B.E.

Fifth Semester Examination, Dec, 2009

PRINCIPLES OF OPERATING SYSTEM

Note : Attempt any five questions.

Q. 1. (a) Suppose that there are 5 jobs requiring a time of 10 units, 20 units, 3 units, 7 units and 12 units. If the time slice is 5 units then calculate the turn around time and average waiting time of each job by applying the following algorithms :

(i) FIFO

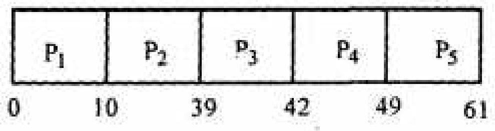
(ii) SJF

(iii) Round Robin.

Ans.

Process	Burst Time
P_1	10
P_2	29
P_3	03
P_4	07
P_5	12

(i) FIFO : The Gantt chart using FIFO Algorithm :



Turn around time per P_1, P_2, P_3, P_4 is : 10, 39, 42, 49, 61 resp.

n/

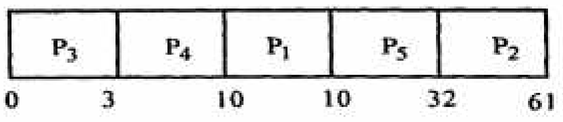
Average turn around time : $= \frac{10+39+42+49+61}{5} = \frac{201}{5}$

$= 40.2$ units

Average waiting time $= \frac{0+10+39+42+49}{5}$

$= \frac{140}{5} = 28$ units

(ii) SJF : The gantt chart using SJF algorithm is :



Turn around time for P_1, P_2, P_3, P_4 & P_5 is : 20, 61, 3, 10, 32 resp.

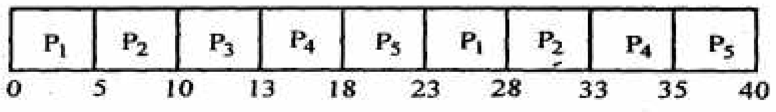
Average Turn around Time : $= \frac{20+61+3+10+32}{5} = \frac{120}{5}$

$= 25.2$ units

Average waiting time $= \frac{10+32+0+3+20}{5} = \frac{65}{5}$

$= 13$ units

(iii) Round Robin : The gantt chart using RR algorithm :



P_2	P_5	P_2	P_2	P_2

40 45 47 52 57 61

Turn around time for P_1, P_2, P_3, P_4, P_5 is : 28, 61, 13, 35, 47 resp.

Average turn around time : $= \frac{28+61+13+35+47}{5}$

$= 184/5 = 36.8$ units.

Average waiting time $= (0+(23-5))+(5+(28-10))+(40-33)+(47-45)+10+(13+(33-18))$

$+ \frac{(18+(35-23)+(45-40))}{5}$

$= \frac{18+5+18+7+2+10+13+15+18+12+5}{5}$

$\frac{123}{5} = 26.6$ units.

Q. 1. (b) What are concurrent processes?

Ans. Concurrent Processes :

A process is an execution of a program; it actually performs the actions specified in it. An operating system considers processes as entities for scheduling most programming languages contain features for concurrent programming. During execution, a program coded using these features informs the OS about its parts that are to be executed concurrently. The OS considers each of these executions as a process. Hence, the processes have a many-to-one relationship with the program. We call such a program a concurrent program. Processes that co-exist in the system at some time are called concurrent processes. Concurrent processes have opportunities to interact with one another during their execution.

Q. 1. (c) What is critical section? What are the ways by which critical section problem can be resolved?

Ans. Consider a system consisting of n processes $\{P_0, P_1, P_2 ------- P_{n-1}\}$. 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 sections. Thus, the execution of critical sections by the process is mutually exclusive in time. The critical-section problem is to design a protocol that the process 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.

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

The diff., ways to resolve critical section problem are :

1. Two-process solutions.
2. Multiple-process solutions.
3. Synchronization hardware
4. Semaphores
5. Binary semaphores.

Q. 2. (a) An OS contains 2 resource classes. The number of resource units in these classes is 4 and 5 respectively. The current resource allocation state is shown below :

I

Process Allocated Resources Maximum Required

	R_1	R_2	R_1	R_2
P_1	1	3	2	5
P_2	2	1	3	2

Would the following requests be granted in the current state? Give reason in support of your answer :

(i) Process P_2 requests (1, 1)

(ii) Process P_1 requests (0, 1)

(iii) Process P_1 requests (1, 0)

Ans.

