Operating System
(May 2018)

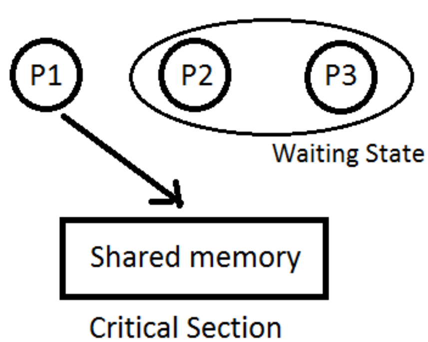
Q.1. a) Explain the difference between monolithic and micro kernel **5M**

➔

Monolithic Kernel	Micro Kernel
1. If virtually entire operating system code is executed in kernel mode, then it is a monolithic program that runs in a single address space.	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 same address space for user mode as well as kernel mode.	2. There is different address space for user mode as well as kernel mode.
3. It has a large space as compared to micro kernel.	3. It has a small space as compared to monolithic kernel.
4. Execution speed is faster than micro kernel.	4. Execution speed is slower than monolithic kernel.
5. If one service crashes whole operating system fails.	5. If one service crashes whole operating system do not fails, it does not affect working of other part micro kernel.
6. Kernel calls the function directly for communication.	6. Communication is done through message passing.

b) What is Mutual Exclusion? Explain its Significance. **5M**

➔ At the same time as one processor executes the shared variable, all remaining processes wishing to accomplish so at the same instant should be kept waiting; when that process has over executing the shared variable, one of the processes waiting to perform so should be permitted to carry on. In this manner, each process executing the shared variable keep outs all others from doing so at the same time. Only one process should be allowed to execute in critical section. This is called as Mutual Exclusion.



Significance:
1) It avoids Race around condition.
2) Prevents multiple threads to enter critical section at the same time.

c) Discuss various Scheduling Criteria. 5M

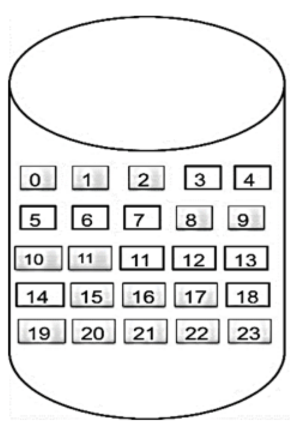
➔ A number of scheduling algorithms are being designed that can be applied to different processes having different properties. The scheduling criteria are mentioned below.
* CPU Utilization – It is amount of time CPU remains busy.
* Throughput – Number of jobs processed per unit time.
* Turnaround Time – Time elapsed between submission of jobs and completion of its execution.
* Waiting Time – Processes waits in ready queue to get CPU. Sum of times spent In ready queues waiting time.
* Response Time – Time from submission till the first response is produced.
* Fairness – Every process should get fair share of the CPU time.

d) Explain various file allocation techniques. 5M

➔ 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.



File	Start	Length
A	0	3
Hello	8	4
Marks	15	3
Photos	19	5

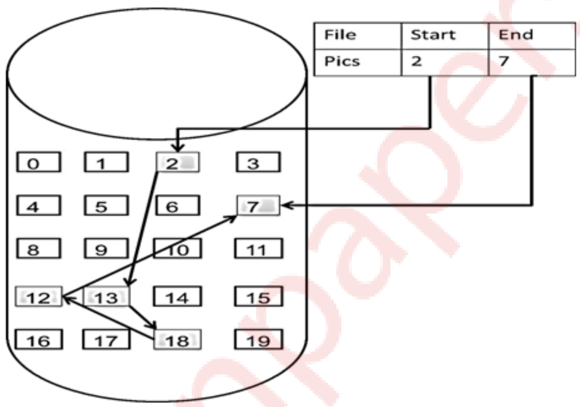
Advantage:

• 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.

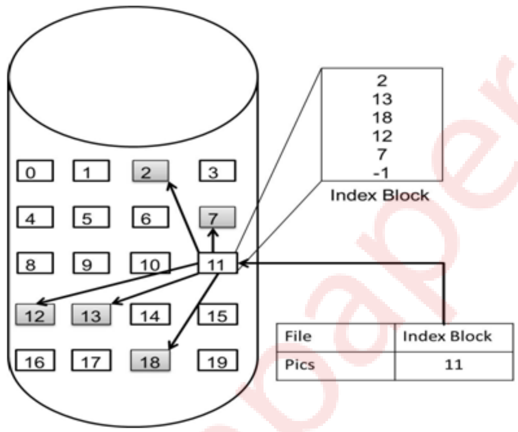
3) **Linked List allocation using table in memory:** Each block needs to store pointer information; therefore, entire block is not fully used to store file content. This limitation can be overcome by keeping pointer information in table which always remains in memory. Refer Linked list fig for Table.

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.

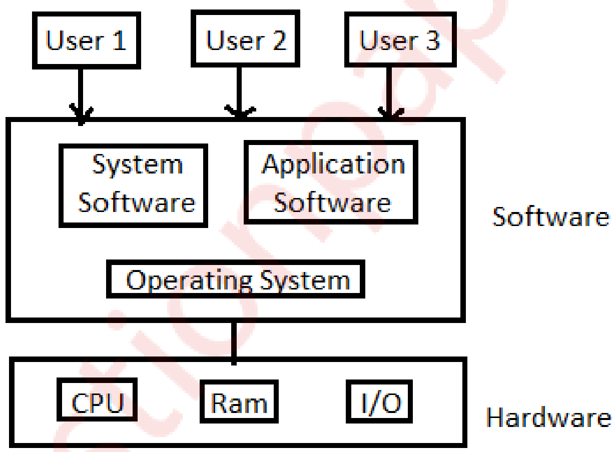
File Attributes	→	
Address of disk block 0	→	
Address of disk block 1	→	
Address of disk block 2	→	
Address of disk block 3	→	
Address of block of pointers	→	Disk Block Containing additional disk address

e) Explain the disk cache. **5M**

➔ 1) In main memory, a buffer is reserved for disk sectors called as cache.
2) The cache keeps copy of data in some of the sectors on the disk.
3) To fulfill an I/O request for a particular sector, first disk cache is checked to see whether the sector is in disk cache.
4) If it is present, then request is fulfilled via the cache.
5) If not present, then required sector is read into the disk cache from the disk.
6) The disk cache improves the performance as some request is satisfied from it.
7) The property of locality of reference is used.
8) When a block of data is present in cache to fulfill a single I/O request, it is expected that same block will be needed in future.

Q.2. a) What is operating System? Explain various functions and objectives. **10M**

➔ Definition: Operating System is an interface between user & the hardware. It is an Software used for communication between user and the hardware and also controls the execution of application programs. It is also called as "Resource Management".



Functions: Operating systems have various functions mentioned below

1) Process management –
* Creation, Execution & Deletion of user and system processes.
* Control the execution of user's application.
* Scheduling of process.
* Synchronization, inter-process communication and deadlock handling for processes.

2) Memory management -
* It allocates primary as well as secondary memory to the user and system and system process.
* Reclaims the allocated memory from all the processes that have finished its execution.
* Once used block become free, OS allocates it again to the processes.

* Monitoring and keeping track of how much memory used by the process.

3) File management -
* Creation & Deletion of files and directories.
* It allocates storage space to the files by using different file allocations methods.
* It keeps back-up of files and offers the security for files.

4) Device management -
* Device drivers are open, closed and written by OS.
* Keep an eye on device driver. communicate, control and monitor the device driver.

5) Protection & Security -
* The resources of the system are protected by the OS.
* Keeps back-up of data.
* Lock the system for security purpose by password.

Objectives: There are three objectives of operating system.

* Convenient – Because pf operating system it is very convenient to use computers.

* Efficiency – Because of operating system efficiency of computer increases. We can work efficiently. We can use resource efficiently.

* Ability to Evolve – If we add more modules in our computers then also our computer works It does not get damage.

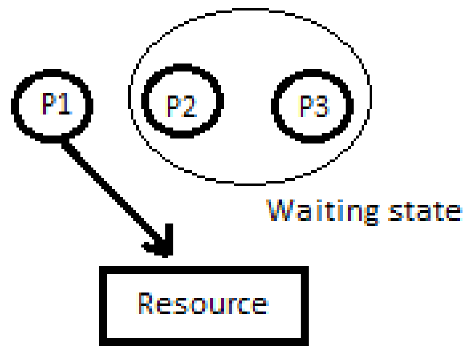
b) What is deadlock? Explain the necessary and sufficient condition for deadlock. What is the difference between deadlock avoidance and prevention? 10M

➔ 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.

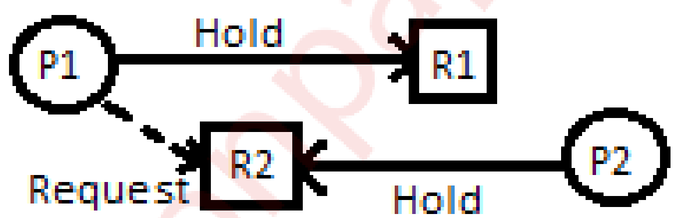
The necessary and sufficient conditions for deadlock to occur are:

- **Mutual Exclusion**
  - A resource at a time can only be used by one process.

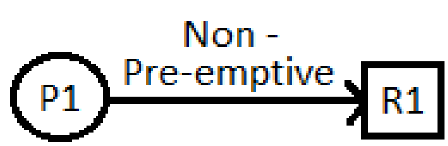
- If another process is requesting for the same resource, then it must be delayed until that resource is released.



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

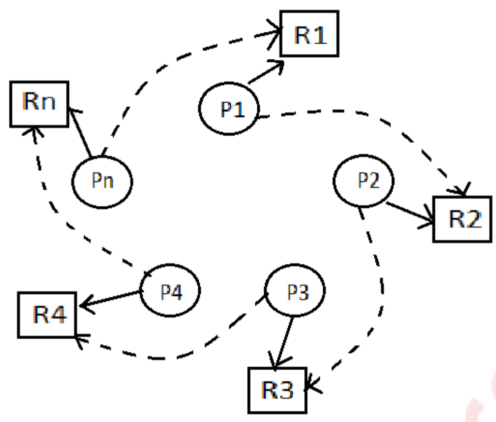


- **No Pre-emption:**
    - Resources cannot be pre-empted
    - Resource can be released only by the process currently holding it based on its voluntary decision after completing the task



- **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.

Deadlock Avoidance	Deadlock Prevention
• The system dynamically considers every request and decides whether it is safe to grant it at this point. • <u>Deadlock Avoidance Techniques:</u> A deadlock avoidance algorithm dynamically examines the resource allocation state to ensure that a circular wait condition can never exist. The resource allocation state is defined by the number of available and allocated resources, and the maximum demand of the process. There are two algorithms: 1) <u>Resource allocation graph</u>. 2) <u>Banker's algorithm</u>.	• Preventing deadlock by constraining how requests for resources can be made in system and how they are handled. • <u>Deadlock prevention Techniques:</u> <u>Mutual Exclusion</u> ◊ this condition is needed to be checked for non-sharable resources (e.g. Printer) <u>Hold and Wait</u> ◊ It requires a process to request a resource and get allocated before execution or allow process to request resources when the process has none. <u>No preemption</u> ◊ If a process that is holding some resources requests another resource that cannot be immediately allocated to it, then all

• The goal is to ensure that the resource which are allocated to the process should be in safe state to avoid deadlock. • A major drawback of this method is that it is difficult to know at the beginning itself of the maximum resource required.	resources currently being held are released Circular Wait◊we can impose a total ordering of all resource types, and ask that each process requests resources in an increasing order of enumeration. • The goal is to ensure that at least one of the above mentioned conditions for deadlock can never hold. • Disadvantage is that it can lead to low device utilization and reduced system throughput.

Q.3. a) Explain the following in brief: **10 M**

i) Process Synchronization – Process Synchronization means sharing system resources by processes in such a way that, Concurrent access to shared data is handled thereby minimizing the chance of inconsistent data.

