P3 (takes 6 seconds). If arrive in order P1, P2, P3, what is

(i) Waiting Time ?

(ii) Turnaround Time ?

What about if process come in order P2, P1, P3 ?

What is

(i) Waiting Time ?

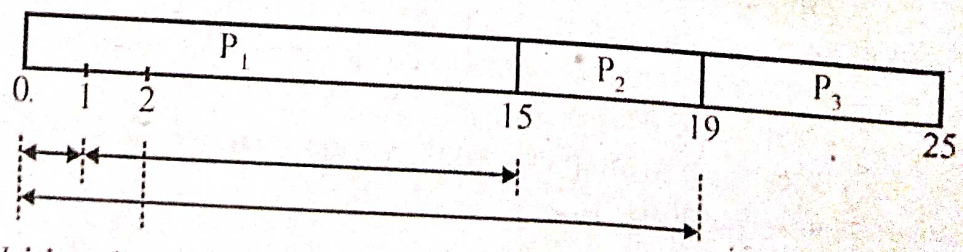
(ii) Turnaround Time ?

Ans. If processes arrive in order P_1, P_2, P_3 let us assume at 0 ms, 1ms & 2ms respec ely. (20)

Process	Brust time	Arrival time
P_1	15	0
P_2	4	1
P_3	6	2

Since, FCFS is a non-preempative algorithm, so when a process execuation is started it is not preempted until its excecution is completed.

The Gantt chart will be



(i) Waiting time for process $P_1 = 0$ ms

Waiting time for process $P_2 = 15 - 1 = 14$ ms

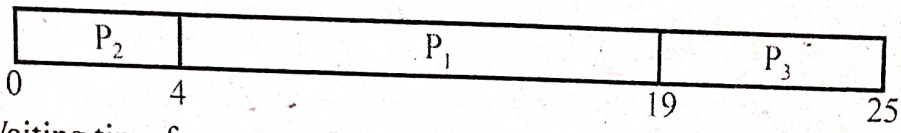
Waiting time for process $P_3 = 19 - 2 = 17$ ms

$\therefore$ Average waiting time $= \frac{0+14+17}{3}$ ms $= \frac{31}{3} = 10.33$ ms **Ans.**

(ii) Turnaround time for $P_1 = 15 - 0 = 15$
Turnaround time for $P_2 = 19 - 1 = 18$
Turnaround time for $P_3 = 25 - 2 = 23$

Average turnaround time $= \frac{15+18+23}{2} = 18.67$ ms **Ans.**

If the process P_1, P_2, P_3 arrives at the same time say at 0 ms in the system and in the order of P_2, P_1, P_3 then the Gantt chart will be



(i) Waiting time for process $P_2 = 0$ ms
Waiting time for process $P_1 = 4$ ms
Waiting time for process $P_3 = 19$ ms

Average waiting time $= \frac{0+4+9}{3} = \frac{23}{3} = 7.67$ ms **Ans.**

(ii) Turnaround time for $P_2 = 4$ ms
Turnaround time for $P_1 = 19$ ms
Turnaround time for $P_3 = 25$ ms

Average turnaround time $= \frac{4+19+25}{3} = 16$ ms **Ans.**

Q.3.(a) What is a process? Describe its various states adn explain various operations on processes. **(10)**

Ans. Process : A process is a program in execuation. The execuation of a process must progress in a sequential fashion.

A process is the smallest unit of work individually schedulable by an operating system. Such a system consists of a collection of processes : operating system processes execute system code, and user processes execute user code. All these processes may execute concurrently. Potential benefits of concurrent execution of programs include increased performance, resource utilization, and responsiveness of a computer system.

Processes are a primary mechanism, for defining and managing concurrent execution of programs under control of an operating system.

The operating system is responsible for the following activites in connection with process and thread management : the *creation* and *deletion* of both user and system processes; and the provision of mechanisms for *communication*, *synchronization* and *deadlock* handling for processes.

Process States : When a process comes in execuation it changes its state.The state of a process is defined in part by the current activity of that process. Each process may be in one of the following state during the time of its execution.

- *New* : In this state, the process is being created
- *Ready* : In the state, the processes is waiting to be assigned to a processor through a short term scheduler.
- *Waiting* : In this state the process is waiting OR blocked for some event to occur such as an input/output completion on reception of a signal.

- *Running* : In this state, instructions are being executed.
- *Terminated* : In the state, the process has finished the execution.

These state names are arbitrary, and they vary across operating systems. However, the states that they represent are found on all systems.

In a uniprocessor system only one process can be in a running state at a single instant of time, although many process may be in ready OR waiting state.

The state transition diagram corresponding to these states is presented is fig.(1).

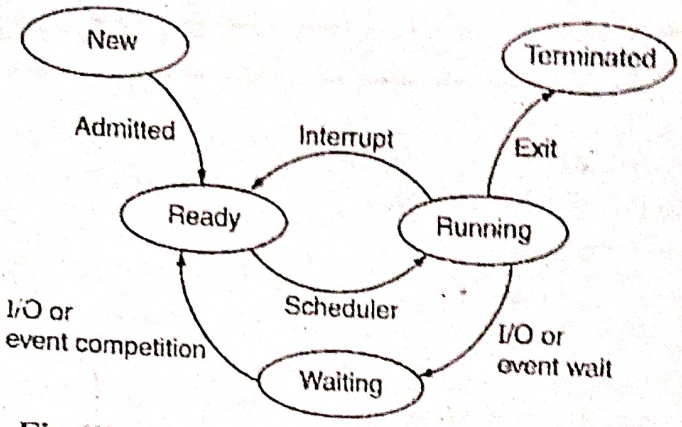


Fig.(1) : Process State Transition Diagram.

Operation On a Process : As we know that the process is sequential. The scenario is that there are many processs and a single CPU. So, we use time slicing or multitasking.This is a type of pseudo-parallel processing and is also known as concurrent sequential processing. Please note that in a true parallel system, many CPUs are there, Now, the various operations that can be done on processes are as follows :

(a) Create a process
(b) Destory a process
(c) Run a process
(d) Change a process priority
(e) Get process information
(f) Set process information

Let us discuss now about two main operations

(a) Process creation.
(b) Process termination.

(a) Process creation : It is the job of our O.S to create process. There are four ways of achieving it. They are as follows-

(i) For a *batch environment*, a process is created in response to the submission of a job.

(ii) In an *interactive environment*, a process is created when a new user attempts to log on.

(iii) The *operating system* can create a process to perform a function on behalf of a user program.

(iv) A number of processes can be generated from the main process. For the purpose of *modularity* or to exploit *parallelism*, a user program can crete a number of processes.

(b) Process termination : There are two types of terminations They are as follows-

(i) Normal Termination

(ii) Forced Termination

(i)Normal Termination : Normal Termination of processes is done by using an exist system call. It returns some output data to its parent process. All the resources allocated to the process are freed by it at this time.

(ii) Forced Termination : Forced Termination of a process is done by its parent process when it exccutes an abort system call. When a parent process terminates the child process is also terminated automatially. This is due several like-

(a) The child process has exceeded allocated resources.

(b) Task assigned to child is no longer required.

(c) Parent is terminating and O.S does not allow a child a to continue. This is known as cascading termination. For example, UNIX supports caseding termination. There are many other reasons for process termination also like expiration of time slot, memory boundary violation. I/O failure etc.

Q.3.(b) Compare batch and multiprogramming, network and distributed operating system. (10)

Ans. Difference Between Batch OS and Multiprogramming OS :

Batch Processing Operating System : Batch processing system is also known as off line or sequential or stacked processing system. It is one of the oldest operating system. In this operating system jobs with similar need batched together by operator and run as a group on a computer system. Users cannot directly interact with the system, just prepared job and submit it to operator. When output produced it a submitted to appropriate user. The major job is batch operating system to transfer control automatically from one job to another. It is appropriate for executing large jobs that need little interactions.

Advantages :

- User /Customer need to wait in queue
- Huge Amount of a data can be processed efficiently.
- It reduces idle time of a computer becaues operator intervention is not required in automatic job to job transiton.

Disadvantages :

- It reduces timeless in some cases.
- Priority scheduling is difficult to achieve.
- Turnaround time increases in some cases.

Multi-programming Operating System : In batch operating system job batches are stored in memory for processing. The operating system picks one job and executes it. Eventually jobs have to wait for some input/output operation. That CPU sits idle for that time hence system performance degrades. Multiprogramming increase CPU utilization by organizing jobs such that the CPU always has one job to execute. It is possible by allowing switching between jobs. In multiprogramming operating system more than one job resides in main memory and hence the memory is utilized. The CPU picks one job and starts executing it, when the job requires to perform input/output operation, the CPU does not sit idle and picks the next job and start executing it's instruction, in this way is single CPU is in demand all the time and hence it is fully utillized.

Advantages :

- Increased throughput
- Shorter response time
- Ability to assign priorities to jobs.

Disadvantages :

- Decreased operating system overhead

Difference Between Distributed OS and Network OS :

Distributed operating System : Distributed systems use multiple central processors to serve multiple real time application and multiple users. Data processing jobs are distributed among the processors accordingly to which one can perform each job most efficiently.

The processors communicate with one another through various communication lines (such as high-speed buses or telephone lines). These are referred as loosely coupled systems or distributed systems. Processors in a distributed system may vary in size and function. These processors are referred as sites, nodes, and computers and so on.

The advantages of distributed systems are following.

(1) With resource sharing facility user at one site may be able to use the resources available at another.

(2) Speedup the exchange of data with one another via electronic mail.

(3) If one site fails in a distributed system, the remaining sites can potentially continue operating.

(4) Better service to the customers.

(5) Reduction of the load on the host computer.

(6) Reduction of delays in data processing.

Network operating System : Network Operating System runs on a server and and provides server the capability to manage data, users, groups, security, applications, and other networking functions. The primary purpose of the network operating system is to allow shared file and printer access among multiple computers in a network, typically a local area network (LAN), a private network or to other networks. Examples of network operating systems are Microsoft Windows Server 2003, Microsoft Windows Server 2008, UNIX, Linux, Mac OS X, Novell NetWare, and BSD.

The advantages of network operating systems are following.

(1) Centralized servers are highly stable.

(2) Security is server managed.

(3) Upgrades to new technologies and hardware can be easily integrated into the system.

(4) Remote access to servers is possible from different locations and types of systems.

The disadvantages of network operating systems are following.

(1) High cost of buying and running a server.

(2) Dependency on a central location for most operations.

(3) Regular maintenance and updates are required.

Section -B

Q.4.(a) Describe various page replacement algorithms. **(10)**

Ans. Page Replacement Algorithms : When the processor needs to execute a page, and if that page is not available in main memory then this sitution is called "page fault". For bringing in the required page into main memory, if the space is not available in memory then we need to remove the page from the main memory for allocating the space to the new page which needs to be executed. In this process the page which we need to remove from the main memory is called a "victim page". For the selection of victim page in main memory different algorithms are available called page replacement algorithms. Some of the widely using page replacement algorithms are as follows :

(i) First-in, First out (FIFO) algorithm
(ii) Optimal page replacement algorithm
(iii) Least recently used (LRU) algorithm
(iv) Least frequenctly used (LFU) algorithm
(v) Most frequently used (MFU) algorithm

(i) First-in, First out (FIFO) algorithm : First-in, first out policy treates the page frams allocated to a process as a circular buffer and pages are removed in round robin fashion, that is first-in, first-out (first inputed page will remove first) from. It is a very simple algorithm.

Here to replace the page that has been in memory longest. It is shown in Table.(1). Regions of programs or data that are heavily used throughout the life of a program. In this technique number of page faults are more when compared with other algorithm, In table.(1) 'yes' indicates that page fault has occurred and replaced page is shown in box in that column with respect to the concerned frame number.

Table.(1) : FIFO System

Address / Frame	3	1	2	3	4	2	3	0	3	1	3
0	$\boxed{3}$	3	3	3	$\boxed{4}$	4	4	4	4	$\boxed{1}$	1
1		$\boxed{1}$	1	1	1	1	$\boxed{3}$	3	3	3	3
2			$\boxed{2}$	2	2	2	2	$\boxed{0}$	0	0	0
Page fault	Yes	Yes	Yes	No	Yes	No	Yes	Yes	No	Yes	No

(ii) Optimal page replacement algorithm : Optimal policy selects the page for replacement when the page fault is to occur for which the time to the next reference is the longest. It is shown in Table.(2).

Advantages : (a) The number of page faults are few, when compared with the other algorithms

(b) It reduces the total page replacement timings.

Disadvantages : Opitmal policy is impossible to implacement, because it requires the operating system should have perfect knowledge of future events.

Table.(2) : Optimal policy

Address / Frame	3	1	2	3	4	2	3	0	3	1	3
0	3	3	3	3	3	3	3	3	3	3	3
1		1	1	1	4	4	4	0	0	1	1
2			2	2	2	2	2	2	2	2	2
Page fault	Yes	Yes	Yes	No	Yes	No	No	Yes	No	Yes	No

(iii) Least recently used (LRU) algorithm : According to the least recently used policy, it replaces the page in memory that has not been referenced for the longest time. It is shown in Table.(3) for the address stream of pages 3 1 2 3 4 2 3 0 3 1 3

Advantages : It is beneficial for the pages which are not refer after certain period, i.e., after removal from main memory.

Table.(3) : LUR policy

Address / Frame	3	1	2	3	4	2	3	0	3	1	3
0	3	3	3	3	3	3	3	3	3	3	3
1		1	1	1	4	4	4	0	0	0	0
2			2	2	2	2	2	2	2	1	1
Page fault	Yes	Yes	Yes	No	Yes	No	No	Yes	No	Yes	No

Disadvantages : (a) It is complicated to implement, becuase it needs to maintain the time of its last reference for all the pages which are there in main memory.

(b) It needs to maintain a stack of references.

(iv) Least frequenctly used (LFU) algorithm : The least frequently used algorithm selects a page for replacement, with least frequency of reference upto the situtation of page fault occurrence. For this maintain the counter with each page frame. Increase the counter when the page is to refer. Whenever the page fault is to occur, select the page for replacement which is having the least count. Table(4) is showing the LEU policy with address stream 3 1 2 3 4 2 3 0 3 1 3

Table.(4) : LRU policy

Address / Frame	3	1	2	3	4	2	3	0	3	1	3
0	3	3	3	3	3	3	3	3	3	3	3
1		1	1	1	4	4	4	0	0	1	1
2			2	2	2	2	2	2	2	2	2
Page fault	Yes	Yes	Yes	No	Yes	No	No	Yes	No	Yes	No

(v) Most frequently used (MFU) algorithm : Most frequently used and least frequently used algorithms are the counting algorithms. The most frequently used algorithm selects a page for replacement, with most frequency of reference. Like least frequently used for this also maintain the counter with each page frame. Increase the counter when the page is to refer. Whenever the page fault is to occur, replace the page with highest count and reset the count. It is shown in Table.(5).

Table.(5) : MFU policy

Address / Frame	3	1	2	3	4	2	3	0	3	1	3
0	3	3	3	3	4	4	4	0	0	0	0
1		1	1	1	1	1	1	1	1	1	1
2			2	2	2	2	3	3	3	3	3
Page fault	Yes	Yes	Yes	No	Yes	No	Yes	Yes	No	No	No

Q.4.(b) Describe First fit, Best fit and Worst fit mechanism for memory allocation. (10)

Ans. Best fit : Best fit chooses the block that is closest in size to the request. It allocates the smallest hole (block) that is big enough. The memory manager searches the list of free holes and places a process in the smallest block of unallocated memory in which it will fit. For example, suppose a process requests 10 kB of memory and the memory manger currently has a list of unalocated blocks of 7 KB, 13KB, 17KB, 11KB, and 15KB blocks. The best-fit strategy will allocate the 11 KB block to the process. An incoming job is placed in the hole in main storage in which it fits most tightly and leaves the smallest amount of unused space. The best-fit technique uses the smallest free area with size $\geq n$. It avoids needless splitting of large areas, however it tends to generate very small free areas due to the splits. Hence in the long run it, too, may suffer from the problem of numerous small free areas. Hence memory fragmentation exits in both schemes. It is slower than first fit as it searches the entire list every time.

Worst fit : Allocate the largest hole. It searches the entire list. The memory manger places a process in the largest block of unallocated memory available. The idea is that this placement will create the largest hole after the allocations, thus increasing the possibility that, compared to best fit, another process can use the hole created as a result of external fragmentation. If the list of unallocated blocks consists of 7 KB, 13KB, 11KB, and 15KB blocks and a process requests a 10 KB block, then worst fit will split the 17KB block into a 10 KB block to allocate to the process, leaving a 7 KB block for future use. Worst fit says to place a program in main storage in the hole in which worst fits, i.e., the largest possible hole. The intuitive appeal is simple: after placing the program in this large hole, the remaining hole often is also large and is thus able to hold a relatively large new program.

First fit : Allocate first hole that is big enough. First fit begins to scan memory from the beginning and chooses the first available block that is large enough. It scans the list until it finds a hole big enough to accommodate the requesting process. It is fast as it searches as little as possible. If there are many holes in the memory, the scheduler, in order to reduce the amount to

time it spends analyzing the free list, begins traversing the list from the beginning and allocates memory from the first hole it encounters large enough to satisfy the request. If the list of unallocated blocks consists of 7 KB, 13 KB, 17KB, 11KB and 15KB blocks and a process requests a 10 KB block, first fit will allocate the 13KB block to the process. An incoming job is placed in the main storage in the first available hole large enough to hold it. First fit also has intuitive appeal in that it allows the placement decision to be made quickly. To service a request for n words of memory, the first fit technique uses the first free area it can find whose size is $\geq n$ words, n words are allocated to the request, and the remaining part of the area is put into free list. This technique suffers from the problem that memory areas become sucsessively smaller as a system operates; hence it becomes difficult to handle for large memory areas.

Q.5.(a) Explain the concept of virtual memory. **(10)**

Ans. Definition of Virtual Memory : Virtual memory is a memory hierarchy consisting of a computer system's memory and a disk, that enables a process to operate with only some portions of its address space in memory.

Advantages of Using Virtual Memory : Users Point of View : The details of virtual memory management are generallly transparent to programmers. Being provided with the illustion of a much larger memory than may actually be available, programmers are practically relieved of the burden of trying to fit a program into limited memory. Moreover, the same program may run witho t reprogramming or recompilation on systems with significantly different capacities of installed memory. Thus, programmers may develop applications that exploit the full benefit of a given computer platform regardless the differenes in installed physical memory among different user configurations.

Virtual memory makes the programming task easier as now the programmer should not worry about the size of the available physical memory fig.(1).

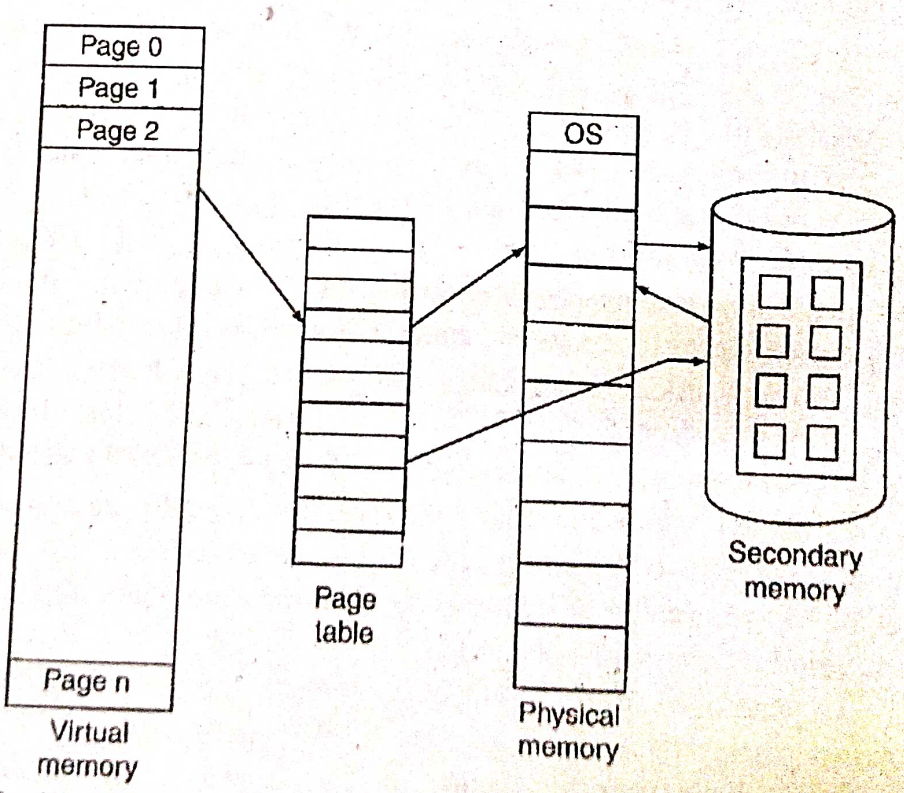


Fig.(1) : Size of virtual memory larger than physical memory.

Advantage of using Virtual Memory : System point of view : The ability to execute a partially loaded process is also advantageous from the operating systems point of views. For example, a process may be loaded into an arbitrary sized space of memory. This may be used to reduce to reduce the external fragmentation without the need to change the scheduled order of process executions. Moreover, the amount of space in use by a given process may be varied during its memory residence. As a result, the operating system may speed up the execution of important processes by allocating them more real memory. Alternatively, by reducing the real-memory holdings of resident processes, the degree of multiprogramming can be increased by using vacated space to activate more processes.

Thus, runnings a program that is not entirely in memory would benefit both the system and the user.

Q.5.(b) Explain demand paging. **(10)**

Ans. Demand Paging : In Demand paging, a page is brought into the memory for its execution only when it is demanded,l otherwise it is remained in backing storage (disk). The name seems to have been derived form demand feeding – a policy used for feeding the baby by mothers in which the food is given to the baby only when he cries for it.

This method is the combination of *paging* and *swapping* method. The main requirement of this method is that the complete program should be present in the backing storage (disk) in the form of pages. Swapping is used to swap the contents of the program from disk to main memory when it is required.

In this method a "lazy swapper" is used, which swaps the pages when it is needed. Swapping requires removing the unused contents from main memory to disk and loading the required contents of program from disk to main memory [fig.(1)].

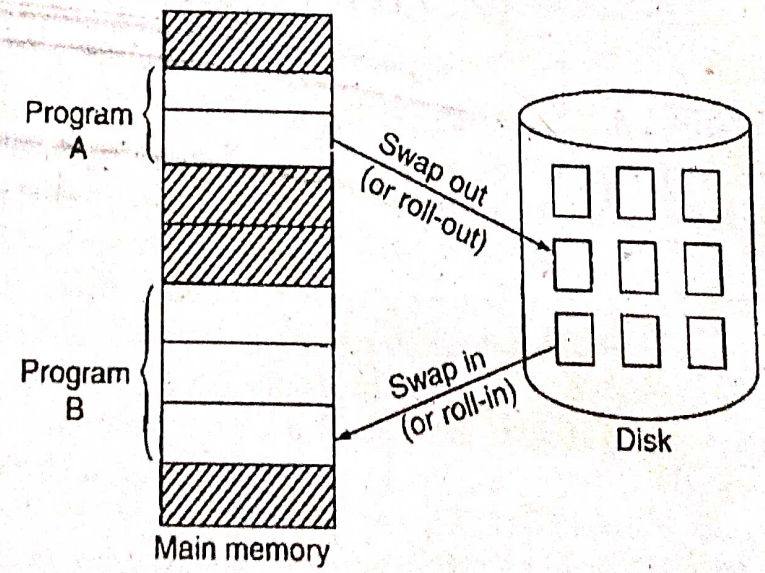


Fig.(1) : Swapping with Paging.

Advantage :

(i) Reduced memory requirement

(ii) Swap time is also reduced

(iii) Increase degree of multiprogramming

Disadcvantages : Using this method swaping is done using hit-and-trial method. For example, if a program is having 5 pages : 0, 1, 2, 3, 4 and out of these only pages 0, 3, 4 are loaded into memory at the time of execution state. If the program tries to access the address of page number 2, then there will be an error; as this page is not present in the main memory since it was not loaded at the time of swapping from disk to main memory. This error is called as page fault error.

The error is noticed by paging hardware while translating the address using the page table. The entire program is present in back storage in the form of pages fig.(2).

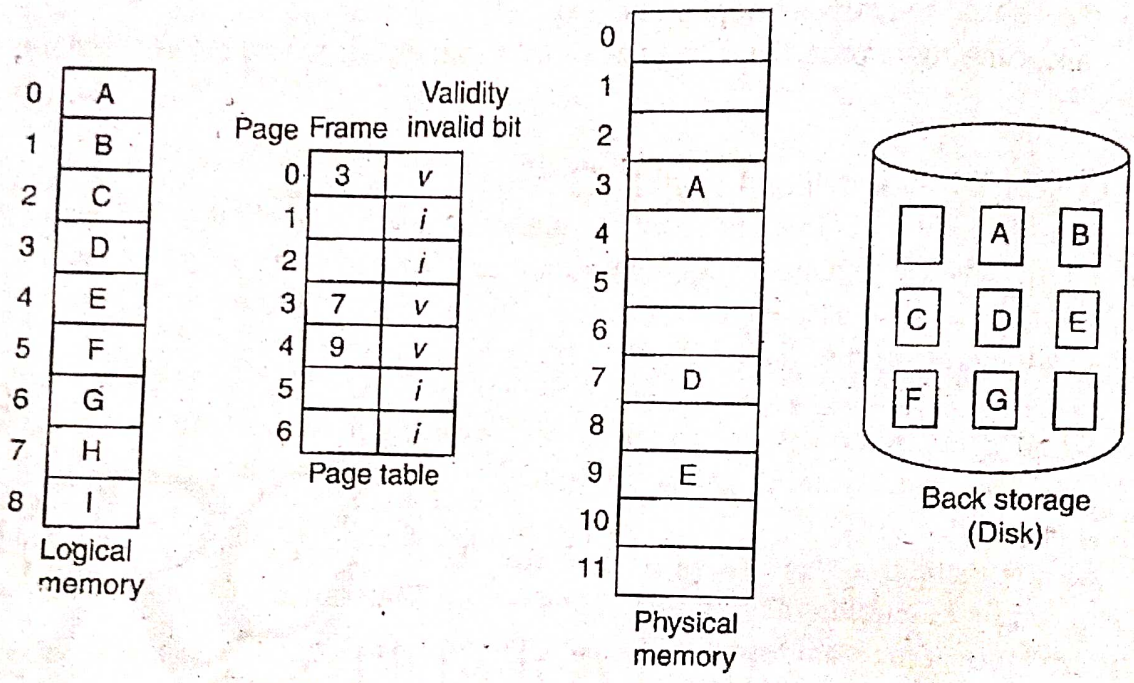


Fig.(2) : Example of demand paging.

Section -C

Q.6. What is a deadlock ? What are the necessary conditions for a deadlock ? Explain the mechanism for deadlock recovery. (20)

Ans. Deadlock : A set of processes is in a *deadlock* state if each process in the set is waiting for an event that can be caused by only another process in the set. In other words, each member or the set of deadlock processes is waiting for a resources that can be released only by a deadlock process. None of the processes can run, none of them can release any resources, and none of them can be awakened. It is important to note that the number of processes and the number and kind of resources possessed and requested are unimportant.

