58567

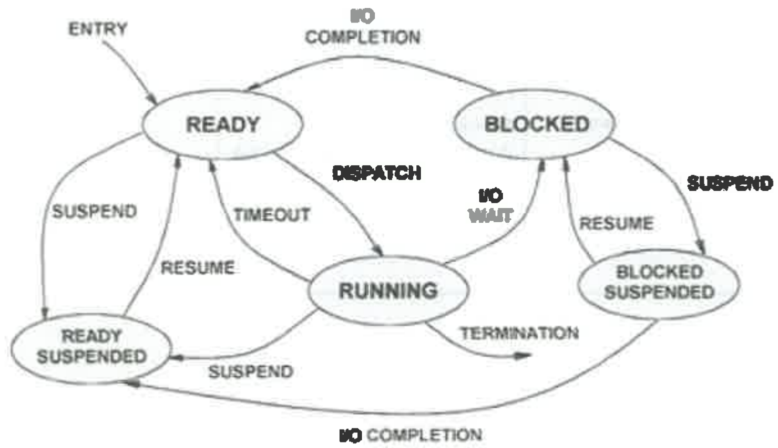
Solution

1. Question 1 is compulsory.
2. Attempt any **three** from remaining five questions.
3. Assume suitable data where required.

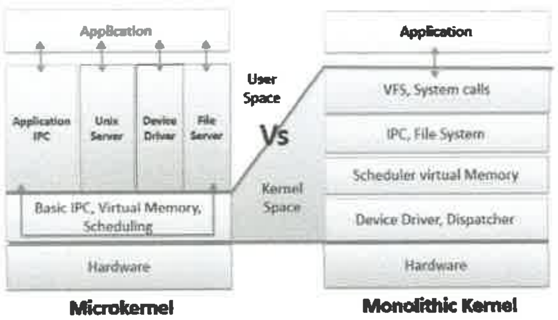
<table>
<tr><th>S N o</th><th>Questions</th><th>Marks</th></tr>
<tr><td>1</td><td>

a. Discuss Operating System as a resource manager
- Main Memory management
- I/O management
- Secondary memory management

b. Draw process state diagram and explain the following states:
1. New
2. Ready
3. Running
4. Wait
5. Suspended ready
6. Suspended wait



c. Describe Microkernel with a diagram.



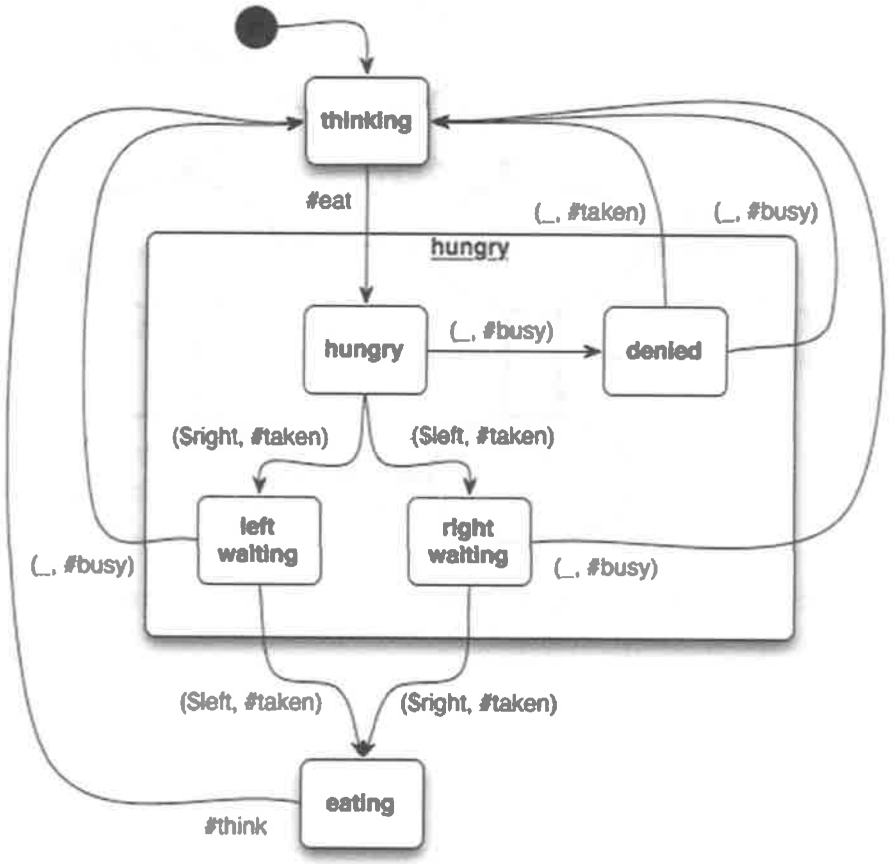
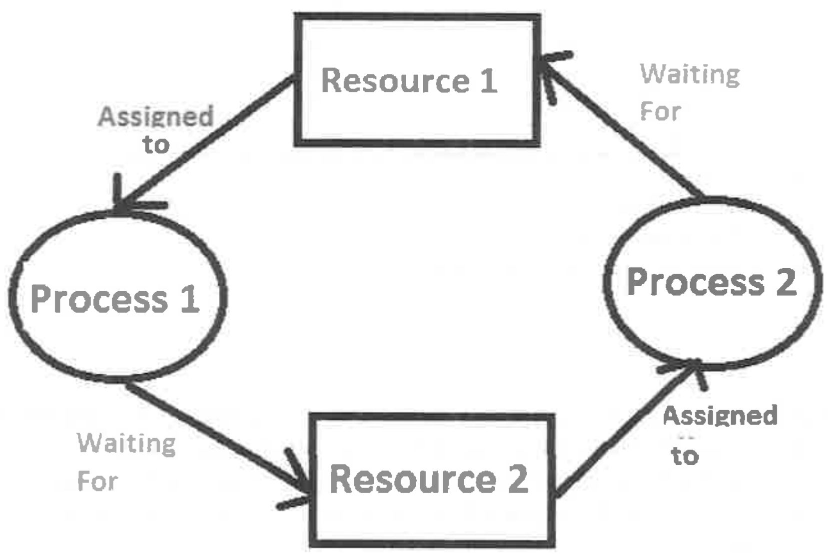
</td><td>[5]