- Each process executes its own operations on shared variables sequentially, as specified by its own program. Nevertheless, different processes may execute their operations on the same shared variable concurrently. That is, operation executions of different processes may overlap, and they may affect one another.
- Each operation on a shared variable, when executed indivisibly, transforms the variable from one consistent value to another. However, when the operations arc executed concurrently on a shared variable, the consistency of its values may not the guaranteed.
- The behaviors of operation executions on shared variables must, be predictable for effective inter-process communication.
- Thus, operation executions on shared variables may need to be coordinated to ensure their consistency semantics.
- Coordination of accesses to shared variables is called synchronization.
- A synchronization solution coordinates accesses from processes to shared variables. where all accesses to shared variables are channeled through access controllers
- The controllers do the coordination. Most operating systems implement a few different synchronization schemes for process coordination purposes. Each scheme

supports a set of primitives. The primitives are used when it is absolutely necessary to have orderly executions of operations on shared variables in a particular manner.

ii) Inter-Process Communication –

* Synchronization and communication are two basic requirements should be satisfied When processes communicate with each other.
* Synchronization of processes is required to achieve the mutual exclusion.
* Independent processes do not communicate with each other but cooperating processes may need to exchange information. Cooperative processes either communicates through shared memory or message passing.
* Cooperating processes require an inter process communication (IPC) mechanism that will allow them to exchange data and information.
* There are two fundamental models of inter process communication.

1) Shared Memory
- In the shared memory model, a region of memory that is shared by cooperating processes are established.
- Processes can then exchange information by reading and writing data to the shared region.
- In the message passing model, communication takes place by means of message exchanged between the cooperating process.

2) Message Passing
- Following are the two primitives used in message passing:
  Send (destination, message)
  Receive (Source. message)
- This is the minimum two operations required for processes to send and receive the message.
- A process sends data in the form of a message to another process indicated by destination.
- A process receive data by executing the received primitives, indicating the source and the message.

b) Consider the following set of processes, assuming all are arriving at time 0. 10M

Process	**Burst time**	**Priority**
P1	**2**	**2**
P2	**1**	**1**
P3	**8**	**4**
P4	**4**	**5**
P5	**5**	**3**

Calculate average waiting time and turn-around time for FCFS, SJF (Non-pre-emptive), Priority and RR (Quantum=2).

➔1) FCFS – Mode- Non-Pre-emptive
Criteria- Arrival Time

P1	P2	P3	P4	P5

0 2 3 11 15 20

process	Arrival Time (AT)	Burst time (BT)	Completion Time (CT)	Turn Around Time (TAT) (TAT=CT-AT)	Waiting Time (WT) (WT=TAT-BT)
P1	0	2	2	2	0
P2	0	1	3	3	2
P3	0	8	11	11	3
P4	0	4	15	15	11
P5	0	5	20	20	15

ATAT = $\frac{2+3+11+15+20}{5} = 10.2$

AWT = $\frac{0+2+3+11+15}{5} = 6.2$

2) <u>SJF (Non-Pre-emptive)</u> – Mode - Non-Pre-emptive
Criteria – Burst Time.

P2	P1	P4	P5	P3

0 1 3 7 12 20

process	Arrival Time (AT)	Burst time (BT)	Completion Time (CT)	Turn Around Time (TAT) (TAT=CT-AT)	Waiting Time (WT) (WT=TAT-BT)
P1	0	2	3	3	1
P2	0	1	1	1	0
P3	0	8	20	20	12
P4	0	4	7	7	3
P5	0	5	12	12	7

ATAT = $\frac{3+1+20+7+12}{5} = 8.6$

AWT = $\frac{1+0+12+3+7}{5} = 4.6$

3) <u>Priority</u> – Mode – Non-Pre-emptive (Solution for Pre-emptive will be same for this problem
Criteria- Priority as arrival time is same or 0)

P2	P1	P5	P3	P4

0 1 3 8 16 20

process	Arrival Time (AT)	Burst time (BT)	Priority	Completion Time (CT)	Turn Around Time (TAT) (TAT=CT-AT)	Waiting Time (WT) (WT=TAT-BT)
P1	0	2	2	3	3	1
P2	0	1	1	1	1	0
P3	0	8	4	16	16	8
P4	0	4	5	20	20	16
P5	0	5	3	8	8	3

ATAT = $\frac{3+1+16+20+8}{5} = 9.6$

AWT = $\frac{1+0+8+16+3}{5} = 5.6$

4) <u>RR(Quantum=2)</u> – Mode – Pre-emptive
Criteria – Arrival time

Queue:

P1	P2	P3	P4	P5	P3	P4	P5	P3	P5	P3

P1	P2	P3	P4	P5	P3	P4	P5	P3	P5	P3

0 2 3 5 7 9 11 13 15 17 18 20

process	Arrival Time (AT)	Burst time (BT)	Completion Time (CT)	Turn Around Time (TAT) (TAT=CT-AT)	Waiting Time (WT) (WT=TAT-BT)
P1	0	2	2	2	0
P2	0	1	3	3	2
P3	0	8	20	20	12
P4	0	4	13	13	9
P5	0	5	18	18	13

ATAT = $\frac{2+3+20+13+18}{5} = 11.2$

AWT = $\frac{0+2+12+9+13}{5} = 7.2$

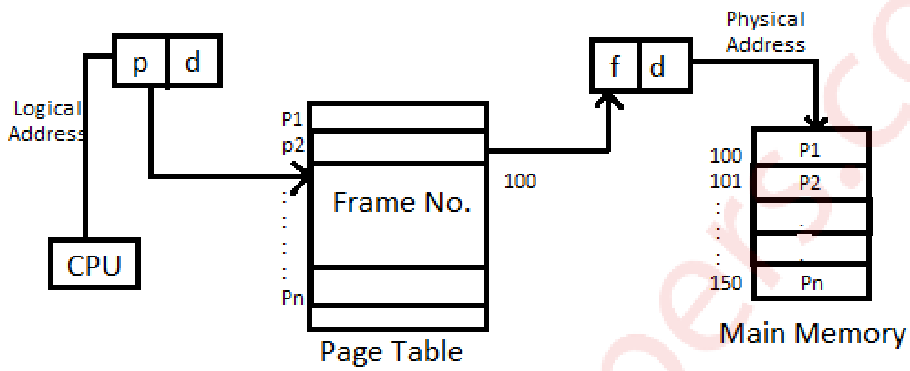
Q.4. a) What is paging? Explain LRU, FIFO and Optimal page replacement policy for the following String. Page frame size is 4.
1,2,3,4,5,3,4,1,6,7,8,7,8,9,7,8,9,5,4,5,4,2. 10M

➔ <u>Paging</u>**:**

1. The date required by the application is brought from external memory to the main memory in form of blocks(pages) This process is called as Paging.

2. Paging is a memory management scheme which allows the physical address of a process to be non-contiguous.

3. Paging concept is used to remove the problem fragmentation. Here we are able to allocate physical memory to the process in non-contiguous manner wherever memory is available.
4. Paging avoids external fragmentation and need for compaction.



There are three types of page replacement policies explained below:

1) FIFO- In this page replacement policy the page which has arrived first is removed first.
   Example:

1	**2**	**3**	**4**	**5**	**3**	**4**	**1**	**6**	**7**	**8**	**7**	**8**	**9**	**7**	**8**	**9**	**5**	**4**	**5**	**4**	**2**
1	1	1	1	5	5	5	5	5	5	8	8	8	8	8	8	8	8	8	8	8	2
	2	2	2	2	2	2	1	1	1	1	1	1	9	9	9	9	9	9	9	9	9
		3	3	3	3	3	3	6	6	6	6	6	6	6	6	6	5	5	5	5	5
			4	4	4	4	4	4	7	7	7	7	7	7	7	7	7	4	4	4	4
M	**M**	**M**	**M**	**M**	**H**	**H**	**M**	**M**	**M**	**M**	**H**	**H**	**M**	**H**	**H**	**H**	**M**	**M**	**H**	**H**	**M**

Page Hit = 9 Page Hit ratio = $\frac{9}{22}$ * 100 = 40.90%

Page Miss = 13 Page Miss ratio = $\frac{13}{22}$ * 100 = 59.10%

Advantage – Easy to implement
Disadvantage – Performance is not always good. It may suffers from belady’s anomaly.

2) LRU- In this page replacement policy we replace that page that was used farthest back in past.

Example:

1	2	3	4	5	3	4	1	6	7	8	7	8	9	7	8	9	5	4	5	4	2
1*	1	1	1	5*	5	5	5	6*	6	6	6	6	6	6	6	6	5*	5	5*	5	5
	2*	2	2	2	2	2	1*	1	1	1	1	1	9*	9	9	9*	9	9	9	9	9
		3*	3	3	3*	3	3	3	7*	7	7*	7	7	7*	7	7	7	4*	4	4*	4
			4*	4	4	4*	4	4	4	8*	8	8*	8	8	8*	8	8	8	8	8	2*
M	**M**	**M**	**M**	**M**	**H**	**H**	**M**	**M**	**M**	**M**	**H**	**H**	**M**	**H**	**H**	**H**	**M**	**M**	**H**	**H**	**M**

Page Hit = 9　　　Page Hit ratio = $\frac{9}{22}$ * 100 = 40.90%

Page Miss = 13　　　Page Miss ratio = $\frac{13}{22}$ * 100 = 59.10%

3) Optimal- In this page replacement policy we replace that page which will not be used for the longest period of time. Here we see future of that Page. If it will be used in future, then we won't replace it.
Example:

1	2	3	4	5	3	4	1	6	7	8	7	8	9	7	8	9	5	4	5	4	2
1	1	1	1	1	1	1	1	6	6	6	6	6	9	9	9	9	9	4	4	4	4
	2	2	2	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
		3	3	3	3	3	3	3	7	7	7	7	7	7	7	7	7	7	7	7	7
			4	4	4	4	4	4	4	8	8	8	8	8	8	8	8	8	8	8	2
M	**M**	**M**	**M**	**M**	**H**	**H**	**H**	**M**	**M**	**M**	**H**	**H**	**M**	**H**	**H**	**H**	**H**	**M**	**H**	**H**	**M**

Page Hit = 11　　　Page Hit ratio = $\frac{11}{22}$ * 100 = 50%

Page Miss = 11　　　Page Miss ratio = $\frac{11}{22}$ * 100 = 50%

b) Explain Banker's algorithm in detail **10M**

➔ Banker's algorithm is a deadlock avoidance algorithm. It is applicable to the resource allocation system with multiple instance of each resource type.
* It is less efficient than resource allocation graph algorithm.
* Newly entered process should declare maximum number of instance of each resource type which it may require.
* The request should not be more than total number of resource in the system.
* System checks if allocation of requested resource will leave the system in safe state. If it will the requested resources are allocated.
* If system determines that resources cannot be allocated as it will go in unsafe state, the requesting process should wait until other process free the resource.

Following Data Structure is used in Banker's algorithm:
- **A[m]:** Array A of size m shows the number of available resources.
- **Max[n][m]:** A 2-D matrix where each process stores the maximum demand of resource.
- **Allocation [n][m]:** A 2-D matrix which stores the current allocation status of all

resource types to various process in the system.

- **Need [n][m]:** A 2-D matrix which tells the current remaining resource need of each process. It is the maximum demand of a process and the current allocation status.

Algorithm:

Let process Pi → Request i

Step1: if Request i ≤ Need
Go to step2 else error

Step2: if Request i ≤ Available
Go to step3 else wait

Step3: Available i = Available i – Request i
Allocation i = Allocation i + Request i
Need i = Need i – Request i

Step4: Check if new state is in safe state or not.

Safety Algorithm

Step1: Work = Available.
Finish(i) = false

Step2: find an i such that finish[i]=false and need i ≤ work.
If no such i, go to step4

Step3: finish[i] = true
Work = work + allocation.
Go to step2

Step4: If finish[i] = True for all i
Then system is safe

Q.5. a) What is system call? Explain any five system call in details. 10M

➔ 1) The interface between OS and user programs is defined by the set of system calls that the operating system offers. System call is the call for the operating system to perform some task on behalf of the user's program. Therefore, system calls make up the interface between processes and the operating system.

2) The system calls are functions used in the kernel itself. From programmer's point view, the system call is a normal C function call.

3) Due to system call, the code is executed in the kernel so that there must be a mechanism to change the process mode from user mode to kernel mode.

