Operating System
(DEC 2019)

Q.P. Code – 58567

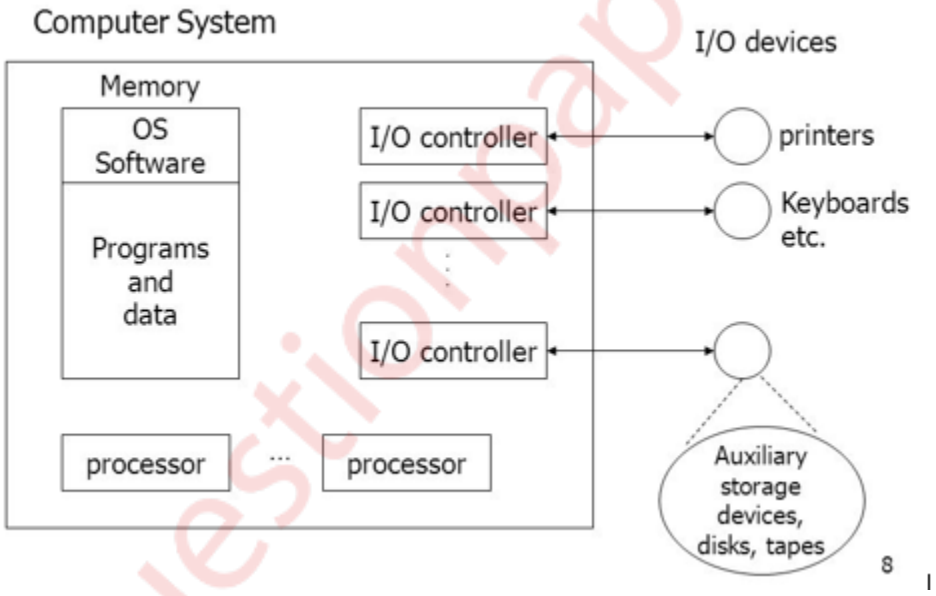
Q 1

a) Discuss Operating System as a Resource Manager. **5M**

- An operating system is a program that manages the computer hardware.
- An operating system may be viewed as an organized collection of software extension of hardware, consisting of control routines for operating a computer and for providing an environment for execution of program.
- An operating system is an important part of almost every computer system.
- Internally an Operating System acts as a manager of resources of the computer system such as processor, memory, files, and I/O device.

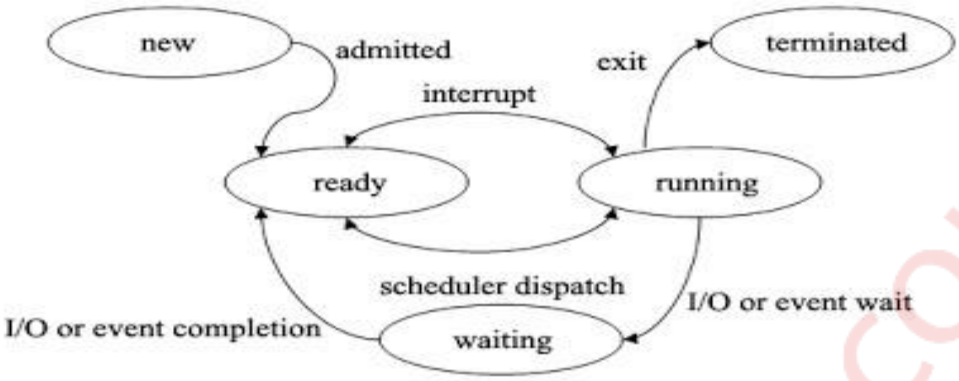
- In this role, the operating system keeps track of the status of each resource, and decides who gets a resource, for how long and when.

- In system that supports concurrent execution of program, the operating system resolves conflicting requests for resources in manner that preserves system integrity, and in doing so attempts to optimize the resulting performance.



b) Draw process state diagram and explain following states: **5M**

1. New
2. Ready
3. Running
4. Wait
5. Suspended ready
6. Suspended wait



- Process can have one of the following five states at a time.

1. New state: A process that just has been created but has not yet been admitted to the pool of execution processes by the operating system. Every new operation which is requested to the system is known as the new born process.

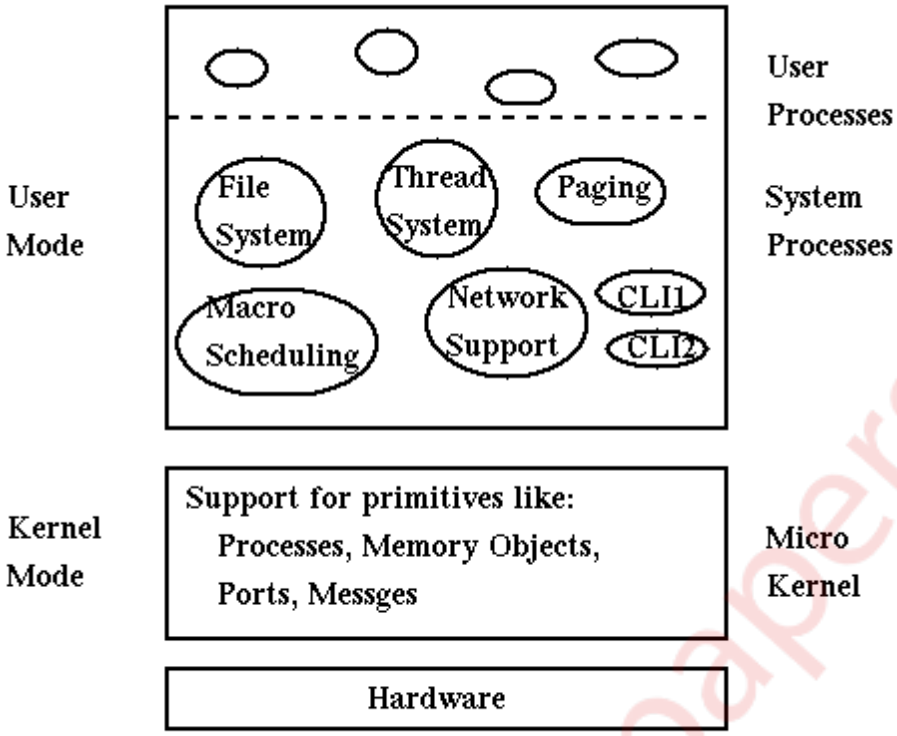
2. Ready state: When the process is ready to execute but he is waiting for the CPU to execute then this is called as the ready state. After completion of the input and output the process will be on ready state means the process will wait for the processor to execute.