[5]

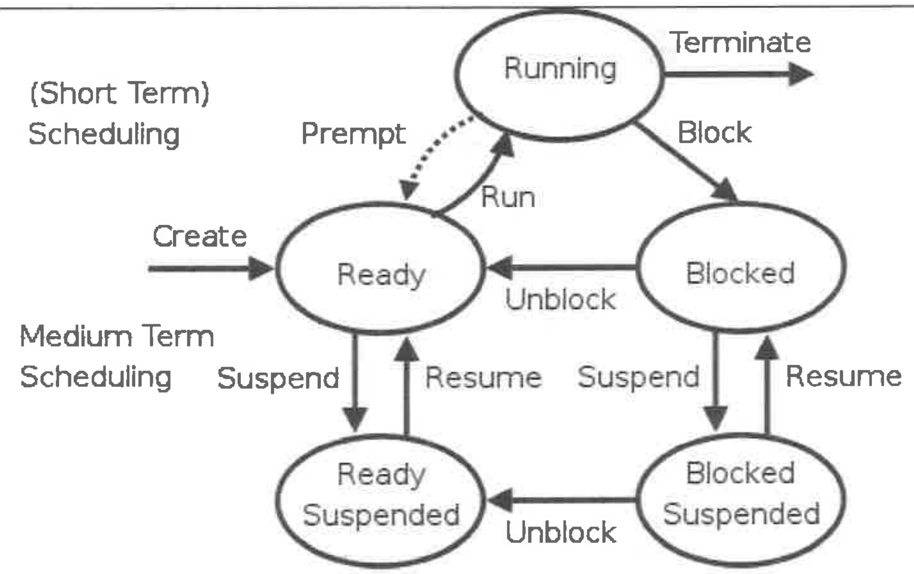
[5]</td></tr>
</table>

d. Discuss the importance of "Multithreading". Differentiate between kernel and user thread. [5]

User level thread	**Kernel level thread**
User threads are supported above the kernel and 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.	Kernel threads are supported directly by the operating system. The kernel performs thread creation, scheduling, and management in kernel space.
When threads are managed in user space, each process needs its own private thread table to keep track of the threads in that process	No run-time system is needed in each. Also, there is no thread table in each process. Instead, the kernel has a thread table that keeps track of all the threads in the system.
User-level *threads requires non-blocking systems call*, that means a multithreaded kernel. Otherwise, entire process will blocked in the kernel, even if there are runnable threads left in the processes. For example, if one thread causes a page fault, the process blocks.	Kernel threads do not require any new, non-blocking system calls. If one thread in a process causes a page fault, the kernel can easily check to see if the process has any other runnable threads, and if so, run one of them while waiting for the required page to be brought in from the disk.
User-level threads are generally fast to create and manage	The kernel-level threads are slow and inefficient. For instance, threads operations are hundreds of times slower than that of user-level threads.