Necessary conditions for a deadlock : There are four conditions that must hold simultaneously for a deadlock occurance. They are :

1. *Mutual Exclusion Condition* : The resources involved are non-shareable. At least one resource (thread) must be held in a non-shareable mode, that is, only one process at a time claims exclusive control of the resource. If another process requests that resource, the requesting process must be delayed until the resources has been released.

2. *Hold and Wait Condition* : Requesting process hold already, resources while waiting for requested resources. There must exist a process that is holding a resource already allocated to it while waiting for additional resource that are currently being held by other processes.

3. *Non-Preemptive Condition* : Resources already allocated to a process can not be preempted. Resources can not be removed from the processes are used to completion or released voluntarily by the process holding it.

4. *Circular Wait Condition* : The processes in the system form a circular list or chain where each process in the list is waiting for a resource held by the next process in the list.

Recovery from Deadlock : Once a deadlock in the system is detectedd, the system must take attempt to recover from the deadlock. The deadlock should be broken by removing one or more of the necessary conditions. Typically, many processes will lose some or all the work they have accomplished. The under mentioned four approaches can be adopted to overcome a deadlock. One approach would be to simply abort/terminate one or more processes in order to remove the circular wait. The second approach is to preempt some resources from one or more of the deadlock processes. The third approach is to back up each deadlocked process to some previously defined point, and restart all the processes. This is also called Rollback. The problem with this approach is that the original deadlock may reoccur. Fourth approach is, sucessively abort deadlocked processes untile deadlock no longer exists. The order in which processes for termination should be on the basis of some criterion, with minimum cost. Some factors which can be considered are as follows :

- The priority of the processes.
- How long the process has computed, and how much longer the process will compute before completing its designated task.
- How many and what type of resources the process has used (for example, are the resources simple to preempt?)
- How many more resources the process needs in order to complete.

Ir-impelmentation of any of the four methods of deadlock recovery, after termination each time the deadlock detection algorithm will have to be invoked to check whether a deadlock exists or not.

Q.7. Write notes on :

(a) Various directory structures. **(10)**

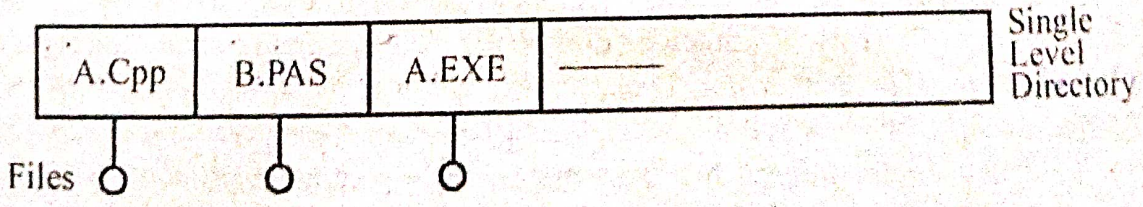
(b) Disk Scheduling. **(10)**

Ans.(a) Various directory structures are as follows :

(i) Single Level Directory : This is simplest directory structure. All files are contained in the same directory structure.

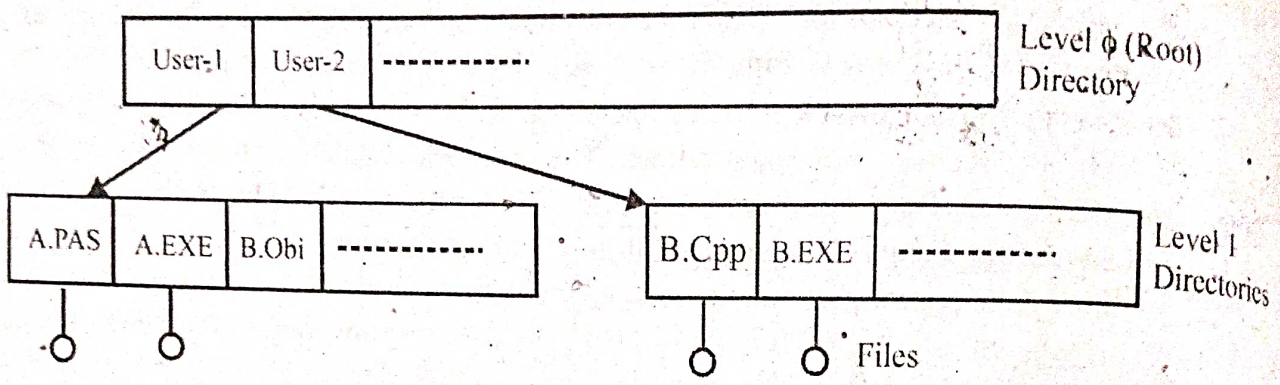
Distadvantages : (a) It becomes highly unwiedly when system has more than one users or the number of files is large.

(b) Two files by different users, with same name, cannot be accommondated ; one of the files will need to the renamed.

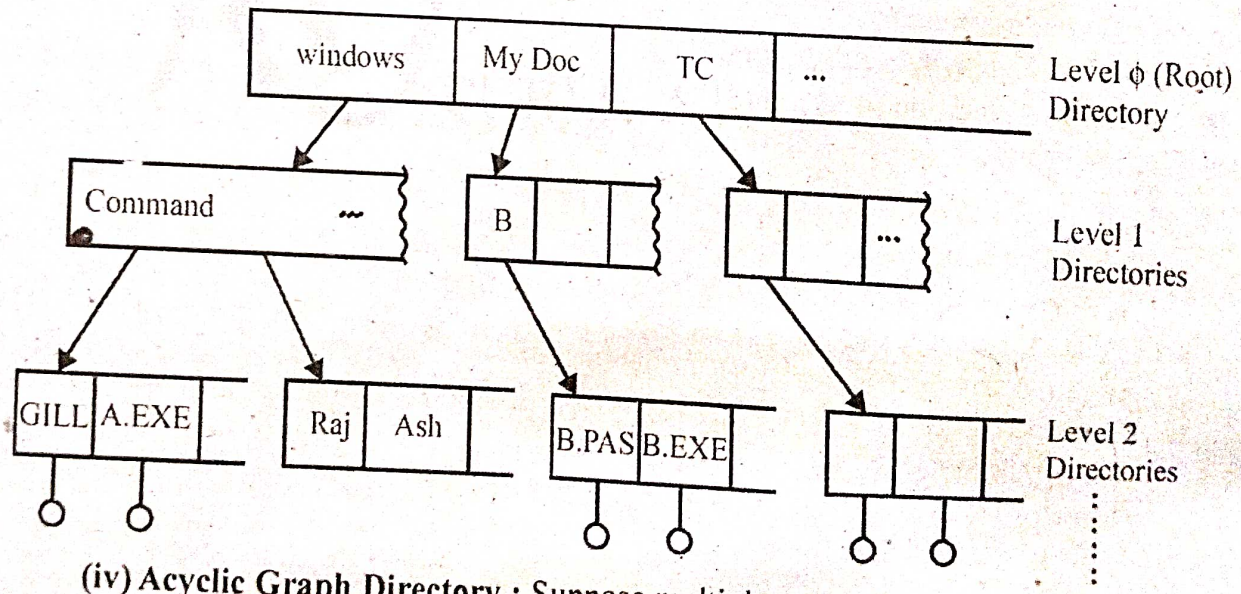


(ii) Two Level Directory : This solution can be used to create multiple User file Directories (UFDs), one for each user, under a Maste file Directory (MFD). The MFD is indexed

by the user name or user account number, to provide an access to the relevant UFD. When a user references any of its files, a search is carried out only in the User file Directory (UFD) of that particular user.



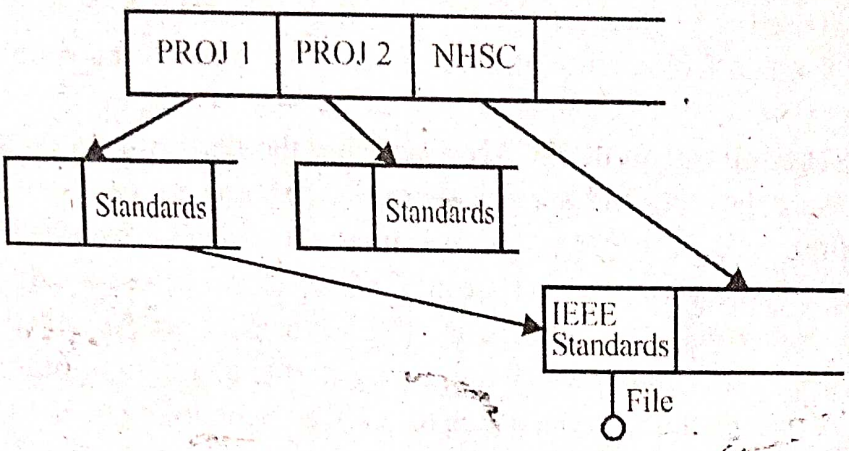
(iii) Tree Structured Directory : The two-level directory is in fact a two-level tree. The tree structured directory is a generalization of the two level. Directory and it forms a tree of an arbitrary height. This permits the Users to create their own sub-directions under the respective UFDs. A directory may contain a set of files and a set of sub-directories. Each file in the system has a unique access path name. A path name specifies the path from the root directory to the file. So, path name of a file is its unique identification. When a user logs in, he is provided access to his UFD. User may use absolute path name or relative path name (relative to the current directory being accessed by the user) to access his sub-directions. New sub-directions can be created and existing sub-directories can be deleted. Some systems insist that a sub-directory can be deleted only if it is empty.



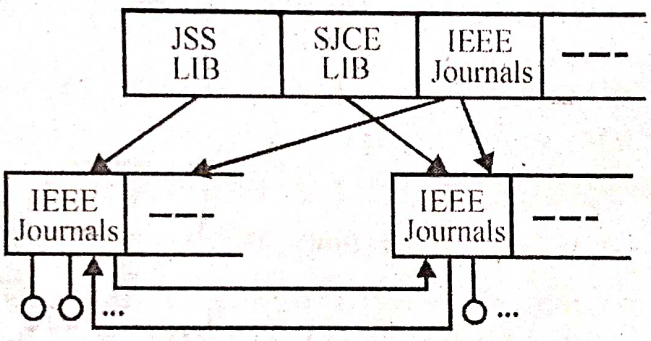
(iv) Acyclic Graph Directory : Suppose multiple users are working on a project, the project files can be stored in a common sub-directory of the multiple users. This type of directory is called acylic graph directory. The common directory will be declared a shared directory. Same way, system may provide shared files. However, the graph constains no cycles. With shared files, changes made by one user are made visible to other users. One advantage of acyclic graph directory is the ease to determine when there are no more references to a file/directory. The issues, which need to be resolved now are

(a) A file may now have multiple absoulte paths. So, distirct file names may refer to same file.

(b) Whenever, a shared directory/file is deleted, all pointers to the directory/file are also to be removed.



(v) General Graph Directory : General graph directory permits cycles. One major disadvantage is that is poorly designed search algorithm may get into infinite loop while searching for a file.



Ans. (b) Disk scheduling : As we know that on a single Computer we can Perform Many Operations at a Time so that Management is also necessary on all the Running Processes those are running on the System at a Time. With the help or Advent of the Multi-programming we can Execute Many Programs at a Time. So for Controlling and providing the Memory to all the Processes Operating System uses the Concept of Disk Scheduling.

In this the Time of CPU is divided into the various Processes and also determines that all the processes will Work Properly. So that Disk Scheduling Will Specifies that at which time which Process will be executed by the CPU. So that the Scheduling means to Execute all the Processes those are given to a CPU at a Time. The Scheduling is used for Divide the Total Time of the CPU between the Number or Processes So that the Processes can execute Concurrently at a Single Time. For Sharing the Time or For Dividing the Total Time of the CPU, the CPU uses the following the Scheduling Techniques.

(1) FCFS or First Come First Serve: In this Jobs or Processes are Executed in the Manner in which they are entered into the Computer. In this Operating System Creates a Queue which contains the Sequence Order in which they are to be Executed and the Sequence in which the CPU will Execute the Process.. In this all the Jobs are performed according to their Sequence Order as they have entered. In this the Job which had Requested first will firstly performed by the CPU. And the Jobs those are entered Later will be Executed in to their Entering Order.

(2) SSTF or Shortest Seek Time First : In this Technique The Operating System will Search for the Shortest time means this will search which job will takes a Less Time of CPU for

Running. And After Examining all the jobs, all the Jobs are arranged in the Sequence wise or they are Organized into the Priority Order. The Priority of the Process will be the Total Time which a Process will use For Execution. The Shortest Seek Time Will Include all the Time means Time to Enter and Time to Completion of the Process. Means the Total Time which a Process Will Take For Execution.

(3) Scan Scheduling : In the SCAN algorithm, the disk arm starts at one end of the disk, and moves toward the other end, servicing requests as it reaches each cylinder, until it gets to the other end of the disk. At the other end, the direction of head movement is reversed, and servicing continues. The head continuously scans back and forth across the disk.

(4) C-Scan Scheduling: In the C-Scan all the Processes are Arranged by using Some Circular List. Circular List is that in which there is no start and end point of the list means the End of the List is the Starting Point of the list. In the C-Scan Scheduling the CPU will search for the Process from Start to end and if an End has Found then this again start from the Starting Process. Because Many Times When a CPU is executing the processes then may a user wants to enter some data means a user wants to enter Some data So that at that Situation the CPU will Again Execute that Process after the Input Operation. So that C-Scan Scheduling is used for Processing Same Processes again and Again.

(5) Look Scheduling : In the Look Scheduling the CPU Scans the List from Starting to End of the Disk in which the various Processes are Running and in the Look Scheduling the CPU will Scan the Entire Disk from one End to the Second end.

Section - D

Q.8.(a) What are the difference between user level threads and kernel level threads ? Under what circumstances is one type better than other ? (10)

Ans. User level Thread : In a user level thread, all the work of thread management is done by the application and the kernal is not aware of the existence of threads. The thread library contains code for creating and destroying threads, for passing message and data between threads, for scheduling thread execution and for saving and restoring thread contexts. The application begins with a single thread and begins runnings in that thread. Fig.(1). shows the user level thread. User level thread are generally fast to create and manage.

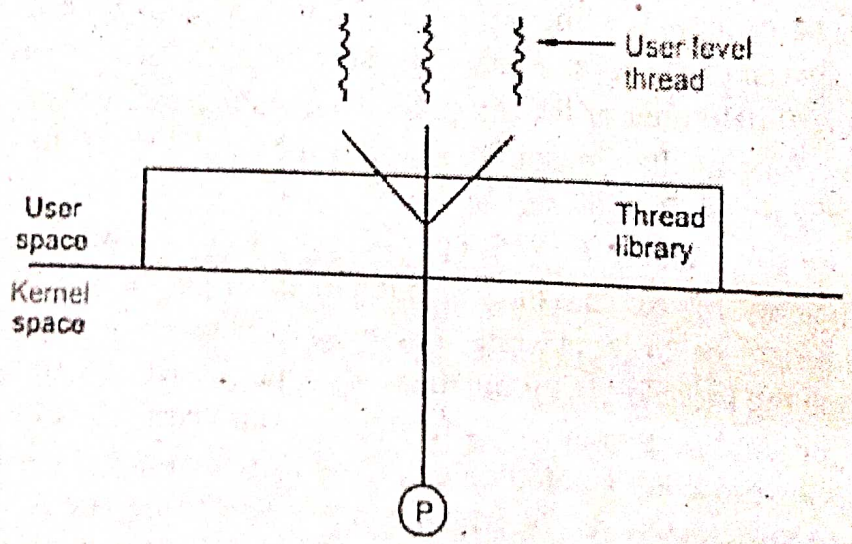


Fig.(1) : User level thread.

Advantage of user level thread over kernal level thread :

(i) Thread switching does not require kernal mode privileges

(ii) User level thread can run on any operating system.

(iii) Scheduling can be application specific

(iv) User level threads are fast to create and manage.

Disadvantages of user level thread :

(i) In a typical operating system, most system calls are blocking.

(ii) Multithreaded application cannot take advantage of multiprocessing.

Kernal Level Threads : In kernal level thread, thread managment is done by the kernal. There is no thread management code in the application area. Kernal threads are supported directly by the operating system. Any application can be programmed to be multithreaded. All of the threads within an application are supported within a single process. The kernal maintains context information for the process as a whole and for individulas threads within the process.

Scheduling by the kernal is done on a thread basis. The kernal performs thread creation scheduling and managment in kernal space. Kernal threads are generally slower to crete and manage than the user threads.

Advantages of kernal level thread :

(i) Kernal can simultaneously schedule multiple threads from the same process on multiple process.

(ii) If one thread in a process is blocked, the kernal can schedule another thread of the same process.

(iii) Kernal routines themselves can be multithreaded.

Disadvantages : (i) Kernal threads are generally slower to create and manage than the user threads.

(ii) Transfer of control from one thread to another within same process requires a mode switch to the kernal.

Difference between User Level and Kernel Level Thread

Sr.No.	User level threads	Kernel level threads
1.	User level threads are faster to create and manage.	Kernel level threads slower to create and and manage.
2.	Implemented by a thread library at the user level.	Operating system support directly to kernel threads.
3.	User level thread can run on any operating system.	Kernel level threads are specific to the operating system.
4.	Support provided at the user level called user level thread.	Support may be provided by Kernel is called kernal level thread.
5.	Multithread application cannot take advantage of multiprocessing.	Kernel routines themselves can be multi-threaded.

User-level threads are much faster to switch between, as there is no context switch; further, a problem-domain-dependent algorithm can be used to schedule among them. CPU-

bound tasks with interdependent computations, or a task that will switch among threads often, might best be handled by user-level threads. Kernel-level threads are scheduled by the OS, and each thread can be granted its own timeslices by the scheduling algorithm. The kernel scheduler can thus make intelligent decisions among threads, and avoid scheduling processes which consist of entirely idle threads (or I/O bound threads). A task that has multiple threads that are I/O bound, or that has many threads (and thus will benefit from the additional timeslices that kernel threads will receive) might best be handled by kernel threads. "Kernel-level threads require a system call for the switch to occur; user-level threads do not.

A user thread is more appropriate for low-level tasks, whereas a kernel thread is better for high-priority tasks that should get high priority to system resources.

Q.8.(b) Explain file system in Unix. **(10)**

Ans. Unix File System : The files are normally used in UNIX for long term storage purpose. The data, programs and text all are stored as files, which are generally stored on mass storage devices like hard disks. These can also be stored on other storage media floppy disks, CD, tape etc. UNIX maintains the files in hierarchical structure named as directory structure. There are six types of files which are referenced in UNIX.

- **Regular files :** These files contain sequence of types that generally corresponds to the data or code. These can be referenced via the standard I/O system calls.
- **Directory files :** These files are stored in a special format on the disk and it consititutes the backbone of the system. These files can be referenced only means of directory specific system calls.
- **Special files :** It corresponds to the peripherals such as printers and disks, and interprocess communication mechanisms such as pipes and sockets. These can be reference by means of the standard I/O system calls.
- **Named pipes :** A pipe is an interprocess communictions facility. A pipe file buffers data received in its input so that a process that reads from the pipe's output, receives the data on a first-in first-out basis.
- **Links :** In essence, a link is an alternative file name for an existing file.
- **Symbolic links :** This is a data file that contains the name of the file it is linked to.

Theoretically, a UNIX file is a linear sequence of bytes. The UNIX kernel does not support the hierchical structure for file structure such as block, records, or fields. As it is apparent from the system call lseek (), which allows positioning the file pointer only in terms of a byte offset. Mostly the user is concerned with the handling of ordinary files, which correspond to what most systems treat as files.

All the blocks belonging to a Unix File System are logically divided into four parts, thus the Unix file system as a whole, has the following structure :

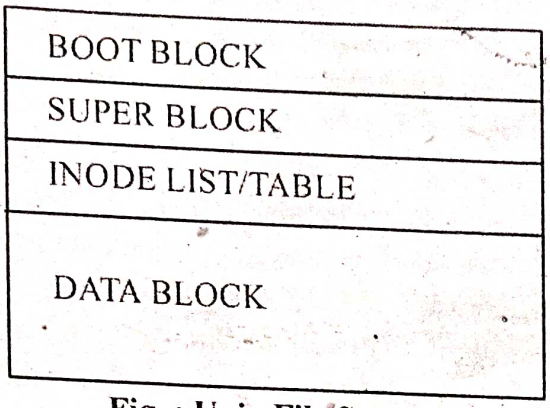


Fig. : Unix File System.

(i) Boot Block : The Boot block represents the beginning of the file system. It contains a program called BOOT STRAP LOADER, which is executed when we boot or initialize the host system (Unix Server). Only one boot block is needed to startup the Unix system, so all the file systems contain one (possibly empty) boot block

(ii) Super Block : The super block describes the state of the Unix file system, i.e. how large it is, how many files it can store and where to find empty files along with other relevant information.

(iii) Incode List/Table : We know that all entities in Unix are treated as files. The information related to these files (not the file data contents) is stored in an inode table on the disk. For each file, there is an inode entry in the table. Each inode entry is made up of 64 bytes and contains the following relevant details for that file.

- File Owner
- File Type
- Group to which owner belongs
- File Access permissions
- Date & Time of last access
- Date & Time of last modification
- Number of links to the files
- Size of the file
- Address of block (disk address) where file is physically present

(iv) Data Block : These contain the actual file contents created by the user or the system itself. In other words, they contain both file data and administrative data. An allocated data block can belong to only one file in the file system. This block can be used for storing any other files contents unless the file to which it originally belongs is deleted.

Q.9. Write notes on :

(a) Windows NT. **(10)**

(b) I/O interface. **(10)**

Ans.(a) Windows NT : The Microsoft windows NT OS is a 32-bit preemptive multitasking operating system for modern microprocessors. NT is portable to a variety of processor architecture.

The two varsions of NT are Windows NT Workstation and Windows NT server.

Design Principles : For the design goals that NT include following feature :

(i) Extensibility : It is an important property of any operating system that hopes to keep up with advancements in computing technology. So that changes are facilitated over time. NT is implemented using a layered architecture. The NT executive which runs in kernel or protected mode that provide basic system services.

(ii) Portability : An operating system is portable if it can be moved from one hardware architechure to another with relatively few changes. NT is designed to be portable.

(iii) Reliability : Reliability is the ability to handle error conditions that including the ability of the operating system to protect itself and its user from defective software.

(iv) Compatibility : NT provides source-level compatibility to application that follows the IEEE standard. Thus, they can be complied to run on NT without changing the source code.

(v) Performance : NT is designed to afford good performance. The subsystem that comprises NT can communicate with one another efficiently by a local procedure call facility that provides high-performance message passing.

(vi) International : NT is also design for international use. It provides supports for different locales vin the national language support API.

(vii) User mode : The user mode is made up of subsystems which can pass I/O requests to the appropriate kernel mode drivers via the I/O manager (which exists in kernel mode). Two subsystems make up the user mode layer of windows NT : the Environment subsystem and the integral subsystem.

The environment subsystem was designed to run applications written for may different types of operating systems. None of the environment subsystems can directly access hardware, and must request access to memory resources through the virtual Memory Manager that runs in kernel mode. Also, applications run at a priority than kernel mode processes.

The integral subsystem looks after operating system specific functions on behalf of the environment subsystem. It consists of a security subsystem, a workstation service and a server service. The security subsystem deals with security tokens, gransts or denies access to user accounts based on resource permissions, handles login requests and initiates login authentication and determines which system resources need to be audited by Window NT. It also looks after Active Directory. The workstation service in an API to the network redirector, which provides the computers access to the network. The server service is an API that allows the computer to provide network services.

(viii) Kernel mode : Windows NT kernel mode has full access to the hardware and system resources of the computer and runs code in a protected memory area. It controls access to schelduling, thread prioritization, memory management and the interaction with hardware. The karnel mode stops user mode services and applications from accessing critical areas of the operating that they should not have access to ; user mode processes must ask the kernel mode to perform such operations on their behalf.

Kernel mode consists of executive services, which is itself made up on many modules that do specific tasks, kernal drivers, a kernal and a Hardware Abstraction Layer, or HAL.

Ans.(b)Input Output Interface : Input-output interface provides a method for transferring information between internal storage and external I/O devices. Peripherals connected to a computer need special communication links for interfacing them with the central processing unit. The purpose of the communication link is to resolve the differences that exist between the central computer and each peripheral. The major differences are:

(1) Peripherals are electromechanical and electromagnetic devices and their manner of operation is different from the operation of the CPU and memory, which are electronic devices. Therefore, a conversion of signal values may be required.

(2) The data transfer rate of peripherals is usually slower than the transfer rate of the CPU, and consequently, a synchronization mechanism may be needed.

(3) Data codes and formats in peripherals differ from the word format in the CPU and memory.

(4) The operating modes of peripherals are different from each other and each must be controlled so as not to disturb the operation of other peripherals connected to the CPU.

To resolve these differences, computer systems include special hardware components between the CPU and peripherals to supervise and synchronize all input and output transfers. These components are called interface units because they interface between the processor bus and the peripheral device.

Need for I/O interface :

(1) Peripherals are electromechanical devices.But CPU and Memory are electronic devices. Therefore conversion of signal values may be required.

(2) Data codes and formats in peripherals differ from the word format in CPU and memory.

(3) Data transfer rate of peripherals are slower than CPU, So synchronization may be needed.

(4) The operating modes of peripherals are different. So they must be controlled so as not to disturb the operation of other peripherals that are connected to CPU.

PRINCIPLES OF OPERATING SYSTEM

Dec 2015

Paper Code: CSE-301-F

Note: Attempt five questions in all, selecting one question from each section. Q. No. 1 is compulsory.

Q.1(a) Give Difference between Multiprogramming and Multiuser Operating System. (5)

Ans. Multi programming: Multiprogramming is the technique of running several programs at a time using timesharing. It allows a computer to do several things at the same time. Multiprogramming creates logical parallelism. The concept of multiprogramming is that the operating system keeps several jobs in memory simultaneously. The operating system selects a job from the job pool and starts executing a job, when that job needs to wait for any i/o operations the CPU is switched to another job. So the main idea here is that the CPU is never idle.

Multi tasking: Multitasking is the logical extension of multi-programming .The concept of multitasking is quite similar to multiprogramming but difference is that the switching between jobs occurs so frequently that the users can interact with each program while it is running. This concept is also known as time-sharing systems. A time-shared operating system uses CPU scheduling and multiprogramming to provide each user with a small portion of time-shared system. For example, let us say you are printing a documet of 100 pages. While your computer is performing that, you still can do other jobs like typing a new document. So, more than one task is performed.

One of the main differences between multiprogramming and multitasking is, "In multiprogramming, a user cannot interact (everything is decided by OS, like picking the next program and sharing on time basis, etc...) where as in multitasking, a user can interact with the system (you can type a letter, while the other task of printing is going on)".

Q.1(b) What are buffering and spooling ? (5)

Ans. Buffering : In computing, a buffer is a region of memory used to temporarily hold data while it is being moved from one place to another. Typically, the data is stored in a buffer as it is retrieved from an input device (such as a keyboard) or just before it is sent to an output device (such as a printer). However, a buffer may be used when moving data between processes within a computer. This is comparable to buffers in telecommunication. Buffers can be implemented in either hardware or software, but the vast majority of buffers are implemented in software. Buffers are typically used when there is a difference between the rate at which data is received and the rate at which it can be processed, or in the case that these rates are variable, for example in a printer spooler.