System call is categorized in five groups.

Sr. No.	**Group**	**Examples**
1	Process control	End, abort, load, execute, create process, get process attributes, set process attributes, wait for time, wait event, allocate and free memory.
2	Device Manipulation	Request device, release device, read, write, reposition, get device attributes, set device attributes, logically attach or detach devices.

3	Communications	Create, delete communication connection, send, receive message, transfer status information, attach or detach devices.
4	File Manipulation	Create file, delete file, open, close, read, write, reposition, get file attributes, set file attributes.
5	Information Maintenance	Get time or date, set time or date, get system data, set system data, get process, file, or device attributes, set process, file or device attributes.

Some examples of System Calls

Open()	A program initializes access to a file in a file system using the open system call.
Read()	A program that needs to access data from a file stored in a file system uses the read system call.
Write()	It writes data from a buffer declared by the user to a given device, maybe a file. This is primary way to output data from a program by directly using a system call.
Exec()	exec is a functionality of an operating system that runs an executable file in the context of an already existing process, replacing the previous executable
Fork()	fork is an operation whereby a process creates a copy of itself. Fork is the primary method of process creation on Unix-like operating systems.

b) Explain paging hardware with TLB along with protection bits in page table. 10M

➔ The (Translation look aside buffer) TLB is similar to the page table. Important or most used pages are stored in TLB. First pages are searched in TLB if page is found then it proceeds further else it searches pages in page table. TLB reduce execution time. TLB is a piece of very fast, associative memory, capable of searching many areas of memory simultaneously. This means that many table entries can be searched at the same time for a logical-page entry. This type of memory is very expensive which means that not much of it is used; Memory Management Unit (MMUs) usually use TLBs with between 64 and 1024 entries.

- A new page table causes all the entries stored in the TLB to become useless. When a new page table is used for example, on a context switch the TLB entries must be erased to make sure the new process's logical-address space maps to the old process's physical-address space.

- Protection bits
    - With all the paging and replacement that goes on, it might be easy to forget that there needs to be protective barriers thrown up around memory pages.

- We need to prevent code from straying beyond the boundaries of its pages. Our scheme must embrace the swapping and paging ideas we have developed. Protection in a paged environment mug focus on physical-memory frames. Since every memory reference goes through the page table to reference frames, we can add protection bits for frames and store them in the page table. These bits can give certain properties for frames: read-only or read-write.

0	4	Valid
1		Invalid
2	6	Valid
3		Invalid
4		Invalid
5	9	Valid
6		invalid

- For further protection, we can add an execute bit. To support paging, we can add a valid-invalid bit, which indicates if the page being accessed is actually in memory. Note that these protection schemes are focused on the page table. Property bits are the easiest way to provide protection.
- When logical-to-physical translation is taking place, the nature and owner of the request is also analyzed.

- If the process that issues the address is not the process that owns the page or if the requested operation is something that is not permitted by the access bits, the attempt is ruled illegal and the operating system is notified. The valid-invalid bit is set when paging occurs. If the bit is set to invalid, this means the page has been swapped to virtual memory and a page fault should be triggered.

Q.6. Write short notes on (any two) 20M

a) Linux Virtual file system.

➔ 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.

b) Process Control Block

➔ Process Control Block (PCB) is the data structure used by the operating system to keep track on the processes. Any process is identified by its PCB.

- All the information associated with process is kept in PCB. There is separate PCB for each process.
- Traffic controller module keeps a track on the status of the processes. PCB's of the processes which are in the same state (ready, waiting, executing, etc) are linked together giving a specific name to the list such as ready list.
- If may processes wait for the device, the PCB's of all these processes are linked together in chain waiting for that device.
- If the processes are available in that device chain, then again it is placed back to ready state to request that device again.

Pointer	Current State
Process ID	
Priority	
Program counter	
Registers	
Event information	
List of open files	
. . . . .	

Process Control Block

a) **Pointer:** This field points to other process PCB. The scheduling list is maintained by pointer.
b) **Current State**: Currently process can be in any of the state from new, ready, executing, waiting, etc.
c) **Process ID**: Identification number of the process. Operating system assign this number to process to distinguish it from other processes.
d) **Priority:** Different process can have different priority. Priority field indicate the priority of the process.
e) **Program Counter**: The counter indicates the address of the next instruction to be executed for this process.
f) **Registers:** The registers vary in number and type, depending on the computer architecture. They include accumulators, index registers, stack pointers and general purpose registers, plus any condition code information.
g) **Event Information**: For a process in the blocked state this field contains information concerning the event for which the process is waiting.
h) **List of Open Files**: Files opening, reading, writing, closing is done by the process.

After creation of process, hardware registers and flags are set as per the details supplied by loaders or linkers. At any time, the process is blocked, the processor register's content are generally placed on the stack and the pointer to the respective stack frame is stored in the PCB. In this fashion, the hardware state can be restored when the process is scheduled and resume execution again.

c) Reader and Writer problem using semaphore.

➔ While defining the reader/writer's problem, It is assumed that, many processes only read the file(readers) and many write to the file(writers). File is shared among a many number of processes. The conditions that must be satisfied are as follows

1) Simultaneously reading of the file is allowed to many readers.
2) Writing to the file is allowed to only one writer at any given point of time.
3) Readers are not allowed to read the file while writer is writing to the file.

- in this solution, the first reader accesses the file by performing a down operation on the semaphore file. Other readers only increment a counter, read count. When readers finish the reading counter is decremented.
- when last one end by performing an up on the semaphore, permitting a blocked writer, if there is one, to write. suppose that while a reader is reading a file, another reader comes

along. Since having two readers at the same time is not a trouble, the second readers and later readers can also be allowed if they come.

- After this assumes that a writer wants to perform a write operation on the file. The writer can't be allowed to write the file, since writer is suspended. The writer will suspend until no reader is reading the file. If new reader arrives continuously with short interval and perform reading, the writer will never obtain the access to the file.
- To stop this circumstance, the program is written in a different way: when a reader comes and at the same time a writer is waiting, the reader is suspended instead of being allowed immediately.
- Now writer will wait for readers that were reading and about to finish but does not have to wait for readers that came along with him. The drawback of this solution is that it achieves less concurrency and thus lower performance.

Algorithm

Writer process

```
While(TRUE)
     {
Wait(w)
   # perform write operation
Signal(w)
     }
```

Reader Process

```
While(TRUE)
     {
Wait(m)
Read_count ++;                    //reader enters
If (Read_count == 1)
   Wait(w)                        //blocks writer
   Signal(m)
   # perform read operation
   Wait(m)
   Read_count - -;
If (Read_count == 0)              //No reader
   Signal(w)                      //Writer can write
   Signal(m)
     }
```

d) Explain disk scheduling algorithms.

➔

- 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)

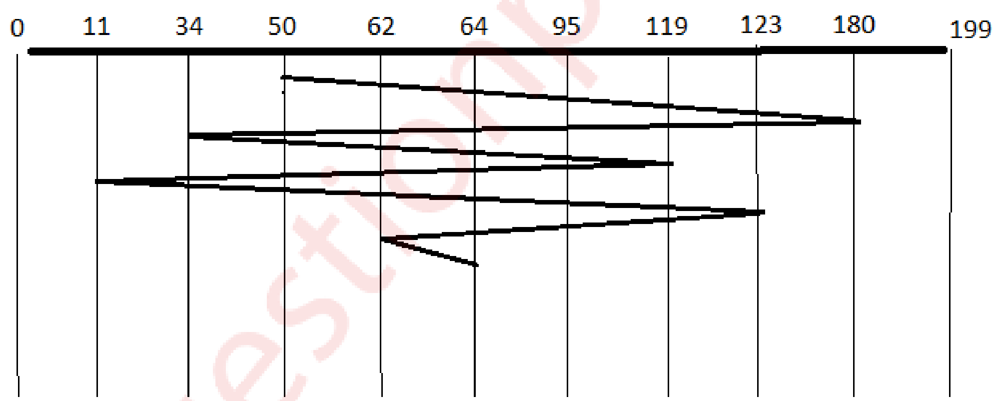
5) LOOK

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):

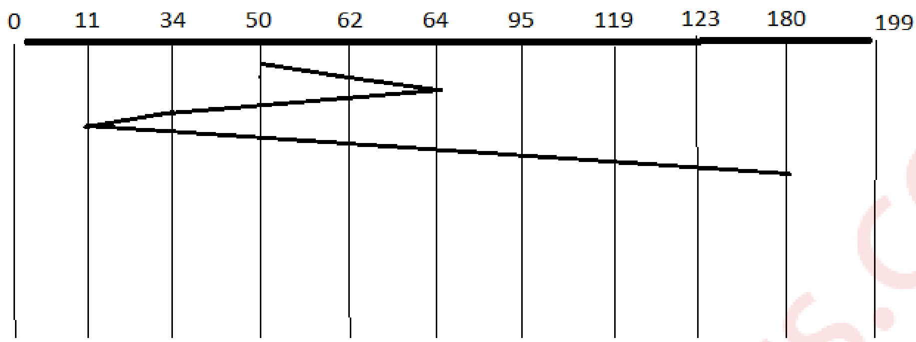
All incoming requests are placed at the end of the queue. Whatever number that is next in the queue will be the next number served. Using this algorithm doesn't provide the best results. To determine the number of head movements you would simply find the number of tracks it took to move from one request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

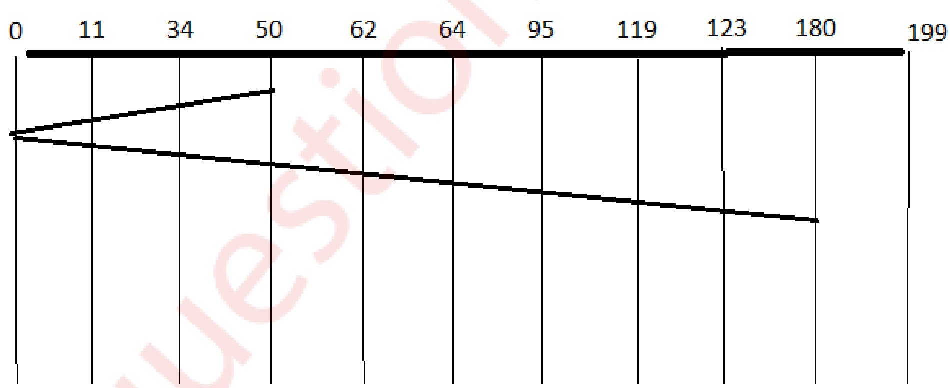
In this case request is serviced according to next shortest distance. Starting at 50, the next shortest distance would be 62 instead of 34 since it is only 12 tracks away from 62 and 16 tracks away from 34. The process would continue until all the process are taken care of. For example, the next case 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):

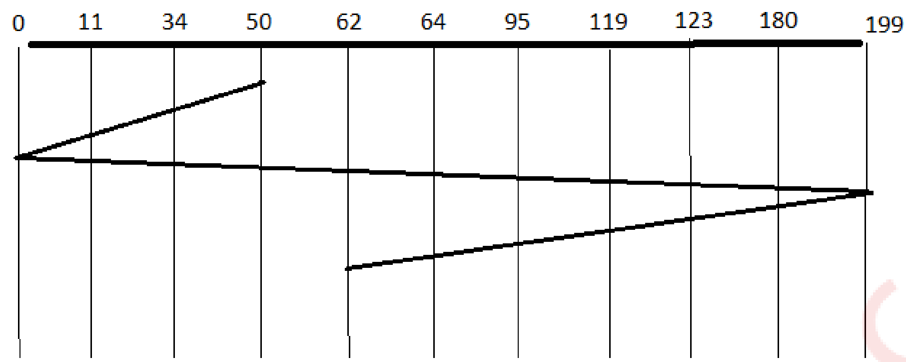
This approach works like an elevator does. It scans down towards the nearest end and then when it hits the bottom it scans up servicing the requests that it didn't get going down. If a request comes in after it has been scanned it will not be serviced until the process comes back down or moves back up.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

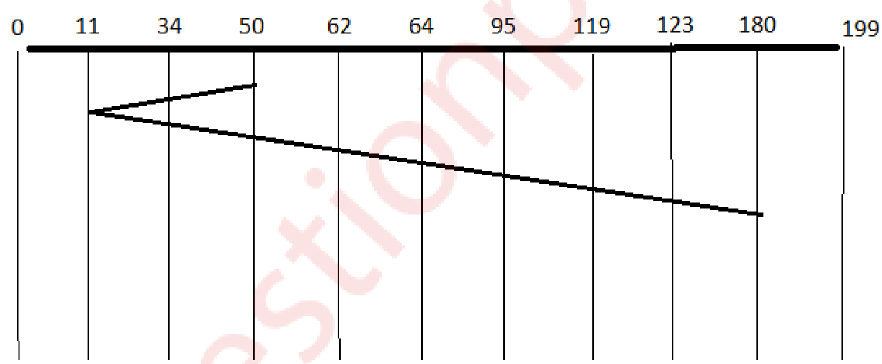
Circular scanning works just like the elevator to some extent. It begins its scan toward the nearest end and works it way all the way to the end of the system. Once it hits the bottom or top it jumps to the other end and moves in the same direction. Keep in mind that the huge jump doesn't count as a 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.