	 	
	 	4
	For the Banker's algorithm to work, it needs to know three things: • How much of each resource each process could possibly request[MAX]	6

2



Unblock is done by another task (a.k.a. wakeup, release, V)
Block is a.k.a. sleep, request, P)

a. Calculate AWT, ATAT, Response Time and Throughput of the following processes using Shortest job first (Non Pre-emptive).

Process	Arrival Time (ms)	Burst Time (ms)
P1	1	7
P2	2	5
P3	3	1
P4	4	2
P5	5	8

- AWT=6ms [3]
- ATAT=10.6ms [3]
- Throughput=20.8 [2]
- Response Time: P1=0; P2=9; P3=5; P4=5; P5=11 [2]

3

a. What are Semaphores? Differentiate between Counting and Binary Semaphores. Discuss Readers Writers Problem. Write sudo code of its solution using semaphores.
- Synchronization tool used for synchronization, It uses wait and signal. [2]
- Binary is variable is either 0 or 1; in Counting semaphore can be initialized to more than 1 [2]

[6]

- How much of each resource each process is currently holding[ALLOCATED]
- How much of each resource the system currently has available[AVAILABLE]

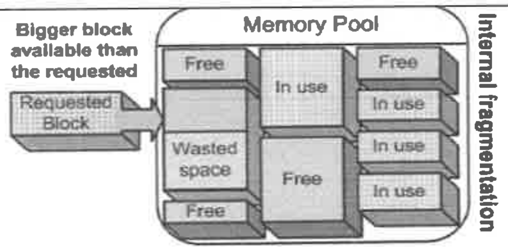
Resources may be allocated to a process only if the amount of resources requested is less than or equal to the amount available; otherwise, the process waits until resources are available.
Some of the resources that are tracked in real systems are memory, semaphores and interface access.
The Banker's Algorithm derives its name from the fact that this algorithm could be used in a banking system to ensure that the bank does not run out of resources, because the bank would never allocate its money in such a way that it can no longer satisfy the needs of all its customers[citation needed]. By using the Banker's algorithm, the bank ensures that when customers request money the bank never leaves a safe state. If the customer's request does not cause the bank to leave a safe state, the cash will be allocated, otherwise the customer must wait until some other customer deposits enough.
Basic data structures to be maintained to implement the Banker's Algorithm:
Let n be the number of processes in the system and m be the number of resource types. Then we need the following data structures:

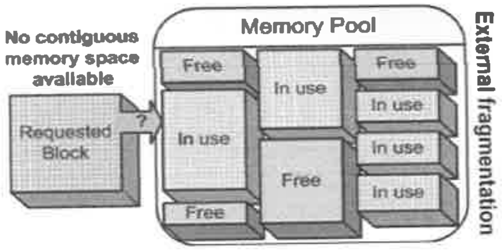
- Available: A vector of length m indicates the number of available resources of each type. If Available[j] = k, there are k instances of resource type R_j available.
- Max: An $n \times m$ matrix defines the maximum demand of each process. If Max[i,j] = k, then P_i may request at most k instances of resource type R_j.
- Allocation: An $n \times m$ matrix 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.
- Need: An $n \times m$ matrix indicates the remaining resource need of each process. If Need[i,j] = k, then P_i may need k more instances of resource type R_j to complete the task.

Note: Need[i,j] = Max[i,j] - Allocation[i,j]. n=m-a.

4

a. [4] [4]





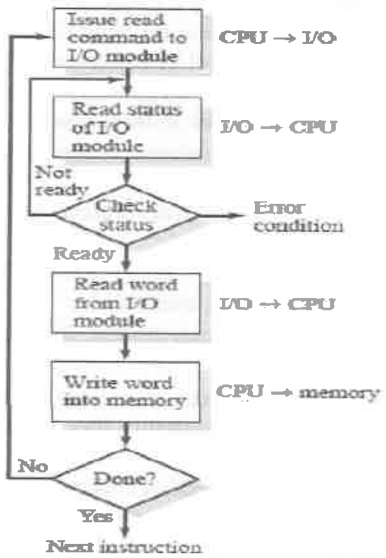
b. Compare the performance of FIFO, LRU and Optimal based on number of page hit for the following string. Frame size = 3; String (pages): 1 2 3 4 5 2 1 3 3 2 4 5 [4] [4] [4]