3. Running state: The process that is currently being executed. When the process is running under the CPU, or when the program is executed by the CPU, then this is called as the running process and when a process is running then this will also provide us some outputs on the screen.

4. Waiting or blocked state: A process that cannot execute until some event occurs or an I/O completion. When a process is waiting for some input and output operations then this is called as the waiting state and in this process is not under the execution instead the process is stored out of memory and when the user will provide the input and then this will again be on ready state.

5. Suspended ready: After the completion of the process, the process will be automatically terminated by the CPU. So this is also called as the terminated state of the process. After executing the complete process the processor will also deallocate the memory which is allocated to the process. So this is called as the terminated process.

c) Describe microkernel with a diagram. **5M**



Micro–Kernel Architecture

1. In microkernel, set of modules for managing the hardware is kept which can uniformly well be executed in user mode. A small microkernel contains only code that must execute in kernel mode. It is the second part of operating system.
2. There is different address space for user mode as well as kernel mode.
3. It has a small space as compared to monolithic kernel.

4. Execution speed is slower than monolithic kernel.
5. If one service crashes whole operating system do not fails, it does not affect working of other part micro kernel.
6. Communication is done through message passing.
7. To write microkernel more code is required.
8. It is easily extendible.
9. Example: QNX, Symbian, L4Linux, etc.
10. It is more flexible.
11. In microkernel, communication is through message passing.

d) Discuss the importance of "Multithreading". Differentiate between kernel and user thread. **5M**

Importance of Multithreading:

- In operating system that supports multithreading, process can consist of many threads, Theses threads run in parallel improving the application performance.
- Each such thread has it's own CPU state and stack, but they share the address space of the process and the environment.
- Considering the advantages of user level and kernel level threads, a hybrid threading model using both types of threads, a hybrid threading model using both types of threads can be implemented.
- The Solaris operating system supports this hybrid model.
- In this implementation, all the thread management functions are carried out by user level thread package at user space. So operations on thread do not require kernel intervention.
- Java language supports for development of multithreaded process that offers scheduling and memory management for java applications.
- Java application that can benefit directly from multicore resources comprise application servers.
- Single application can also be benefitted from multicore architecture by running multicore instances of the application in parallel.
- If multiple application instances require some degree of isolation, virtualization technology can be used to offer each of them with it's own separate and secure environment.

User level	Kernel level
1. Kernel is unaware of the thread.	1. The thread management is carried out by kernel.
2. All of the work of thread management is done by thread package.	**2. All thread management activities are carried out in kernel space.**
3. Kernel threads are generally requires more time to create and manage than the user thread.	**3. creating and destroying threads require less time.**
4. if one thread is blocked on I/O, entire process gets blocked.	**4. Blocking of one thread does not blocked entire process.**
5. User level threads are platform independent.	**5. Kernel level threads are platform dependent.**

Q2

a) Differentiate between short term, medium term and long term scheduler with a diagram. **10M**

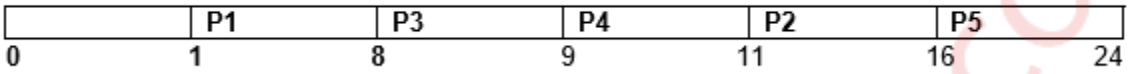
Sr. No	Long-term Scheduler	Short-term scheduler	Medium-term Scheduler
1	Select processes from the queue and loads them into the memory for execution.	Chose the process from ready queue and assign it to the CPU.	Swap in and out the processes from memory.
2	Speed is less than short term scheduler.	Speed is very fast and invoked frequently than long term scheduler.	Speed is in between both the short term and long term.
3	Transition of process state from new to ready.	Transition of process state from ready to executing.	No process state transition
4	Not present in time sharing system.	Minimal in time sharing system.	Present in time sharing system.
5	Supply a reasonable mix of jobs, such as I/O bound CPU bound.	Select new process to allocate to CPU frequently.	Processes are swapped in and out for balanced process mix.
6	It controls degree of multiprogramming	It has control over degree of multiprogramming	Reduce the degree of multiprogramming
7	It is also called as job scheduler.	It is also called as CPU scheduler.	Swapping scheduler
8	The decision to add to the pool of processes to be execute	The decision to add to the number of processes that are partially or fully in main memory.	The decision as to which available processes will be executed by the processor.

b) Calculate AWT, ATAT, Response Time and Throughput of the following processes using Shortest job first (Non-Pre-emptive).
10M

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

➔

Process	AT	BT	CT
P1	1	7	8
P2	2	5	16
P3	3	1	9
P4	4	2	11
P5	5	8	24



TAT (CT-AT)	WT (TAT-BT)
7	0
14	9
6	5
7	5
19	11
Total= 53	Total= 30

Avg TAT= 53/5= 37.8
WT= 30/5= 21.2

Q3
a) What are Semaphores? Differentiate between counting and Binary Semaphores. Discuss Dinning Philosopher problem. **10M**

Definition:
A semaphore S is integer variable whose value can be accessed and changed only by two operations wait and signal. Wait and signal are atomic operations.