Spooling : Simultaneous Peripheral Operations On-line, *spooling* refers to as a process that putting jobs in a buffer or say spool, or temporary storage area, a special area in memory or on a disk where a device can access them when it is ready. Spooling is useful because devices access data at different rates. The buffer provides a waiting station where data can rest while the slower device catches up.However, unlike a spool of thread, the first jobs sent to the spool are the first ones to be processed (FIFO, not LIFO).

The most common spooling application is print spooling. In print spooling, documents are loaded into a buffer (usually an area on a disk), and then the printer pulls them off the buffer at its own rate. Because the documents are in a buffer where they can be accessed by the printer, you can perform other operations on the computer while the printing takes place in the background. Spooling also lets you place a number of print jobs on a queue instead of waiting for each one to finish before specifying the next one.

Q.1(c) Explain semaphores.

Ans. Semaphore : An integer value used for signaling among processes. Only three operations may be performed on a semaphore, all of which are atomic : initialize, decrement, and increment. The decrement operation may result in the blocking of a process, and the increment operation may result in the unblocking of a process. Also known as a counting semaphore or a general semaphore.

Mutual exclusions with semaphores :

– A semaphore is a non-negative integer value

– Two atomic operations are supported on a semaphore

– *down* = decrements the value, since the value cannot be negative, the process blocks if the value is zero

– *up* = increments the value; if there are any processes waiting to perform a down, then they are unblocked

– Initializing a semaphore to 1 creates a mutual exclusion (mutex) semaphore

– Initializing a semaphore to 0 creates a signaling semaphore

– Initializing a semaphore to other values can be used for resource allocation and synchronization.

Example :

```
/* program mutualexclusion */
const int n = /* number of processes */;
semaphore s = 1; /* Initialize s to 1 */
void P (int i)
{
        while (true) {
                semWait (s);
                /* critical section */;
                /* remainder */;
        }
}
void main ( )
{
        parbegin (P(1), P(2), ...., P(n));
}
```

Q.1(d) What is I/O Hardware ? **(5)**

Ans. Computers operate on many kinds of devices. General types include storage devices (disks, tapes), transmission devices (network cards, modems), and human-interface devices

(screen, keyboard, mouse). Other devices are more specialized. A device communicates with a computer system by sending signals over a cable or even through the air.

The device communicates with the machine via a connection point termed a port (for example, a serial port). If one or more devices use a common set of wires, the connection is called a bus.In other terms, a bus is a set of wires and a rigidly defined protocol that specifies a set of messages that can be sent on the wires.

Daisy chain : When device A has a cable that plugs into device B, and device B has a cable that plugs into device C, and device C plugs into a port on the computer, this arrangement is called a daisy chain. It usually operates as a bus.

Controller : A controller is a collection of electronics that can operate a port, a bus, or a device. A serial-port controller is an example of a simple device controller. This is a single chip in the computer that controls the signals on the wires of a serial port.

The SCSI bus controller is often implemented as a separate circuit board (a host adapter) that plugs into the computer. It contains a processor, microcode, and some private memory to enable it to process the SCSI protocol messages. Some devices have their own built-in controllers.

I/O port : An I/O port typically consists of four registers, called the status , control, data-in, and data-out registers.

Polling : Polling is a process by which a host waits for controller response.It is a looping process, reading the status register over and over until the busy bit of status register becomes clear. The controller uses/sets the busy bit when it is busy working on a command, and clears the busy bit when it is ready to accept the next command. The host signals its wish via the command-ready bit in the command register. The host sets the command-ready bit when a command is available for the controller to execute.

- In the following example, the host writes output through a port, coordinating with the controller by handshaking
- The host repeatedly reads the busy bit until that bit becomes clear.
- The host sets the write bit in the command register and writes a byte into the data-out register.
- The host sets the command-ready bit.
- When the controller notices that the command-ready bit is set, it sets the busy bit.
- The controller reads the command register and sees the write command.
- It reads the data-out register to get the byte, and does the I/O to the device.
- The controller clears the command-ready bit, clears the error bit in the status register to indicate that the device I/O succeeded, and clears the busy bit to indicate that it is finished.

Direct Memory Access (DMA) : Many computers avoid burdening the main CPU with programmed I/O by offloading some of this work to a special purpose processor. This type of processor is called, a Direct Memory Access(DMA) controller. A special control unit is used to transfer block of data directly between an external device and the main memory, without intervention by the processor. This approach is called Direct Memory Access(DMA).

DMA can be used with either polling or interrupt software. DMA is particularly useful on devices like disks, where many bytes of information can be transferred in single I/O operations. When used with an interrupt, the CPU is notified only after the entire block of data has been transferred. For each byte or word transferred, it must provide the memory address and all the

bus signals controlling the data transfer. Interaction with a device controller is managed through a device driver.

Device Controllers : A computer system contains a many types of I/O devices and their respective controllers

- network card
- graphics adapter
- disk controller
- DVD-ROM controller
- serial port
- USB
- sound card .

Section-A

Q.2.(a) Explain Operating System and its type in detail. **(10)**

Ans. Operating System : The Operating system (OS) is the main software component with which users and applications interact. The OS provides the basis platform that defines how the user can interact with the computer. The operating system has two basis functions. First, it communicates with the PC's hardware; it gives applications access to things like the hard drive, it receives input from the keyboard, and it outputs data to the monitor. The second function is to provide a user interface and interpret commands so a user can communicate with the hardware. An operating system is also referred to as a command interpreter. Several type of operating systems performs basic functions.

Types of Operating Systems : Within the broad family of operating systems, there are generally four types, categorized based on the types of computers they control and the sort of application they supports. The categories are :

(i) Real Time Operating System (RTOS) : Real time operatings system are used to control machinery, scienctific instruments and industrial system. An RTOS typically has very little user interface capability, and no end-user utilities, since the system will be a "sealed box" when delivered for use. A very important part of an RTOS is managing the resources of the computer so that a particular operation executes in precisely the same amount of time, every time it occurs. In a complex machine, having a part move more quickly just because system resources are available may be just as catastrophic as having it not move at all because the system is busy.

(ii) Singel-User, Single-Task : As the name implies, this operating system is designed to mangage the computer so that one user can effectively do one thing at a time. The palm OS for Palm handheld computers is a good example of a modern single-user, signle-task operating system.

(iii) Single-User, Multi-Tasking : This is the type of operating system most people use on their desktop and laptop computers today, Microsoft's Windows and Apple's Mac OS platforms are both examples of operating system that will let a single user have several programs in operation at the same time. For example, it's entirely possible for a windows user to be writing a note in a word processor while downloading a file from the internet while printing the next of an e-mail message.

(iv) Multi User : A multi user operating system allows many different users to take advantage of the computer's resources simultaneously. The operating system must make sure that the requirements of the various users are balanced, and that each of the program they are using has sufficient and separate resources so that a problem with one user doesn't affect at entire community of users. UNIX, VMS and mainframe operating systems such as MVS, are examples of multi-user operating systems.

Q.2.(b) (i) Differentiate between process and program. (5)

(ii) What is process scheduling. (5)

Ans. (i) Differentiate between process and program : The differences between a program and a process are :

Program	Process
1. A program consists of a set of instructions in a programming language.	1. A process is a sequence of instruction exection.
2. A program is a static object that exists in a file	2. .A process is a dynamic object i.e. a program in execuation.
3. A program is loaded into the secondary storage devices.	3. A process is loaded into the main memory.
4. The time span of a program in unlimited.	4. The time span of a process is limited.
5. A program is a passive entity.	5. A process is an active entity.

(ii) Process scheduling : The assignment of physical processor to processes allows processor to accomplish work. The problem in determining when processors should be assigned and to which processes, is called *processor scheduling* or *CPU scheduling*. When more than process is runable, the operating system must decide which one first. The part of the operating system concerned with this decision is called the scheduler, and algorithm it uses is called the scheduling algorithm.

Types of Schedulers : Genrally, there are three types of schedulers, which may coexist in a complex operating system :

(i) Long term schedular
(ii) Medium term schedular
(iii) Short term scheduler.

Fig.(1) shows the possible traversal path of jobs and programs through the components and queues, in a computer system.

As shown in fig.(1) a submitted batch job joins the batch queue while waiting to be processed by the long term scheduler. Once, scheduled for execution, processes spawned by the batch job, enter ther ready queue to await the processor allocation by the short term scheduler. After becoming suspended in waiting queue, the running process may be removed from memory and swapped out to secondary stroge. Such processes are subsequently admitted to main by the medium term scheduler in order to be considered for execution by the short term schedular.

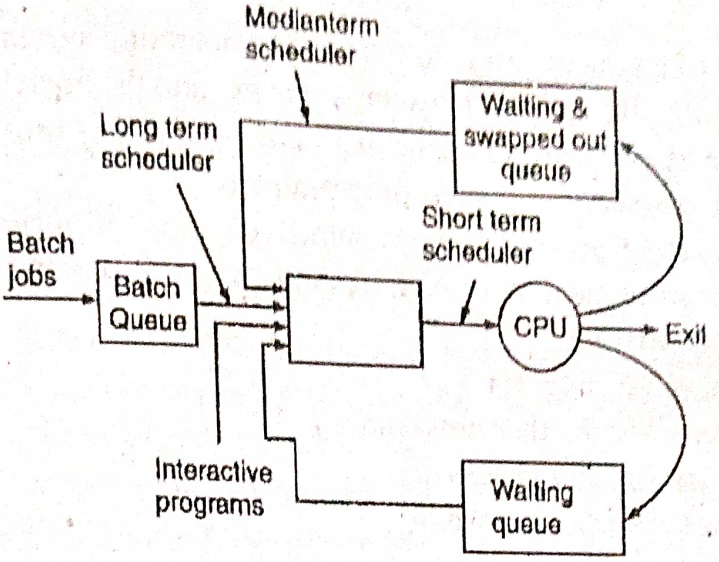


Fig.(1) : Scheduler.

Q.3.(a) Explain System program and System Calls. **(10)**

Ans. System Programs : System programs provide a convenient environment for program development and execuation. Some of them provides a user interface to system calls ; other are more complex.

System programs can be divided into several categories :

(i) *File Management* : These programs create, delete,copy, rename, print, dump, list and generally manipulates files and directories.

(ii) *Status Information* : Some programs simply ask the operating system for date, time, amount of available memory or disk space, number of users or similar status information.

(iii) *File Modification* : Several text editors may be available to create or modify the content of files stored on disk.

(iv) *Programming-language support* : Compilers, assemblers, and interpreters for common programming languages (such as C, C++, Java, Basic, PERL) are after provided to the user with the operating system.

(v) *Programming Loading and Execution* : When a program is assembled or compiled it must be loaded into memory for its executation. The system may provide absolute loaders, relocatable loaders, linkage editors, and overlay loaders. Debugging systems for either high level language or machine languague are also needed.

(vi) *Communication* : These programs provide the mechanism for creating the virtual connections among processes, users different computer systems. They allow users to send message through web browers, E mail services from one machine to another.

(vii) *System Utilities* : Most operating system are supplied with programs that solve common problems, or perform common operations such as web browsers, word processors, statistical-analysis packages, spread sheets database sytems, etc.

System Calls : User programs communicate with an operating system and request services from it by making system calls. Each *system call* has a *library procedure* that the *user program* calls. This procedure puts the parameters of the system call in registers and then issues a *TRAP* instruction which is a *protected procedure call* to start the operating system. The library procedure *hides* the details of TRAP instructions and makes system calls appear as on ordinary procedure calls.

The *purpose* of a system call is to request the operating system to perform some task. So a system call provides an interface between a process and the operating system. These calls are generally avaiable as assembly language instructions, and they usually listed in the various *manuals* used by the *assembly language* programmers.

Systems call can be grouped roughly into five major categories : process control, file management, device management, information maintenance and communications.

1. **Process Control :**
   (i) end/abort this program
   (ii) load/execture another program
   (iii) creat/terminate a process
      (*a*) get /set process attribute
      (*b*) Wait for specified time
      (*c*) Wait for an event
      (*d*) allocate the process in memory or free the process from the memory.
2. **File Management :**
   (i) create/open /read/write/close/delete file
   (ii) get/set file attributes.
3. **Device Management :**
   (i) request/read/write/release device
   (ii) get/set device attributes
   (iii) logically attach/detach devices.
4. **Information Maintenance :**
   (i) get/set time or date
   (ii) get/set system data
   (iii) get/set process/file/device attributes
5. **Communication :**
   (i) create/delete communication connection
   (ii). send/receive messages
   (iii) Transfer status information
   (iv) attach/detach remote devices.

Q.3.(b) Explain Priority scheduling and Multilevel feedback queue scheduling. **(10)**

Ans. Priority scheduling : Priority can be defined either internally or externally. Internally defined priorites use some measurable quantities or qualities to compute priority of a prcoerss. Examples of Internal periorities are Time limits, Memory requirements, File requirement, for example, number of open files, CPU Vs I/O requirments Externally defined priorities are set by criteria that are external to operating system such as the importance of process, type of amount of funds being paid for computer use, the department sponsoring the work.

Priority scheduling can be either preemptive or non-preemptive :

(a) A preemptive priority algorithm will preemptive the CPU if the priority of the newly arrival process is higher than the priority of the currently running process.

(b) A non-preemptive priority algorithm will simply put the new process at the head of the ready queue.

A major problem with priority scheduling is indefinite blocking or starvation. A solution to the problem of indefinite blockage of the low-priority process is aging. Aging is a technique of gradually increasing the prioirty of process that wait in the system for a long period of time.

Multilevel Feedback Queue Scheduling : A scheduling mechanism is expected to

(i) favour short jobs.

(ii) Favour input/output bound jobs to get good input/output device utilization.

(iii) Determine the nature of job quickly and schedule it accordingly.

(iv) MFQS provides and structure that accomplishes these goals.

(v) A new process enters the queuing network at the back of the top queue. It moves through the FIFO Queue until it gets the CPU.

(vi) If the job completes or relinquishes the CPU to wait for input/output completion or completion of some event, the job leaves the queuing network.

(vii) If the quantum expires before the process voluntarily relinquishes the CPU, the process is placed at the back of next lower level queue.

The process is next serviced when it reaches the head of the queue of the first queue is empty.

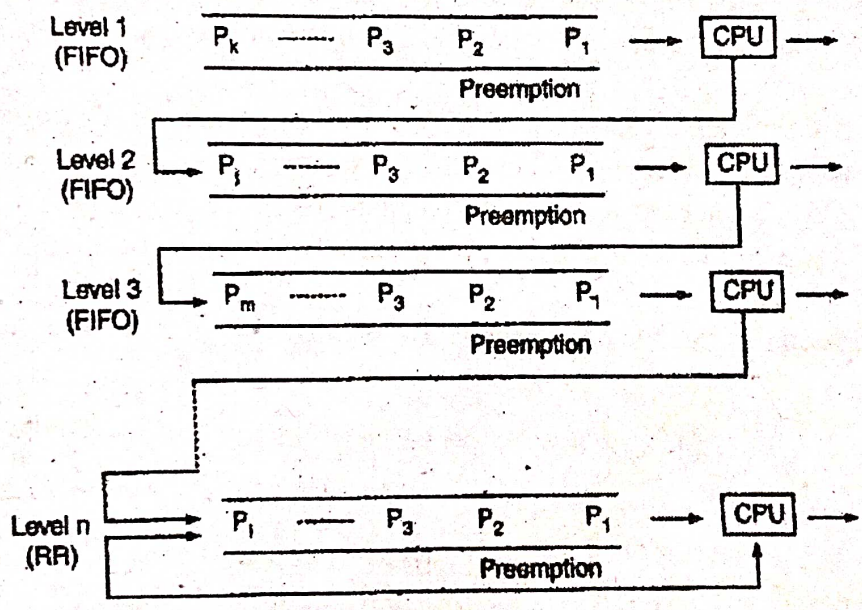


Fig.(1) : Multiple Feedback Queue.

(viii) As long as the process uses the full quantum provided at each level, it continues to move to the back of the next lower queue.

(ix) In many multilevel feedback queue schemes, the quantum given to the process as it moves to each lower level queue becomes larger. Thus the longer a process has been in the queuing network, the large quantum it is assigned each time it gets the CPU.

(x) Multilevel feedback queues are ideal for separating processes into categories based on their need for the CPU. In a time sharing system each time a process leaves the queuing network, it can be "stamped" with the identity of the lowest level queue in which it resides. When the process reenters the queuing network, it can be sent directly to the queue in which it last completed operation.

Here, the schedular is using the heuristic that the recent past behaviour of a process is a good indicator of its near future behaviour. So, a CPU bound process returning to the queuing networking is not put into the higher level queues, where it would interfere with service to high priority short processers OR inunit/output bound processes.

Section-B

Q.4. Explain paging and Segmentation techniques. (20)

Ans. Paging : Paging is a memory management schemes that removes the requirement of contiguous allocation of physical memory. This scheme permits the physical address space of a process to be non-contiguous. Paging in its various forms is commonly used in most operating systems. Initially, paging was used with the hardware support. But current designs of an operating system have implemented paging by closely integrating the hardware and operating system, especially on 64 bit microprocessors.

The physical memory is conceptually divided into a number of fixed-size blocks, called frames and the logical address space (or virtual addres space) is also splitted into fixed size blocks, called pages. When a process is to be executed, its pages are loaded into any available memory frames from the backing store. The backing store is also divided into fixed size blocks that are of the same size of the frames i.e., the size of a frame is same as the size of a page for a particular hardware.

Allocation of memory consists of finding a sufficient number of unused page frames for loading the pages of a requesting process for its execution. An address translation scheme is used to map the logical pages to their physical counterparts. Since each page is mapped separately, different page frames allocated to a single process need not occupy contiguous areas of physical memory.

Fig. (1) demonstrate the basic principle of paging.

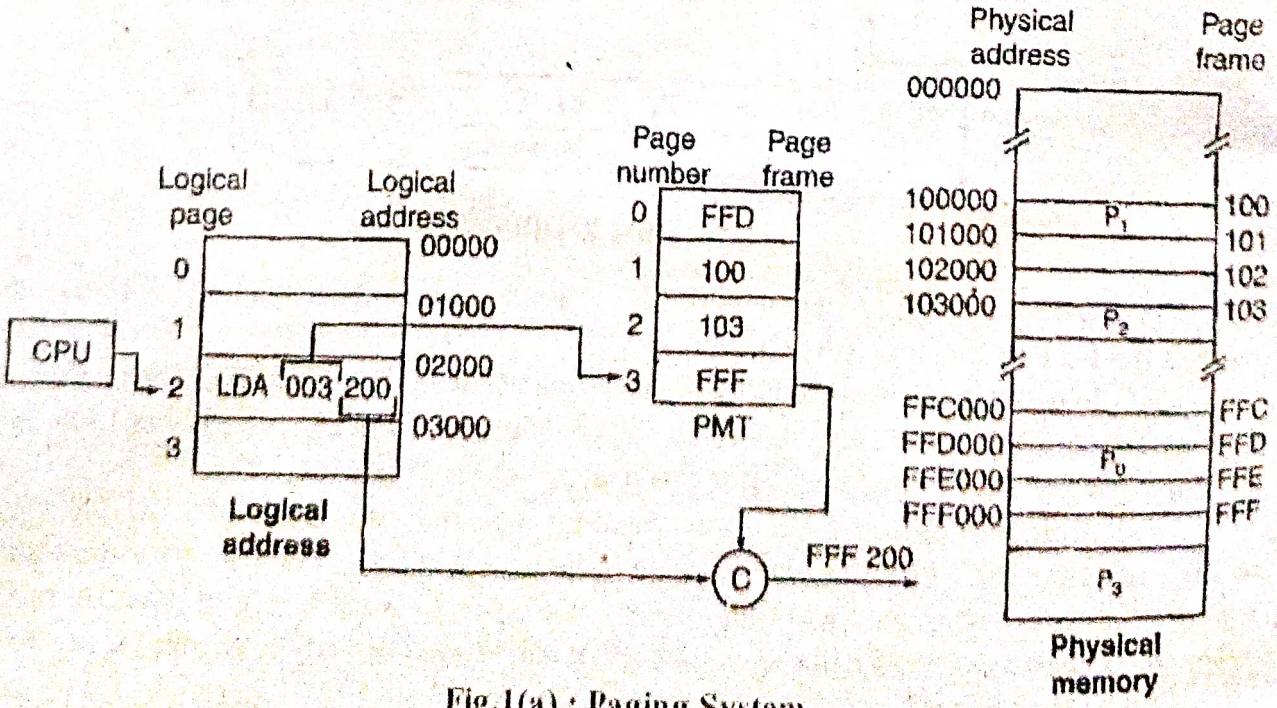


Fig.1(a) : Paging System.

It illustrates a sample of 16 MB system in which logical and physical addresses are assumed to be 24 bits each in size. The logical address space is divided into four pages numbered from 0 to 3. The mapping of logical addresses to physical addresses in paging system is performed at page level. Particularly, each virtual address (physical address) is divided into two parts : the page number (p) and the offset (d) within that page. Since page (p) and frames (f) have identical sizes, offsets within each are identical and need not be mapped.

In above example, example 24 bit logical address may be regarded as a 12-bit page numbers (high order bits) and a 12 bits offset within the page.

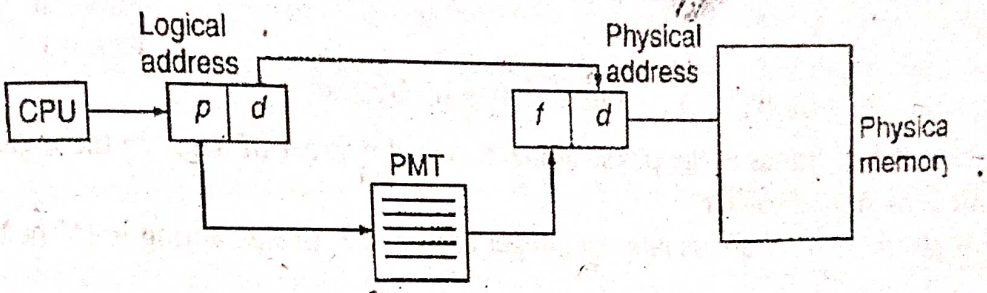


Fig.1(b) : Paging System.

In paging system, address translation is performed with the help of a mapping table called as page map table. (PMT) as shown in fig.1(b). PMT is constructed as process loading time in order to establish the correspondence between the virtual and physical address. In PMT, there is one entry for each page of a process indexed with the page number. The value of each entry is the number of frames (composed of high order, page level bits) in the physical memory when the corresponding page is placed.

Fox example, Page 0 is placed in the frame whose starting address is FFD000H (in hexadecimal). With each frame being 1000 H in size, the corresponding frame number is FFDH as indicated on the right side of physical memory in fig.1(a). This value is stored in the first entry of PMT. All other entries are filed with frame numbers of the region where the corresponding pages are actually loaded.

The logic addresses translation in paged system is illustrated in fig.(2) on the example of virtual addresses 3200 H.

The virtual address is splitted into two parts :

003	200

003 H indicates the page numbers and 200 H indicates the offset within that page. Page number is used to the index of PMT and to obtain the corresponding physical frame number (FFFH in out example). This value is then concatenated with the offset (200 H) to produce the physical address (FFF 200 H) which is used as a reference address in physical memory.

FFF	200

If the capacity of physical memory is 'm' and the size of each page is 'p' then the number of frame in the physical memory will be

$$f = \frac{m}{p} \quad \text{...(i)}$$

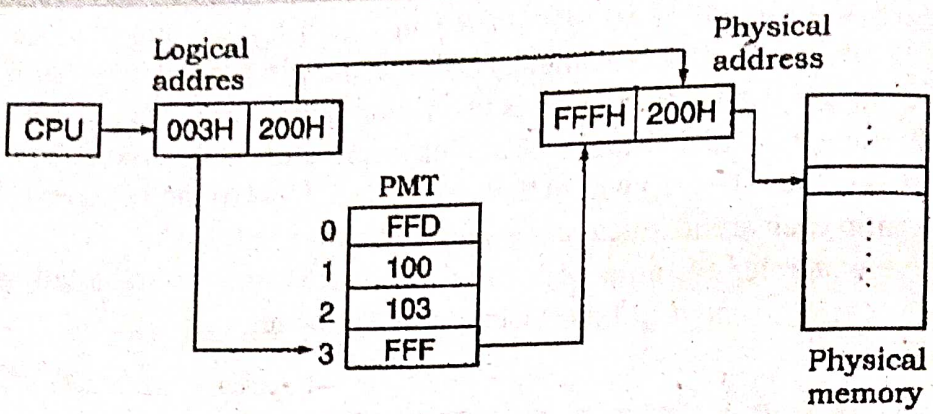


Fig.(2) : Paging System.

Number of franes in the physical memory and number of pages in the logical memory are identical as started earlier.

Both '*m*' and '*p*' are usually an integer power of 2, thus resulting in '*f*' being also can integer.

The page size is similar to the frame size is defined by the hardware and it varies according to the architecture of computer system. For the convenience of mapping, page sizes are usually an integer power of 2. If the size of logical address space is 2^m and page size is 2^n units than the high order $m - n$ units of logical address designate the page number and the n low-order units designate the page offset.

Thus, the logical address is defined as :

Page number	Page offset
p	d
m–n	n

where '*p*' is the index number in the page table and '*d*' is the value (displacement).

Segmentation : Segmentation is another technique for the noncontiguous storage allocation. It is different from paging as pages are physical in nature and hence are of fixed size, whereas segments are logical divisions of a program and hence are of variable size.

It is a memory management schemes that supports the user view of memory rather than system view of memory as in paging, in which we are concerned on the division of physical memory into fixed size blocks called frames. In segmentation, we divide the logical address space into different segments. The general division can be : main program, set of subroutines, procedures, functions and set of data structure (stack, array etc). Each segment has a name and length which is loaded into physical memory as it is. For simplicity, the segments are referred by a segment number, rather than segement name.

Thus, a logical address consists of two tuples, :

< segement_name, offset >

Hence, address is considered as two dimensional. The size of a segement varies according to the data stored in it or the nature of operation performed on that segment. This change in size does not affect other segment in the memory, as segments are large in size.

So, segmentation is a method of dividing memory into independent address spaces called segement. Each segment consists of linear sequence (0, 1, 2, 3, ...) of address starting from 0 to maximum value depending upon the size of segement.

Like in paging system we face the same problem in segmentation that how the compiler will generate 2-*d* array and how it is same address translation ? When we put all segments in a conceptual manner one after the other as shown in fig.(3), then it is clear that the compiler itself has to generate the two dimesional address (s, d) i.e., (segment no, offest).

Segment No.	0	1	2	3
Limit address	0–999	1000–1699	1700–2499	2500–3399

Fig.(3) : Virtual address segmentation.

This is different from the paging system in which a single dimensional virtual address and a two dimensional address would be exactly same in the binary as the page size is an integral power of 2.

In segmentation it is not possible as the segment size is unpredicatable. So we need to express the address in 2d form.

So the virtual address space is divided into two parts in which high order units refer to '*s*' *i.e.* segment number and lower order units refer to '*d*' i.e., displacement (limit value).

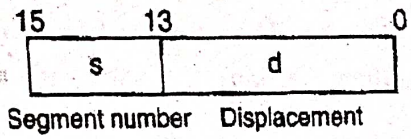


Fig.(4) : Virtual address space or logical address space (16 bit

Q.5. Consider the following page-reference string : 2 3 2 1 5 2 4 5 3 2 5 2
How many page faults occur in the following replacement algorithms, assuming there frames.