Process	Allocated	Resources	Maximum	Required
	R_1	R_2	R_1	R_2
P_1	1	3	2	5
P_2	2	1	3	2

The need available matrix is :

Process	Need		Available	
	R_1	R_2	R_1	R_2
P_1	1	2		
		0 1	01	
P_2	1	1		

(i) Process P_2 requests (1, 1) :

Process P_2 request 1 resource of R, & 1 resource of type P_2. As both are available, the request of P_2 is satisfied & it will execute completely & resources are free to use for process P_1. So, for this request both process request will be granted for the current state.

(ii) Process P_1 requests (0, 1) : As process P_1 request only resource type P_2. It is available but as it needs 2 R_2 resources & one R_1 resources. It doesn't satisfies any need. So for this request it is not execute completely or not satisfies its need for the current state.

(iii) Process P_1 requests (1, 0) : Process P_1 requests only resource type R_1. It is available, its maximum need for R_1 1's only one so it is allocated to P_1 & it will use R_1 as its needs completely then free it but in case of R_2 its request is not granted as still it needs 2 instances of resource R_2.

Q. 2. (b) Describe the following :

(i) Short, long and middle term scheduler

(ii) Dispatcher.

Ans. (i) Short, Long and Middle Term Scheduler :

A process migrates between the various scheduling queues throughout its lifetime. The operating system must select, for scheduling purposes, processes from these queues in some fashion. The selection process is carried out by the appropriate scheduler. The long-term or job scheduler, selects processes from this pools & loads them into memory for execution. The short-term scheduler, or CPU scheduler, selects from among the processes that are ready to execute, and allocates the CPU to one of them. The primary distinction between these two schedulers is the frequency of their execution. The long-term scheduler executes much less frequently than short-term schedular. The long-term controls the degree of multiprogramming.

Some operating systems, such as time-sharing systems, may introduce an intermediate level of scheduling. This medium-term scheduler, removes processes from memory and thus reduces the degree of multiprogramming. At some later time, the process can be reintroduced into memory & its execution can be continued where it left off. This scheme is called swapping. The process is swapped out and is later swapped in by the medium-term scheduler.

(ii) Dispatcher :

Dispatcher is a component involved in the CPU scheduling function. The dispatches is the module that gives control of the CPU to the process selected by the short-term scheduler. This function involves :

/

Switching context

Switching to user mode

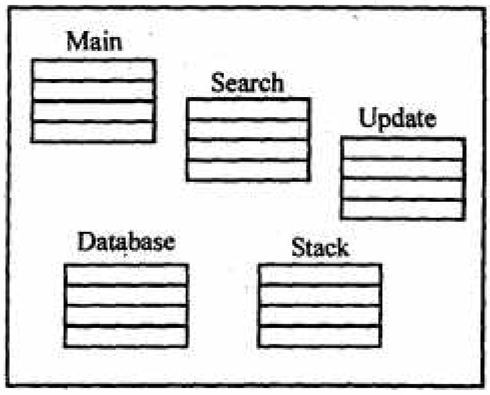
Jumping to the proper location in the user

Program to restart that program.

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

Q. 3. (a) Discuss the similarities and differences between paging and segmentation. What are advantages of combining paging with segmentations?

Ans. Both paging & segmentation have advantages & disadvantages. In segmentation with paging approach, an each segment in a program is paged separately. Accordingly, on integral number of pages is allocated to each segment. This approach simplifies memory allocation and speeds it up and also avoids external fragmentation. A page table is constructed for each segment, and a pointer to the page table is kept in the segment's entry in the segment table. Address translation for a logical address (si, bi) is now done in two stages. In the first stage, the entry of Si is located in the segment table and the address of its page table is obtained. The byte no. bi is now split into a pass (psi, bpi), where Psi is the page number is segment si, bpi is the byte no. in page Pi. The effective address calculation is now completed as in paging, i.e., the frame number of psi is obtained & bpi is concatenated with it to obtain the effective address.