FIFO: No of Hits=2; page fault=10
LRU: No of Hits=2; page fault=10

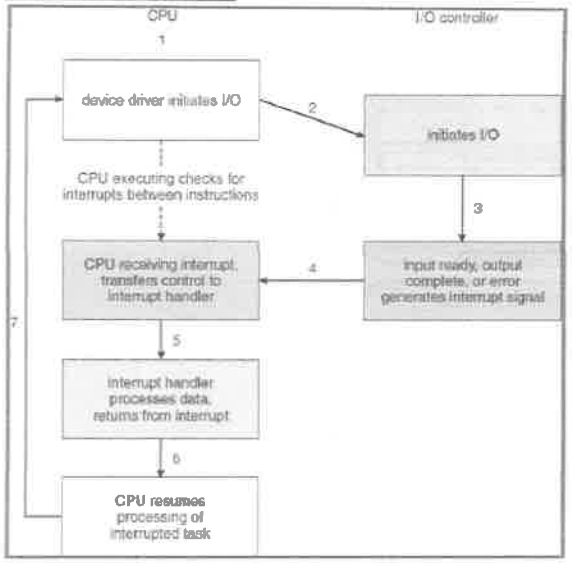
Optimal: No of hits=5; page fault=7

5 a. **Explain Interrupt driven IO and discuss the advantages of Interrupt driven IO over programmed IO.** [5]

Programmed I/O



Interrupt Driven I/O [5]



TYPES OF DISK SCHEDULING ALGORITHMS

Although there are other algorithms that reduce the seek time of all requests, I will only concentrate on the following disk scheduling algorithms: [2]

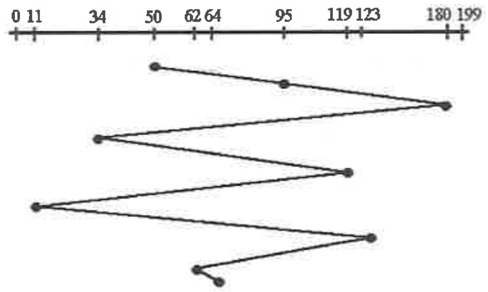
First Come-First Serve (FCFS)

Shortest Seek Time First (SSTF)

Elevator (SCAN)
Circular SCAN (C-SCAN)
LOOK
C-LOOK
These algorithms are not hard to understand, but they can confuse someone because they are so similar. What we are striving for by using these algorithms is keeping Head Movements (# tracks) to the least amount as possible. The less the head has to move the faster the seek time will be. I will show you and explain to you why C-LOOK is the best algorithm to use in trying to establish less seek time.

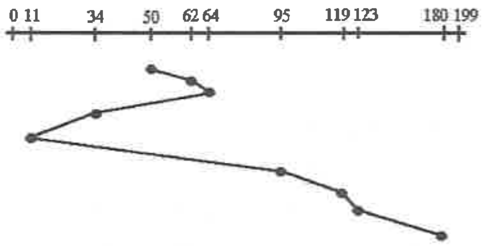
Given the following queue -- 95, 180, 34, 119, 11, 123, 62, 64 with the Read-write head initially at the track 50 and the tail track being at 199 let [2]
us now discuss the different algorithms.

1. *First Come -First Serve*



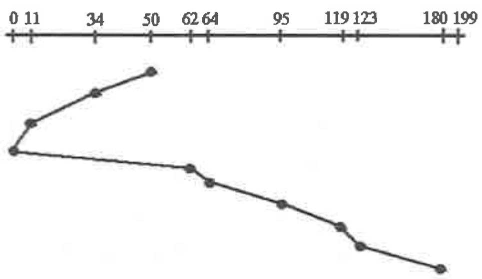
(FCFS) All incoming requests are placed at the end of the queue. Whatever number that is next in the queue will be the next number served. Using this algorithm doesn't provide the best results. To determine the number of head movements you would simply find the number of tracks it took to move from one request to the next. For this case it went from 50 to 95 to 180 and so on. From 50 to 95 it moved 45 tracks. If you tally up the total number of tracks you will find how many tracks it had to go through before finishing the entire request. In this example, it had a total head movement of 640 tracks. The disadvantage of this algorithm is noted by the oscillation from track 50 to track 180 and then back to track 11 to 123 then to 64. As you will soon see, this is the worse algorithm that one can use.