(*i*) FIFO (*ii*) LRU (*iii*) Optimal

Ans. (i) FIFO

frame	2	3	2	1	5	2	4	5	3	2	5	2
0	2	2	2	2	5	5	5	5	3	3	3	3
1		3	3	3	3	2	2	2	2	2	5	5
2				1	1	1	4	4	4	4	4	2
page fault	*	*		*	*		*		*	*		

Number of page fault = 9

(ii) LRU

frame	2	3	2	1	5	2	4	5	3	2	5	2
0	2	2	2	2	2	2	2	2	3	3	3	3
1		3	3	3	5	5	5	5	5	5	5	5
2				1	1	1	4	4	4	2	2	2
page fault	*	*		*	*		*		*	*		

Number of page fault = 7

(iii) Optimal

frame	2	3	2	1	5	2	4	5	3	2	5	2
0	2	2	2	2	2	2	4	4	4	2	2	2
1		3	3	3	3	3	3	3	3	3	3	3
2				1	5	5	5	5	5	5	5	5
page fault	*	*		*	*		*			*		

Number of page fault = 6

Section-C

Q.6.(a) Explain file system and its types. **(10)**

Ans. File system : A file system is a subsystem of an operating system that performs file management activities such as organiszation, storing, retrieval, naming, sharing and protection of files. It is designed to allow programs to use a set of operations that characterize the file abstraction and free the programmers from conerns about and details of space allocation and layout of the secondary storage device. Therefore, a file system provides an abstraction of a storage device ; that is, it a conventent mechanism for storing and retrieving information from the storage device.

Types of file systems : Following are the types of file system :

(1) Disk file systems : A disk file system takes advantages of the ability of disk storage media to randomly address data in a short amount of time. Additional considerations include the speed of accessing data following that initially requested and the anticipation that the following data may also be requested. This permits multiple users (or processes) access to various data on the disk without regard to the sequential location of the data. Examples include FAT (FAT12, FAT16, FAT32), exFAT, NTFS, HFS and HFS+, HPFS.

(2) Flash file systems : A flash file system considers the special abilities, performance and restrictions of flash memory devices. Frequently a disk file system can use a flash memory device as the underlying storage media but it is much better to use a file system specifically designed for a flash device.

(3) Tape file systems : A tape file system is a file system and tape format designed to store files on tape in a self-describing form. Magnetic tapes are sequential storage media with significantly longer random data access times than disks, posing challenges to the creation and efficient management of a general-purpose file system.

In a disk file system there is typically a master file directory, and a map of used and free data regions. Any file additions, changes, or removals require updating the directory and the used/free maps. Random access to data regions is measured in milliseconds so this system works well for disks.

Tape requires linear motion to wind and unwind potentially very long reels of media. This tape motion may take several seconds to several minutes to move the read/write head from one end of the tape to the other.

Consequently, a master file directory and usage map can be extremely slow and inefficient with tape. Writing typically involves reading the block usage map to find free blocks for writing.

updating the usage map and directory to add the data, and then advancing the tape to write the data in the correct spot. Each additional file write requires updating the map and directory and writing the data, which may take several seconds to occur for each file.

Tape file systems instead typically allow for the file directory to be spread across the tape intermixed with the data, referred to as streaming, so that time-consuming and repeated tape motions are not required to write new data.

(4) Network file systems : A network file system is a file system that acts as a client for a remote file access protocol, providing access to files on a server. Programs using local interfaces can transparently create, manage and access hierarchical directories and files in remote network-connected computers. Examples of network file systems include clients for the NFS, AFS, SMB protocols, and file-system-like clients for FTP and WebDAV.

(5) Shared disk file systems : A shared disk file system is one in which a number of machines (usually servers) all have access to the same external disk subsystem (usually a SAN). The file system arbitrates access to that subsystem, preventing write collisions. Examples include GFS2 from Red Hat, GPFS from IBM, SFS from DataPlow, CXFS from SGI and StorNext from Quantum Corporation.

(6) Flat file systems : In a flat file system, there are no subdirectories. When floppy disk media was first available this type of file system was adequate due to the relatively small amount of data space available. CP/M machines featured a flat file system, where files could be assigned to one of 16 user areas and generic file operations narrowed to work on one instead of defaulting to work on all of them. These user areas were no more than special attributes associated with the files, that is, it was not necessary to define specific quota for each of these areas and files could be added to groups for as long as there was still free storage space on the disk. The early Apple Macintosh also featured a flat file system, the Macintosh File System. It was unusual in that the file management program (Macintosh Finder) created the illusion of a partially hierarchical filing system on top of EMFS. This structure required every file to have a unique name, even if it appeared to be in a separate folder.

While simple, flat file systems become awkward as the number of files grows and makes it difficult to organize data into related groups of files.

Q.6.(b) Explain Semaphore in detail. **(10)**

Ans. A semaphore is a special kind of synchronization data that can be used only through specific synchronization primitives. The primitives are simple operations that can be used to implement both mutual exculsion and control synchronization. It also ensures the lack of race conditions. Fig.(1) shows a semantics of *wait and signal* operation. Whan a process performs a *wait* operation on a semaphore, the operation checks whether the value of the semaphore > 0. If so, it decrements the value of semaphore and lets the process execution be complted; otherwise it blocks the process on the semaphore. A *signal* operation on the semaphore activates a blocked process on the semaphore, if any, or increments the value of semaphore by 1. Indivisibility of the wait and signal operations is ensured by the programming language or the OS that implements it.

```
operation wait (S)
begin
    if S > 0
        then S = S – 1;
```

```
        else block the process on S ;
end ;
Operation signal (S)
begin
    if some processes are blocked on S
    then activate one blocked process ;
    else S = S + 1 ;
end ;
```

Fig. (1) : Semantics of wait and signal operations

Process use wait signal operations to sychronize their execuation w.r.t. one another. The initial value of semaphore determines how many process can get past the wait operation. A process that does not get past a wait operation is blocked on the semaphore. This feature avoids busy waits.

Q.7.(a) What are advantage of a distributed file system over a file system in a centralized system ? (10)

Ans. Advantages of Distributed system over centrilized systems :

Item	Description
Economics	Microprocessors offer a better price/performance than Mainiframes. 10,000 CPUs executing 50 MIPS yields executing 500,000 MIPS. NOT possible for a single CPU to achieve
Speed	A distributed system may have more total computing power than mainframe
Inherent distribution	Some applications involve spactially separated machines
Reliability	If one machine crashes, the system as whole can Still service
Incremental growth	Computer power can be added in small increments.

Q.7.(b) What are algorithms for deadlock prevention ? (10)

Ans. Banker's Algorithm : The resource allocation graph algorithm is not applicable to a resource allocation system with multiple instances (resource units) of each resource type (resource class). The deadlock avoidance scheme that is applicable to such a system, but is less efficient than RAG scheme. This algorithm is commonly known as *Banker's Algorithm.*

The name was chosen because this algorithm could be used in a banking system to ensure that the bank never alloctes its available cash that is can no logner satisfy the needs of al its customers.

When a new process enters the system, it must declare the maximum number of instances of each resource type it may need. This number may not exceed the total number of resources

in the system. When a user requests a set of resources, the system must determine whether the allocation of these resources will leave the system in safe state. If it will, the resources are allocated, otherwise, the process must wait until some other process releases enough resources.

Several data structure required to implement the Banker's Algorithm.These data structures encode the satee of the resource allocation system. Let 'n' be the number of process in the system and 'm' be the number of resource types. We need following data structures :

(i) *Available* : A vector of length 'm' indicates the number of avaiable resources of each type. If available [j] = k, then there 'k' instances of resource type R_1 available. It is a 1-dimensional array of length 'm'

(ii) *Max* : It is a 2-dimensional array (matrix) of length $n \times m$ that defines the maximum demand of each process in a system. If max [i, j] = k, then process P_i may request at most 'k' instance of resource type R_j.

(iii) *Allocation* : It is also a 2-d array (matrix) of length $n \times m$ that defines the number of resources of each type currently allocated to each process.

If allocation [i, j] = k, then process P_i is currently allocated 'k' instances of resource type R_j.

(iv) *Need* : It is also a 2-d array (matrix) that of length $n \times m$ that indicates the remaining resources need of each process.

If need [i, j] = k, then process P_i may need 'k' more instances of resources type R_j to complete its task.

Note that need [i, j] = max [i, j] – Allocation [i, j]

These data structures vary over time in both size and value.

We can treat each row in the matrices allocation and need as vectors and refers them Allocation_i and Need_i respectively.

Allocation_i specifices the process currently allocated to process P_i and vector Need, specifices the additional resources that process P_i may still request to complete its task.

Safty 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, ..., n

(2) Find an i such that both

(i) Finish [i] = false

(ii) $\text{Need}_i \leq$ work

If no such i exists go to step (4)

(3) Work = Work + Allocationi

Finish [i] = true

goto step (2)

(4) if Finish [i] = true for all i,

then the system is in safe state.

This algorithm will require an order of $m \times n^2$ operations to decide whether a state is safe.

Resources-Request Algorithm : Let $Request_i$ be the request vector for process P_i. If $Request_i [j] = k$ then process P_i wants k instances of resource R_j. When a request for resources is made by process P_i, the following actions are taken :

(1) If $Request_i \leq Need_i$
goto step (2) ; otherwise.
raise an error codition, since the process has exceeded its maximum claim.

(2) If $Request_i \leq Available$
go to 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 P_i by modiflying the state as follows :
$Available = Available - Request_i$
$Allocation_i = Allocation_i + Request_i$
$Need_i = Need_i - Request_i$

If the resulting resource - allocation state is safe, the transaction is completed and process P_i is allocated its resources. However, if the new state is unsafe, then P_i must wait for $Request_i$ and the old resource-allocation state is restored.

Section-D

Q.8.(a) What are various layers of I/O software ? Explain function of each layer. **(10)**

Ans. I/O software is often organized in four layers, as shown in Fig.(1).

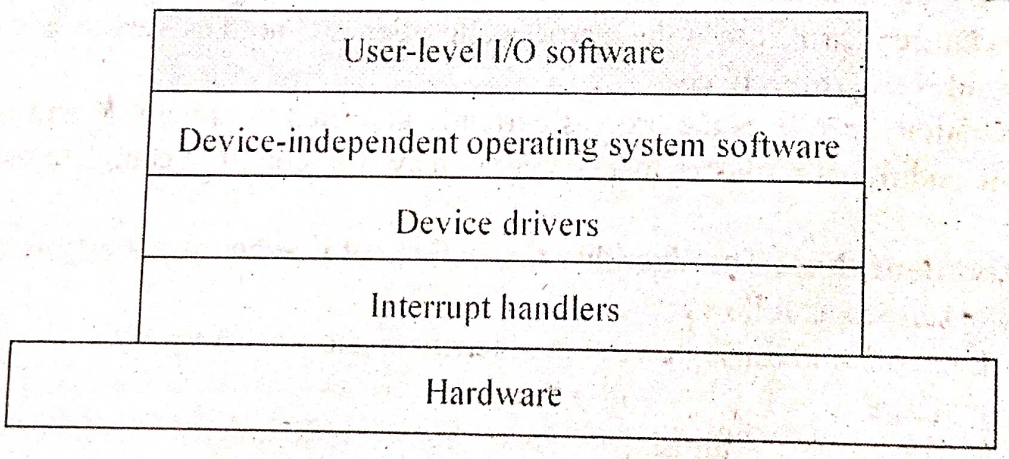


Fig.(1) : Layers of the I/O software system.

Interrupt Handlers : Interrupts are an unpleasant fact of life; although they cannot be avoided, they should be hidden away, deep in the bowels of the operating system, so that as little of the operating system as possible knows about them. The best way to hide them is to have the driver starting an I/O operation block until the I/O has completed and the interrupt occurs. The driver can block itself by doing a down on a semaphore, a wait on a condition variable, a receive on a message, or something similar, for example.

When the interrupt happens, the interrupt procedure does whatever it has to in order to handle the interrupt. Then it can unblock the driver that started it. In some cases it will just complete up on a semaphore. In others it will do a signal on a condition variable in a monitor. In

still others, it will send a message to the blocked driver. In all cases the net effect of the interrupt will be that a driver that was previously blocked will now be able to run. This model works best if drivers are structured as independent processes, with their own states, stacks, and program counters.

Device Drivers : Each I/O device attached to a computer needs some device-specific code for controlling it. This code, called the device driver, is generally written by the device's manufacturer and delivered along with the device on a CD-ROM. Since each operating system needs its own drivers, device manufacturers commonly supply drivers for several popular operating systems.

Each device driver normally handles one device type, or one class of closely related devices. For example, it would probably be a good idea to have a single mouse driver, even if the system supports several different brands of mice. As another example, a disk driver can usually handle multiple disks of different sizes and different speeds, and perhaps a CD-ROM as well. On the other hand, a mouse and a disk are so different that different drivers are necessary.

In order to access the device's hardware, meaning the controller's registers, the device driver traditionally has been part of the system kernel. This approach gives the best performance and the worst reliability since a bug in any device driver can crash the entire system. MINIX 3 departs from this model in order to enhance reliability. As we shall see, in MINIX 3 each device driver is now a separate user-mode process.

Operating systems usually classify drivers as block devices, such as disks, orcharacter devices, such as keyboards and printers. Most operating systems define a standard interface that all block drivers must support and a second standard interface that all character drivers must support. These interfaces consist of a number of procedures that the rest of the operating system can call to get the driver to do work for it.

In general terms, the job of a device driver is to accept abstract requests from the device-independent software above it and see to it that the request is executed. A typical request to a disk driver is to read block n. If the driver is idle at the time a request comes in, it starts carrying out the request immediately. If, however, it is already busy with a request, it will normally enter the new request into a queue of pending requests to be dealt with as soon as possible.

The first step in actually carrying out an I/O request is to check that the input parameters are valid and to return an error if they are not. If the request is valid the next step is to translate it from abstract to concrete terms. For a disk driver, this means figuring out where on the disk the requested block actually is, checking to see if the drive's motor is running, determining if the arm is positioned on the proper cylinder, and so on. In short, the driver must decide which controller operations are required and in what sequence.

Once the driver has determined which commands to issue to the controller, it starts issuing them by writing into the controller's device registers. Simple controllers can handle only one command at a time. More sophisticated controllers are willing to accept a linked list of commands, which they then carry out by themselves without further help from the operating system.

After the command or commands have been issued, one of two situations will apply. In many cases the device driver must wait until the controller does some work for it, so it blocks itself until the interrupt comes in to unblock it. In other cases, however, the operation finishes

without delay, so the driver need not block. As an example of the latter situation, scrolling the screen on some graphics cards requires just writing a few bytes into the controller's registers. No mechanical motion is needed, so the entire operation can be completed in a few microseconds.

In the former case, the blocked driver will be awakened by the interrupt. In the latter case, it will never go to sleep. Either way, after the operation has been completed, it must check for errors. If everything is all right, the driver may have data to pass to the device-independent software (e.g., a block just read). Finally, it returns some status information for error reporting back to its caller. If any other requests are queued, one of them can now be selected and started. If nothing is queued, the driver blocks waiting for the next request.

Dealing with requests for reading and writing is the main function of a driver, but there may be other requirements. For instance, the driver may need to initialize a device at system startup or the first time it is used. Also, there may be a need to manage power requirements, handle Plug 'n Play, or log events.

Device-Independent I/O Software : Although some of the I/O software is device specific, a large fraction of it is device independent. The exact boundary between the drivers and the device-independent software is system dependent, because some functions that could be done in a device-independent way may actually be done in the drivers, for efficiency or other reasons. The functions shown in Fig. 3-6 are typically done in the device-independent software. In MINIX 3, most of the device-independent software is part of the file system. Although we will study the file system in Chap. 5, we will take a quick look at the device-independent software here, to provide some perspective on I/O and show better where the drivers fit in.

Functions of the device-independent I/O software :

(i) Uniform interfacing for device drivers
(ii) Buffering
(iii) Error reporting
(iv) Allocating and releasing dedicated devices
(v) Providing a device-independent block size

Q.8.(b) What are the difference between user level threads and kernel level threads ? Under what circumstances is one type better than other ? (10)

Ans. User level Thread : In a user level thread, all the work of thread management is done by the application and the kernal is not aware of the existence of threads. The thread library contains code for creating and destroying threads, for passing message and data between threads, for scheduling thread execution and for saving and restoring thread contexts. The application begins with a single thread and begins runnings in that thread. Fig.(1). shows the user level thread. User level thread are generally fast to create and manage.

Advantage of user level thread over kernal level thread :

(i) Thread switching does not require kernal mode privileges
(ii) User level thread can run on any operating system.
(iii) Scheduling can be application specific
(iv) User level threads are fast to create and manage.

Disadvantages of user level thread :

(i) In a typical operating system, most system calls are blocking.
(ii) Multithreaded application cannot take advantage of multiprocessing.

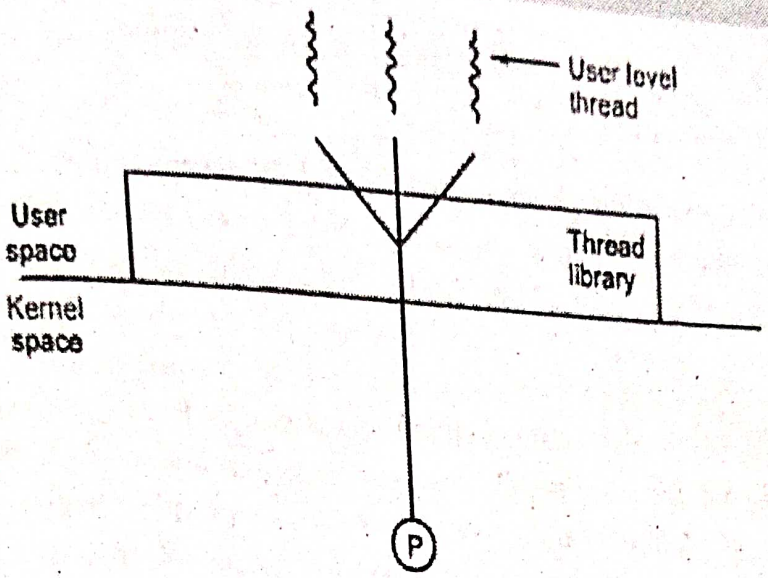


Fig.(1) : User level thread.

Kernal Level Threads : In kernal level thread, thread managment is done by the kernal. There is no thread management code in the application area. Kernal threads are supported directly by the operating system. Any application can be programmed to be multithreaded. All of the threads within an application are supported within a single process. The kernal maintains context information for the process as a whole and for individulas threads within the process.

Scheduling by the kernal is done on a thread basis. The kernal performs thread creation scheduling and managment in kernal space. Kernal threads are generally slower to crete and manage than the user threads.

Advantages of kernal level thread :

(i) Kernal can simultaneously schedule multiple threads from the same process on multiple process.

(ii) If one thread in a process is blocked, the kernal can schedule another thread of the same process.

(iii) Kernal routines themselves can be multithreaded.

Disadvantages : (i) Kernal threads are generally slower to create and manage than the user threads.

(ii) Transfer of control from one thread to another within same process requires a mode switch to the kernal.

Difference between User Level and Kernel Level Thread

Sr.No.	User level threads	Kernel level threads
1.	User level threads are faster to create and manage.	Kernel level threads slower to create and and manage.
2.	Implemented by a thread library at the user level.	Operating system support directly to kernel threads.
3.	User level thread can run on any operating system.	Kernel level threads are specific to the operating system.
4.	Support provided at the user level called user level thread.	Support may be provided by Kernel is called kernal level thread.

5.	Multithread application cannot take advantage of multiprocessing.	Kernel routines themselves can be multi-threaded.

User-level threads are much faster to switch between, as there is no context switch; further, a problem-domain-dependent algorithm can be used to schedule among them. CPU-bound tasks with interdependent computations, or a task that will switch among threads often, might best be handled by user-level threads. Kernel-level threads are scheduled by the OS, and each thread can be granted its own timeslices by the scheduling algorithm. The kernel scheduler can thus make intelligent decisions among threads, and avoid scheduling processes which consist of entirely idle threads (or I/O bound threads). A task that has multiple threads that are I/O bound, or that has many threads (and thus will benefit from the additional timeslices that kernel threads will receive) might best be handled by kernel threads. "Kernel-level threads require a system call for the switch to occur; user-level threads do not.

A user thread is more appropriate for low-level tasks, whereas a kernel thread is better for high-priority tasks that should get high priority to system resources.

Q.9.(a) Explain Windows NT architecture. **(10)**

Ans. Windows NT Architecture : Windows NT architecture is illustrated in fig.

Interrupt-Driven Input/Output : The problem with programmed input/output is that the processor has to wait a long time for the input/output module of concern to be ready for either reception or transmission of more data. The processor, while waiting, must repeatedly interrogate the status of the input/output module. As a result, the level of performance of the entire system is degraded.

An alternative approach for this is : for the processor to issue an I/O command to a module and then go on to do some other useful work. The input output module will then interrupt the processor to request service, when it is ready to exchange data with the processor. The processor than executes the data transfer as before and then resumes its format processing.

Fig. shows the use of interrupt. Input/output for reading in a block of data. However, interrupt input/output still consumers a lot of time because every data has to pass with processor.

Vertical Memory Manager : The virtual memory manager has two major duties : to create and manage address maps for process and central physical memory allocation.

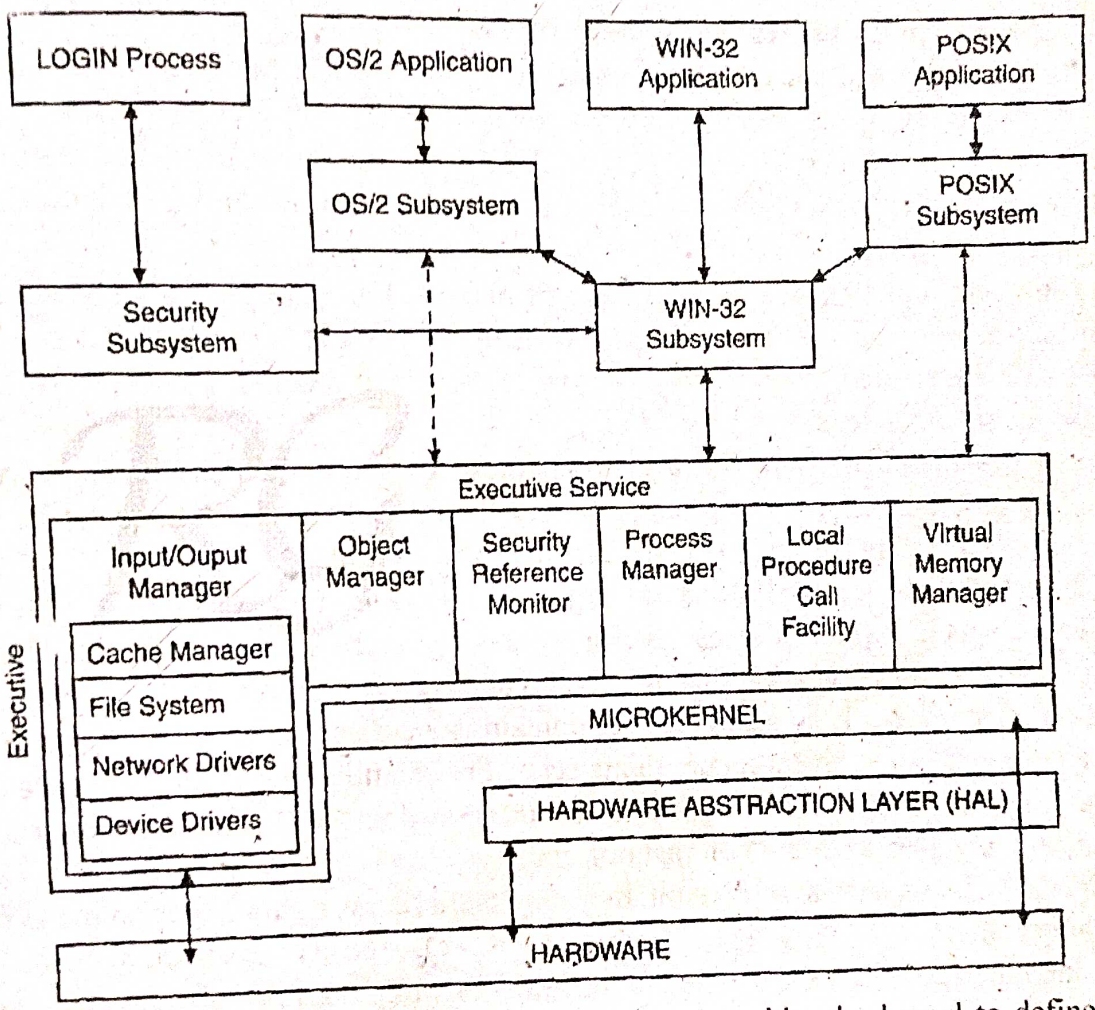
The virtual memory manager implements demand-page virtual memory, which means it manage memory in individual pages. The total memory application requirement can exceed the computer's physical memory on the hard disk in page files. The virtual memory manager transfers data to physical memory from a paging file when an application requests the data .

Security Reference Monitor : The Security Reference Monitor is closely associated with the object manager. The object manager calls the security reference monitor for an access check before letting an application, open an object. The Security Monitor (SRM) implements a security model based on identifiers and Access Control Lists (ACLs).

ACLs specify the action that particular IDs perform on an object. An ACL can contain any number of access control entries (ACEs), including no entries, which contain the information, about the action that IDs can perform. NT references the ACE when a user attempt to open the object.

Cache Manager : The cache manager closely with the virtual memory manager and file system drivers. The cache manager maintain's NT's global file system, cache. When the working set tuner takes memory containing modified file data, away from the cache manager, the I/O manager invokes file systems that manage the moved file to write there data back to the disk.

I/O Manager : The I/O manager is responsible for integrating add on device with NT. The I/O manager supports 64-bit fill offsets and layered device drivers. Using 64-bit offsets lets NT's file system address extremely large file and less disk device drivers address extremely large disk.



Process Manager : The process manager works withe the kernel to define process and thread objects. The process manager wraps the kernel's process object and add to it a process identifies (PID), the access taken, an address map and a handle table.

The process manager exports an interface that lets over executive and subsystem and user mode application manipulates process and threads.

NT's Kernel : NT's kernel operates more closely with hardware than the executive does, and it contain CPU-specific code. NT's thread scheduler called the dispatcher by NT's developers, resides in the kernel.

The NT dispatcher is a preemptive scheduler. The CPU times is divided into slices called quantum.

NT implements most synchronization primitives in kernel.

NT has a rich set of synchronization types including semaphores, events and spin locks.

Hardware Abstraction Layer (HAL) : The HAL is NT's interface to the now CPU. To make the portability feasible NT's developers isolate as much CPU specific code as possible into a separate, dynamically replaceable module, the HAL. The HAL exports a common processor model that masks the difference in various processor clips from NT.

A common difference between motherboards in the same processor family is that some are for multiprocessor systems and other are for uniprocessor system.

Microsoft provides the debug version of NT for device-driver developers.

Q.9.(b) Explain protection domain inUNIX. (10)

Ans. Domain of Production : A computer system can be said that it is a collection of process and objects. Objects are two types : *hardware objects* such as CPU, memory, printers, disks, tape drives etc., and *software objects* such as files, programs, semaphores etc. Objects are essentially abstract data types and each can be accessed only through well-defined and meaningful operations.