The wait operation on semaphore S,

```
Wait(S): If S>0
         THEN S: =S-1
         ELSE (wait on S)
```

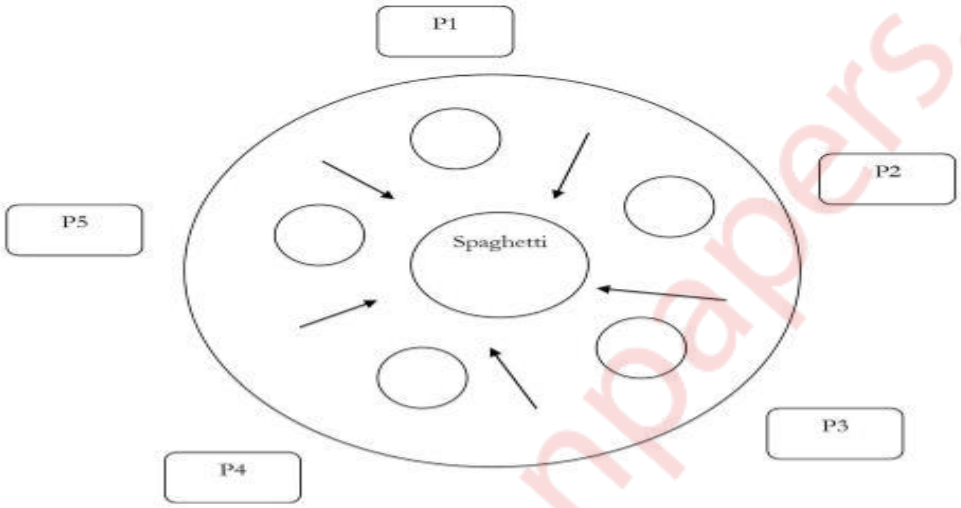
The signal operation on semaphore S

```
Signal(S): IF(one or more process are waiting on S)
           THEN (let one of these processes proceed)
           ELSE S: =S+1
```

Difference between counting and binary semaphore

Counting Semaphore	**Binary Semaphore**
No mutual Exclusion	Mutual exclusion
Any integer value	Value only 0 and 1
More than one slot	Only one slot
Provide a set of processes	It has a mutual exclusion mechanism

The Dining Philosopher Problem: Five philosophers live in a house, where a dining table has been laid for them. The five philosophers have agreed to eat only spaghetti considering the fact that it is the best for their lifestyle and health. All the philosophers require two forks to eat. The table is arranged as shown below:



The eating arrangements are as follows: a round table as above where five plates of spaghetti are kept for each philosopher and five forks.

- A philosopher who wishes to eat goes to his/her assigned place on the table and eats the spaghetti plate in front of him using the two forks before him.
- The Aim of the Dinging philosopher's problem is to allow all the philosophers to eat.
- It must also satisfy the principles of mutual exclusion (no two philosophers can use the same fork at the same time) while avoiding deadlock and starvation.

- Solution (using Semaphores)

  - Over here, each philosopher picks up the fork on his left first and then the fork on his right.
  - After the philosopher finishes eating, the fork is replaced back on table.
  - How ever if all the philosophers sit down together, and all pick up the left fork, and then reach out to the other fork which is not present for anyone and they go on waiting indefinitely (deadlock).
  - To overcome this, we can buy five additional forks.

/* program dining philosopher*/

do {

wait(chopstick[i]);

wait(chopstick[(i+1) % 5])

EATING THE RICE

signal(chopstick[i]);

signal(chopstick[(i+1) % 5])

THINKING

} while(1);

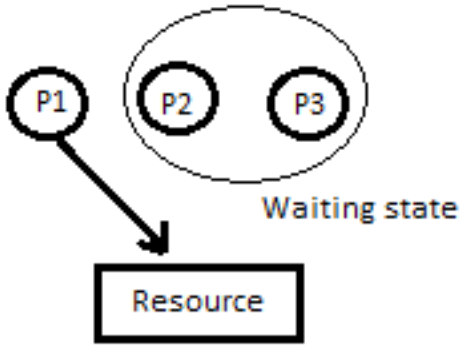
b) What do you understand by a deadlock? Explain deadlock avoidance method. 10M

➔ Deadlock:

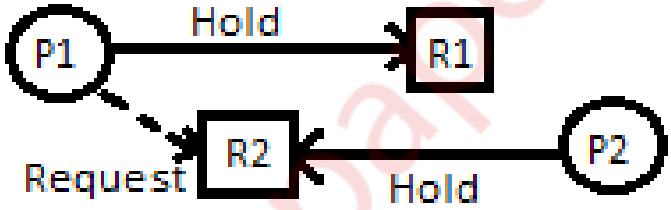
* We know that processes need different resources in order to complete the execution.
* So in a multiprogramming environment, many processes may compete for a multiple Number of resources.
* In a system, resources are finite. So with finite number of resources, it is not possible to fulfill the resource request of all processes.
* When a process requests a resource and if the resource is not available at that time. The process enters a wait state. In multiprogramming environment, it may happen with many processes.
* There is chance that waiting processes will remain in same state and will never again change state.
* It is because the resource they have requested are held by other waiting processes. When such type of situation occurs then it is called as Deadlock.