Name	Size	Page table address
Main	476	—

Database	21600	—
Update	320	—
Search	600	—
Stack	500	—

Fig. shows a process in a system using segmentation with paging. Each segment is paged independently, so internal fragmentation exists in the last page of each segment. Each segment table entry now contains a pointer to the page table of the segment. The size field in a segment's entry is used to facilitate a bound check for memory protection.

Q. 3. (b) Consider the following page reference string :

A, B, C, D, B, A, E, F, B, A, B, C, E, G, C, B, A, B

How many page faults would occur for the following replacement algorithms assuming four available frames? All frames are initially empty :

(i) LRU

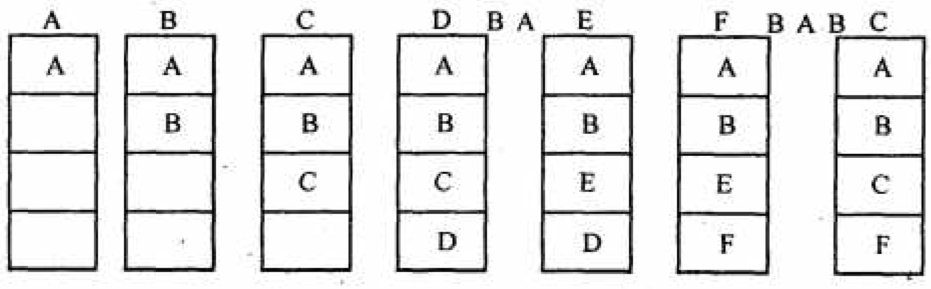
(ii) FIFO

(iii) Optimal.

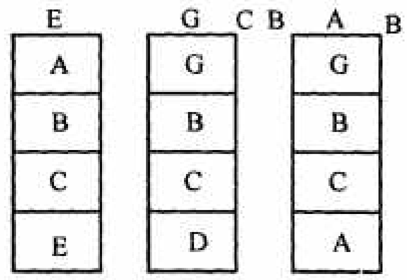
Ans. Page Reference Strings :

A, B, C, D, B, A, E, F, B, A, B, C, E, G, C, B, A, B.

(i) LRU : In this we will replace the page that has not been used for the longest period of time. This approach is least recently used algorithm.

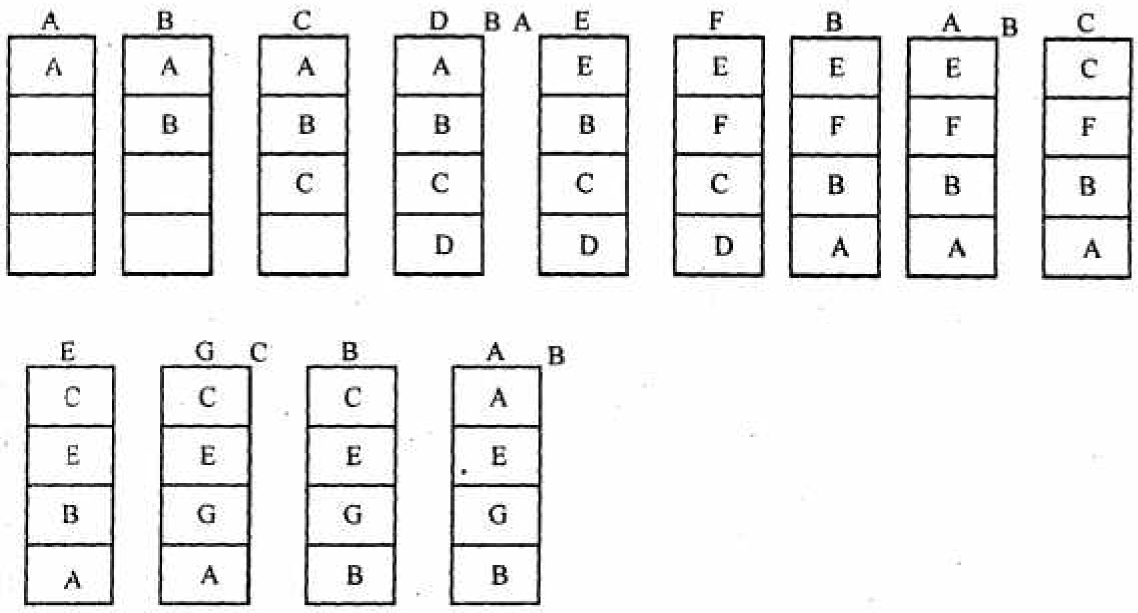


Page Frames :