The operations depends on the nature of objects. For example : a CPU can only be executed on, memory can be read and written whereas CDROM or DVD ROM can be only read. Files can be created, opened, read, written, closed and deleted. Program can be read, written, executed and deleted.

A process should be allowed to access only those resoures for which it has authorization. The complier should be able to access any arbitrary file but only a well defined subset of files (such a source file, listing file etc.) related to the file to be compiled.

To facilitate the scheme of domain structure, a process operations within a protection domain, which specifies the resources that the process may access. Each domain defines a set of objects and the type of operations that may be invoked on each object. The ability to execute an operation on an object is an acces right. A domain is a collection of access rights, each of which is an ordered pair < object-name, rights-set >. For example, if domain D has the access righ < file F, { read, write}>, then a process executing in domain D can read and write F; it cannot perform any other operation on that objects.

Domains do not need to be disjoint; they may share access rights. For example in fig.(1) we have three domains : D_1, D_2 and D_3. The access righ < O_4, {print}> is shared by both D_2 and D_3, implying that a process executing either of these domains can print object O_4. A process must be executing in domain D_1 to read and write object O_1 while only processes in domain D_3 may execture object O_1.

The assocation between a process and domain may be either static or dynamic. If the set of resources avaiable to a process is fixed throughout thin lifetime then the assocation is static otherwise dynamic. Establishing dynamic protection is more complicated than establishing static protection domain.

If the association is fixed then a mechanism must be available to change the content of domain. A process may execute in two different phases, i.e., it may need read access in one phase and write access in another phase.

If the association is dynamic then a mechanism must be available to allow a process to switch from one domain to another. We may also allow the content of a domain to be changed.

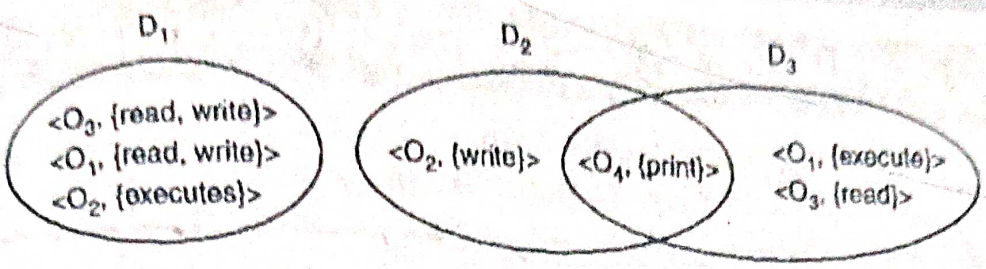


Fig.(1) : System with three protection domains.

A domain can be realized a verietly of ways.

(i) Each *user* may be a domain. In this case, the set of objects that can be accessed depends on the identity of the user. Domain switching occurs when the user is changed generally when one user logs out and other user logs in.

(ii) Each *process* may be a domain. In this case, the set of objects that can be accessed depends on the identity of the process. Domain switching corresponding to one process sending a message to another process, and then waiting for a response.

(iii) Each *procedure* may be a domain. In this case, the sets of objects that can be accessed corresponds to the total variables defined within the procedure. Domain switching occur when a procedure call is made.

Considering the standard dual mode (monitor-user mode) model of operating system execution. When a process executes in monitor mode, it can execute privileged instructions and thus gain complete control of the computer system. While if the process executes in user mode. It can invoke only non-privileged instructions. Consequently, it can execute only within its predefined memory space. These two modes prodect the operating system (executing in moniter domain) from the user processes (executing in user domain). In a multiprogammed operating system, two protection domains are insufficent, since users also want to be protected from one another. Therefore a more elaborate scheme in needed.

PRINCIPLES OF OPERATING SYSTEM

Dec - 2016

Paper Code:-CSE-301-F

Note : *Attempt five questions in all, selecting one question from each Section. Question No. 1 is compulsory. All questions carry equal marks.*

(4×5 = 20)

Q.1. Write a short note on the following :

(a) System programs and system calls.

(b) Logical and Physical address.

(c) Critical Section.

(d) Shell interpreter.

Ans. (a) System Programs : System program provide a more convenient environment for program development and execution. They are divided into several categories.

File Manipulation : These programs create, delete, copy, rename, print, dump, list and generally manipulate files and directories.

Status Information : Status information is the date, time, the amount of memory available or disk space, number of users.

File Modifications : This is to modify the content of files stored on disk or tape.

Programming-Language Support : Compliers, assemblers and interpreters for common programming languages are provided to the user with the operating system.

Program Loading and Execution : After the program is assembled or compiled it has to be loaded into memory to get executed.

Communications : The users can communicate with each other. By sending messages to each other's screens and larger messages as electronic mail etc.

System Calls : System calls provide the interface between a process and the operating system. These calls are generally available as asembly-language instructions, and are usually listed in the manuals used by assembly-language programmers.

System calls may be made directly from higher level language, *e.g.*, C provides system calls directly.

Ans.(b) Logical and Physical Address Space : Compile time and load time address binding generates logical and physical addresses which are identical, whereas address binding generated at execution time, results in differing logical and physical addresses. Logical addresses are generated by CPU whereas the address which refers in main memory obtained form memory address register area called physical address. Set of all logical addresses generated by a program is called logical address space and set of all physical address which corresponds to these logical addresses is called physical address space.[Fig.(1)].

Both logical and physical addresses are contiguous in nature, meaning that each consecutive address refers to a cell. Physical addresses are used in system mode while logical addresses are used in user mode. If the size of physical memory is 4 megabytes (2^{22} bytes) then the physical addresses will run from 0 to $2^{22} - 1$ under the assumption that physical memory is divided into small memory cells of size 1 byte (8 bit).

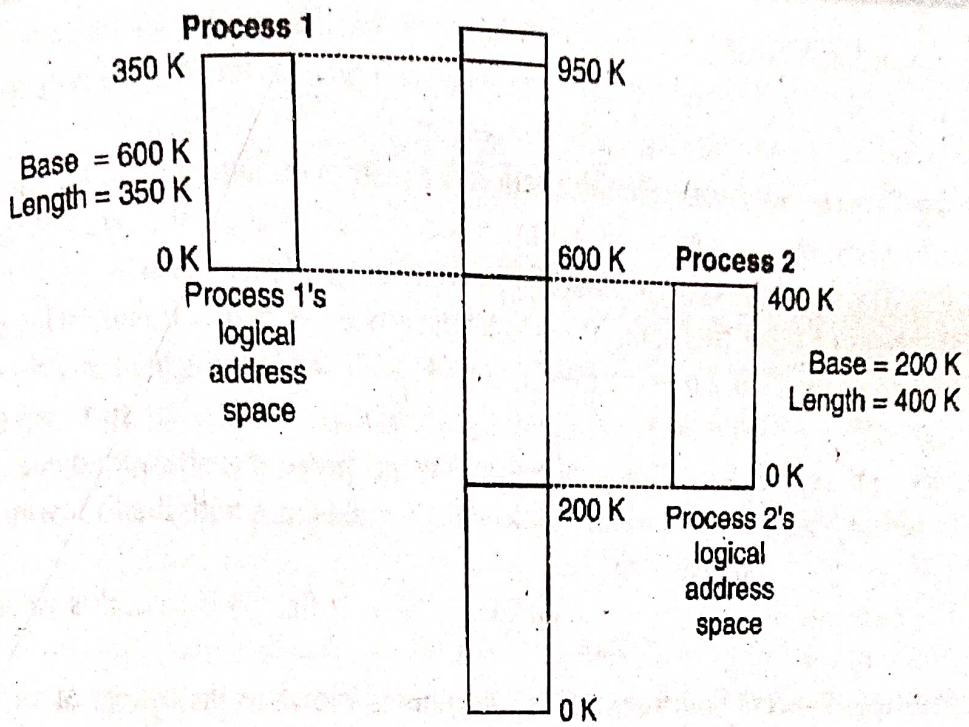


Fig.(1) : Computer's physical address space.

A compute system has only one physical address space, but can have served logical address space [Fig.(1)].

Ans.(c) Critical Section : Consider a system consisting of *n* processes $\{P_0, P_1, ...P_{n-1}\}$. Each process has a segment of code, called a *critical section*, in which the process may be changing common variables, updating a table, writing a file, and so on. The important feature of the system is that, when one process is executing in its critical section, no other process is to be allowed to execute in its critical section. Thus, the execution of critical sections by the processes is mutually exclusive in time. The critical-section problem is to design a protocol that the processes can use to cooperate. Each process must request permission to enter its critical section. The section of code implementing this request is the *entry section*. The critical section may be followed by an *exit section*. The remaining code is the *remainder section*. The general structure of a typical process P_i is as follows :

```
do {
    entry section
        critical section
    exit section
        remainder section
} while (1);
```

A solution to the cirtical-section problem must satisfy the following three requirements :

1. Mutual Exclusion : If process P_i is executing in its critical section, then no other processes can be executing in their critical sections.

2. Progress : If no process is executing in its critical section and some processes wish to enter their critical sections, then only those processes that are not executing in their remainder section can participate in the decision on which will enter its critical section next, and this selection cannot be postponed indefinitely.

3. Bounded Waiting : There exists a bound on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

Two-Process Solutions : In this section, we restrict our attention to algorithms that are applicable to only two processes at a time. The processes are numbered P_0 and P_1. For convenience, when presenting P_i, we use P_j to denote other process; that is, $j == 1 - i$.

Algorithm 1 : Our first approach is to let the processes share a common integer variable turn initialized to 0 (or 1). If turn == i, then process P_i is allowed to execute in its cirtical section.

Algorithm 2 : The problem with algorithm 1 is that it does not retain sufficient information about the state of each process; it remembers only which process is allowed to enter its critical section. To remedy this problem, we can replace the variable turn with the following array :

boolean flag [2];

The elements of the array are initialized to false. If flag [i] is true, this value indicates that P_i is ready to enter thecritical section.

Multiple-Process Solutions : This algorithm is known as the bakery algorithm and it is based on a scheduling algorithm commonly used in bakeries, ice-cream stores, deli counters, motor-vehicle registries, and other locations where order must be made out of chaos. This algorithm was developed for a distributed environment, but at this point we are concerned with only those aspects of the algorithm that pertain to a centralized environment.

Ans.(d) Command-Interpreter System/Shall Interpreter : One of the most important system program for an operating system is the command interpreter, which provides an interface between the user and the operating system. User give commands which are executed by operating system (usually by system calls). The main function of command interpreter system is to get and execute the next command specified by the user.

Command-Interpreter system is usually not a part of the Kernel of an operating system, since multiple command interpreters (shell in UNIX) may be supported by an operating system, and they are in no need to run in Kernel mode.

There are two main advantages to separate the command interpreter system from the Kernel :

(*i*) If we want to change the interface of command interpreter then we are not able to change the code of kernel. So, we cannot modify the interface.

(*ii*) If the command interpreter is a part of kernel then it is possible for a malicious process to gain access to a certain part of kernel which will be harmful for an operating system.

SECTION – A

Q.2.(a) What is Operating System. Explain its types in detail. (10)

Ans. Operating System : The Operating system (OS) is the main software component with which users and applications interact. The OS provides the basis platform that defines how the user can interact with the computer. The operating system has two basis functions. First, it communicates with the PC's hardware; it gives applications access to things like the hard drive, it receives input from the keyboard, and it outputs data to the monitor. The second function is to provide a user interface and interpret commands so a user can communicate with the hardware.

An operating system is also referred to as a command interpreter. Several type of operating systems performs basic functions.

Types of Operating Systems : Within the broad family of operating systems, there are generally four types, categorized based on the types of computers they control and the sort of application they supports. The categories are :

(i) Real Time Operating System (RTOS) : Real time operatings system are used to control machinery, scienctific instruments and industrial system. An RTOS typically has very little user interface capability, and no end-user utilities, since the system will be a "sealed box" when delivered for use. A very important part of an RTOS is managing the resources of the computer so that a particular operation executes in precisely the same amount of time, every time it occurs. In a complex machine, having a part move more quickly just because system resources are available may be just as catastrophic as having it not move at all because the system is busy.

(ii) Singel-User, Single-Task : As the name implies, this operating system is designed to mangage the computer so that one user can effectively do one thing at a time. The palm OS for Palm handheld computers is a good example of a modern single-user, signle-task operating system.

(iii) Single-User, Multi-Tasking : This is the type of operating system most people use on their desktop and laptop computers today. Microsoft's Windows and Apple's Mac OS platforms are both examples of operating system that will let a single user have several programs in operation at the same time. For example, it's entirely possible for a windows user to be writing a note in a word processor while downloading a file from the internet while printing the next of an e-mail message.

(iv) Multi User : A multi user operating system allows many different users to take advantage of the computer's resources simultaneously. The operating system must make sure that the requirements of the various users are balanced, and that each of the program they are using has sufficient and separate resources so that a problem with one user doesn't affect at entire community of users. UNIX, VMS and mainframe operating systems such as MVS, are examples of multi-user operating systems.

Q.2.(b) Explain First come First serve and shortest job first in detail with suitable example. **(10)**

Ans. First-Come, First-Served (FCFS) Scheduling Algorithm : FCFS is the simplest of all scheduling algorithms. The key concpet of this algorithm is "allocated the CPU in the order in which the processes arrive." That is the process that requests the CPU first is allocated the CPU first. The implementation of FCFS policy is managed with a FIFO (First-in-first out) queue. When a process enters the ready queue, its PCB is linked into the tail of the queue. When the CPU is free, it is allocated to the process at the head of the queue. The running process is then removed from the queue. The code for FCFS scheduling is simple to write and understand.

FCFS Scheduling Algorithm is Non-preemptive : Once the CPU has been given to a process, that process keeps the CPU until it releases the CPU either by terminating or by requesting I/O devices. The FCFS algorithm is suitable for Batch operating system. The average waiting time under FCFS policy is often quite long especially if requests of short CPU burst waits

behind the long ones. FCFS is not suitable for time-sharing systems where it is important that each user should get the CPU for an equal amount of arrival time.

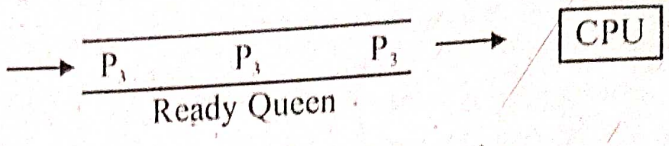


Fig.(1) : FCFS Scheduling

Shortest-job-First (SJF) Scheduling Algorithm : This algorithm associates with each process, the length of the latter's next burst. When the CPU is available, it is assigned to the process that has the smallest next CPU burst. If the two processes have the same length of CPU burst then FCFS scheduling algorithms is followed. As scheduling is done by examining the length of next CPU burst of a process rather than its total length so it is also called shortest next CPU scheduling algorithm.

The key concept of this algorithm is "CPU is allocated to the process with least CPU-burst time". This algorithm is considered to be an optimal algorithm, as it gives the minimum average waiting time as a result.

The disadvantage of this algorithm is the problem to know the length of time for which CPU is needed by a process. A prediction formula may be used to predict the amount of time for which CPU may be required by a process. Let t_n be the length of nth CPU burst and let τ_{n-1} our predicted value for the next CPU burst. Defining a parameter 'α' that controls the relative weight of recent and past history in our prediction such that $0 \leq \alpha \leq 1$ then a formula called exponential average is defined as :

$$\tau_{n-1} = \alpha t_n + (1 - \alpha) \tau_n$$

where t_n contains our most recent information; and τ_n stores the past history in prediction

if $\alpha = 0$, then $\tau_{n-1} = \tau_n \Rightarrow$ recent history has no effect (current conditions are assumed to be transient)

If $\alpha = 1$, then $\tau_{n-1} = t_n \Rightarrow$ in this case only the most recent CPU burst matters.

If $\alpha = 1/2$ then recent and past history are equally weighted.

The SJF algorithm may be either preemptive or non-preemptive : The choice arises when a new process arrives in the ready queue while a previous process is executing.

A preemptive SJF will preempt this currently executing process and starts the execution of newly entered process while non-preemptive SJF will allow the currently executing process to complete its burst time without any interruption in its execution.

Preemptive SJF scheduling is sometimes called Shortest-Remaining-Time-First (SRTF) scheduling.

Q.3.(a) Differentiate between Distributed Operating System and Network Operating System. **(10)**

Ans. Differences between DOS and NOS :

Differentiating factors	DOS	NOS
Definition	An operating system which manages a collection of independent computers and makes them appear to the users of the system as a single computer.	A network operating system (NOS) is a computer operating system that is designed primarily to support work–station, personal computer, and, in some instances, older terminal that are connected on a local area network (LAN).
Autonomy	• Singles system–wide OS and a part of the global OS are run on each computer. • Global set of system calls available on all machines. • A relativley low degree of autonomy	• Each computer in the network has its own local OS–they may be same or different. • Two processes running on different computers communicate through a mutually agreed communication protocol.
Resemblance	It looks like a virtual uniprocessor.	It does not look like a virtual uniprocessor.
System image	• It hides the existence of multiple computers • Same set of system calls are used to access local and remote resources. • Transparency is the key concept.	• User view of computing system is a collection of distinct machines connected by communication subsystem. • Requires special login commands (remote login) and the other commands to use remote resources. • User is aware of all the remote resource one is using.
Fault Tolerance Capability	In distributed operating system, most of the users are normally unaffected by the failed machines and can continue to perform their work normally. Therefore, the fault tolerance capability of a distributed operating system is usually very high as compared to that of a network operating system.	A network operating system provides little or no fault tolerance capability.
Access	Different machines accessed without use knowledge.	User must log into different machines (rlogin, telnet)

Semantics	Well defined semantics for file sharing	Usually no well defined semantics for file sharing
Mode of Communication	Communication via messages	Communication via Shared files.

Q.3.(b) Explain the concept of process in detail. **(10)**

Ans. Process : A process is a program in execuation. The execuation of a process must progress in a sequential fashion.

A process is the smallest unit of work individually schedulable by an operating system. Such a system consists of a collection of processes : operating system processes execute system code, and user processes execute user code. All these processes may execute concurrently. Potential benefits of concurrent execution of programs include increased performance, resource utilization, and responsiveness of a computer system.

Processes are a primary mechanism, for defining and managing concurrent execution of programs under control of an operating system.

The operating system is responsible for the following activites in connection with process and thread management : the *creation* and *deletion* of both user and system processes; and the provision of mechanisms for *communication*, *synchronization* and *deadlock* handling for processes.

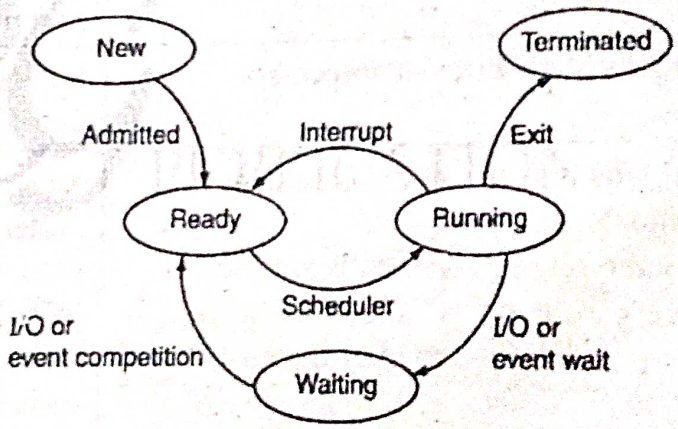


Fig.(1) : Process State Transition Diagram.

Process States : When a process comes in execuation it changes its state.The state of a process is defined in part by the current activity of that process. Each process may be in one of the following state during the time of its execution.

- *New* : In this state, the process is being created
- *Ready* : In the state, the processes is waiting to be assigned to a processor through a short term scheduler.
- *Waiting* : In this state the process is waiting OR blocked for some event to occur such as an input/output completion on reception of a signal.
- *Running* : In this state, instructions are being executed.
- *Terminated* : In the state, the process has finished the execution.

These state names are arbitrary, and they vary across operating systems. However, the states that they represent are found on all systems.

In a uniprocessor system only one process can be in a running state at a single instant of time, although many process may be in ready OR waiting state.

The state transition diagram corresponding to these states is presented is fig.(1).

Operation On a Process : As we know that the process is sequential. The scenario is that there are many processs and a single CPU. So, we use time slicing or multitasking.This is a type of pseudo-parallel processing and is also known as concurrent sequential processing. Please note that in a true parallel system, many CPUs are there, Now, the various operations that can be done on processes are as follows :

(a) Create a process
(b) Destory a process
(c) Run a process
(d) Change a process priority
(e) Get process information
(f) Set process information

Let us discuss now about two main operations :

(a) Process creation.
(b) Process termination.

(a) Process creation : It is the job of our O.S to create process. There are four ways of achieving it. They are as follows-

(i) For a *batch environment*, a process is created in response to the submission of a job.

(ii) In an *interactive environment*, a process is created when a new user attempts to log on.

(iii) The *operating system* can create a process to perform a function on behalf of a user program.

(iv) A number of processes can be generated from the main process. For the purpose of *modularity* or to exploit *parallelism*, a user program can crete a number of processes.

(b) Process termination : There are two types of terminations They are as follows-

(i) Normal Termination
(ii) Forced Termination

(i)Normal Termination : Normal Termination of processes is done by using an exist system call. It returns some output data to its parent process. All the resources allocated to the process are freed by it at this time.

(ii) Forced Termination : Forced Termination of a process is done by its parent process when it exccutes an abort system call. When a parent process terminates the child process is also terminated automatially. This is due several like-

(a) The child process has exceeded allocated resources.
(b) Task assigned to child is no longer required.
(c) Parent is terminating and O.S does not allow a child a to continue. This is known as cascading termination. For example, UNIX supports caseding termination. There are many other reasons for process termination also like expiration of time slot, memory boundary violation. I/O failure etc.

SECTION – B

Q.4. Explain virtual Memory Management. (20)

Ans. Virtual Memory : Virtual memory is a memory hierarchy consisting of a computer system's memory and a disk, that enables a process to operate with only some portions of its address space in memory.

Advantages of Using Virtual Memory : Users Point of View : The details of virtual memory management are generallly transparent to programmers. Being provided with the illustion of a much larger memory than may actually be available, programmers are practically relieved of the burden of trying to fit a program into limited memory. Moreover, the same program may run without reprogramming or recompilation on systems with significantly different capacities of installed memory. Thus, programmers may develop applications that exploit the full benefit of a given computer platform regardless the differenes in installed physical memory among different user configurations.

Virtual memory makes the programming task easier as now the programmer should not worry about the size of the available physical memory fig.(1).

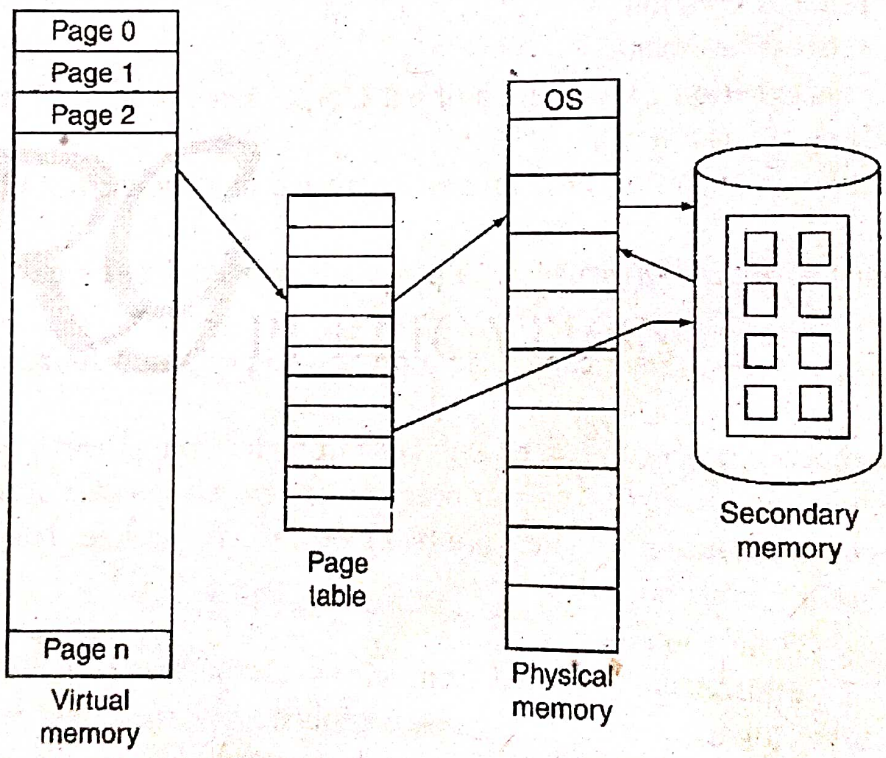


Fig.(1) : Size of virtual memory larger than physical memory.

Advantage of using Virtual Memory : System point of view : The ability to execute a partially loaded process is also advantageous from the operating systems point of views. For example, a process may be loaded into an arbitrary sized space of memory. This may be used to reduce to reduce the external fragmentation without the need to change the scheduled order of process executions. Moreover, the amount of space in use by a given process may be varied during its memory residence. As a result, the operating system may speed up the execution of important processes by allocating them more real memory. Alternatively, by reducing the real-memory holdings of resident processes, the degree of multiprogramming can be increased by using vacated space to activate more processes.

Thus, runnings a program that is not entirely in memory would benefit both the system and the user.

Implementation of Virtual Memory : Virtual memory is implemented by demand paging. It can also be implemented in a segmentation system. Several systems proved paged segmentation where segments are broken into pages but the operating system can implement this scheme with demand paging. Demand segmentation can also be used to provide virtual memory. The segment replacement algorithm are very complex in use because of variable sizes of segments while page replacement algorithm.

Q.5. Explain swapping in detail. **(20)**

Ans. Swapping : Swapping is a mechanism in which a process can be swapped temporarily out of main memory (or move) to secondary storage (disk) and make that memory available to other processes. At some later time, the system swaps back the process from the secondary storage to main memory.

Though performance is usually affected by swapping process but it helps in running multiple and big processes in parallel and that's the reason Swapping is also known as a technique for memory compaction.

A process needs to be in the memory to be executed. A process, however, can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution.

For example, consider a multiprogramming environment with a round-robin CPU-scheduling algorithm. When a quantum expires, the memory manager will start to swap out the process that just finished, and to swap in another process to the memory space that has been freed. In the meantime, the CPU scheduler will allocate a time slice to some other process in memory. When each process finishes its quantum, it will be swapped with another process. Ideally, the memory manager can swap processes fast enough that there are always processes in memory, ready to execute, when the CPU schedular wants to reschedule the CPU. The quantum must also be sufficiently large that reasonable amounts of computing are done between swaps.