The necessary and sufficient conditions for deadlock to occur are:

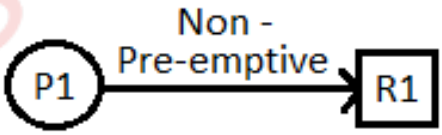
- **Mutual Exclusion**
  - A resource at a time can only be used by one process.
  - If another process is requesting for the same resource, then it must be delayed until that resource is released.



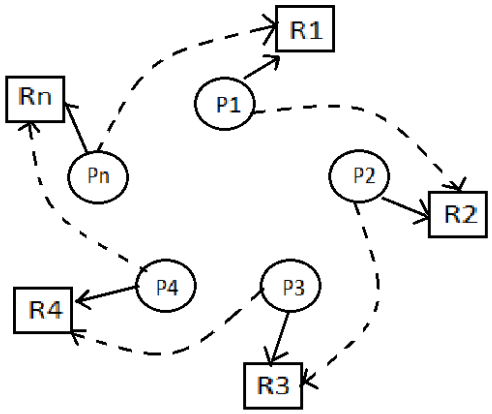
- **Hold and Wait**
  - A process is holding a resource and waiting to acquire additional resources that are currently being held by other processes.



- **No Pre-emption:**
  - Resources cannot be pre-empted
  - Resource can be released only by the process currently holding it based on its voluntary decision after completing the task



- **Circular wait:**
  - A set of processes { P1,….,Pn-1,Pn } such that the process P1 is waiting for resource held by P2,P2 is waiting for P3 ,and Pn is waiting for P1 to release its resources.
  - Every process holds a resource needed by the next process.



All the four above mentioned conditions should occur for a deadlock to occurs.

Q4

a) Explain different types of memory fragmentation. 8M

- As processes are loaded and removed from memory, the free memory space is broken into little pieces.
- It happens after sometimes that processes cannot be allocated to memory blocks considering their small size and memory blocks remains unused. This problem is known as Fragmentation.

Fragmentation is of two types –

External Fragmentation:

- It exists when there us enough total memory space available to satisfy a request, but available memory space are not contiguous.
- Storage space is fragmented into large number of small holes.
- Both first fit and best fit strategies suffer from this.
- First fit is better in some systems, whereas best fit is better for other.
- Depending on the total amount of memory storage, size, external fragmentation may be minor or major problem.
- Statistically N allocated block, Another 0.5 N blocks will be lost to fragmentation. The 1/3 of memory is unusable.

Internal Fragmentation:

- Consider a multiple partition allocation scheme with a hole of 18,462 bytes. The next process request with 18,462 bytes. If we allocate, we are left with a hole of 2 bytes.
- The general approach to avoid this problem is to:

- Break physical memory into fixed sized blocks and allocate memory in units based on block size.
- Memory allocated to a process may be slightly large than the requested memory.

Solution to Internal Fragmentation

1) Compaction- The goal is to shuffle the memory content. so as to place all free memory together in one large block.

It is not always possible due to :-

If relocation is static and done at assembly or load time

It is possible

Only if relocation is dynamic and is done at execution time

2) Permit the logical address space to the processes to be non-contagious.

b) Compare the performance of FIFO, LRU and Optimal based on number of pages hit for the following string. Frame size =3; String(pages): 1 2 3 4 5 2 1 3 3 2 4 5 12M

1. FIFO

Frame	1	2	3	4	5	2	1	3	3	2	4	5
0	1	1	1	4	4	4	1	1	1	1	1	5
1		2	2	2	5	5	5	3	3	3	3	3
2			3	3	3	2	2	2	2	2	4	4
PF	Y	Y	Y	Y	Y	Y	Y	Y	-	-	Y	Y

2. LRU

Frame	1	2	3	4	5	2	1	3	3	2	4	5
0	1	1	1	4	4	4	1	1	1	1	4	4
1		2	2	2	5	5	5	3	3	3	3	3
2			3	3	3	2	2	2	2	2	2	2
PF	Y	Y	Y	Y	Y	Y	Y	Y	-	-	Y	Y

3. Optimal

Frame	1	2	3	4	5	2	1	3	3	2	4	5
0	1	1	1	1	1	1	1	3	3	3	3	3
1		2	2	2	2	2	2	2	2	2	4	4
2			3	4	5	5	5	5	5	5	5	5
PF	Y	Y	Y	Y	Y	-	-	Y	-	-	Y	-

Performance:

Algorithm	FIFO	LRU	Optimal
Page Hit	10	10	7
Page Miss	2	2	5

Q 5

a) Explain Interrupt driven IO and discuss the advantages of Interrupt driven IO over programmed IO. 10M

- When CPU issues I/O command in support of process two possibilities exist.

- Interrupt driven I/O is an alternative scheme dealing with I/O. Interrupt I/O is a way of controlling input/output activity whereby a peripheral or terminal that needs to make or receive a data transfer sends a signal. This will cause a program interrupt to be set.