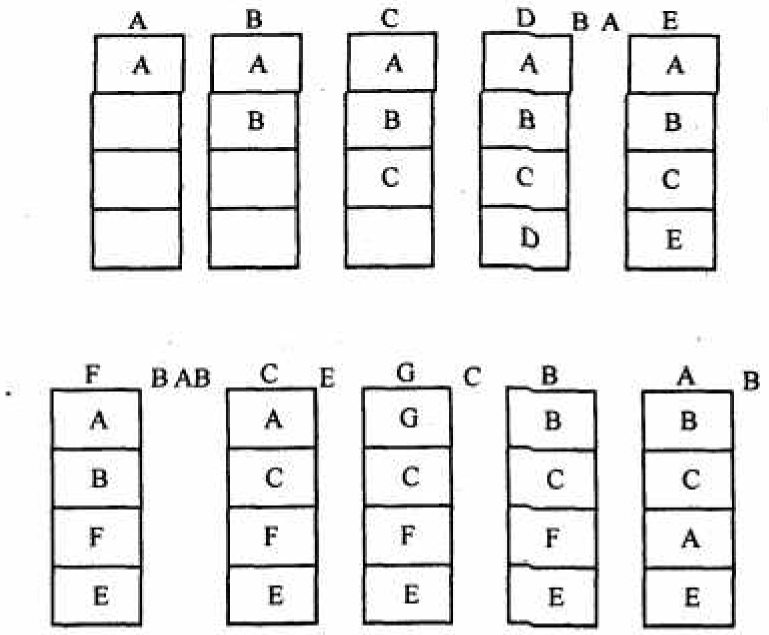
These are total 12 page faults.

(ii) FIFO : It is simple algorithm replaces the first page entered in memory.



There are 13 page faults.

(iii) Optimal : Replace the page that will not be used for the longest period of time.



There are 10 page faults.

Q. 4. (a) Discuss various file allocation and access methods. Compare their advantages and disadvantages.

Ans. File store information, when it is used, this information must be accessed and read into computer memory. The information in the file can be accessed in several ways.

1. Sequential Access :

Information in the file is processed in order, one record after the other. This mode of access is most common; for example, editors & compilers usually access files in this fashion.

2. Direct Access :

Another method is direct access or relative access. The direct-access method is based on a disk-model of a file, since disks allow random access to any file block. A direct access file allow arbitrary blocks to be read or written. There is no restriction on the order of reading or writing for a direct-access file.

Direct-access files are of great use for immediate access to large amount of information. Databases are often of this type.

3. Indexed Sequential Access Method :

It uses a small master index that points to disk blocks of a secondary index. The secondary index blocks points to the actual file blocks. The file is kept stored on a defined key. To find a particular item, we first make a binary search of the master index, which provides the block number of the secondary index.

Allocation Methods :

1. Contiguous Allocation :

This requires each file to occupy a set of contiguous blocks on the disk. Disk addresses define a linear ordering on the disk with this ordering, assuming that only one job is accessing the disk, accessing block between after block normally requires no head movement.

Continuous allocation of a file is defined by the disk address & length. Accessing a file that has been allocated contiguously is easy difficulty with this is finding space for a new file. Another problem is of determining how much space is needed for a file.

2. Linked Allocation :

It solves all problems of contiguous allocation. With linked allocation, each file is a linked list of disk blocks; the disk blocks may be scattered anywhere on the disk. The directory contains a pointer to the first and last blocks of the file.

The problem with this is that it can be used effectively only for sequential access files. Another disadvantage is the space required for the pointers. An another problem with linked allocation is reliability.