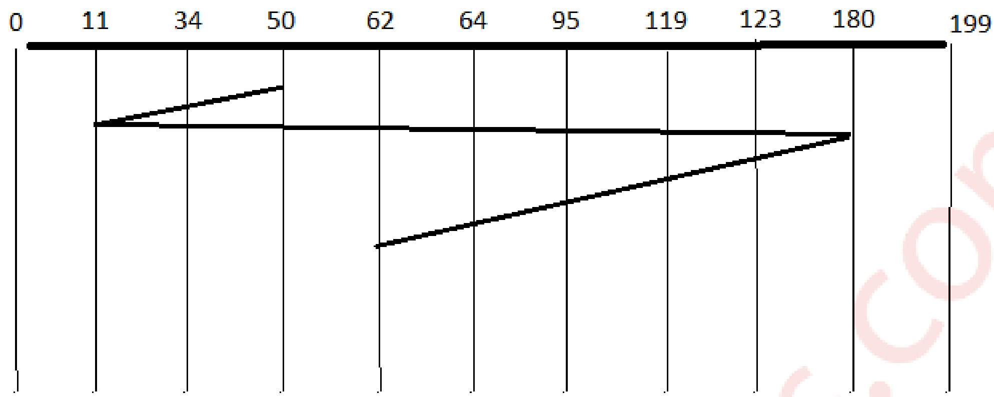


Total head moment = (50-34)+(34-11)+(62-11)+(64-62)+(95-64)+(119-95)+

(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**

****

Operating System
(Nov 2018)

Q.P. Code 55382

Q.1 Attempt any FOUR

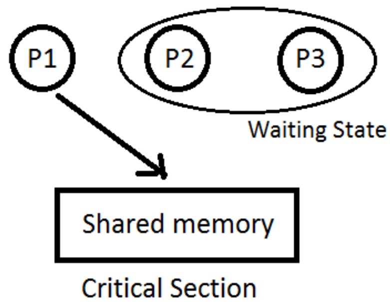
a) Explain the difference between monolithic and micro kernel **5M**

➔

Monolithic Kernel	Micro Kernel
1. If virtually entire operating system code is executed in kernel mode, then it is a monolithic program that runs in a single address space.	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 same address space for user mode as well as kernel mode.	2. There is different address space for user mode as well as kernel mode.
3. It has a large space as compared to micro kernel.	3. It has a small space as compared to monolithic kernel.
4. Execution speed is faster than micro kernel.	4. Execution speed is slower than monolithic kernel.
5. If one service crashes whole operating system fails.	5. If one service crashes whole operating system do not fails, it does not affect working of other part micro kernel.
6. Kernel calls the function directly for communication.	6. Communication is done through message passing.

b) What is Mutual Exclusion? Explain its Significance. **5M**

➔ At the same time as one processor executes the shared variable, all remaining processes wishing to accomplish so at the same instant should be kept waiting; when that process has over executing the shared variable, one of the processes waiting to perform so should be permitted to carry on. In this manner, each process executing the shared variable keep outs all others from doing so at the same time. Only one process should be allowed to execute in critical section. This is called as Mutual Exclusion.



Significance:
1) It avoids Race around condition.
2) Prevents multiple threads to enter critical section at the same time.

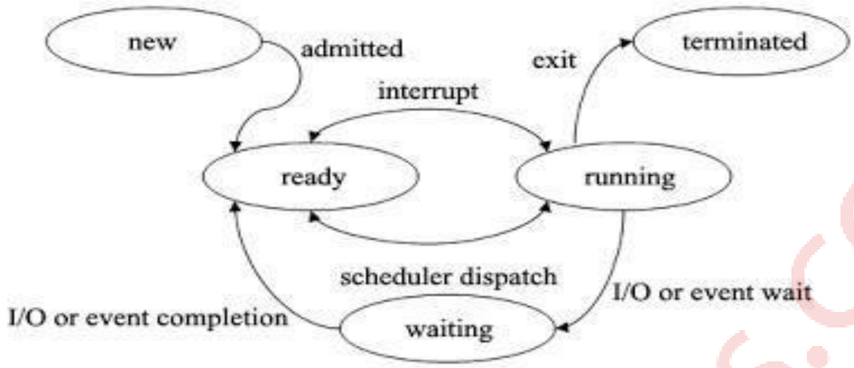
c) Discuss various types of scheduler. **5M**

➔ Different Types of Schedulers are

1. **Long Term Scheduler**
   - The job scheduler or long-term scheduler selects processes from the storage pool in the secondary memory and loads them into the ready queue in the main memory for execution.
   - The long-term scheduler controls the degree of multiprogramming. It must select a careful mixture of I/O bound and CPU bound processes to yield optimum system throughput. If it selects too many CPU bound processes, then the I/O devices are idle and if it selects too many I/O bound processes then the processor has nothing to do.
   - The job of the long-term scheduler is very important and directly affects the system for a long time.
2. **Short Term Scheduler**
   - The short-term scheduler selects one of the processes from the ready queue and schedules them for execution. A scheduling algorithm is used to decide which process will be scheduled for execution next.
   - The short-term scheduler executes much more frequently than the long-term scheduler as a process may execute only for a few milliseconds.
   - The choices of the short term scheduler are very important. If it selects a process with a long burst time, then all the processes after that will have to wait for a long time in the ready queue. This is known as starvation and it may happen if a wrong decision is made by the short-term scheduler.
3. **Medium Term Scheduler**
   - The medium-term scheduler swaps out a process from main memory. It can again swap in the process later from the point it stopped executing. This can also be called as suspending and resuming the process.
   - This is helpful in reducing the degree of multiprogramming. Swapping is also useful to improve the mix of I/O bound and CPU bound processes in the memory.

d) Explain various process states with diagram **5M**

➔



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. **Terminated state:** 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.

e) What is the effect of page size on performance of operating systems? **5M**

➔ **Effect of page size on performance**

- The number of frames is equal to the size of memory divided by the page-size. So and increase in page size means a decrease in the number of available frames.
- Having a fewer frame will increase the number of page faults because of the lower freedom in replacement choice.

- Large pages would also waste space by Internal Fragmentation.
- On the other hand, a larger page-size would draw in more memory per fault; so the number of fault may decrease if there is limited contention.
- Larger pages also reduce the number of TLB misses.

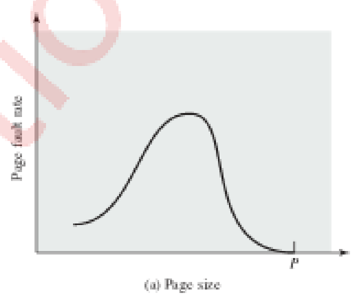
An important hardware design decision is the size of page to be used. There are several factors to consider.

Internal fragmentation:

- Clearly, the smaller the page size, the lesser is the amount of internal fragmentation. To optimize the use of main memory, we would like to reduce internal fragmentation.
- On the other hand, smaller the page, the greater is the number of pages required per process which could mean that some portion of page tables of active processes must be in virtual memory, not in main memory. This eventually leads to double page fault for a single reference of memory.

Rate at which page fault occurs:

- If the page size is very small, then ordinarily a large number of pages will be available in main memory for process, which after some time will contain portions of process near recent references leading to low page fault rate.
- As the size of page is increased, each individual page will contain locations further and further from any particular recent reference. Thus, the effect of the principle of locality is weakened and the page fault rate begins to rise.
- Eventually, however the page fault rate will begin to fall as the size of page approaches the size of the entire process.



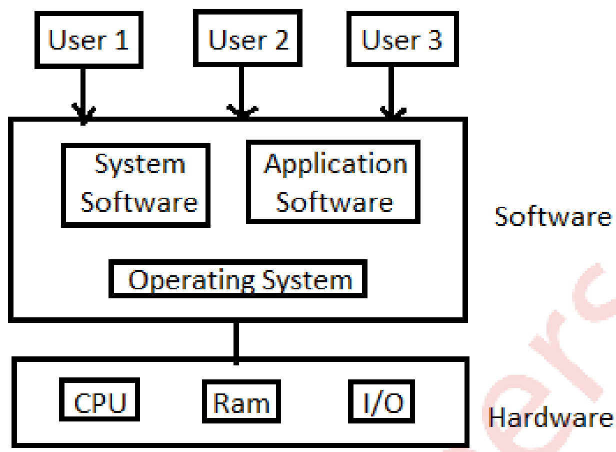
P = size 0f entire pr0cess
W = w0rking set size
N = t0tal number 0f pages in pr0cess

Size of physical main memory and program size:

- For a given size of TLB, as the memory size of processes grows and s locality decreases, the hit ratio on TLB declines. Under these circumstances, the TLB can become a performance bottleneck.
-

Q.2. a) What is operating system? Explain various functions and objectives. 10M

➔ Definition: Operating System is an interface between user & the hardware. It is an Software used for communication between user and the hardware and also controls the execution of application programs. It is also called as "Resource Management".



Functions: Operating systems have various functions mentioned below

1) Process management –
   * Creation, Execution & Deletion of user and system processes.
   * Control the execution of user's application.
   * Scheduling of process.
   * Synchronization, inter-process communication and deadlock handling for processes.

2) Memory management -
   * It allocates primary as well as secondary memory to the user and system and system process.
   * Reclaims the allocated memory from all the processes that have finished its execution.
   * Once used block become free, OS allocates it again to the processes.
   * Monitoring and keeping track of how much memory used by the process.

3) File management -
   * Creation & Deletion of files and directories.
   * It allocates storage space to the files by using different file allocations methods.
   * It keeps back-up of files and offers the security for files.

4) Device management -
   * Device drivers are open, closed and written by OS.

* Keep an eye on device driver. communicate, control and monitor the device driver.

5) Protection & Security -
* The resources of the system are protected by the OS.
* Keeps back-up of data.
* Lock the system for security purpose by password.

Objectives: There are three objectives of operating system.

* Convenient – Because pf operating system it is very convenient to use computers.

* Efficiency – Because of operating system efficiency of computer increases. We Can work efficiently. We can use resource efficiently.

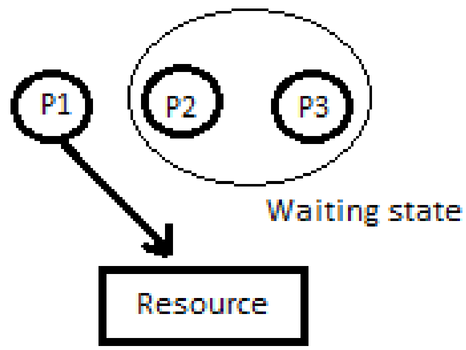
* Ability to Evolve – If we add more modules in our computers then also us Computer works It does not get damage.

b) What is deadlock? Explain the necessary and sufficient condition for deadlock. 10M

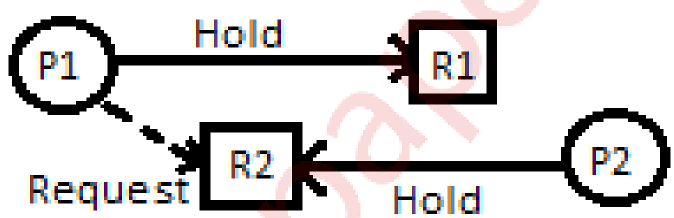
➔ Deadlock:
* We know that processes need different resources in order to complete the execution.
* So in a multiprogramming environment, many processes may compete for a multiple Number of resources.
* In a system, resources are finite. So with finite number of resources, it is not possible to fulfill 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:

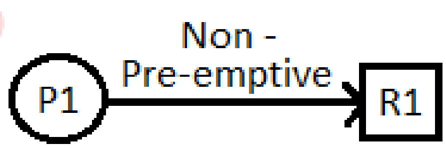
- **Mutual Exclusion**
  - A resource at a time can only be used by one process.
  - If another process is requesting for the same resource, then it must be delayed until that resource is released.