- At a time appropriate to the priority level of the I/O interrupt. Relative to the total interrupt system, the processors enter an interrupt service routine.

- The function of the routine will depend upon the system of interrupt levels and priorities that is implemented in the processor.

- Basic operation of Interrupt

1. CPU issues read command.
2. I/O module gets data from peripheral whilst CPU does other work.
3. I/O module interrupts CPU.
4. CPU requests data.
5. I/O module transfers data.

- **Design Issues**

There are 2 main problems for interrupt I/O, which are:

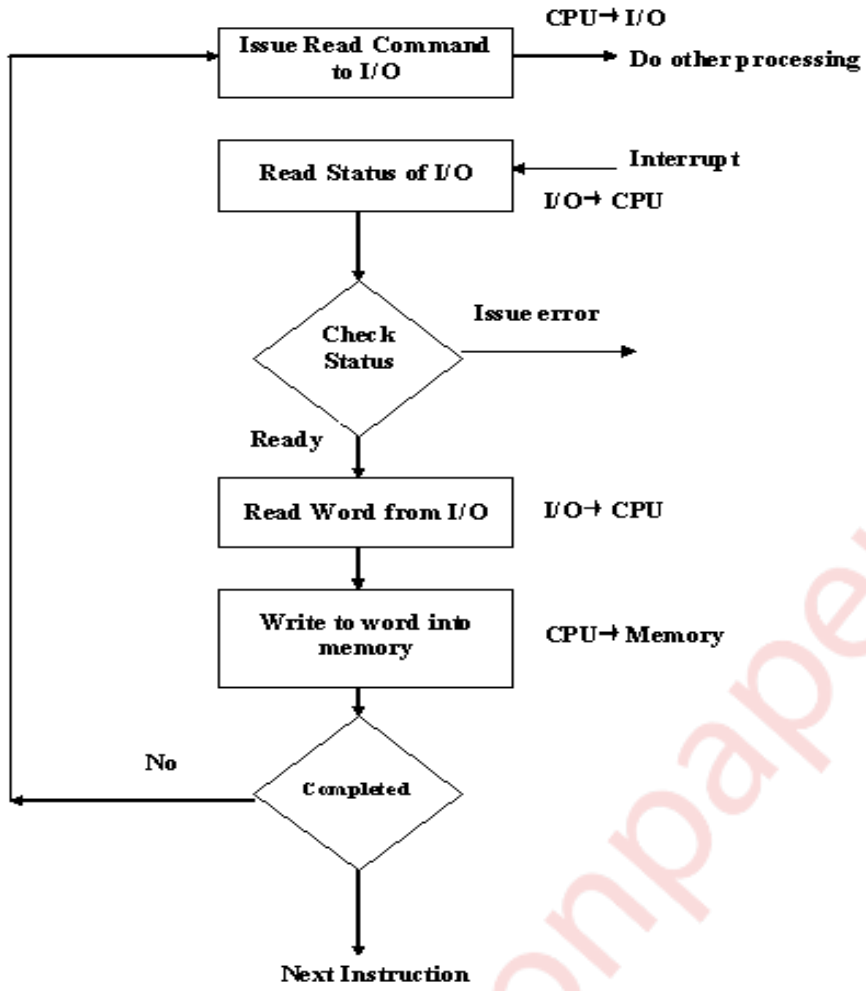
-There are multiple I/O modules, how should the processor determine the device that issued the interrupt signal?

-How does the processor decide which module to process when multiple interrupts have occurred?

- There are 4 main ways to counter these problems, which are:

1.Multiple Interrupt Lines
2. Software Poll
3. Daisy Chain (Hardware Poll, Vectored)
4. Bus Arbitration (Vectored)

- Advantages of Interrupt driven I/O
- It is fast.
- It is efficient.



b) Discuss various disk scheduling methods. **10M**

➔

- In operating systems, seek time is very important. Since all device requests are linked in queues, the seek time is increased causing the system to slow down. Disk Scheduling Algorithms are used to reduce the total seek time of any request.
- **TYPES OF DISK SCHEDULING ALGORITHMS**

  1) First Come-First Serve (FCFS)

  2) Shortest Seek Time First (SSTF)

  3) Elevator (SCAN)

  4) Circular SCAN (C-SCAN)

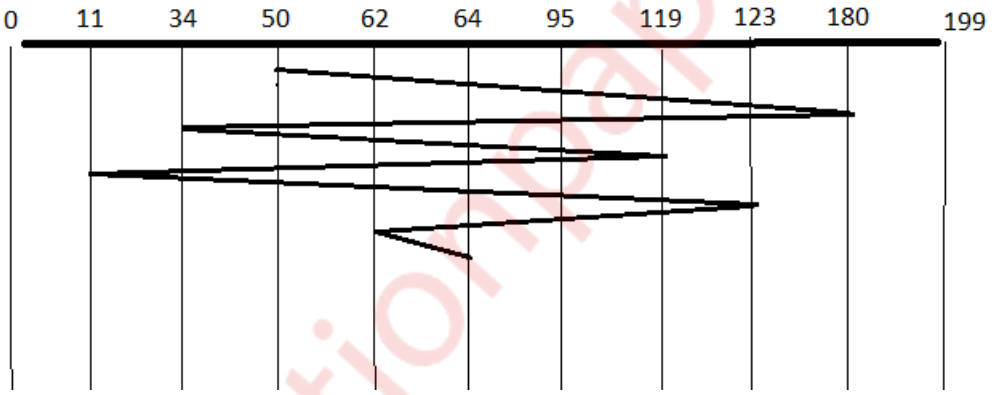
  5) LOOK

6) C-LOOK

- 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 us now discuss the different algorithms.