3. Indexed Allocation :

Linked allocation solves the external fragmentation & size-declaration problems of contiguous allocation. However, in the absence of FAT, linked allocation can't support efficient direct access, since the pointers to the blocks are scattered with the blocks themselves all rev the disk & need to be retrieved in order. Indexed allocation solves this problem by bringing all the pointers together into one location; the index block. Index allocation does suffer from waste space.

Q. 4. (b) What is fragmentation? What are its types? How they are minimized in different memory management schemes?

Ans. Fragmentation :

Memory fragmentation can be internal as well as external. As processes are loaded & removed from memory the free memory space is broken into little pieces. External fragmentation exists when enough total memory space exists to satisfy a request, but it is not contiguous; storage is fragmented into a large number of small holes.

Consider a multiple-partition allocation scheme with a hole of 18, 464 bytes. Suppose that the next process requests 18,462 bytes. If we allocate exactly the requested block, we are left with a hole of 2 bytes. The overhead to keep track of this hole will be substantially larger than the hole itself. The general approach is to break the physical memory into fixed-size blocks, and allocate memory in unit of block sizes. With this the memory allocated to a process may be slightly larger than the requested memory. The diff., between these two numbers is internal fragmentation-memory that is internal to a partition but is not being used. One solution to the problem of external fragmentation is compaction. The goal is to suffle the memory contents to place all free memory together in one large block compaction is not always possible. The simplest compaction algorithm is simply to move all processes toward one end of memory; all holes move in the other direction, producing one large hole of available memory. This scheme can be expensive. Another possible solution to the external fragmentation problem is to permit the logical-address space of a process to be non-contiguous, thus allowing a process to be allocated physical memory wherever the latter is available. Two complementary techniques achieve this scheme : 1. Paging 2. Segmentation.

Q. 5. (a) Discuss the following :

(i) Compaction

(ii) I/O buffering

Ans. (i) Compaction :

One solution to the problem of external fragmentation is compaction. The good is to shuffle the memory contents to place all free memory together in one large block. Compaction is not always possible. If relocation is static and is done at assembly or load time, compaction cannot be done, compaction is possible only if relocation is dynamic, and is done at execution time. If addresses are relocated dynamically, relocation requires only moving the program and data and then changing the base register to reflect the new base address. When compaction is possible, we must determine its cost. The simple compaction algorithm is simply to move all processes toward one end of memory; all holes move in the other direction, producing one large hole of available memory. This scheme can be expensive compaction may be of two types :

1. Partial Compaction :

Only free space that is alternatively present in memory can be compact together.

2. Full Compaction :

All free space available can be compacted together to form a large free space of memory.

(ii) I/O Buffering :

It is basically the process of temporarily storing data that is passing between a processor and a peripheral.

The usual purpose is to smooth out the difference in rates at which the two devices can handle data. When we are using a printer or a scanner as the speed of CPU is much more than that of input/output devices. So, we first store the input or output in a buffer and then process it. This buffer act as a temporary storage of data. This temporarily storage is more speed than that of input/output.

A second use of buffering is to adopt between devices that have different data-transfer sizes.

A third use of buffering is the support copy semantics for application input/output. For example, suppose we want to print some pages, as the speed of printer is very slow we first copy all pages to the buffer & then access these pages from the buffer & CPU is kept busy in other operation.

Q. 5. (b) What is a semaphore? How it is different from Monitors? Implement wait( ) and signal ( ) without busy wait for binary semaphores.

Ans. A semaphore S is an integer variable that, apart from initialization, is accessed only through two standard atomic operations; waits signal. These two operations were originally termed P (for wait) & V (for signal). The classical definition of wait is :

```
Wait (s) {

while (S ≤ 0)

; // no-op

S --;}
```

The classical definition of signal is :

```
Signal (S) {

S++;

}.
```

Modifications to the integer value of the semaphore in the wait and signal operations must be executed indivisibly. That is, when one process modifies the semaphore value, no other process can simultaneously modify that same semaphore value.

We can implement this scheme readily by letting P_1 & P_2 share a common semaphore synch, initialized to 0 and by inserting the statements.