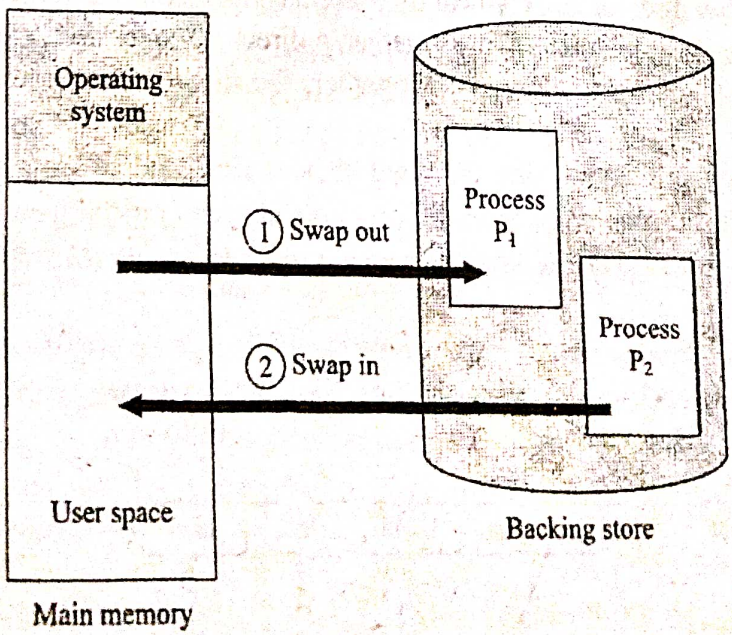


Fig. : Swapping of two processes using a disk as a backing store.

A variant of this swapping policy is used for priority-based scheduling algorithms. If a higher priority process arrives and wants service, the memory manager can swap out the lower-priority process so that it can load and execute higher-priority process. When the higher-priority process finishes, the lower-apriority process can be swapped back in and continued. This variant of swapping sometimes called roll, out roll in.

Normally, a process that is swapped out will be swapped back into the same memory space that it occupied previously. This restriction is dictated by the method of address binding. If binding is done at assembly time or load time, then the process cannot be moved to different locations. If execution-time binding is being used, then it is possible to swap a process into a different memory space, because physical addresses are computed during execution time.

Swapping requires a backing store. The backing store is commonly a fast disk large enough to accommodate copies of all memory images for all users, and must provide direct access to these memory images. The system maintains a ready queue consisting of all processes whose memory-images are on the backing store or in memory and are ready to run. Whenever the CPU scheduler decides to execute a process, it calls the dispatcher. The dispatcher checks to see whether the next process in the queue is in memory. If the process is not, and there is no free memory region, the dispatcher swaps out a process currently in memory and swaps in the desired process. It then reloads registers as normal and transfers control to the selected process.

SECTION – C

Q.6.(a) Explain directory structures in detail. **(10)**

Ans. Directory Structure : An organizational unit, or container, used to organize folders and files into a hierarchical structure. Directories contain bookkeeping information about files that are, figuratively speaking, beneath them in the hierarchy. You can think of a directory as a file cabinet that contains folders that contain files. Many graphical user interfaces use the term folder instead of directory.

In computing, a directory, catalog, or folder, is an entity in a file system which contains a group of files and other directories. A typical file system contains thousands of files, and directories help organize them by keeping related files together. A directory contained inside another directory is called a subdirectory of that directory. Together, the directories form a hierarchy, or tree structure.

The directory is the first one mounted at boot time and it contains the base system necessary to prepare the operation system for multi-user operation. The root directory also contains mount points for other file system that are mounted during the transition to multi-user operation.

Following are some schemes for defining the logical structure of a directory :

Single-level Directory : It is the simplest directory structure. All files are contained in the same directory, which is easy to support and understand [fig.(1)].

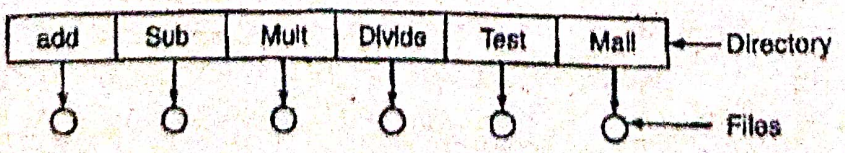


Fig.(1) : Single level directory.

A single-level directory has certain limitations, when the number of files increases or when the system has more than one user. If two users call their data file test, then the unique name rule is violated. So, with multiple users when number of file increases it becomes difficult to remember the names of all files in order to create only files with unique names.

Two-level Directory : The disadvantage of single-level directory is the confusion of file names between different users. The solution for this problem is to create a directory for each user [fig.(2)].

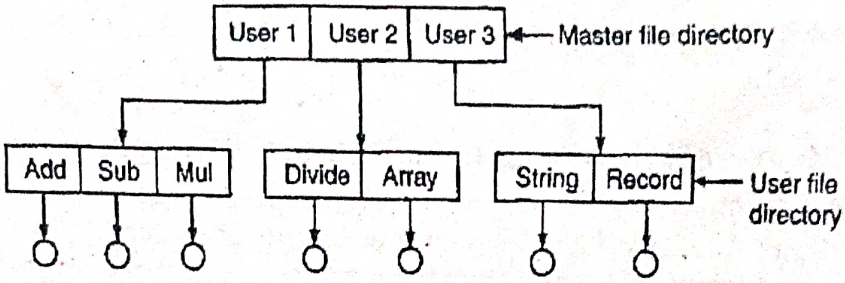


Fig.(2) : Two level directory.

In two-level directory structure each user has his own user file directory. Each user has similar structure but lists only the files of a single user. When user login, the system's master file directory is searched. The master file directory is indexed by user name or account no-for example. user 1, user 2, ..., as in fig.(2) and each entry point to the user directory for that user.

When users refer to a particular file, only their own user file directory is searched. Thus different users may have files with the same name, as long as all file names within each user file directory are unique.

The problem of this directory structure is that, it isolates one user from another. This is a disadvantage when users want to interact on some task or process. Every file in this system has a unique path name.

Tree-Structural Directories : The tree-structure allows user to create their own sub-directories and organize their files accordingly. The tree has a root directory. Every file in the systems has a unique path name. A path is the path from the root through all the sub-directories to a specified file. A directory contains a set of files and or sub-directories.

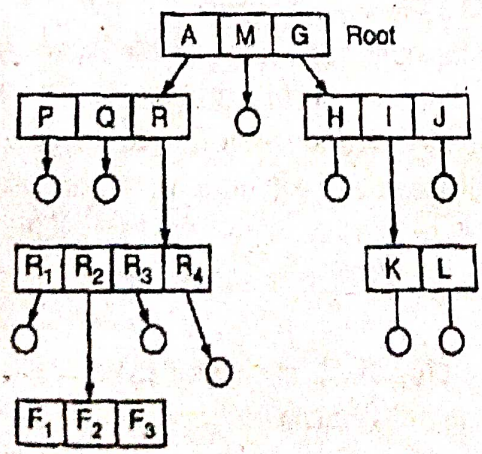


Fig.(3) : Tree structure directory.

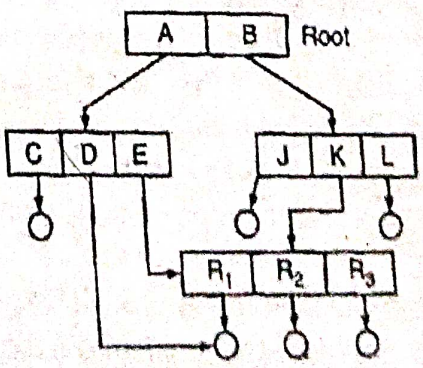


Fig.(4) : Acyclic structure directory.

Acylic graph Directories : A shared directory or file will exist in the file system in two or more places at once. A shared directory or file is not the same as two copies of the file. With a shared file there is only one actual field and any changes made by one person would be immediately visible to the other.

An acyclic graph allows directories to have shared sub-directories and files. The same file or sub directory may be in two or more process exist in the file system at a time. An acyclic graph directory structure is more flexible than a simple structure but it is also more complex. Like a file, we have multiple complex path name. Consequently, distinct file name may refer to the same file. If we are trying to traverse the entire file system (to find or copy all files to backup storage) this problem becomes significant, since we do not want to traverse shared structure more than once.

Q.6.(b) What are the advantages of a Distributed file system over a file system in a centralized system. **(10)**

Ans. Advantages of Distributed system over centrilized systems :

Item	Description
Economics	Microprocessors offer a better price/performance than Mainiframes. 10,000 CPUs executing 50 MIPS yields executing 500,000 MIPS. NOT possible for a single CPU to achieve
Speed	A distributed system may have more total computing power than mainframe
Inherent distribution	Some applications involve spactially separated machines
Reliability	If one machine crashes, the system as whole can Still service
Incremental growth	Computer power can be added in small increments.

Q.7.(a) Explain distributed file system. **(10)**

Ans. Distributed file system : A distributed file system (DFS) is a distributed implemen tion of the classical time-sharing model of a file system, where multiple users share files and storage resources. The purpose of a DFS is to support the same kind of sharing when the files are physically dispersed among the various sites of a distributed system.

A distributed system is a collectio of loosely coupled machines interconnected by a communication network. We use the term machine to denote either a mainframe or a workstation. From the point of view of a specific machine in a distributed system, the rest of the machines and their respective resources are remote, whereas the machine's own resources are referred to as local.

To explain the structure of a DFS, we need to define the terms service, saw, and client. A service is a software entity running on one or more machines and providing a particular type of function to a priori unknown clients. A server is the service software running on a single machine.

A client is a process that can invoke a service using a set of operations that forms its client interface. Sometimes, a lower-level interface is defined for the actual cross-machine interaction, which we refer to as the intermachine interface.

A DFS is a file system whose clients, servers, and storage devices are dispersed among the machines of a distributed system. Accordingly, service activity has to be carried out across the network, and instead of a single centralized data repository, there are multiple and independent storage devices.

As will become evident, the concrete configuration and implementation of a DFS may vary. There are configurations where servers run on dedicated machines, as well as configurations where a machine can be both a server and a client. A DFS can be implemented as part of a distributed operating system, of alternatively by a software layer whose task is to manage the communicational between conventional operating system and file systems. The distinctive features of a DFS are the multiplicity and autonomy of clients and servers.

Ideally, a DFS should look to its clients like a conventional, centralized file system. The multiplicity and dispersion of its servers and storage device should be made transparent. That is, the client interface of a DFS should not distinguish between local and remote files. It is up to the DFS to locate the file and to arrange for the transport of the data. A transparent DFS facilitates use mobility by bringing over the user's environment (that is, home directory) wherever a user log's in.

The most important performance measurement of a DFS is the amount of time needed to satisfy various service requests. In conventional systems, time consists of disk access time and a small amount of CPU processing time.

In a DFS, however, a remote access has the additional overhead attributed the distributed structure. This overhead includes the time needed to deliver the request to a server, as well as the time for getting the response across network back to the client. For each direction, in addition to the actual time of the information, there is the CPU overhead of running the communication protocol software. The performacne of a DFS can be viewed as another dimension of the DFS's transparency.

The fact that a DFS manages a set of dispersed storage devices is the DFS's key distinguishing feature. The overall storage space managed by a DFS is composed of different, and remotely located smaller storage spaces. Usually, there is correspondence between these constituent storage spaces and sets of files.

Q.7.(b) What is semaphore ? Which operation done on semaphore. Explain in detail. **(10)**

Ans. Semaphore : An integer value used for signaling among processes. Only three operations may be performed on a semaphore, all of which are atomic : initialize, decrement, and increment. The decrement operation may result in the blocking of a process, and the increment operation may result in the unblocking of a process. Also known as a counting semaphore or a general semaphore.

A semaphore is a special kind of synchronization data that can be used only through specific synchronization primitives. The primitives are simple operations that can be used to implement both mutual exculsion and control synchronization. It also ensures the lack of race conditions. Fig.(1) shows a semantics of *wait and signal* operation. Whan a process performs

a *wait* operation on a semaphore, the operation checks whether the value of the semaphore > 0. If so, it decrements the value of semaphore and lets the process execution be complted; otherwise it blocks the process on the semaphore. A *signal* operation on the semaphore activates a blocked process on the semaphore, if any, or increments the value of semaphore by 1. Indivisibility of the wait and signal operations is ensured by the programming language or the OS that implements it.

```
operation wait (S)
begin
    if S > 0
        then S = S – 1 ;
        else block the process on S ;
end ;
Operation signal (S)
begin
    if some processes are blocked on S
    then activate one blocked process ;
    else S = S + 1 ;
end ;
```

Fig. (1) : Semantics of wait and signal operations

Process use wait signal operations to sychronize their execuation w.r.t. one another. The initial value of semaphore determines how many process can get past the wait operation. A process that does not get past a wait operation is blocked on the semaphore. This feature avoids busy waits.

SECTION – D

Q.8. Explain Window NT file system. **(10)**

Ans. Windows NT Architecture : Windows NT architecture is illustrated in fig.

Interrupt-Driven Input/Output : The problem with programmed input/output is that the processor has to wait a long time for the input/output module of concern to be ready for either reception or transmission of more data. The processor, while waiting, must repeatedly interrogate the status of the input/output module. As a result, the level of performance of the entire system is degraded.

An alternative approach for this is : for the processor to issue an I/O command to a module and then go on to do some other useful work. The input output module will then interrupt the processor to request service, when it is ready to exchange data with the processor. The processor than executes the data transfer as before and then resumes its format processing.

Fig. shows the use of interrupt. Input/output for reading in a block of data. However, interrupt input/output still consumers a lot of time because every data has to pass with processor.

Vertical Memory Manager : The virtual memory manager has two major duties : to create and manage address maps for process and central physical memory allocation.

The virtual memory manager implements demand-page virtual memory, which means it manage memory in individual pages. The total memory application requirement can exceed the

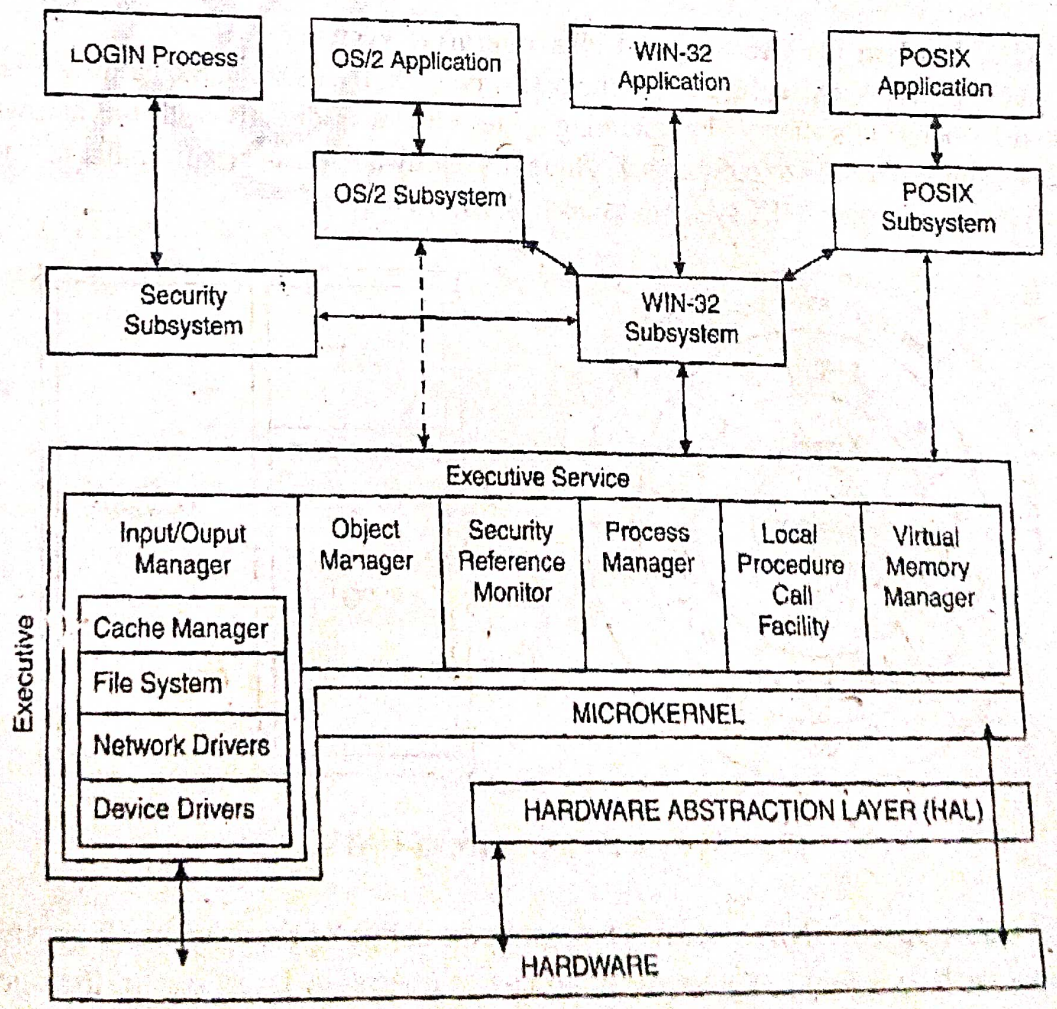
computer's physical memory on the hard disk in page files. The virtual memory manager transfers data to physical memory from a paging file when an application requests the data .

Security Reference Monitor : The Security Reference Monitor is closely associated with the object manager. The object manager calls the security reference monitor for an access check before letting an application, open an object. The Security Monitor (SRM) implements a security model based on identifiers and Access Control Lists (ACLs).

ACLs specify the action that particular IDs perform on an object. An ACL can contain any number of access control entries (ACEs), including no entries, which contain the information, about the action that IDs can perform. NT references the ACE when a user attempt to open the object.

Cache Manager : The cache manager closely with the virtual memory manager and file system drivers. The cache manager maintain's NT's global file system, cache. When the working set tuner takes memory containing modified file data, away from the cache manager, the I/O manager invokes file systems that manage the moved file to write there data back to the disk.

I/O Manager : The I/O manager is responsible for integrating add on device with NT. The I/O manager supports 64-bit fill offsets and layered device drivers. Using 64-bit offsets lets NT's file system address extremely large file and less disk device drivers address extremely large disk.



Process Manager : The process manager works withe the kernel to define process and thread objects. The process manager wraps the kernel's process object and add to it a process identifies (PID), the access taken, an address map and a handle table.

The process manager exports an interface that lets over executive and subsystem and user mode application manipulates process and threads.

NT's Kernel : NT's kernel operates more closely with hardware than the executive does, and it contain CPU-specific code. NT's thread scheduler called the dispatcher by NT's developers, resides in the kernel.

The NT dispatcher is a preemptive scheduler. The CPU times is divided into slices called quantum.

NT implements most synchronization primitives in kernel.

NT has a rich set of synchronization types including semaphores, events and spin locks.

Hardware Abstraction Layer (HAL) : The HAL is NT's interface to the now CPU. To make the portability feasible NT's developers isolate as much CPU specific code as possible into a separate, dynamically replaceable module, the HAL. The HAL exports a common processor model that masks the difference in various processor clips from NT.

A common difference between motherboards in the same processor family is that some are for multiprocessor systems and other are for uniprocessor system.

Microsoft provides the debug version of NT for device-driver developers.

Q.9. Explain architecture of UNIX operating system. (10)

Ans. UNIX Architecture : The interaction between the user and the hardware happens through the operating system. The operating system interacts directly with the hardware. It provides common services to programs and hides the hardware intricacies from them. The high level architecture of the UNIX system has been shown in Fig.

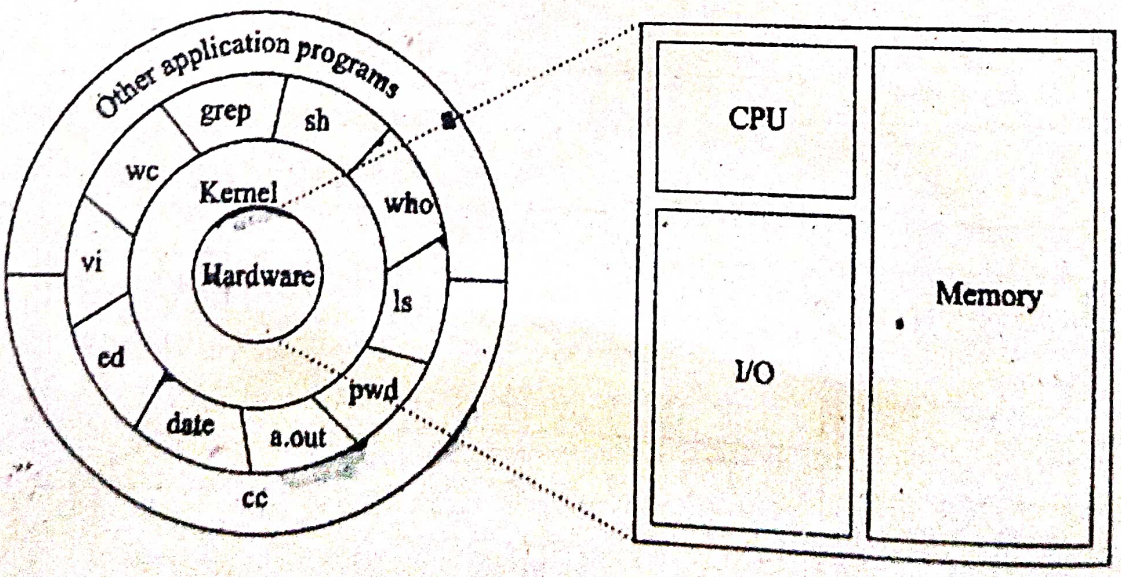


Fig. : System architecture of UNIX.

The hardware of a UNIX system is present in the centre of the diagram. It provides the basic services such as memory management, processor execution level, etc. ot the operating system. The UNIX system seems to be organized as a set of layers. The system Kernal is also

called the operating system. The user programs are independent of the hardware on which they are running. Therefore, it becomes very easy to run programs on UNIX system running on different hardware if the programs do make special assumptions. The programs such as the shell and editors like (ed and vi) interact with the Kernel by invoking a well defined set of system calls. The system calls get various actions done from the Kernel for the calling program. They interchange data between the Kernel and the program. There are many other programs in this layer which from a part of the standard system configurations. These programs are known as commands. But there are several other user created programs present in the same layer. It is shown by the program whose name is a. out. 'a.out 'is the standard name for all the executable files produced by the C compiler. The outer most layers contain other application programs which can be build on top of lower level programs. For instance, the C compiler, cc, appears in the outermost layer of the figure. It invokes a C preprocessor, compiler, assembler and link loader. These are all separate lower level programs. The programming style offered by the UNIX system helps us to fulfill a task by combining the existing programs.

PRINCIPAL OF OPERATING SYSTEM

Dec - 2017

Paper Code:-CSE-301-F

Note : *Attempt five questions in all, selecting one question from each Section. Question No. 1 is compulsory. All questions carry equal marks.*

Q.1.(a) What is system call ? Explain.

Ans. System Calls : User programs communicate with an operating system and request services from it by making system calls. Each *system call* has a *library procedure* that the *user program* calls. This procedure puts the parameters of the system call in registers and then issues a *TRAP* instruction which is a *protected procedure call* to start the operating system. The library procedure *hides* the details of TRAP instructions and makes system calls appear as on ordinary procedure calls.

The *purpose* of a system call is to request the operating system to perform some task. So a system call provides an interface between a process and the operating system. These calls are generally avaiable as assembly language instructions, and they usually listed in the various *manuals* used by the *assembly language* programmers.

Systems call can be grouped roughly into five major categories : process control, file management, device management, information maintenance and communications.

Q.1.(b) What is distributed operating system ?

Ans. Distributed Operating System : Distribution operating system are the operating system for a network of autonomous computers connected by a communication network through a message passing mechanism. A distributed operating system controls and manages the hardware and software resources of a distributed system. When a program is executed on a distributed system, user is not aware of where the program is executed or the location of the resources accessed.

The basic design issue of a distributed OS is same as in traditional operating system, viz. process synchronization, deadlocks, scheduling, file systems, inter-process communication, memory management etc.

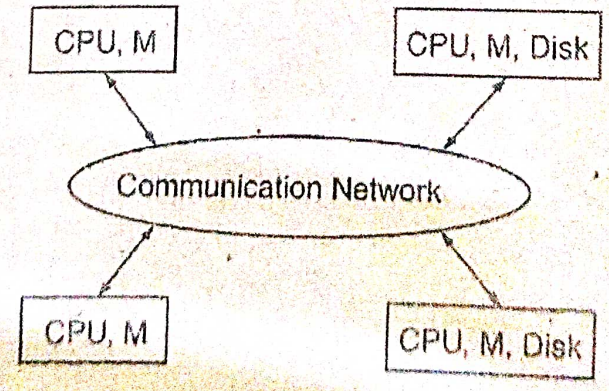


Fig. : Architecture of distributed system

But some additional feature like the absence of both shared memory and global clock,

difficult. For example: Alpha, Kernel, Amoeba, Mach are few examples of distributed operating systems.

Q.1.(c) Differentiate between contiguous and non-contiguous memory allocation.

Ans. Key Differences Between Contiguous and Noncontiguous Memory Allocation :

1. The basic difference between contiguous and noncontiguous memory allocation is that contiguous allocation allocates one single contiguous block of memory to the process whereas, the noncontiguous allocation divides the process into several blocksand place them in the different address space of the memory i.e. in a noncontiguous manner.

2. In contiguous memory allocation, the process is stored in contiguous memory space; so there is no overhead of address translationduring execution. But in noncontiguous memory allocation, there is an overhead of address translation while the process execution, as the process blocks are spread in the memory space.

3. Process stored in contiguous memory executes faster in comparison to process stored in noncontiguous memory space.

4. The solution for contiguous memory allocation is to divide the memory space into the fixed-sized partition and allocate a partition to a single process only. On the other hands, in noncontigous memory allocation, a process is divided into several blocks and each block is placed at different places in memoryaccording to the availability of the memory.

5. In contiguous memory allocation, operating system has to maintain a table which indicates which partition is available for the process and which is occupied by the process. In noncontiguous memory allocation, a table is maintained for each process which indicates the base address of each block of the process placed in the memory space.

Q.1.(d) What is critical section problem ?

Ans. Critical section problem : In an operating system, there may be a segment of code, which will be shared by many processes.but when two processes shares the code and if one updates the code while other reading it,there occurs a conflict.this is called *critical section problem* . To avoid this problem a lock is created on that block of code which establishes that no new process can access that block until the first process release it after it's usage.

Thus there would be no chance conflict between processes while reading that block simultaneously.

List of constraints Dijkstra placed on solutions to the critical-section problem :

(a) Simultaneous execution is equivalent to sequential execution in unknown order.

(b) Speeds of the processes are independent of each other.

(c) A process in noncritical section can't prevent other processes fromentering the critical section.

(c) Selection of admitting one process into critical section can't be indefinitely post-poned.

Q.1.(e) What is polling ?

Ans. Polling : Polling, or polled operation, in computer science, refers to actively sampling the status of an external device by a client program as a synchronous activity. Polling is most often used in terms of input/output (I/O), and is also referred to as polled I/O or software-driven I/O.

Polling has the disadvantage that if there are too many devices to check, the time required to poll them can exceed the time available to service the I/O device. Polling can be described in following steps :

– The host repeatedly reads the busy bit of the controller until it becomes clear.

– When clear, the host writes in the command register and writes a byte into the data-out register.

– The host sets the command-ready bit (set to 1).

– When the controller senses command-ready bit is set, it sets busy bit.

– The controller reads the command register and since write bit is set, it performs necessary I/O operations on the device. If the read bit is set to one instead of write bit, data from device is loaded into data-in register, which is further read by the host.

– The controller clears the command-ready bit once everything is over, it clears error bit to show successful operation and reset busy bit (0).

Section – A

Q.2.(a) What is an Operating System ? Discuss the services provided by an operating system.