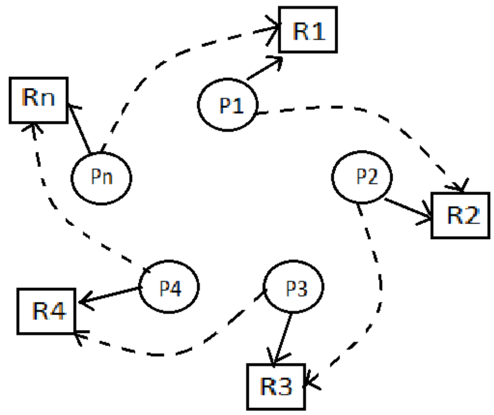
- **Hold and Wait**
  - A process is holding a resource and waiting to acquire additional resources that are currently being held by other processes.



- **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.

Q.3. a) Explain counting semaphore with example **10M**

➔

- A semaphore is hardware or a software tag variable whose value indicates the status of a common resource. Its purpose is to lock the resource being used.
- A process which needs the resource will check the semaphore for determining the status of the resource followed by the decision for proceeding.
- A semaphore is a synchronization object that controls access by multiple processes to a common resource in a parallel programming environment.
- Semaphores are widely used to control access to files and shared memory.
- Semaphores are used for mutual exclusions where the semaphore has an initial value of one, and P () and V () are called before and after the critical sections.

There are two types of semaphores:

1) Binary Semaphore

2) Counting Semaphore

Counting Semaphore:

- Counting Semaphore may have value to be greater than one, typically used to allocate resources from a pool of identical resources.
- A counting semaphore is a synchronization object that can have an arbitrarily large number of states. The internal state is defined by a signed integer variable, the counter. The counter value (N) has a precise meaning:

a. **Negative**, there are exactly -N threads queued on the semaphore.

b. **Zero**, no waiting threads, a wait operation would put in queue the invoking thread.

c. **Positive** no waiting threads, a wait operation would not put in queue the invoking thread.

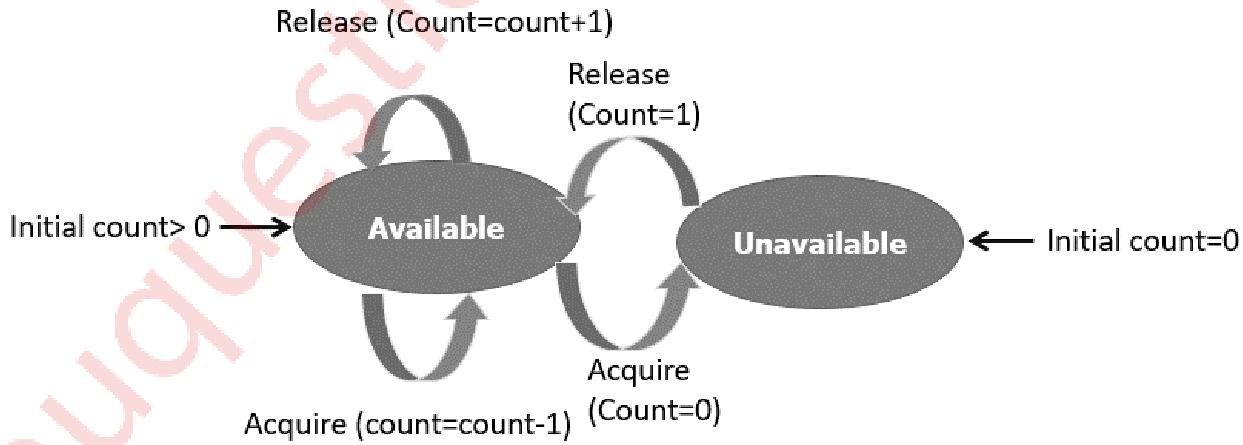
Two operations are defined for counting semaphores:

- **Wait:** This operation decreases the semaphore counter; if the result is negative then the invoking thread is queued. After the semaphore value is decreased, which becomes negative, the command is held up until the required conditions are satisfied.

```
Copy CodeP(S)
{
   while (S<=0);
   S--;
}
```

- **Signal:** This operation increases the semaphore counter, if the result is non-negative then a waiting thread is removed from the queue and resumed.

```
Copy CodeP(S)
{
   while (S>=0);
   S++;
}
```



- Counting semaphores have no ownership attribute and can be signaled by any thread or interrupt handler regardless of who performed the last wait operation.
- Because there is no ownership concept a counting semaphore object can be created with any initial counter value as long it is non-negative.

- The counting semaphores are usually used as guards of resources available in a discrete quantity. For example, the counter may represent the number of used slots into a circular queue, producer threads would “signal” the semaphores when inserting items in the queue, consumer threads would “wait” for an item to appear in queue, this would ensure that no consumer would be able to fetch an item from the queue if there are no items available. Note that this is exactly how I/O queues are implemented in ChibiOS/RT, very convenient.

Example of Semaphore

The below-given program is a step by step implementation, which involves usage and declaration of semaphore.

```
Shared var mutex: semaphore = 1;
Process i
  begin
  .
  .
  P(mutex);
  execute CS;
  V(mutex);
  .
  .
  End;
```

b) Consider the processes P1, P2, P3, P4 given in below table, arrives for execution in the same order, with Arrival Time 0, and given Burst Time. Draw the Gantt chart and find the average waiting time using the FCFS and SJF (Non-Pre-emptive) scheduling algorithm.

10M

process	Burst time
P0	**21**
P1	**3**
P2	**6**
P3	**2**

➔ 1) FCFS – Mode- Non-Pre-emptive
Criteria- Arrival Time

P0	P1	P2	P3

0 21 24 30 32

process	Arrival Time (AT)	Burst time (BT)	Completion Time (CT)	Turn Around Time (TAT) (TAT=CT-AT)	Waiting Time (WT) (WT=TAT-BT)
P0	0	21	21	21	0
P1	0	3	24	24	21
P2	0	6	30	30	24
P3	0	2	32	32	30

ATAT = $\frac{21+24+30+32}{4}$ = 26.75

AWT = $\frac{0+21+24+30}{4}$ = 18.75

2) SJF (Non-Pre-emptive) – Mode - Non-Pre-emptive
Criteria – Burst Time.

P3	P1	P2	P0

0 2 5 11 32

process	Arrival Time (AT)	Burst time (BT)	Completion Time (CT)	Turn Around Time (TAT) (TAT=CT-AT)	Waiting Time (WT) (WT=TAT-BT)
P0	0	21	32	32	11
P1	0	3	5	5	2
P2	0	6	11	11	5
P3	0	2	2	2	0

ATAT = $\frac{32+5+11+2}{4}$ = 12.5

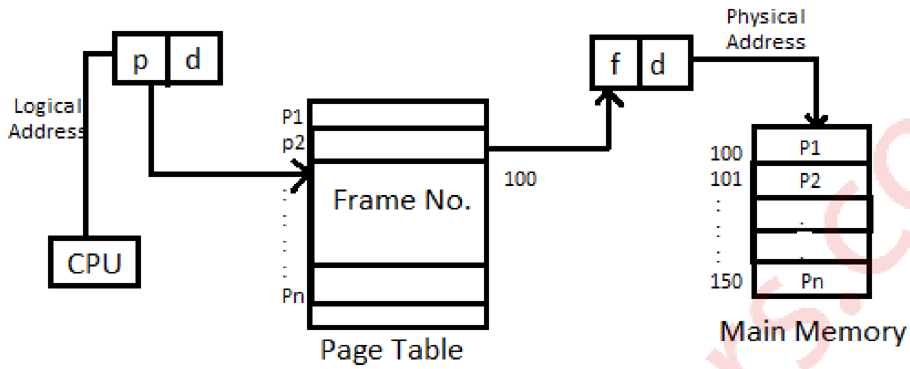
AWT = $\frac{11+2+5+0}{4}$ = 4.5

Q.4. a) What is paging? Explain LRU, FIFO and Optimal page replacement policy for the following String. Page frame size is 4.
1,2,3,4,5,3,4,1,6,7,8,7,8,9,7,8,9,5,4,5,4,2. **10M**

➔ Paging:

1. The date required by the application is brought from external memory to the main memory in form of blocks(pages) This process is called as Paging.
2. Paging is a memory management scheme which allows the physical address of a process to be non-contiguous.

3. Paging concept is used to remove the problem fragmentation. Here we are able to allocate physical memory to the process in non-contiguous manner wherever memory is available.
4. Paging avoids external fragmentation and need for compaction.



There are three types of page replacement policies explained below:

1) FIFO- In this page replacement policy the page which has arrived first is removed first.

Example:

1	**2**	**3**	**4**	**5**	**3**	**4**	**1**	**6**	**7**	**8**	**7**	**8**	**9**	**7**	**8**	**9**	**5**	**4**	**5**	**4**	**2**
1	1	1	1	5	5	5	5	5	5	8	8	8	8	8	8	8	8	8	8	8	2
	2	2	2	2	2	2	1	1	1	1	1	1	9	9	9	9	9	9	9	9	9
		3	3	3	3	3	3	6	6	6	6	6	6	6	6	6	5	5	5	5	5
			4	4	4	4	4	4	7	7	7	7	7	7	7	7	7	4	4	4	4
M	**M**	**M**	**M**	**M**	**H**	**H**	**M**	**M**	**M**	**M**	**H**	**H**	**M**	**H**	**H**	**H**	**M**	**M**	**H**	**H**	**M**

Page Hit = 9 Page Hit ratio = $\frac{9}{22}$ * 100 = 40.90%

Page Miss = 13 Page Miss ratio = $\frac{13}{22}$ * 100 = 59.10%

Advantage – Easy to implement

Disadvantage – Performance is not always good. It may suffer from belady's anomaly.

2) LRU- In this page replacement policy we replace that page that was used farthest back in past.

Example:

1	**2**	**3**	**4**	**5**	**3**	**4**	**1**	**6**	**7**	**8**	**7**	**8**	**9**	**7**	**8**	**9**	**5**	**4**	**5**	**4**	**2**
1*	1	1	1	5*	5	5	5	6*	6	6	6	6	6	6	6	6	5*	5	5*	5	5
	2*	2	2	2	2	2	1*	1	1	1	1	1	9*	9	9	9*	9	9	9	9	9
		3*	3	3	3*	3	3	3	7*	7	7*	7	7	7*	7	7	7	4*	4	4*	4
			4*	4	4	4*	4	4	4	8*	8	8*	8	8	8*	8	8	8	8	8	2*
M	**M**	**M**	**M**	**M**	**H**	**H**	**M**	**M**	**M**	**M**	**H**	**H**	**M**	**H**	**H**	**H**	**M**	**M**	**H**	**H**	**M**

Page Hit = 9 Page Hit ratio = $\frac{9}{22}$ * 100 = 40.90%

Page Miss = 13 Page Miss ratio = $\frac{13}{22}$ * 100 = 59.10%

3) Optimal- In this page replacement policy we replace that page which will not be used for the longest period of time. Here we see future of that Page. If it will be used in future, then we won't replace it.
Example:

1	**2**	**3**	**4**	**5**	**3**	**4**	**1**	**6**	**7**	**8**	**7**	**8**	**9**	**7**	**8**	**9**	**5**	**4**	**5**	**4**	**2**
1	1	1	1	1	1	1	1	6	6	6	6	6	9	9	9	9	9	4	4	4	4
	2	2	2	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
		3	3	3	3	3	3	3	7	7	7	7	7	7	7	7	7	7	7	7	7
			4	4	4	4	4	4	4	8	8	8	8	8	8	8	8	8	8	8	2
M	**M**	**M**	**M**	**M**	**H**	**H**	**H**	**M**	**M**	**M**	**H**	**H**	**M**	**H**	**H**	**H**	**H**	**M**	**H**	**H**	**M**

Page Hit = 11 Page Hit ratio = $\frac{11}{22}$ * 100 = 50%

Page Miss = 11 Page Miss ratio = $\frac{11}{22}$ * 100 = 50%

b) Explain data structures used in banker's algorithms with example. 10M

➔

- Banker's Algorithm is a deadlock avoidance algorithm that checks for safe or unsafe state of a System after allocating resources to a process.
- When a new process enters into system, it must declare maximum no. of instances of each resource that it may need. After requesting operating system run banker's algorithm to check whether after allocating requested resources, system goes into deadlock state or not. If yes, then it will deny the request of resources made by process else it allocate resources to that process.
- No. of requested resources (instances of each resource) may not exceed no. of available resources in operating system and when a process completes it must release all the requested and already allocated resources.
- Following **Data structures** are used to implement the Banker's Algorithm:

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

Available:

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

Max:

- It is a 2-d array of size '**n*m'** that defines the maximum demand of each process in a system.