1. First Come -First Serve (FCFS):

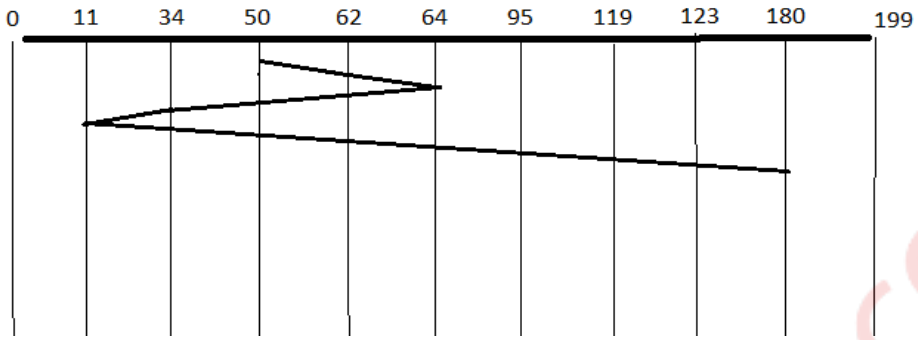
All incoming requests are placed at the end of the queue. Whatever number that is next in the queue will be the next number served. Using this algorithm doesn't provide the best results. To determine the number of head movements you would simply find the number of tracks it took to move from one request 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.



Total head moment = (95-50)+(180-95)+(180-34)+(119-34)+(119-11)+(123-11)+(123-62)+(64-62) = **644**

2. Shortest Seek Time First (SSTF):

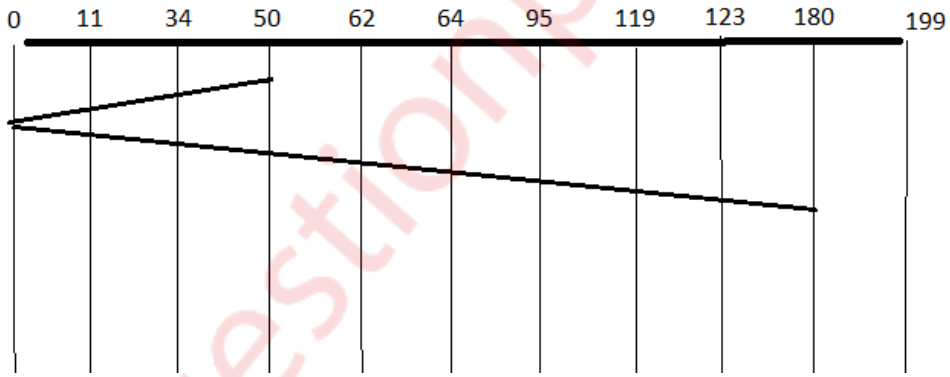
In this case request is serviced according to next shortest distance. Starting at 50, the next shortest distance would be 62 instead of 34 since it is only 12 tracks away from 62 and 16 tracks away from 34. The process would continue until all the process are taken care of. For example, the next case 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 each other the other requests will never be handled since the distance will always be greater.



Total head moment = (62-50)+(64-62)+(64-34)+(34-11)+(95-64)+(119-95)+(123-119)+(180-123) = **238**

3. Elevator (SCAN):

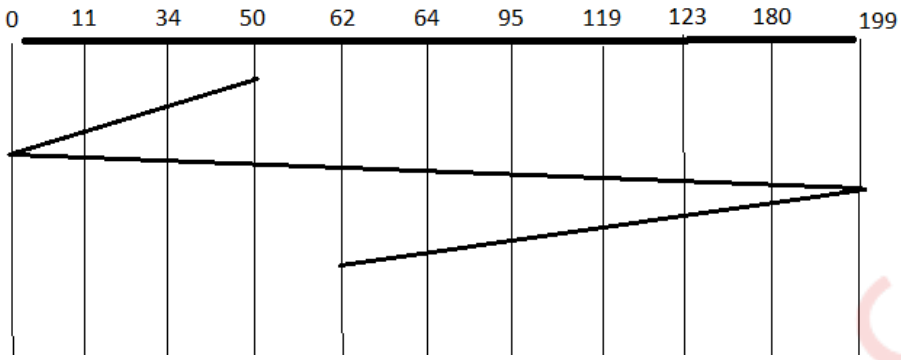
This approach works like an elevator does. It scans down towards the nearest end and then when it hits the bottom it scans up servicing the requests that it didn't get going down. If a request comes in after it has been scanned it will not be serviced until the process comes back down or moves back up.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



Total head moment = (50-34)+(34-11)+(11-0)+(62-0)+(64-62)+(95-64)+(119-95)+(123-119)+(180-123) = 230