Ans. Operating System : The Operating system (OS) is the main software component with which users and applications interact. The OS provides the basis platform that defines how the user can interact with the computer. The operating system has two basis functions. First, it communicates with the PC's hardware; it gives applications access to things like the hard drive, it receives input from the keyboard, and it outputs data to the monitor. The second function is to provide a user interface and interpret commands so a user can communicate with the hardware. An operating system is also referred to as a command interpreter. Several type of operating systems performs basic functions.

Services of operating system : An operating system provides an environment for the execution of programs. The services of operating systems are :

(1) Program Execution : The system must be able to load a program into memory and run it. The program must be able to end its execution, either normally or abnormally.

(2) I/O Operations : A running program may require I/O. This I/O may involve a file or an I/O device. For specific devices, special functions may be desired. For efficiency and protection, users usually cannot control I/O devices directly.

(3) File-System Manipulation : The file-system is of particular interest. It should be obvious that programs need to read and write files. They also need to create and delete and delete files by name.

(4) Communications : Communication is exchange of information between processes. There are two ways in which communication can occur. The first one is communication between processes execution on the same computer. The second place between processes execution on different computer systems.

(5) Error Detection : The operating system constantly needs to be aware of possible errors. Error may occur in the CPU and memory hardware such as memory error a power failure.

(6) Resource Allocating : When there are multiple users or multiple jobs running at the same time, resources must be allocated to each of them. Many different types of resources are managed by the operating system.

(7) Accounting : Accounting is the information about which users use how much and what kind of computer resources.

(8) Protection : The owners of information stored in a multiuser computer system may want to control its use. Protection involves ensuring that all access to system resources is controlled. Security of the system from outsiders is also important.

Q.2.(b) Differentiate between multitasking, multiprogramming and multithreading.

Ans. Multitasking : Multitasking is the ability of an operating system to execute more than one task simultaneously on a single processor machine. Though we say so but in reality no two tasks on a single processor machine can be executed at the same time. Actually CPU switches from one task to the next task so quickly that appears as if all the tasks are executing at the same time. More than one task/program/job/process can reside into the same CPU at one point of time.

Multiprogramming : Multiprogramming is also the ability of an operating system to execute more than one program on a single processor machine. More than one task/program/job/process can reside into the main memory at one point of time. A computer running excel and firefox browser simultaneously is an example of multiprogramming.

Multithreading : Multithreading is the ability of an operating system to execute the different parts of a program called threads at the same time. Threads are the light wait processes which are independent part of a process or program. In multithreading system, more than one threads are executed parallely on a single CPU.

Q.3. What is process scheduling ? Explain different types of schedulers. Also describe the different scheduling algorithm.

Ans. Process scheduling : The assignment of physical processor to processes allows processor to accomplish work. The problem in determining when processors should be assigned and to which processes, is called *processor scheduling* or *CPU scheduling*. When more than process is runable, the operating system must decide which one first. The part of the operating system concerned with this decision is called the scheduler, and algorithm it uses is called the scheduling algorithm.

Types of Schedulers : Genrally, there are three types of schedulers, which may coexist in a complex operating system :

(i) Long term schedular
(ii) Medium term schedular
(iii) Short term scheduler.

Fig.(1) shows the possible traversal path of jobs and programs through the components and queues, in a computer system.

As shown in fig.(1) a submitted batch job joins the batch queue while waiting to be processed by the long term scheduler. Once, scheduled for execution, processes spawned by the batch job, enter ther ready queue to await the processor allocation by the short term scheduler.

After becoming suspended in waiting queue, the running process may be removed from memory and swapped out to secondary stroge. Such processes are subsequently admitted to main by the medium term scheduler in order to be considered for execution by the short term schedular.

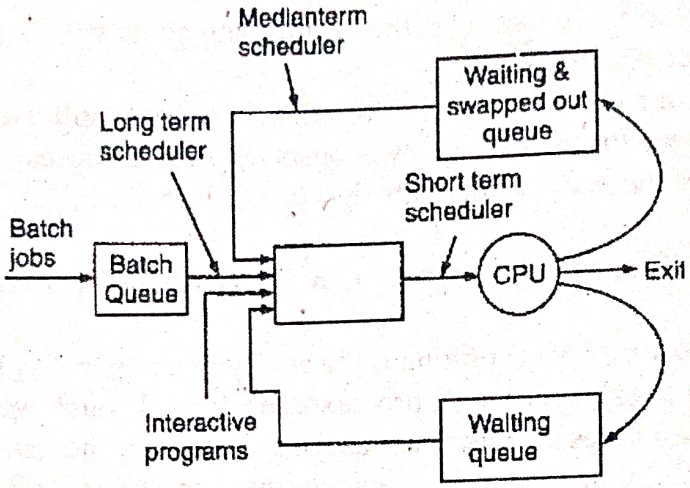


Fig.(1) : Scheduler.

Basically, there are two types of scheduling; pre-emptive or non-pre-emptive scheduling.

(*i*) **Non-pre-emtive scheduling :** If once a process has been given the CPU, and the CPU cannot be taken from that process the scheduling is called non-pre-emptive. In this type of scheduling, the jobs are made to wait by longer jobs, but the treatment of all processes is fairer.

(*ii*) **Pre-emptive :** In pre-emptive scheduling the CPU can be taken away by the allocated process. Pre-emptive scheduling is more useful in high priority, which requires immediate response.

FCFS (First-Cum-First-Serve) : This is the simplest scheduling, the process in this scheduling are served in the order they have arrive. Its implementation is maintained by FIFO (First-In-First-Out) queue. Once the process allocates the CPU, it runs and completes its job and after that releases the CPU.

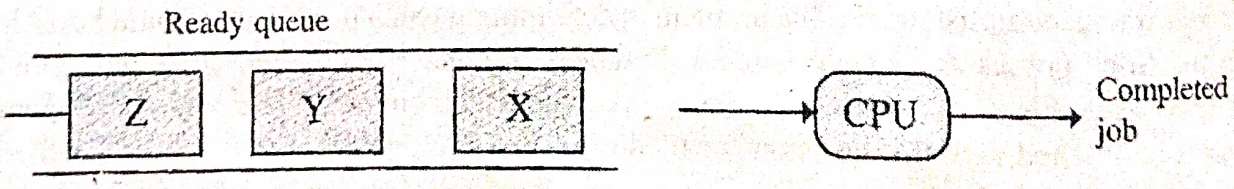


Fig.(2) : First-In-First-Out Scheduling.

The FCFS is non-preemptive scheduling which results in poor performance. In this scheduling there is low rate of CPU utilization and system throughput. The short jobs have to wait for a long time. For example, consider two processes :

Process	CPU Time
P1	30
P2	6

If these processes arrive in the order P1-P2, the turnout time is 30 and 36 units of time respectively thus giving an average time of (30 + 36) / 2 = 33 units of time. Their corresponding

waiting time is 0 and 30 units of time with the average of 15 units of time. However, if the process arrives in reverse order *i.e.* P2-P1, then the turn around times are 6 and 36 units of time respectively giving an average of 21 units of time and the average waiting time is (0 + 6) / 2 = 3.

Shortest-job-First-Scheduling (SJFS) : In shortest job scheduling the scheduling of a job or process is done on the basis it's having shortest execution time. If the two process, have the same execution time then FCFS is used. For example, consider the process given below :

Process	CPU Time
P1	4
P2	10
P3	6
P4	3

According to the shortest job scheduling the processes are served in the order P4-P1-P3-P2.

So, the waiting time is

(0 + 3 + 7 + 13) / 4 = 23 / 4 = 5.75

units of time. Now, lets compare it with the FCFS scheduling, the average waiting time in FCFS is :

(0 + 4 + 14 + 20) / 4 = 38 / 4 = 9.5 units of time.

So, the SJFS scheduling reduces the average waiting time. Shortest job scheduling may be pre-emptive and non-pre-emptive. But in both the case whenever the SJF scheduler is invoked, it searches the ready queue to find the job or the process with the shortest execution time. The difference between the two SJF scheduler lies in the condition that leads to invocation of the scheduler and consequently the frequency of its execution.

Round Robin Scheduling : Round Robin Scheduling is the oldest, simplest and widely used. This scheduling is primarily used in time-sharing and a multi-user system where the primary requirement is the good response time. In this scheduling algorithm, the CPU time is divided into time slices. Each process is allocated a small time slice while it is running. No process can run for more than one time slice when there are others waiting in the ready queue. If the process requires more CPU time then it should go in the end of the ready queue.

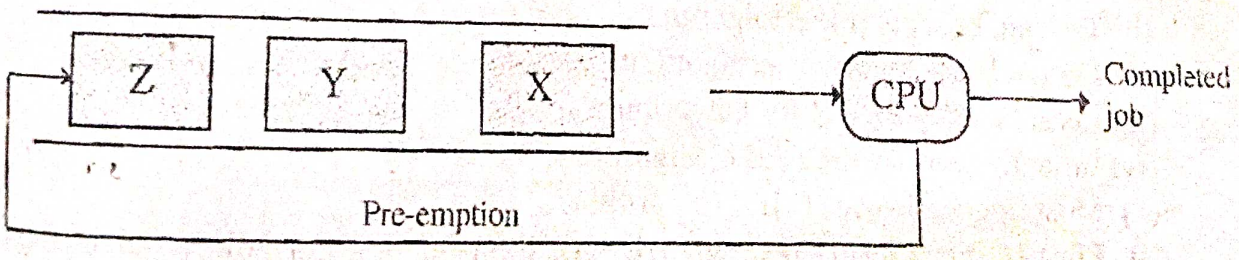


Fig.(3) : Round Robin Scheduling.

In the Round Robin Scheduling small process may be executed in single time slice giving good response time whereas the long processes may require several time slices and thus be forced to pass through ready queue a few times before completion. But in this scheduling the system resources are utilizing in an equitable manner. For example, consider the three processes waiting in the ready queue :

Process	CPU Time
P1	20
P2	5
P3	10

If we use a time slice of 5 units of time, then P1 gets first 5 units of time and the CPU is given to the second process *i.e.* P2. Since P2 needs just 5 units of time, it terminates, as time slicer is expired. The CPU is given to the process P3. Once each process has received one time slicer. Ther CPU is returned to P1 for additional time slicer. Thus, the Round Robin Scheduling is :

P1	P2	P3	P1	P3	P1	P1
0	5	10	20	25	30	35

For implementing a Round Robin Scheduling we require a dedicated timer. The timer is usually set to interrupt the operating system whenever a time slicer expires and thus forces the scheduler to be involved. Processing the interrupt to switch the CPU to another process requires saving all the registers for the old process and then loading the registers for the new process. This task is known as context switching.

Section – B

Q.4. What do you mean by page replacement ? Explain the page replacement algorithm.

Ans. Page Replacement Algorithms : When the processor needs to execute a page, and if that page is not available in main memory then this sitution is called "page fault". For bringing in the required page into main memory, if the space is not available in memory then we need to remove the page from the main memory for allocating the space to the new page which needs to be executed. In this process the page which we need to remove from the main memory is called a "victim page". For the selection of victim page in main memory different algorithms are available called page replacement algorithms. Some of the widely using page replacement algorithms are as follows :

(i) First-in, First out (FIFO) algorithm
(ii) Optimal page replacement algorithm
(iii) Least recently used (LRU) algorithm
(iv) Least frequenctly used (LFU) algorithm
(v) Most frequently used (MFU) algorithm

(i) First-in, First out (FIFO) algorithm : First-in, first out policy treates the page frams allocated to a process as a circular buffer and pages are removed in round robin fashion, that is first-in, first-out (first inputed page will remove first) from. It is a very simple algorithm.

Here to replace the page that has been in memory longest. It is shown in Table.(1). Regions of programs or data that are heavily used throughout the life of a program. In this technique number of page faults are more when compared with other algorithm, In table.(1) 'yes' indicates that page fault has occurred and replaced page is shown in box in that column with respect to the concerned frame number.

Table.(1) : FIFO System

Address / Frame	3	1	2	3	4	2	3	0	3	1	3
0	3	3	3	3	4	4	4	4	4	1	1
1		1	1	1	1	1	3	3	3	3	3
2			2	2	2	2	2	0	0	0	0
Page fault	Yes	Yes	Yes	No	Yes	No	Yes	Yes	No	Yes	No

(ii) Optimal page replacement algorithm : Optimal policy selects the page for replacement when the page fault is to occur for which the time to the next reference is the longest. It is shown in Table.(2).

Advantages : (a) The number of page faults are few, when compared with the other algorithms

(b) It reduces the total page replacement timings.

Disadvantages : Opitmal policy is impossible to implacement, because it requires the operating system should have perfect knowledge of future events.

Table.(2) : Optimal policy

Address / Frame	3	1	2	3	4	2	3	0	3	1	3
0	3	3	3	3	3	3	3	3	3	3	3
1		1	1	1	4	4	4	0	0	1	1
2			2	2	2	2	2	2	2	2	2
Page fault	Yes	Yes	Yes	No	Yes	No	No	Yes	No	Yes	No

(iii) Least recently used (LRU) algorithm : According to the least recently used policy, it replaces the page in memory that has not been referenced for the longest time. It is shown in Table.(3) for the address stream of pages 3 1 2 3 4 2 3 0 3 1 3

Table.(3) : LUR policy

Address / Frame	3	1	2	3	4	2	3	0	3	1	3
0	3	3	3	3	3	3	3	3	3	3	3
1		1	1	1	4	4	4	0	0	0	0
2			2	2	2	2	2	2	2	1	1
Page fault	Yes	Yes	Yes	No	Yes	No	No	Yes	No	Yes	No

Avtantages : It is beneficial for the pages which are not refer after certain period, i.e., after removal from main memory.

Disadvantages : (a) It is complicated to implement, becuase it needs to maintain the time of its last reference for all the pages which are there in main memory.

(b) It needs to maintain a stack of references.

(iv) Least frequenctly used (LFU) algorithm : The least frequently used algorithm selects a page for replacement, with least frequency of reference upto the situtation of page fault occurrence. For this maintain the counter with each page frame. Increase the counter when the page is to refer. Whenever the page fault is to occur, select the page for replacement which is having the least count. Table(4) is showing the LEU policy with address stream 3 1 2 3 4 2 3 0 3 1 3

Table.(4) : LRU policy

Address / Frame	3	1	2	3	4	2	3	0	3	1	3
0	3	3	3	3	3	3	3	3	3	3	3
1		1	1	1	4	4	4	0	0	1	1
2			2	2	2	2	2	2	2	2	2
Page fault	Yes	Yes	Yes	No	Yes	No	No	Yes	No	Yes	No

(v) Most frequently used (MFU) algorithm : Most frequently used and least frequently used algorithms are the counting algorithms. The most frequently used algorithm selects a page for replacement, with most frequency of reference. Like least frequently used for this also maintain the counter with each page frame. Increase the counter when the page is to refer. Whenever the page fault is to occur, replace the page with highest count and reset the count. It is shown in Table.(5).

Table.(5) : MFU policy

Address / Frame	3	1	2	3	4	2	3	0	3	1	3
0	3	3	3	3	4	4	4	0	0	0	0
1		1	1	1	1	1	1	1	1	1	1
2			2	2	2	2	3	3	3	3	3
Page fault	Yes	Yes	Yes	No	Yes	No	Yes	Yes	No	No	No

Q.5.(a) Discuss Logical and Physical address space.

Ans. Logical and Physical Address Space : Compile time and load time address binding generates logical and physical addresses which are identical, whereas address binding

generated at execution time, results in differing logical and physical addresses. Logical addresses are generated by CPU whereas the address which refers in main memory obtained form memory address register area called physical address. Set of all logical addresses generated by a program is called logical address space and set of all physical address which corresponds to these logical addresses is called physical address space [Fig.(1)].

Both logical and physical addresses are contiguous in nature, meaning that each consecutive address refers to a cell. Physical addresses are used in system mode while logical addresses are used in user mode. If the size of physical memory is 4 megabytes (2^{22} bytes) then the physical addresses will run from 0 to $2^{22} - 1$ under the assumption that physical memory is divided into small memory cells of size 1 byte (8 bit).

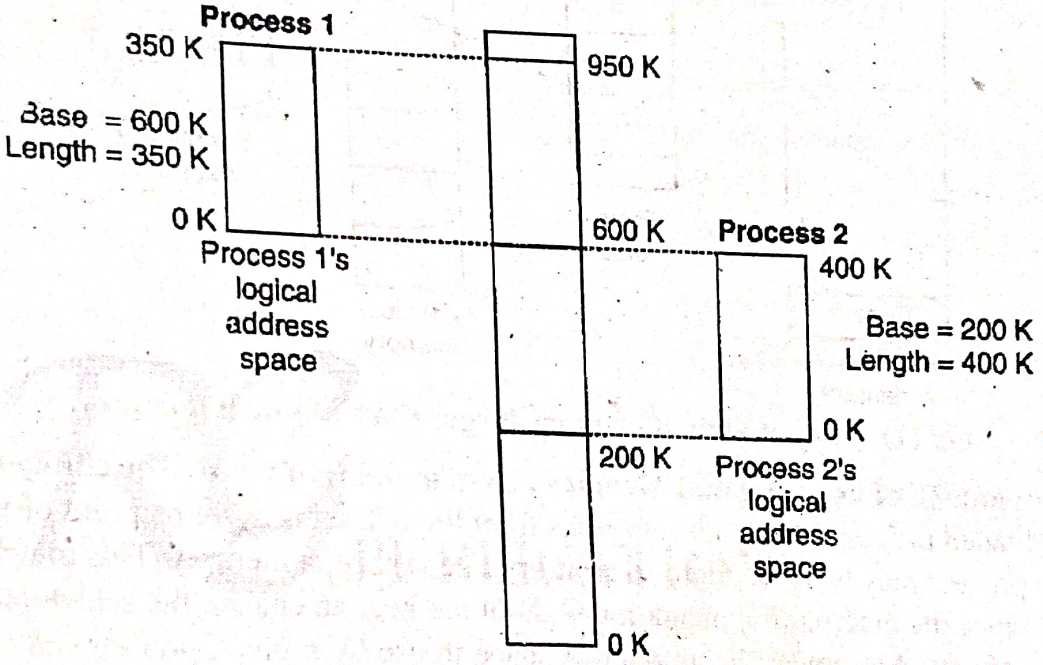


Fig.(1) : Computer's physical address space.

A compute system has only one physical address space, but can have served logical address space [Fig.(1)].

Q.5.(b) Explain the concept of virtual memory.

Ans. Definition of Virtual Memory : Virtual memory is a memory hierarchy consisting of a computer system's memory and a disk, that enables a process to operate with only some portions of its address space in memory.

Advantages of Using Virtual Memory : Users Point of View : The details of virtual memory management are generallly transparent to programmers. Being provided with the illustion of a much larger memory than may actually be available, programmers are practically relieved of the burden of trying to fit a program into limited memory. Moreover, the same program may run without reprogramming or recompilation on systems with significantly different capacities of installed memory. Thus, programmers may develop applications that exploit the full benefit of a given computer platform regardless the differenes in installed physical memory among different user configurations.

Virtual memory makes the programming task easier as now the programmer should not worry about the size of the available physical memory fig.(1).

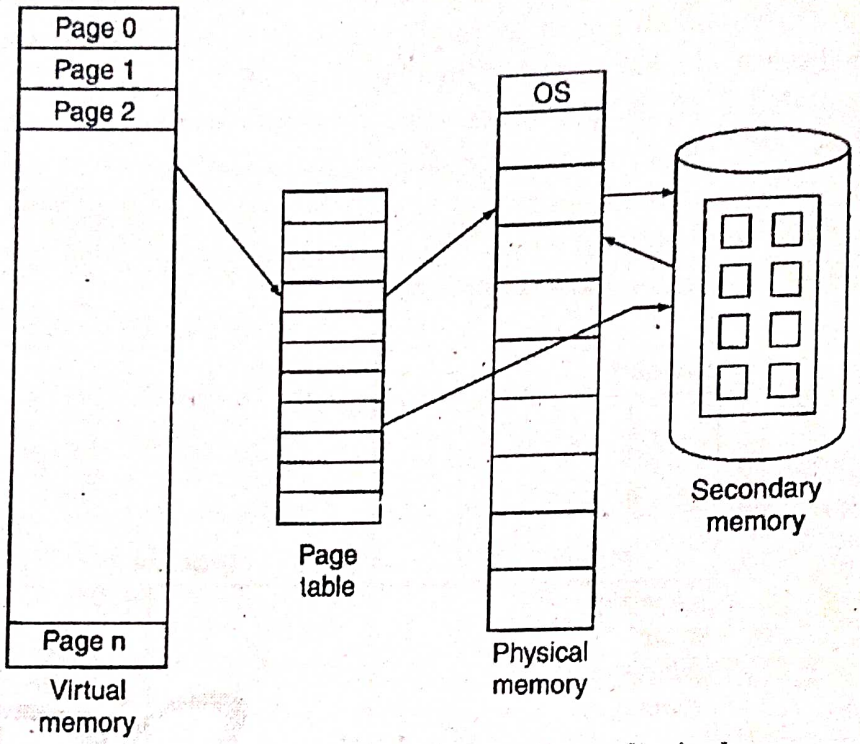


Fig.(1) : Size of virtual memory larger than physical memory.

Advantage of using Virtual Memory : System point of view : The ability to execute a partially loaded process is also advantageous from the operating systems point of views. For example, a process may be loaded into an arbitrary sized space of memory. This may be used to reduce to reduce the external fragmentation without the need to change the scheduled order of process executions. Moreover, the amount of space in use by a given process may be varied during its memory residence. As a result, the operating system may speed up the execution of important processes by allocating them more real memory. Alternatively, by reducing the real-memory holdings of resident processes, the degree of multiprogramming can be increased by using vacated space to activate more processes.

Thus, runnings a program that is not entirely in memory would benefit both the system and the user.

Section – C

Q.6. What do you understand by file system structure ? Describe the various type of allocation method.

Ans. File system structure : A file system is a subsystem of an operating system that performs file management activities such as organiszation, storing, retrieval, naming, sharing and protection of files. It is designed to allow programs to use a set of operations that characterize the file abstraction and free the programmers from conerns about and details of space allocation and layout of the secondary storage device. Therefore, a file system provides an abstraction of a storage device ; that is, it a conventent mechanism for storing and retrieving information from the storage device.

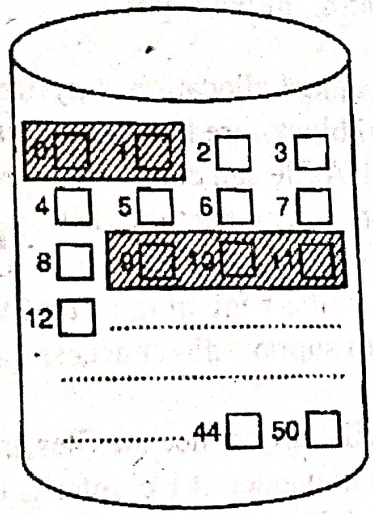
The major methods of allocating disk space are in widely used :

(1) Contiguous allocation
(2) Linked allocation
(3) Indexed allocation

Contiguous Allocation : The contiguous allocation method requires each file to occupy a set of contiguous addresses on disk. Disk addresses define a linear ordering on the disk. With this ordering, assuming that only one job is accessing the disk, accessing block $b + 1$ after block 'b' normally requires no head movement. When head movement is needed, it is only one track.

Contiguous allocation of a file is defined by the disk address of the first block and its length. If the file is 'n'-blocks long and starts at location 'b' then it occupies blocks $b, b + 1, b + 2, \ldots b - n - 1$. The directory entry for each block indicates the address of the starting block and the length of the area allocated for this file.

Accessing a file which has been contiguously allocated is easy. For sequential access, file system remembers the disk address of the last block and when required, reads the next block. For direct access to block 'i' of a file that starts at block 'b', we can immediately access block $b - i$. Thus, both sequential and direct access can be supported by contiguous allocation.



Directory

File	Start	Length
A	0	2
B	9	3
C	17	4
D	31	5

Fig.(1) : Contiguous allocation of disk space.

The difficulty with contiguous allocation is finding space for a new file. The implementation of free space management determines how this task is accomplished. External fragmentation is an other disadvantage of contiguous allocation.

Linked Allocation : Linked association solves all problems of contiguous allocation. In this allocation, each file is a linked list of disk blocks which may be scattered at any place on the disk. The directory contains a pointer to the first block of the file. For example, a file of 5 blocks may start at block 9, continue at block 16 then block 1, block 10 and finally at block 13 [fig.(2)]. Each block contains a pointer to the next block. These pointers are not made available to the user. Thus, if each block of 512 bytes, and a disk address requires 4 bytes, then user sees blocks of 508 bytes.

Creating a new file is an easy task in this allocation scheme. For this we create a new entry in the device directory with linked allocation. Each directory entry has a pointer to the first disk block of the file. This pointer is initialized with a Null value to signify an empty file. A write

to a file remians the first free block from the free space list and write to it. This new block is then linked to the end of the file. To read a file, we simply read blocks by following the pointers from block to block.

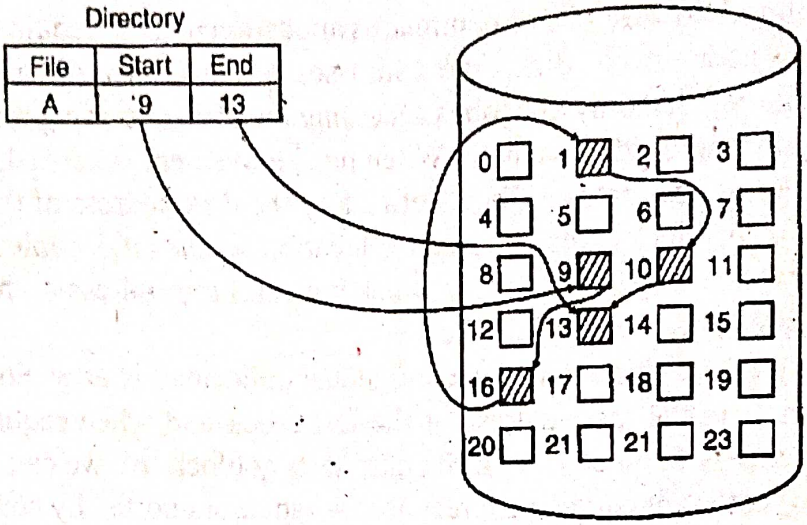


Fig.(2) : Linked allocation of disk space.

There is no external fragmentation with linked allocation. Any free block on the free space list can be used to satisfy a request since all blocks are linked together. Also, there is no need to declare the size of a file when it is created. A file can continue to grow as long as there are free blocks. Consequently, it is never necessary to compact the disk space.

The major problem with linked allocation is that it can be used only for sequential-access files. To find the ith block of a file, we must start at the beginning of that file, and follow the pointers until we get the ith block. It is inefficient to support direct access capability for linked allocation files.

Another problem of linked association is reliability. Since the files are linke together by pointers scattere all over the disk. Consider what will happen if a pointer is lost or damaged. A bug in the OS software or a disk hardware failure might result in picking up the wrong pointer. This error could result in linking into the free-space list or into another file.

Indexed Allocation : Linked allocation solves the problem of external fragmentation for contiguous allocation. Indexed allocation solves the problems arrived in linked allocation.

In Indexed allocation, all pointers are brought together into one location called index block.