- Max[i, j] = k means process $\mathbf{P_i}$ may request at most **'k'** instances of resource type $\mathbf{R_j}$.

Allocation:

- It is a 2-d array of size **'n*m'** that defines the number of resources of each type currently allocated to each process.

- Allocation[i, j] = k means process $\mathbf{P_i}$ is currently allocated **'k'** instances of resource type $\mathbf{R_j}$

Need:

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

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

Example

Consider the following system snapshot using data structure in the Banker's algorithm, with resource A,B,C,D and Process P0 to P4.

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

Check whether the system is in safe state?

If a request from process P4 arrived for additional resources of (1,2,0,0) can Banker's Algorithm grant the request immediately? Show the new system state and other criteria.

Max – Allocation=Need.

	MAX			
	A	B	C	D
P0	6	0	1	2
P1	1	7	5	0
P2	2	3	5	6
P3	1	6	5	3
P4	1	6	5	6

	ALLOCATION			
	A	B	C	D
P0	4	0	0	1
P1	1	1	0	0
P2	1	2	5	4
P3	0	6	3	3
P4	0	2	1	2

	NEED			
	A	B	C	D
P0	2	0	1	1
P1	0	6	5	0
P2	1	1	0	2
P3	1	0	2	0
P4	1	4	4	4

Initialize the array Finish and Work as follows and apply safety algorithm.
Finish = {False, False, False, False, False};
Work = {3, 2, 1, 1};

Is Need of P0 <= Work => {2, 0, 1, 1} <= {3, 2, 1, 1} =>True
So Finish= {True, False, False, False, False};
Work = {3, 2, 1, 1}+{4,0,0,1}(Allocation of P0)= {7,2,1,2}
Safe Sequence= {P0}

Is Need of P1 <= Work => {0,6,5,0} <= {7, 2, 1, 2} => False
So Finish= {True, False, False, False, False};

Is Need of P2 <= Work => {1, 1, 0, 2} <= {7, 2, 1, 2} =>True
So Finish= {True, False, True, False, False};
Work = {7, 2, 1, 2}+{1, 2, 5, 4}(Allocation of P2) = {8,4,6,6}
Safe Sequence= {P0, P2}

Is Need of P3 <= Work => {1, 0, 2, 0} <= {8,4,6,6} =>True
So Finish = {True, False, True, True, False};
Work = {8,4,6,6}+{0, 6, 3, 3}(Allocation of P3)= {8,10, 9, 9}
Safe Sequence= {P0,P2,P3}

Is Need of P4 <= Work => {1, 4, 4, 4} <= {8,10, 9, 9} =>True
So Finish = {True, False, True, True, True};
Work = {8,10, 9, 9}+{0, 2, 1, 2}(Allocation of P4)= {8,12,10,11}
Safe Sequence= {P0,P2,P3,P4}

Is Need of P1<= Work => {0,6,5,0} <= {8,12,10,11}} =>True
So Finish= {True, True, True, True, True};
Work = {8,12,10,11}}+{1,1,0,0}(Allocation of P1)= {9,13,10,11}
Safe Sequence= {P0,P2,P3,P4,P1}

Yes System is in safe state.

Q.5. a) What is system call? Explain any five system call in details. 10M

➔ 1) The interface between OS and user programs is defined by the set of system calls that the operating system offers. System call is the call for the operating system to perform some task on behalf of the user's program. Therefore, system calls make up the interface between processes and the operating system.

2) The system calls are functions used in the kernel itself. From programmer's point view, the system call is a normal C function call.

3) Due to system call, the code is executed in the kernel so that there must be a mechanism to change the process mode from user mode to kernel mode.

System call is categorized in five groups.

Sr. No.	Group	Examples
1	Process control	End, abort, load, execute, create process, get process attributes, set process attributes, wait for time, wait event, allocate and free memory.
2	Device Manipulation	Request device, release device, read, write, reposition, get device attributes, set device attributes, logically attach or detach devices.
3	Communications	Create, delete communication connection, send, receive message, transfer status information, attach or detach devices.
4	File Manipulation	Create file, delete file, open, close, read, write, reposition, get file attributes, set file attributes.
5	Information Maintenance	Get time or date, set time or date, get system data, set system data, get process, file, or device attributes, set process, file or device attributes.

Some examples of System Calls

Open()	A program initializes access to a file in a file system using the open system call.
Read()	A program that needs to access data from a file stored in a file system uses the read system call.
Write()	It writes data from a buffer declared by the user to a given device, maybe a file. This is primary way to output data from a program by directly using a system call.
Exec()	exec is a functionality of an operating system that runs an executable file in the context of an already existing process, replacing the previous executable
Fork()	fork is an operation whereby a process creates a copy of itself. Fork is the primary method of process creation on Unix-like operating systems.

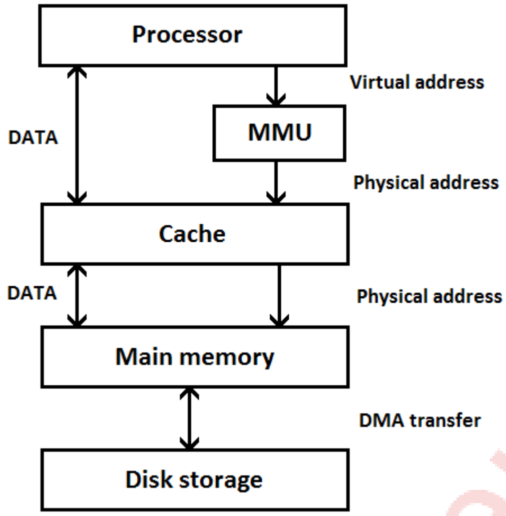
b) Explain virtual memory concept with respect to paging, segmentation and TLB.

➔ **10M**

Virtual Memory:

- In the most computer system, the physical main memory is not as large as address space of the processor.
- When we try to run a program, if it do not completely fit into the main memory the parts of its currently being executed are stored in main memory and remaining portion is stored in secondary storage device such as HDD.
- When a new part of program is to be brought into main memory for execution and if the memory is full, it must replace another part which is already is in main memory.
- As this secondary memory is not actually part of system memory, so for CPU, secondary memory is Virtual Memory.
- Techniques that automatically more program and data blocks into physical memory when they are required for execution are called virtual memory
- Virtual Memory is used to logically extend the size of main memory.
- When Virtual Memory is used, the address field is virtual address.
- A special hardware unit knows as MMU translates Virtual Address into Physical Address.

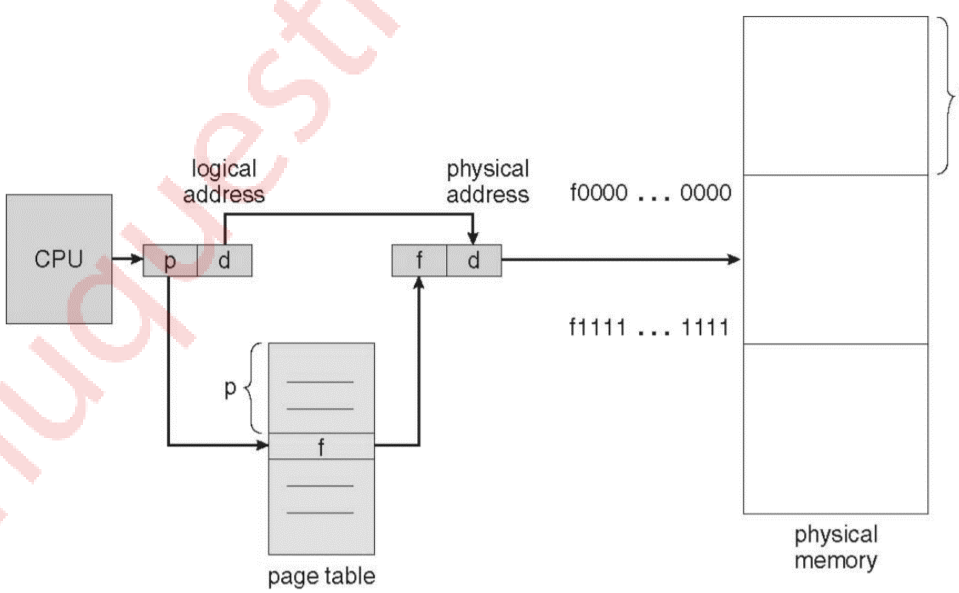
Memory Management Unit



- Address Translation is done by two techniques
  - Paging
  - Segmentation

1.Paging:

- Physical memory is divided into fixed size block know as Frames.
- Logical Memory is divided into blocks of same size knows as Pages.
- When a process is to be executed, its pages are loaded into available memory - - - Paging Hardware:

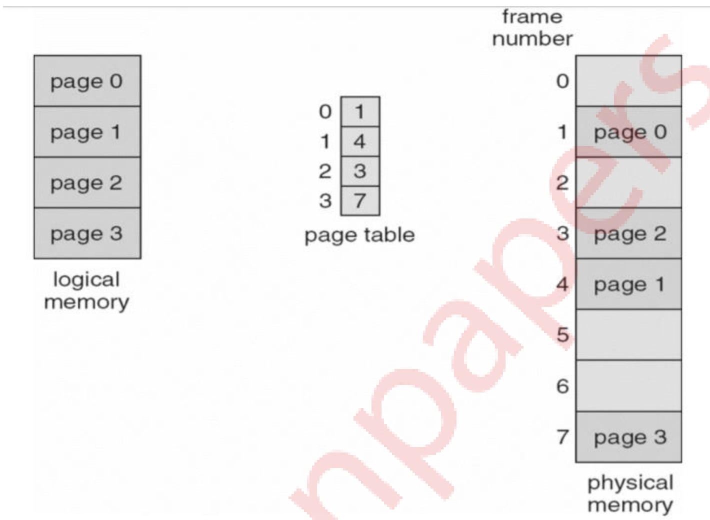


Physical Memory

Page Table (Base Register)

- Every address generated by CPU is divided into two parts:
    - Page Number (P)
    - Displacement/Offset (d)
- The page number is used as index into a page table from page table contains base address (f) of each page in physical memory.
- This base address (f) is combined with the page offset (d) to define the physical memory address.