S_1

```
Signal (synch);
do
{
        wait (mutex);
        critical section
        signal (mutex);
        remainder section
}
while (1);
```

In process P_1 and the statements

```
wait (synch);
S2;
```

in process P_2.

On other, A monitor is characterized by a set of programmer-defined operators. The representation of a monitor type consists of declarations of variables whose values define the state of an instance of the type, as well or bodies of procedures or functions that implement operations on the type.

Q. 6. (a) Differentiate between physical address and logical addresses.

Ans. An address generated by the CPU is commonly referred to as a logical address, where as an address seen by the memory unit that is, the one loaded into the memory-address register of the memory is commonly referred to as a physical address.

The compile-time and load-time address-finding methods generate identical logical & physical addresses. The set of all logical addresses generated by a program is a logical address space. The abstract view of an entity is called the logical view & the arrangement & relationship between components of the entity is called the logical organization. The real view of an entity is called the physical view and the arrangement depicted in it is called physical organization.

A logical address is considered to consist of two parts the id of the process component containing the address and the id of the byte within the component we represent each address by a pair of the form

(comp i, byte i).

Q. 6. (b) A machine has 48 bit virtual addresses and 32 bit physical addresses. Pages are of 8K how many entries are needed for a conventional page table and for an inverted page table.

Ans. Virtual address	= 48 bits
Physical address	= 32 bit
Page size	= 8 k

Number of entries for a conventional page table :

Size of logical address space is 2^{48}, and the

Page size is 8k $= 2^{13}$ bits

$$p = m - n = 48 - 13 = 35$$

$$d = n = 13$$

There are 2^{35} pages total.

35	13
Page number	Page offset

For an inverted page table.

Size of physical address space it 2^{32} & the page size is 8k $= 2^{13}$ bits.

$$p = 32 - 13 = 19$$

$$d = n = 13$$

There are 2^{19} pages total.

19	13
Page number	Page offset

Q. 6. (c) What is thrashing? What is the cause of thrashing? How does the system cope up with thrashing.

Ans. Thrashing :

Look at any process that doesn't have enough frames. Although it is technically possible to reduce the number of allocated frames to the minimum, these is some number of pages in active use. If the process doesn't have this number of frames, it will quickly page fault. At this point, it must replace some page. However, since all its pages are in active use, it must replace a page that is needed again. It quickly page faults again & again. The process continues to fault, replacing pages for which it then faults brings back in right away. This high paging activity is called thrashing. A process is thrashing if it is spending more time paging then executing.

Thrashing results in severe performance problems. To understand the cause of thrashing consider the operating system monitors CPU utilization. If CPU utilization is too loss we increase the degree of multiprogramming by introducing a new process to the system. A global page replacement algorithm is used; it replaces pages with no regard to the process to which they belong. This will decrease the CPU utilization. The CPU scheduler sees the decreasing CPU utilization, and increases the degree of multiprogramming. The new process tries to get started by taking frames from running processes, causing more page faults and a longer queue for the paging device.

As a result, CPU utilization drops even further and the CPU schedular tries to increase the degree of multiprogramming even more. Thrashing has occurred & system throughputs plunge. The page fault rate increases.

Q. 7. (a) What are threads? What are the difference between user level threads and kernel level threads? Under what circumstances is one type better than the others?

Ans. A thread, sometime called a light weight process (LWP) is a basic unit of CPU utilization it comprises a thread ID, a program counter, a register set and a stack. It shares with other threads belonging to the same process its code section, data section, and other operating system resources, such as open files & signals. A traditional process has a single thread of control. If the process has multiple threads of control, it can do more than one task at a time.

Support for threads may be provided at either the user level, for user threads or by the kernel, for kernel threads.

1. User Threads :

They are supported above the kernel & are implemented by a thread library at the user level. The library provides support for thread creation, scheduling and management with no support from the kernel. Because the kernel is unaware of user-level threads, all thread creation & scheduling are done in user space without the need for kernel intervention. Therefore, user level threads are generally fast to create and manage; they have drawbacks, however. For instance, if the kernel is single-threaded, then any user-level thread performing a blocking system call will cause the entire process to block, even if other threads are available to run within the application. User-thread libraries include POSIX P threads, Mach C-threads and Solari's 2 UI threads.