2. *Shortest Seek Time First* [2]



[2]

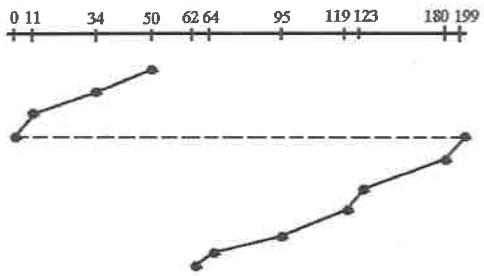
(SSTF)
In this case request is serviced according to next shortest distance. Startingat 50, the next shortest distance would be 62 instead of 34 since it is only 12 tracks away from 62 and 16 tracks away from 34. The process would continue until all the process are taken care of. For example the next case would be to move from 62 to 64 instead of 34 since there are only 2 tracks between them and not 18 if it were to go the other way. Although this seems to be a better service being that it moved a total of 236 tracks, this is not an optimal one. There is a great chance that starvation would take place. The reason for this is if there were a lot of requests close to eachother the other requests will never be handled since the distance will always be greater.



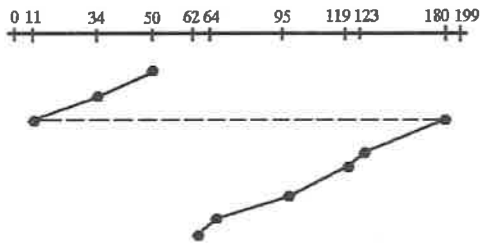
[2]

3. Elevator (SCAN)
This approach works like an elevator does. It scans down towards the nearest end and then when it hits the bottom it scans up servicing the requests that it didn't get going down. If a request comes in after it has been scanned it will not be serviced until the process comes back down or moves back up. This process moved a total of 230 tracks. Once again this

is more optimal than the previous algorithm, but it is not the best.



4. Circular Scan (C-SCAN)
Circular scanning works just like the elevator to some extent. It begins its scan toward the nearest end and works it way all the way to the end of the system. Once it hits the bottom or top it jumps to the other end and moves in the same direction. Keep in mind that the huge jump doesn't count as a head movement. The total head movement for this algorithm is only 187 track, but still this isn't the mose sufficient.



5. C-LOOK
This is just an enhanced version of C-SCAN. In this the scanning doesn't go past the last request in the direction that it is moving. It too jumps to the other end but not all the way to the end. Just to the furthest request. C-SCAN had a total movement of 187 but this scan (C-LOOK) reduced it down to 157 tracks.

From this you were able to see a scan change from 644 total head movements to just 157. You should now have an understanding as to why *your operating system truly relies on the type of algorithm it needs when it is dealing with multiple processes.*

(i)

6\. a. Discuss various File Allocation Mechanism and their advantages.

File access mechanism refers to the manner in which the records of a file may be accessed. There are several ways to access files –

- Sequential access
- Direct/Random access
- Indexed sequential access

Sequential access [2]

A sequential access is that in which the records are accessed in some sequence, i.e., the information in the file is processed in order, one record after the other. This access method is the most primitive one. Example: Compilers usually access files in this fashion.

Direct/Random access

- Random access file organization provides, accessing the records directly. [2]
- Each record has its own address on the file with by the help of which it can be directly accessed for reading or writing.
- The records need not be in any sequence within the file and they need not be in adjacent locations on the storage medium.

Indexed sequential access

- This mechanism is built up on base of sequential access. [2]
- An index is created for each file which contains pointers to various blocks.
- Index is searched sequentially and its pointer is used to access the file directly.

Space Allocation

Files are allocated disk spaces by operating system. Operating systems deploy following three main ways to allocate disk space to files.

- Contiguous Allocation
- Linked Allocation
- Indexed Allocation

Contiguous Allocation

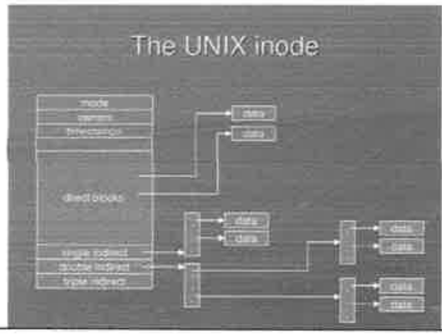
- Each file occupies a contiguous address space on disk.
- Assigned disk address is in linear order. [2]
- Easy to implement.
- External fragmentation is a major issue with this type of allocation technique.

Linked Allocation

- Each file carries a list of links to disk blocks.
- Directory contains link / pointer to first block of a file. [2]
- No external fragmentation
- Effectively used in sequential access file.
- Inefficient in case of direct access file.

Indexed Allocation

- Provides solutions to problems of contiguous and linked allocation.
- A index block is created having all pointers to files.
- Each file has its own index block which stores the addresses of disk space occupied by the file.
- Directory contains the addresses of index blocks of files.

	B Unix iNode Structure  	[10][1]

1