Each file has its own index block which is an array of disk-block addresses. The ith entry in the disk block points the ith block of the file. The directory contains the address of the index block [fig.(3)]. To read the ith block, we use the pointer in the ith index block entry to find and read the desired block.

Index allocation supports direct access without suffering from external fragmentation, because any free block on the disk may satisfy a request for more space.

Indexed allocation does suffer from wasted space. The pointer overhead of the index block is greater than the pointer overhead of liked allocation. Considering a common case in which we have a file of only one or two blocks. With linked allocation, we lose the space of only

one pointer per block. With indexed allocation, an entire index block must be allocated, even iff one or two pointers will be non-varnil.

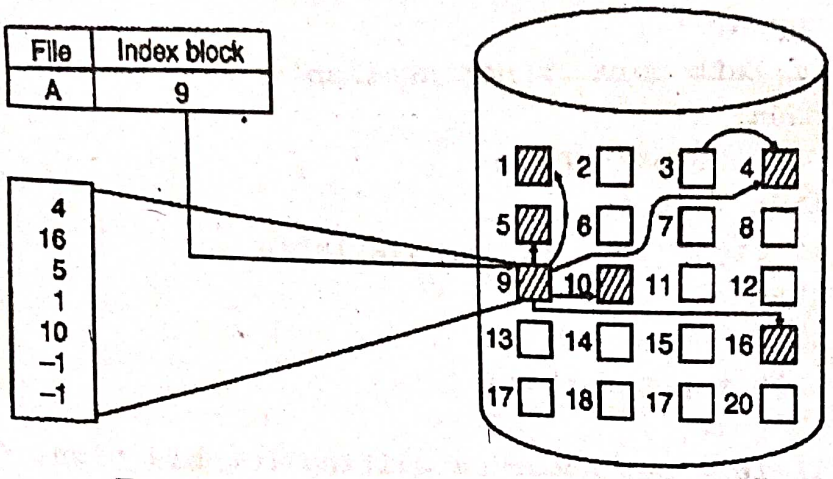


Fig.(3) : Indexed allocation of disk space.

Q.7.(a) What is a semaphore ? Implement the dining-Philosophers problem using semaphore.

Ans. Semaphore : An integer value used for signaling among processes. Only three operations may be performed on a semaphore, all of which are atomic : initialize, decrement, and increment. The decrement operation may result in the blocking of a process, and the increment operation may result in the unblocking of a process. Also known as a counting semaphore or a general semaphore.

Dining Philosophers Problem : Five philosophers are sitting around a table. A plate is kept in front of each philosopher and a fork is placed between each pair of philosopher(fig.(a)). To eat a philosopher must pick up the two forks placed between him, but here we are having 5 forks in total. So, the problem here is to design the processes in such a manner that each philosopher can be able to eat with a pair of forks under the condition that a hungry philosopher should not face indefinite waits when he decides to eat.

Here, the challenge is, to design a solution that does not suffer from deadlocks or starvation considering the outline of a philosopher process P_i shown in fig.(b), where details of process synchronization have been omitted. This solution prone to deadlock because if all philosophers simultaneously pick their left fork none will be able to lift the right fork.

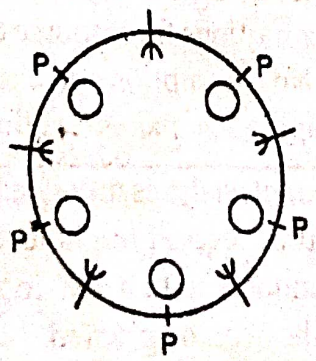


Fig. : (a)

It also contains race-conditions because neighbours might fight over a shared fork. We can avoid deadlocks by modifying the philosopher process, so that if the right fork is not available, the philosopher would defer to his left neighbour by putting down the left fork and repeating the

attempt to take the forks after some time later. However, this approach suffers from starvation as some situation may recover.

```
repeat
if left fork is not available
then
      block (Pi) ;
lift left fork ;
if right fork is not available
then
      block (Pi) ;
lift right fork ;
   {eat} ;
if left neighbour is waiting for his right fork
then activate (left neighbour) ;
if right neighbour is waiting for his left fork
then
   activate (right neighbour);
   {think} ;
   forever ;
```

Fig.(b) : Outline of philosopher process (P*i*)

Q.7.(b) What is deadlock ? How it is handled ?

Ans. Deadlock : A deadlock is a situation wherein two transactions wait for each other to give up their respective locks.

Handling deadlocks : Table describes three fundamental approaches used in deadlock handling. Each approach has different implications for user processes and for the operating system.

Table : Deadlock Handling Approaches

Approach	Description
Deadlock Prevention	The kernel uses a resource allocation policy that ensures that the four conditions for resource deadlock (Mutual Exclusion, Hold-and-wait, No preemption and circular-wait) do not arise simultaneously. This approach makes deadlock impossible.
Deadlock avoidance	The kernel analyzes the allocation state to determine whether granting a resource request might lead to a deadlock later on. Only requests that cannot lead to a deadlock are granted, others are kept pending untill they can be granted.
Deadlock detection and resolution.	The kernel analyzes the resource state to check whether a deadlock exists. If so, it aborts some process (es) and allocates the resources held by them to other processes so that the deadlock classes to exist.

To ensure that deadlocks never occur, the system can use either a *deadlock prevention* or *deadlock avoidance scheme*.

Deadlock avoidance requires that the operating system be given in advance additional information concerning which resources, a process will request and use during its lifetime. With this additional knowledge, we can decide for each request whether or not the process should wait. To decide whether the current request can be satisfied or delayed, the system must consider the resources currently available, the resources currently allocated to each process, and the future request and releases of each process.

If a system does not employ either a *deadlock prevention* or a *deadlock avoidance* algorithm, then a deadlock situation may occur. In this environment, the sysem can provide an algorithm that examines the state of the system to determine whether a deadlock has occurred, and and algorithm to resolve or recover from the deadlock.

If a system does not ensure that a deadlock will never occur and also does not provide a mechanism for *deadlock detection* and *resolution*, then we may arrive at a situation where the system is in a deadlock state yet has no way of recognizing what has happened. In this case, the undetected deadlock will result in the deterioration of the system performance, because resources are being allocated by the processes that cannot run and because more and more processes, as they make requests for resources, enter a deadlock state. Eventually, the system will stop functioning and will need to be restarted manually.

Section – D

Q.8.(a) Explain how to transforming I/O request to hardware operation.

Ans. Typical life cycle of a blocking read request, as depicted in fig. The fig. suggests that an I/O operation requires a great many steps that together consume a tremendous number of CPU cycles.

(i) A process issues a blocking read ( ) system call to a file descriptor of a file that has been opened previously.

(ii) The system-call code in the kernel checks the parameters for correctness. In the case of input, if the data are already available in the buffer cache, the data are returned to the process, and the I/O request is completed.

(iii) Otherwise, a physical I/O must be performed. The process is removed from the run queue and is placed on the wait queue for the device, and the I/O request is scheduled. Eventually, the I/O subsystem sends the request to the device driver. Depending on the operating system, the request is sent via a subroutine call or an in-kernel message.

(iv) The device driver allocates kernel space to receive the data and schedules the I/O. Eventually, the driver sends commands to the device controller by writing into the device-control registers.

(v) The device controller operates the device hardware to perform the data transfer.

(vi) The driver may poll for status and data, or it may have set up a DMA transfer into kernel memory. We assume that the transfer is managed by a DMA controller, which generates an interrupt when the transfer completes.

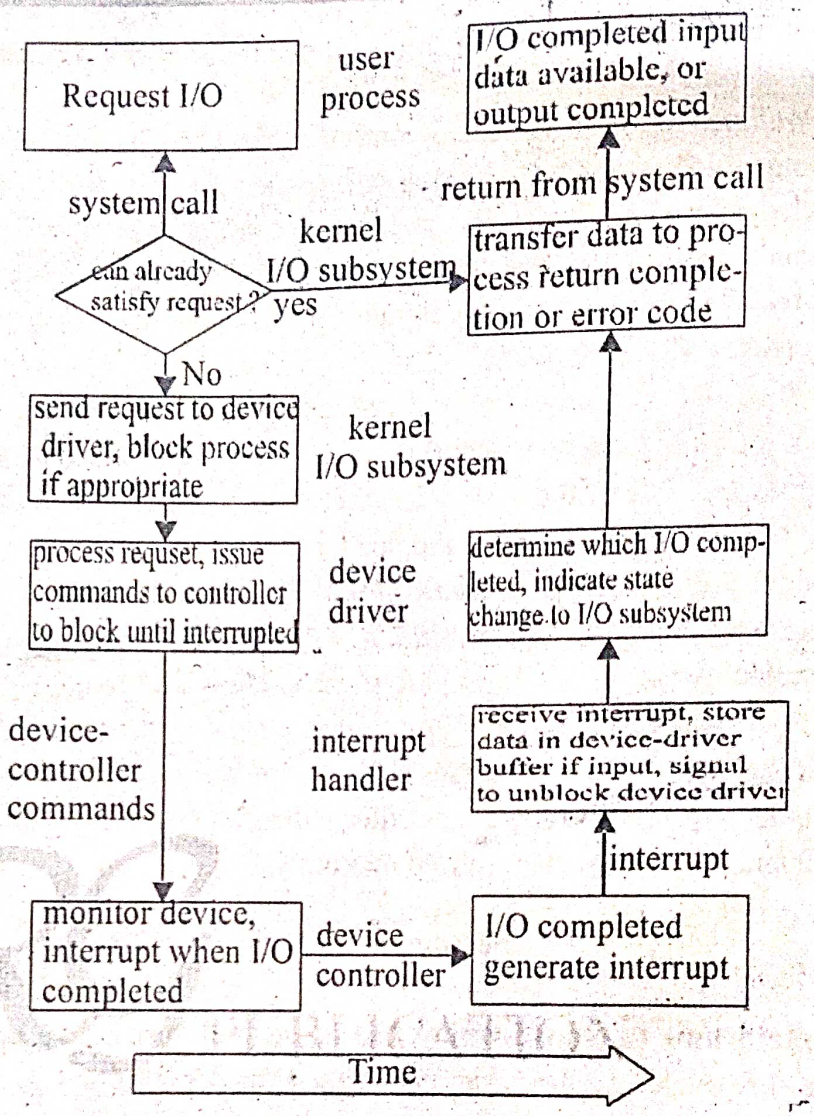


Fig. : The life cycle of an I/O request

(vii) The correct interrupt handler receives the interrupt via the interrupt vector table, stores any necessary data, signals the device drive, and returns from the interrupt.

(viii) The device driver receives the signal, determines which I/O request has completed, determines the request's status, and signals the kernel I/O subsystem that the request has been completed.

(ix) The kernel transfers data or return codes to the address space of the requesting process and moves the process from the wait queue back to the ready queue.

(xi) Moving the process to the ready queue unblocks the process. When the scheduler assigns the process to the CPU, the process resumes execution at the completion of the system call.

Q.8.(b) What are various layers of I/O software ? Explain function of each layer.

Ans. I/O software is often organized in four layers, as shown in Fig.(1).

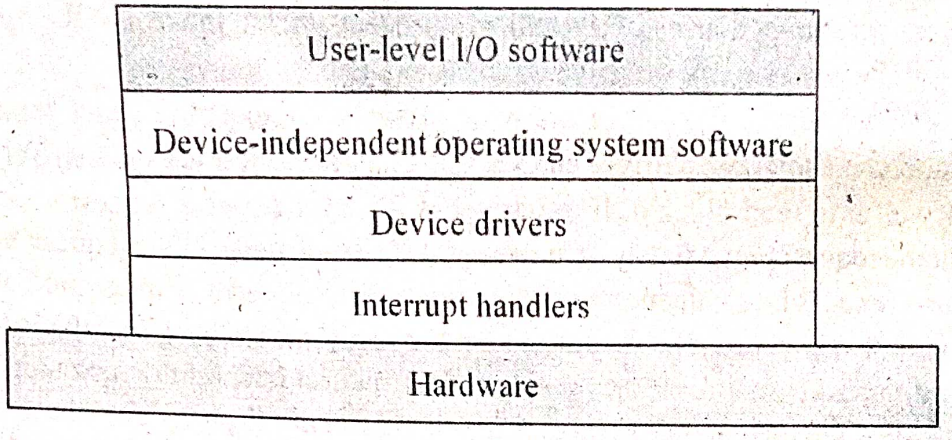


Fig.(1) : Layers of the I/O software system.

Interrupt Handlers : Interrupts are an unpleasant fact of life; although they cannot be avoided, they should be hidden away, deep in the bowels of the operating system, so that as little of the operating system as possible knows about them. The best way to hide them is to have the driver starting an I/O operation block until the I/O has completed and the interrupt occurs. The driver can block itself by doing a down on a semaphore, a wait on a condition variable, a receive on a message, or something similar, for example.

When the interrupt happens, the interrupt procedure does whatever it has to in order to handle the interrupt. Then it can unblock the driver that started it. In some cases it will just complete up on a semaphore. In others it will do a signal on a condition variable in a monitor. In still others, it will send a message to the blocked driver. In all cases the net effect of the interrupt will be that a driver that was previously blocked will now be able to run. This model works best if drivers are structured as independent processes, with their own states, stacks, and program counters.

Device Drivers : Each I/O device attached to a computer needs some device-specific code for controlling it. This code, called the device driver, is generally written by the device's manufacturer and delivered along with the device on a CD-ROM. Since each operating system needs its own drivers, device manufacturers commonly supply drivers for several popular operating systems.

Each device driver normally handles one device type, or one class of closely related devices. For example, it would probably be a good idea to have a single mouse driver, even if the system supports several different brands of mice. As another example, a disk driver can usually handle multiple disks of different sizes and different speeds, and perhaps a CD-ROM as well. On the other hand, a mouse and a disk are so different that different drivers are necessary.

In order to access the device's hardware, meaning the controller's registers, the device driver traditionally has been part of the system kernel. This approach gives the best performance and the worst reliability since a bug in any device driver can crash the entire system. MINIX 3 departs from this model in order to enhance reliability. As we shall see, in MINIX 3 each device driver is now a separate user-mode process.

Operating systems usually classify drivers as block devices, such as disks, orcharacter devices, such as keyboards and printers. Most operating systems define a standard interface that all block drivers must support and a second standard interface that all character drivers must

support. These interfaces consist of a number of procedures that the rest of the operating system can call to get the driver to do work for it.

In general terms, the job of a device driver is to accept abstract requests from the device-independent software above it and see to it that the request is executed. A typical request to a disk driver is to read block n. If the driver is idle at the time a request comes in, it starts carrying out the request immediately. If, however, it is already busy with a request, it will normally enter the new request into a queue of pending requests to be dealt with as soon as possible.

The first step in actually carrying out an I/O request is to check that the input parameters are valid and to return an error if they are not. If the request is valid the next step is to translate it from abstract to concrete terms. For a disk driver, this means figuring out where on the disk the requested block actually is, checking to see if the drive's motor is running, determining if the arm is positioned on the proper cylinder, and so on. In short, the driver must decide which controller operations are required and in what sequence.

Once the driver has determined which commands to issue to the controller, it starts issuing them by writing into the controller's device registers. Simple controllers can handle only one command at a time. More sophisticated controllers are willing to accept a linked list of commands, which they then carry out by themselves without further help from the operating system.

After the command or commands have been issued, one of two situations will apply. In many cases the device driver must wait until the controller does some work for it, so it blocks itself until the interrupt comes in to unblock it. In other cases, however, the operation finishes without delay, so the driver need not block. As an example of the latter situation, scrolling the screen on some graphics cards requires just writing a few bytes into the controller's registers. No mechanical motion is needed, so the entire operation can be completed in a few microseconds.

In the former case, the blocked driver will be awakened by the interrupt. In the latter case, it will never go to sleep. Either way, after the operation has been completed, it must check for errors. If everything is all right, the driver may have data to pass to the device-independent software (e.g., a block just read). Finally, it returns some status information for error reporting back to its caller. If any other requests are queued, one of them can now be selected and started. If nothing is queued, the driver blocks waiting for the next request.

Dealing with requests for reading and writing is the main function of a driver, but there may be other requirements. For instance, the driver may need to initialize a device at system startup or the first time it is used. Also, there may be a need to manage power requirements, handle Plug 'n Play, or log events.

Device-Independent I/O Software : Although some of the I/O software is device specific, a large fraction of it is device independent. The exact boundary between the drivers and the device-independent software is system dependent, because some functions that could be done in a device-independent way may actually be done in the drivers, for efficiency or other reasons. The functions shown in Fig. 3-6 are typically done in the device-independent software. In MINIX 3, most of the device-independent software is part of the file system, Although we will study the file system in Chap. 5, we will take a quick look at the device-independent software here, to provide some perspective on I/O and show better where the drivers fit in.

Functions of the device-independent I/O software :

(i) Uniform interfacing for device drivers
(ii) Buffering
(iii) Error reporting
(iv) Allocating and releasing dedicated devices
(v) Providing a device-independent block size

Q.9.(a) What is shell interpreter ? Explain some basic shell commands.

Ans. Command-Interpreter System/Shall Interpreter : One of the most important system program for an operating system is the command interpreter, which provides an interface between the user and the operating system. User give commands which are executed by operating system (usually by system calls). The main function of command interpreter system is to get and execute the next command specified by the user.

Command-Interpreter system is usually not a part of the Kernel of an operating system, since multiple command interpreters (shell in UNIX) may be supported by an operating system, and they are in no need to run in Kernel mode.

There are two main advantages to separate the command interpreter system from the Kernel :

(*i*) If we want to change the interface of command interpreter then we are not able to change the code of kernel. So, we cannot modify the interface.

(*ii*) If the command interpreter is a part of kernel then it is possible for a malicious process to gain access to a certain part of kernel which will be harmful for an operating system.

Basic shell commands : A shell is a user interface that provides access to an operating system. It is a program that takes your commands you type from the keyboard and gives them the operating system to perform the required task. After the task is completed it displays the output. A shell is an environment in which we can run our commands. A program called 'bash' acts as the shell program in most Linux systems.

The prompt, $, which is called command prompt, is issued by the shell. While the prompt is displayed, you can type a command. The shell reads your input after you press Enter. It determines the command you want executed by looking at the first word of your input. The following are some of the basic shell commands used in linux.

Date : The date command is a simple example of shell command which displays current date and time:

$ date

Sat Aug 13 15:23:50 IST 2016

SSH : In Linux machines you can use the following command to log into your server.

 # ssh root@IP_address -p port_number

The above example command will try to log into your server as root were the 'p' argument being the port that the SSH service listens to. If you don't use -p then it will try to connect on the default SSH listening port 22. Now you are logged into your server via SSH.

PWD : To check in which directory you are currently residing, you can use the pwd shell command. PWD stands for 'print working directory'.

 # root@vps:~# pwd
/root

Ls : ls command is used to list the content of a directory. This command has many options that you can combine with. Ls with no option list files and directories in bare format where we won't be able to view details like file types, size, modified date and time, permission and links etc.

 # ls
Desktop Documents Downloads Music Pictures. Public Templates Videos

Cd : If you want to go back up on one directory, you can use the following command:

 # cd ..

For example if you are in /usr/local and you type "cd .." , you will return to the directory /usr.

To navigate to the /root directory. Type:

 # cd /

Touch : If you want to create a file, use the following command:

 # touch file1

This command will create a file which will be empty. To populate it you can use one of the many text editors in Linux such as vim, vi, nano, emacs etc…

Mkdir : For creating a directory, you can use the mkdir command:

 # mkdir dir1

Rm : If you want to delete the created directory, use the following command:

 # rm -rf dir1

This above command will delete everything in the present dir1 directory.

You can also use the 'rm' command to delete the created file. For this please use the following command:

 # rm -rf file1

Mv : You can use the 'mv' command to rename a file. If you want to rename a file named as file1 to file2 you can use the following command for that:

 # mv file1 file2

'mv' command can also be used to move a file from one directory to another, for this purpose use the command:

 # mv /root/file1 /var/www/file2

Cp : If you need to copy files and directories you can use the 'cp' command:

cp file1 file2

The above command will make a copy of the file in to the same directory.

Head : Head command is used to display the first few lines of a text file. For example, the following will display the first ten lines of the file named file1 in the current directory (i.e., the directory in which the user is currently working):

 # head file1

Tail : You can also use tail command to display the last few lines of a text file.

 # tail file1

The above command will display the last ten lines of the file named file1 in the current directory.

Cat : If you want to display the contents of a file at the *command* line, the simplest way is to use the 'cat' command. To display the contents of a file named as file1, use the following command:

 # cat file1

Cat command can also use for combining copies of text files and creating new text files. These are the basic shell commands used in Linux.

If you need any further assistance please reach our support department.

Q.9.(b) Describe Window NT architecture.

Ans. Windows NT Architecture : Windows NT architecture is illustrated in fig.

Interrupt-Driven Input/Output : The problem with programmed input/output is that the processor has to wait a long time for the input/output module of concern to be ready for either reception or transmission of more data. The processor, while waiting, must repeatedly interrogate the status of the input/output module. As a result, the level of performance of the entire system is degraded.

An alternative approach for this is : for the processor to issue an I/O command to a module and then go on to do some other useful work. The input output module will then interrupt the processor to request service, when it is ready to exchange data with the processor. The processor than executes the data transfer as before and then resumes its format processing.

Fig. shows the use of interrupt. Input/output for reading in a block of data. However, interrupt input/output still consumers a lot of time because every data has to pass with processor.

Vertical Memory Manager : The virtual memory manager has two major duties : to create and manage address maps for process and central physical memory allocation.

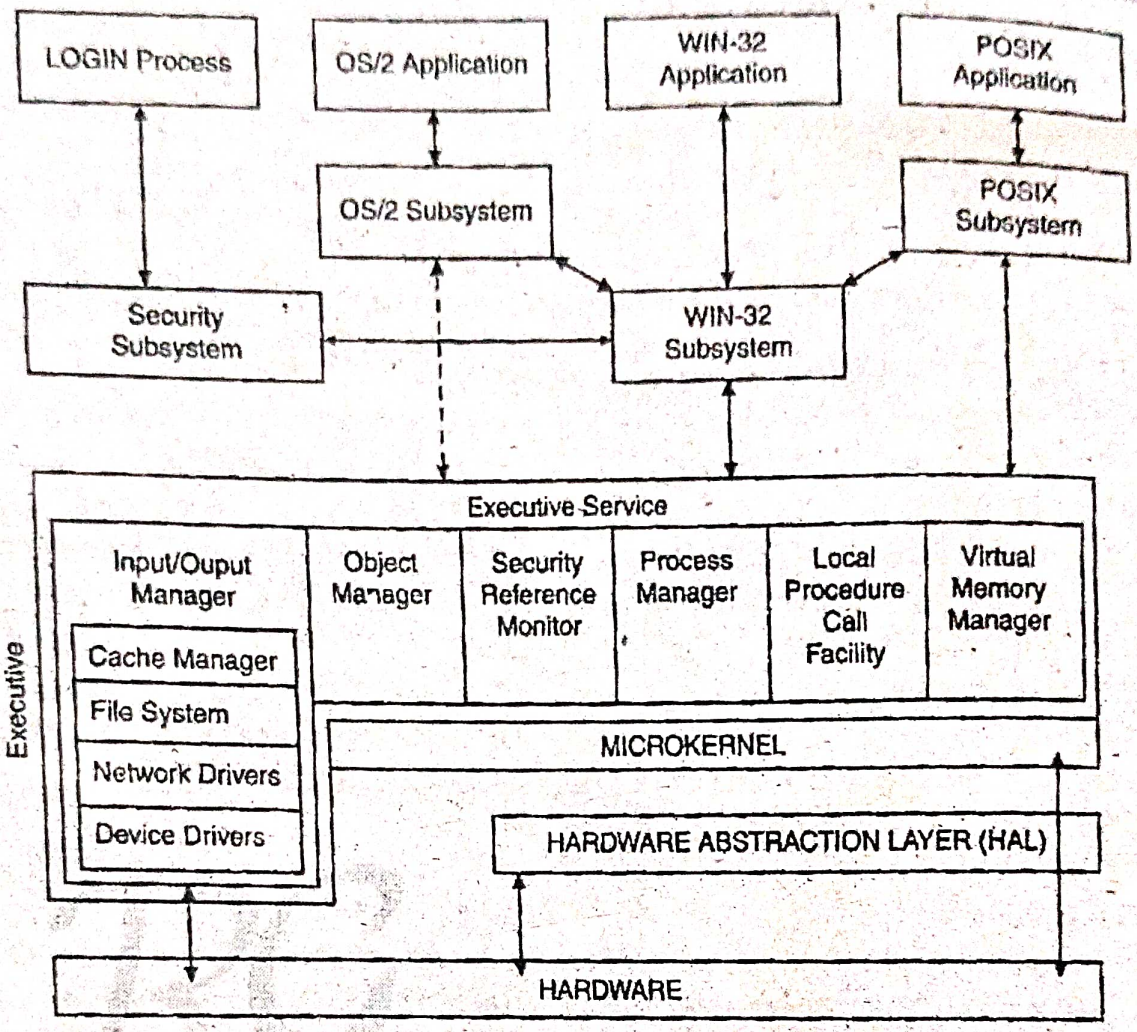
The virtual memory manager implements demand-page virtual memory, which means it manage memory in individual pages. The total memory application requirement can exceed the computer's physical memory on the hard disk in page files. The virtual memory manager transfers data to physical memory from a paging file when an application requests the data .

Security Reference Monitor : The Security Reference Monitor is closely associated with the object manager. The object manager calls the security reference monitor for an access check before letting an application, open an object. The Security Monitor (SRM) implements a security model based on identifiers and Access Control Lists (ACLs).

ACLs specify the action that particular IDs perform on an object. An ACL can contain any number of access control entries (ACEs), including no entries, which contain the information, about the action that IDs can perform. NT references the ACE when a user attempt to open the object.

Cache Manager : The cache manager closely with the virtual memory manager and file system drivers. The cache manager maintain's NT's global file system, cache. When the working set tuner takes memory containing modified file data, away from the cache manager, the I/O manager invokes file systems that manage the moved file to write there data back to the disk.

I/O Manager : The I/O manager is responsible for integrating add on device with NT. The I/O manager supports 64-bit fill offsets and layered device drivers. Using 64-bit offsets lets NT's file system address extremely large file and less disk device drivers address extremely large disk.



Process Manager : The process manager works withe the kernel to define process and thread objects. The process manager wraps the kernel's process object and add to it a process identifies (PID), the access taken, an address map and a handle table.

The process manager exports an interface that lets over executive and subsystem and user mode application manipulates process and threads.

NT's Kernel : NT's kernel operates more closely with hardware than the executive does, and it contain CPU-specific code. NT's thread scheduler called the dispatcher by NT's developers, resides in the kernel.

The NT dispatcher is a preemptive scheduler. The CPU times is divided into slices called quantum.

NT implements most synchronization primitives in kernel.

NT has a rich set of synchronization types including semaphores, events and spin locks.

Hardware Abstraction Layer (HAL) : The HAL is NT's interface to the now CPU. To make the portability feasible NT's developers isolate as much CPU specific code as possible into a separate, dynamically replaceable module, the HAL. The HAL exports a common processor model that masks the difference in various processor clips from NT.

A common difference between motherboards in the same processor family is that some are for multiprocessor systems and other are for uniprocessor system.

Microsoft provides the debug version of NT for device-driver developers.