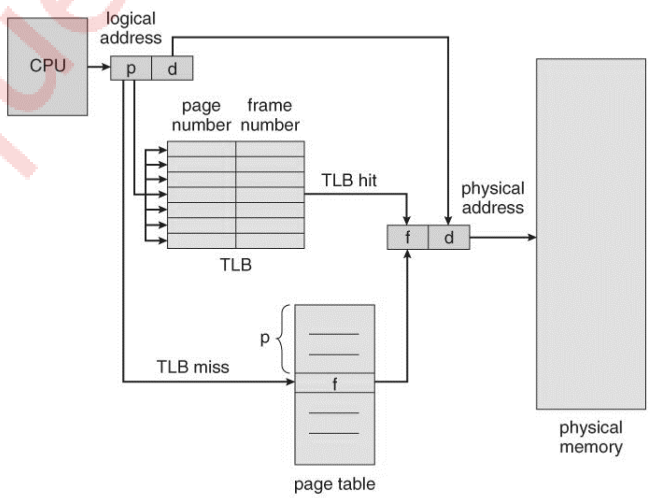
Paging Model of Logical & Physical Memory:



Disadvantages:

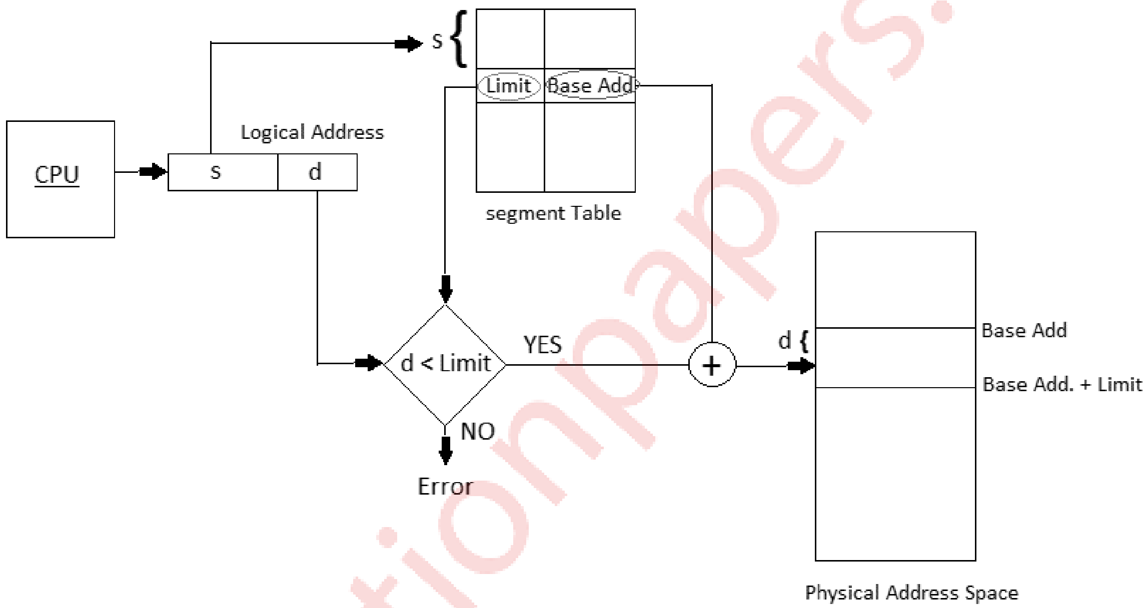
- This approach slow down the memory access by faster of 2.
- So the solution to this problem is to use special small fast cache know as translation Look Aside Buffer (TLB)

Paging Hardware with TLB:



- The TLB contains only few of page table entries.
- When logical address is generated by CPU its page number is used to index the TLB.
- If page number is found, frame number is obtained and we can access memory and is a TLB Hit.
- If page is not in TLB, then it is TLB miss and Reference is masked to page task.
- If TLB is already FULL of entries, then Operating System must select Replacement Policy.
- If the required page is not in main memory, the page must be brought from secondary to main memory.

2.**Segmentation:**



- The mapping is done with help of segment table. Each entry of segment table has base and limits.
- The segment base contains starting physical address where resides in memory whereas limit specifies length of the segments.
- The segment number is used as index for segment table.
- The offset must be between O and limits.
- If it is not less than limit then it is trapped (addressing the error).
- If it is less than limit, then add it to segment base to produce address in physical memory.

Q.6. Write short notes on: (any two): 20M

a) Linux Virtual file system

➔

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) <u>I-node Object</u> – An individual file is represented by I-node Object.
    2) <u>File Object</u> – An open file is represented by file object.
    3) <u>Superblock Object</u> – An entire file system is represented by a Superblock Object.
    4) <u>Dentry Object</u> – 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.

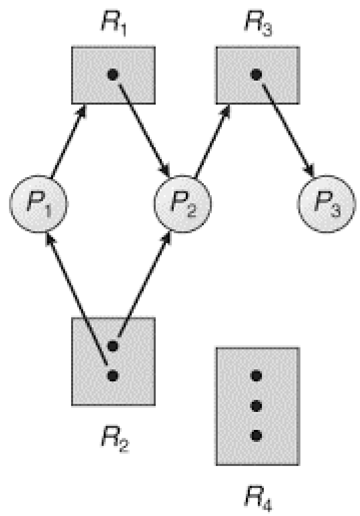
- The super block object represents files of entire file system.
- The operating system kernel keeps a single superblock object for each disk device mounted as a file system and each networked file system at present connected. The main duty of the superblock object is to offer access to i-nodes.
- The 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.

b) Resource Allocation Graph

➔ As <u>Banker's algorithm</u> using some kind of table like allocation, request, available all that thing to understand what is the state of the system. Similarly, if you want to understand the state of the system instead of using those table, actually tables are very easy to represent and understand it, but then still you could even represent the same information in the graph. That graph is called **Resource Allocation Graph (RAG)**.
So, resource allocation graph is explained to us what is the state of the system in terms of **processes and resources**. Like how many resources are available, how many are allocated and what is the request of each process. Everything can be represented in terms of the diagram. One of the advantages of having a diagram is, sometimes it is possible to see a deadlock directly by using RAG, but then you might not be able to know that by looking at the table. But the tables are better if the system contains lots of process and resource and Graph is better if the system contains less number of process and resource.

Resource Allocation Graph

- Deadlock can be described through a resource allocation graph.
- The RAG consists of a set of vertices P={P1,P2 ,…,P n} of processes and R={R1,R2,…,Rm} of resources.
- A directed edge from a processes to a resource, Pi->R j, implies that Pi has requested Rj.
- A directed edge from a resource to a process, Rj->Pi, implies that Rj has been allocated by Pi.
- If the graph has no cycles, deadlock cannot exist. If the graph has a cycle, deadlock may exist.



Components Of RAG-

There are two major components of a Resource Allocation Graph-

1. Vertices
2. Edges

1. Vertices- Vertices are of two types process vertices and Resource vertices

Process Vertices

- Process vertices represent the processes.
- They are drawn as a circle by mentioning the name of process inside the circle.

Resource Vertices-

- Resource vertices represent the resources.
- Depending on the number of instances that exists in the system, resource vertices may be single instance or multiple instance.
- They are drawn as a rectangle by mentioning the dots inside the rectangle.
- The number of dots inside the rectangle indicates the number of instances of that resource existing in the system.

They are further classified into

- **Single instance type resource –** It represents as a box, inside the box, there will be one dot. So the number of dots indicate how many instances are present of each resource type.
- **Multi-resource instance type resource –** It also represents as a box, inside the box, there will be many dots present.

2. Edges- There are two types of edges in a Resource Allocation Graph Assign Edges and Request Edges

Assign Edges-

- Assign edges represent the assignment of resources to the processes.
- They are drawn as an arrow where the head of the arrow points to the process and tail of the process points to the instance of the resource.

Request Edges-

- Request edges represent the waiting state of processes for the resources.
- They are drawn as an arrow where the head of the arrow points to the instance of the resource and tail of the process points to the process.
- If a process requires 'n' instances of a resource type, then 'n' assign edges will be drawn.

c) Readers and Writer problem using Semaphore

➔ While defining the reader/writer's problem, It is assumed that, many processes only read the file(readers) and many write to the file(writers). File is shared among a many number of processes. The conditions that must be satisfied are as follows

1) Simultaneously reading of the file is allowed to many readers.
2) Writing to the file is allowed to only one writer at any given point of time.
3) Readers are not allowed to read the file while writer is writing to the file.

- in this solution, the first reader accesses the file by performing a down operation on the semaphore file. Other readers only increment a counter, read count. When readers finish the reading counter is decremented.
- when last one end by performing an up on the semaphore, permitting a blocked writer, if there is one, to write. suppose that while a reader is reading a file, another reader comes along. Since having two readers at the same time is not a trouble, the second readers and later readers can also be allowed if they come.
- After this assumes that a writer wants to perform a write operation on the file. The writer can't be allowed to write the file, since writer is suspended. The writer will suspend until no reader is reading the file. If new reader arrives continuously with short interval and perform reading, the writer will never obtain the access to the file.
- To stop this circumstance, the program is written in a different way: when a reader comes and at the same time a writer is waiting, the reader is suspended instead of being allowed immediately.
- Now writer will wait for readers that were reading and about to finish but does not have to wait for readers that came along with him. The drawback of this solution is that it achieves less concurrency and thus lower performance.

Algorithm

Writer process

```
While(TRUE)
    {
Wait(w)
  # perform write operation
Signal(w)
    }
```

Reader Process

```
While(TRUE)
    {
Wait(m)
Read_count ++;                    //reader enters
If (Read_count == 1)
  Wait(w)                         //blocks writer
  Signal(m)
  # perform read operation
  Wait(m)
  Read_count - -;
If (Read_count == 0)              //No reader
  Signal(w)                       //Writer can write
  Signal(m)
    }
```

d) Compare disk scheduling algorithms.

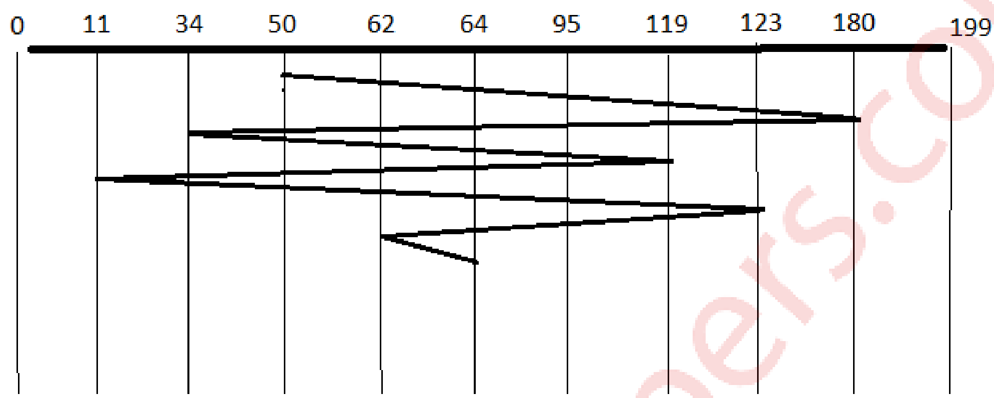
➔ In operating systems, seek time is very important. Since all device requests are linked in queues, the seek time is increased causing the system to slow down. Disk Scheduling Algorithms are used to reduce the total seek time of any request.