4. Circular Scan (C-SCAN)

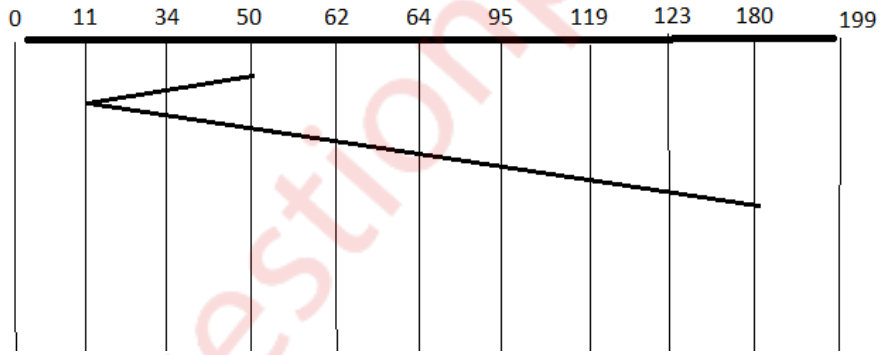
Circular scanning works just like the elevator to some extent. It begins its scan toward the nearest end and works it way all the way to the end of the system. Once it hits the bottom or top it jumps to the other end and moves in the same direction. Keep in mind that the huge jump doesn't count as a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



Total head moment = (50-34)+(34-11)+(11-0)+(199-0)+(199-180)+(180-123)+(123-119)+(119-95)+(95-64)+(64-62) = 386

5. LOOK

This is same as Scan scheduling but in this we do not move till end which reduce total head moment. This is the best scheduling algorithm because it has minimum total head
Moment.



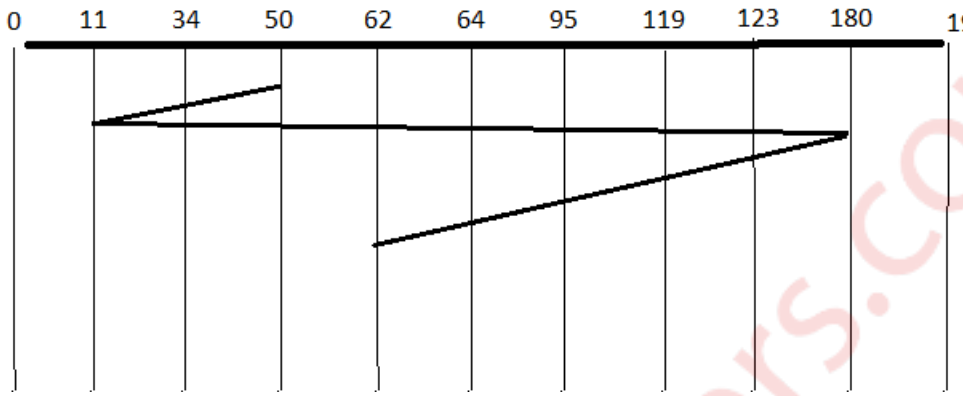
Total head moment = (50-34)+(34-11)+(62-11)+(64-62)+(95-64)+(119-95)+

(123-119)+(180-123) = 208

6. 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.



Total head moment = (50-34)+(34-11)+(180-11)+(180-123)+(123-119)+(119-95)+
(95-64)+(64-62) = **326**

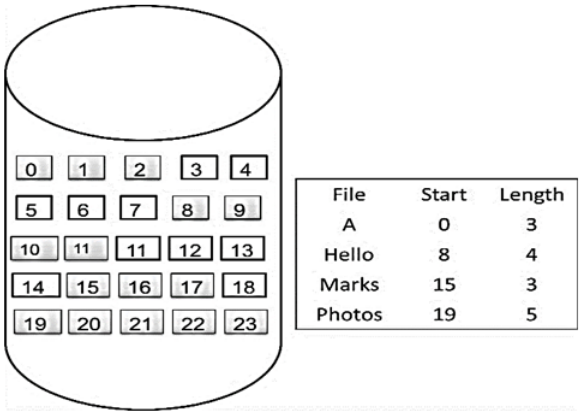
Q6

a) Discuss various File Allocation Mechanism and their advantages. 10M

➔ Files are usually stored on secondary storage devices such as disk. These files are then called back when needed. As part of their implementation, files must be stored in the hard disk. This has to be done in such a way that the disk space is utilized effectively and the files can be accessed quickly. There are following methods used majority in different operating system:

Contiguous allocation
Linked List allocation
Linked List allocation using a table in memory.
Indexed allocation
I-nodes

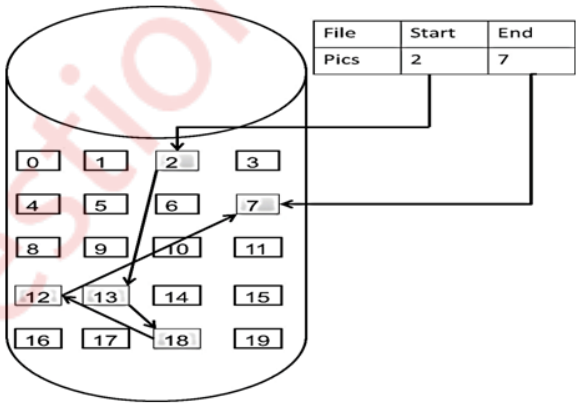
1) **Contiguous Allocation:** Each file takes up a set of contiguous blocks on the disk. Disk address defines a linear ordering on the disk. If each block size on disk is 2 KB, then 30 KB file would be allocated 15 consecutive blocks. The directory entry for each file specifies the address of the starting block and the total number of blocks allocated for this file. Directory entry is shown below. File A starts at block 0 and it is 3 block long occupying block 0.



Advantage:
• Contiguous allocation is easy to implement.