2. Kernel Threads :

These are supported directly by the operating system. The kernel performs thread creation, scheduling & management in kernel space. Because thread management is done by the operating system, kernel threads are generally slower to create & manage than are user threads. However, since the kernel is managing the threads, if a thread performs a blocking system call, the kernel can schedule another thread in the application for execution. Also, in a multiprocessor environment, the kernel can schedule threads on different processors. Windows NT, windows 2000, Solaries 2, BeOS, True 64 UNIX support Kernel threads.

Q. 7. (b) Explain the meaning of drwxr-xr-in a UNIX system?

Ans. **drwxr–xr–in Unix :**

d is used for a directory.

"—" : In any position means that flag is not set.

"r" : File is readable by owner, group or other.

"w" : File is writable on a directory, bisite access means you can add or delete files.

"x" : File is executable. Execute permission on a directory means you can list the files in that directory.

Each user has a default set of permissions which apply to all files created by that user, unless the software explicitly set something else.

We use-rw-r-r- for regular files.

&

drwxr-xr-x for directory.

These are used to define the root directories in UNIX. The meaning of above is only owner can create, rename, and delete files. Everyone can read access files.

Q. 8. Write short notes on following :

(i) **Swapping**

(ii) **Disk Scheduling**

(iii) **Deadlock prevention**

(iv) **Time sharing systems.**

Ans. (i) Swapping :

A process can be swapped temporarily out of memory to a backing store and then brought back into memory for continued execution. For e.g., assume a multiprogramming system with a round-robin CPU-scheduling algorithm. When a quantum expires, the memory manager will start to sea out the process that just finished, and to swap in another process to the memory space that has been freed. A variant of this swapping policy is used for priority-based scheduling algo-If a higher-priority process arrives & wants service, the memory manager can swap out the lower-priority process so that it can load and execute the higher-priority process. When the higher-priority process finishes, the lower-priority process can be swapped back in and continued. This variant of swapping is called roll out, roll in.

Swapping requires a backing store. The backing store is commonly a fast disk.

(ii) Disk Scheduling :

One of the responsibilities of the operating system is to use the hardware efficiently for the disk drives, meeting this responsibility entails having a fast access time & disk bandwidth. The access time has two major components. The seek time is the time for the disk arm to move the head to the cylinder containing the desired sector. The rotational latency is the additional time waiting for the disk to rotate the desired sector to the disk head. The disk band-width is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer. We can improve both the access time & the bandwidth by scheduling the servicing the disk input/output requests in a good order.

The various disk scheduling algorithm are :

1. FCFS scheduling
2. SSTF scheduling
3. SCAN scheduling
4. L-SCAN scheduling
5. Look scheduling.

(iii) Deadlock Prevention :

For a deadlock to occur each of the for necessary conditions must hold. By ensuring that at least one of these conditions can't hold, we can prevent the occurrence of a deadlock.

(i) Mutual Exclusion :

Mutual exclusion condition holds for non-sharable resources. To prevent deadlock to avoid this condition use sharable resources as possible.

(ii) Hold & Wait :

To ensure that hold & wait condition never occur in the system, we must guarantee that, whenever a process requests a resource, it does not hold any other resources, one protocol that can be used requires each process to request and be allocated all its resources before it begins execution.

(iii) No Preemption :

To ensure that this condition doesn't hold, we can use the following protocol. If a process is holding some resources & requests another resource that can't be immediately allocated to it, then all resources currently begin held are preempted. The preempted resources are added to the list of resources for which the process is waiting.

(iv) Time Sharing Systems :

Time sharing is a logical extension of multiprogramming. The CPU executes multiple jobs by switching among them, but the switches occur so frequently that the users can interact with each program while it is running.

/

A time-sharing O.S. allows many users to share the computer simultaneously. Since each action or command in a time-sharing system tends to be short, only a little CPU time is needed for each user.

A time-shared O.S. uses CPU scheduling & multi-programming to provide each user with a small portion of a time-shared computer. They are more complex than multiprogramming systems. As the system switches rapidly from one use to the next, each user is given the impression that the entire computer system is dedicated to his use, even though it is being shared among many users.