Disadvantage:
• When allocated file is deleted, continuously allocated blocks to the file become free.

2) **Linked List Allocation:** This overcomes the disadvantage of contiguous allocation. In linked list allocation, each file is linked list of disk block. The scattered disk on the disk can be allocated to the file. The directory contains a pointer to the first and the last block. Below figure shows thee linked list allocation for file Pics. The file pics of 5 blocks starts at block 2 and continue 13, then block 18, then block 12, and finally block 7.



Advantage:
• Reading file sequentially is simple.

Disadvantage:
• In each block pointer takes some space, so each file requires slightly more disk space rather than its actual size.

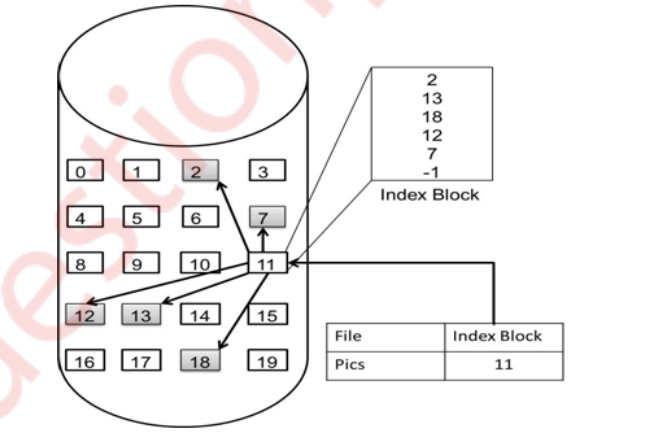
3) **Linked List allocation using table in memory:** Each block needs to store pointer information; therefore, entire block is not fully used to store file content. This limitation can be overcome by keeping pointer information in table which always remains in memory. Refer Linked list fig for Table.

Physical	Next block
0	2 (File Pics Starts from here)
1	13
2	18
3	12

Advantage:
- Random access is much easier

Disadvantage:
- Whole table must be in memory all the time to make it work.

4) **Indexed allocation:** With file allocation table in memory, Linked list allocation support random access, but this entire table must be in memory all the time. In indexed allocation, all the pointers are kept in one location called as index block. There is an index block assigned to each file and this index block holds the disk block addresses of that particular file.



Advantage:
- Indexed allocation supports random access.

Disadvantage:
- The pointer overhead of index block is more with compare to the pointer overhead of linked allocation.

5) **I-nodes**: I-Nodes (Index Nodes) Is the data structure which records the attributes and disk addresses of the files blocks. I-nodes is associated with each file and it keeps track

of which block belongs to which file. If particular file is open, only its i-node to be in memory. This is more beneficial with compare to linked list allocation which requires entire file allocation table in memory. The size of file allocation table is proportional to the number of blocks that disk contains.

File Attributes	➔	
Address of disk block 0	➔	
Address of disk block 1	➔	
Address of disk block 2	➔	
Address of disk block 3	➔	
Address of block of pointers	➔	Disk Block Containing additional disk address

b) Explain Unix iNode structure in detail. **10M**

➔The object oriented principle is used in Virtual File System (VFS).

- It has two modules: a set of definitions that states what file –system objects are permissible to seems to be and software layer for these objects manipulation.
- Following four major object types are defined by VFS:
    1) I-node Object – An individual file is represented by I-node Object.
    2) File Object – An open file is represented by file object.
    3) Superblock Object – An entire file system is represented by a Superblock Object.
    4) Dentry Object – An individual directory entry is represented by Dentry object.
- A set of operations are defined for each of the type of objects. Each object of one of these points to a function table.
- The record of addresses of the actual functions is kept in function table. These functions implement the defined set of operations for that object.
- The VFS software layer need not recognize earlier about what kind of object it is dealing with. It can carry out an operation on one of the file-system objects by invoking the right function from the object's function table.
- The VFS remains unaware about whether an i-node represents a networked file, a disk file, a network socket, or a directory file. The function table contains suitable function to read the file.
- The VFS software layer will invoke that function without worrying about the way of reading the data. The file can be accesses with the help of i-node and file objects.
- An i-node object is a data structure that holds pointers to the disk blocks. The actual file data is present on disk block.
- The file object denotes a point of access to the data in an open file. In order to access the i-node contents, the process first has to access a file object pointing to the i-node.
- The i-node objects do not belong to single process. Whereas file objects belong to single process.
- The i-node object is cached by the VFS to get better performance in near future access of the file. Therefore, although processes is not currently using the file, its i-node is cached by VFS.
- All cached file data are linked onto list in the file's i-node object. The i-node also keeps standard information about each file, for example the owner, size, and time most recently modified.
- Directory files are treated in a different way from other files.

- The programming interface of UNIX defines several operations on directories, for example creation, deletion, and renaming of a file in a directory.
- The system calls for these directory operations do not have need of the user open the files concerned, unlike the case for reading or writing data.
- Therefore, these directory operations are defined by VFS in the i-node object, instead of in the file object.
- The super block object represents files of entire file system.
- The operating system kernel keeps a single superblock object for each disk device mounted as a file system and each networked file system at present connected. The main duty of the superblock object is to offer access to i-nodes.
- The VFS recognize each i-node by a unique file-system / i-node number pair.
- It locates the i-node analogous to a particular i-node number by requesting the superblock object to return the i-node with that number.
- A entry object represents a directory entry that may include the name of a directory in the path name of a file (such as /usr) or the actual file.