Let us compare **various disk scheduling algorithms**:

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**

Pros:

- Simple
- Not complex
- Easy to implement
- No starvation
- Low overhead
- No indefinite delay
- Low overhead

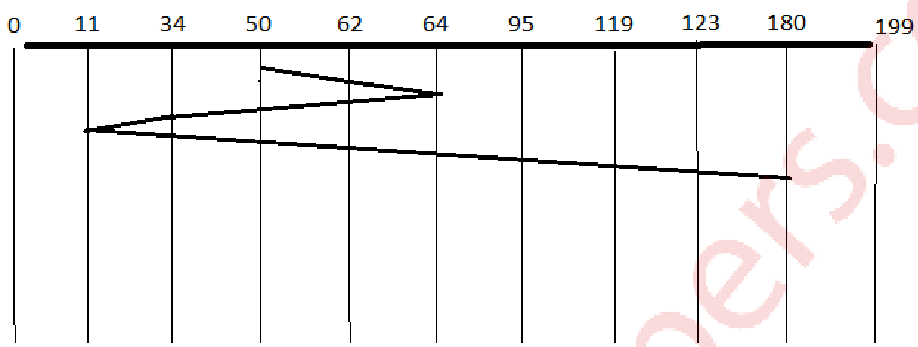
Cons:

- No preemption possible
- Low throughput
- Best services may not be delivered

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**

Pros:

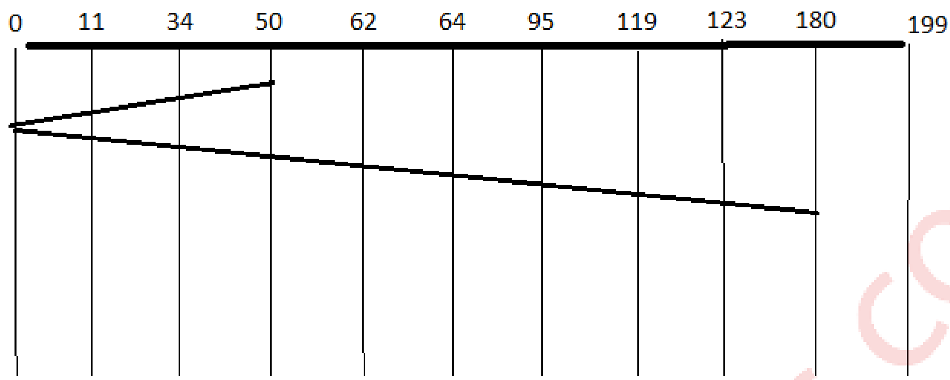
- The average time taken for response is reduced
- Many processes can be processed
- An increase in throughput

Cons:

- Starvation
- Different time is taken for different responses
- Overhead

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**

Pros:

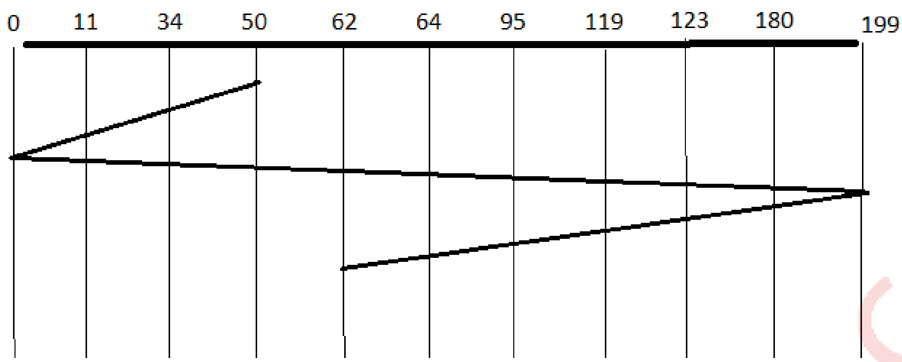
- High throughput
- Almost similar response times

Cons:

- Long waiting times

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**

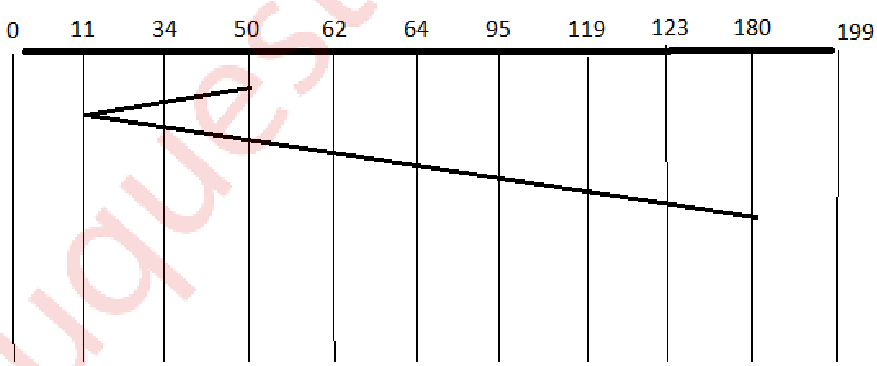
Pros:

- Provides additional invariable wait time related to SCAN algorithm

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

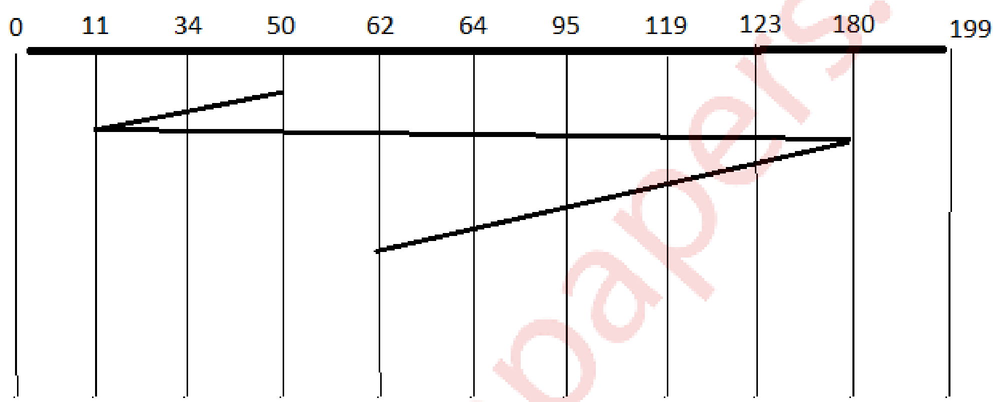
Moment.



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**

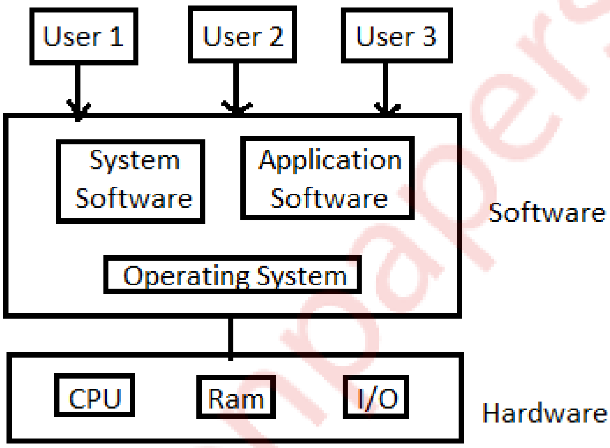
Operating System

(May 2019)

Q-1

a. Define Operating System. Brief the Functions of OS. **5**

➔ Definition: Operating System is an interface between user & the hardware. It is an Software used for communication between user and the hardware and also controls the execution of application programs. It is also called as “Resource Management”.



Functions: Operating systems have various functions mentioned below

1) Process management –
   * Creation, Execution & Deletion of user and system processes.
   * Control the execution of user’s application.
   * Scheduling of process.
   * Synchronization, inter-process communication and deadlock handling for processes.

2) Memory management -
   * It allocates primary as well as secondary memory to the user and system and system process.
   * Reclaims the allocated memory from all the processes that have finished its execution.
   * Once used block become free, OS allocates it again to the processes.
   * Monitoring and keeping track of how much memory used by the process.

3) File management -

* Creation & Deletion of files and directories.
* It allocates storage space to the files by using different file allocations methods.
* It keeps back-up of files and offers the security for files.

4) Device management -

* Device drivers are open, closed and written by OS.
* Keep an eye on device driver. communicate, control and monitor the device driver.

5) Protection & Security -

* The resources of the system are protected by the OS.
* Keeps back-up of data.
* Lock the system for security purpose by password.

b. Explain Shell. Explain use of chmod command in linux. **5**

➔Shell

Shell is a system in which we can run our commands, programs, and shell scripts, There are different flavours of a shell, just as there are different flavour's of operating systems. Each flavour of shell has its own set of recognizable commands and functions.

Chmod

chmod – It change file mode bits.

- chmod changes the permissions of each given file according to given mode, where the mode describes the permissions to modify. Mode can be specified with octal numbers or with letters.

- In Linux, who can do what to a file or directory is controlled through sets of permissions. There are three sets of permissions. One set for the owner of the file, another set for the members of the file's group, and a final set for everyone else.

- In Linux operating systems, **chmod** is the command and system call which is used to change the access permissions of file system .

- Classes of users are used to distinguish to whom the permissions giving. If no classes are specified 'all' is implied. The classes are represented by one or more of the following letters:

Reference	Class	Description
u	user	Owner of file
g	group	File group members
o	others	Not owner and not a member of file system
a	all	All the three of above commands

c. Discuss various scheduling criteria. **5**

➔ A number of scheduling algorithms are being designed that can be applied to different processes having different properties. The scheduling criteria are mentioned below.

- CPU Utilization: It is amount of time CPU remains busy.
- Throughput: Number of jobs processed per unit time.
- Turnaround Time: Time elapsed between submission of jobs an completion of its execution.
- Waiting Time: Processes waits in ready queue to get CPU. Sum of times spent In ready queues waiting time.
- Time: Time from submission till the first response is produced.
- Fairness: Every process should get fair share of the CPU time.

d. Explain the effect of page frame size on performance of page replacement algorithms. **5**

➔

- The number of frames is equal to the size of memory divided by the page-size. So and increase in page size means a decrease in the number of available frames.
- Having a fewer frames will increase the number of page faults because of the lower freedom in replacement choice.
- Large pages would also waste space by Internal Fragmentation.
- On the other hand, a larger page-size would draw in more memory per fault; so the number of fault may decrease if there is limited contention.
- Larger pages also reduce the number of TLB misses.

An important hardware design decision is the size of page to be used. There are several factors to consider.

Internal fragmentation:

- Clearly, the smaller the page size, the lesser is the amount of internal fragmentation. To optimize the use of main memory, we would like to reduce internal fragmentation.
- On the other hand, smaller the page, the greater is the number of pages required per process which could mean that some portion of page tables of active processes must be in virtual memory, not in main memory. This eventually leads to double page fault for a single reference of memory.

Rate at which page fault occurs:

- If the page size is very small, then ordinarily a large number of pages will be available in main memory for process, which after some time will contain portions of process near recent references leading to low page fault rate.
- As the size of page is increased, each individual page will contain locations further and further from any particular recent reference. Thus, the effect of the principle of locality is weakened and the page fault rate begins to rise.
- Eventually, however the page fault rate will begin to fall as the size of page approaches the size of the entire process.

e. Explain Thrashing. **5**

➔ -A process should have some minimum number of frames to support active pages which are in memory.

- It helps to reduce the number of page faults. If these numbers of frames are not available to the process then it will quickly page fault.

- To handle this page fault, it is necessary to replace the existing page from memory.

- Since all the pages fault, it is necessary to replace the existing page from memory.

-Since in paging, pages are transferred between main memory and disk, this has an enormous overhead.

- Because of this frequent movement of pages between main memory and disk, system throughput reduces.

- This frequent paging activity causing the reduction in system throughput called as thrashing.

-Although many processes are in memory, due processes are in memory, due to thrashing CPU utilization goes low.

- When operating system monitors this CPU utilization, it introduce new process in memory to increase the degree of multiprogramming.

- Now it is needed that the existing pages should be replaced for the new process.

-If global page replacement algorithm is used, it replaces the pages of other process and allocates frames to this newly introduced process. So other processes whose pages are replaced are also causes page faults.

- All these faulting processes go in wait state and waits for paging devices. In this case again CPU utilization goes low.

-There is no actual work getting done and processes spend time only in paging.

-This thrashing can be limited by using local page replacement algorithm instead of global page replacement algorithm.

Q-2

a. Differentiate between monolithic, layered and microkernel structure of OS. 10

Monolithic Kernel	**Micro Kernel**
1. If virtually entire operating system code is executed in kernel mode, then it is a monolithic program that runs in a single address space.	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 same address space for user mode as well as kernel mode.	2. There is different address space for user mode as well as kernel mode.
3. It has a large space as compared to micro kernel.	3. It has a small space as compared to monolithic kernel.
4. Execution speed is faster than micro kernel.	4. Execution speed is slower than monolithic kernel.
5. If one service crashes whole operating system fails.	5. If one service crashes whole operating system do not fails, it does not affect working of other part micro kernel.
6. Kernel calls the function directly for communication.	6. Communication is done through message passing.
7. To write monolithic kernel less code is required.	7. To write microkernel more code is required.
8. It is hard to extend.	8. It is easily extendible.
9. Example: Linux, BSDs, etc.	9. Example: QNX, Symbian, L4Linux, etc.
10. It is not flexible as compared to microkernel.	10. It is more flexible.
11. kernel calls the function directly.	11. In microkernel, communication is through message passing.

b. Describe the differences among short term, medium-term, and long term Scheduling. **10**

➔

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.

Q-3

a. Discuss how the following pairs of scheduling criteria conflict in certain setting **10**

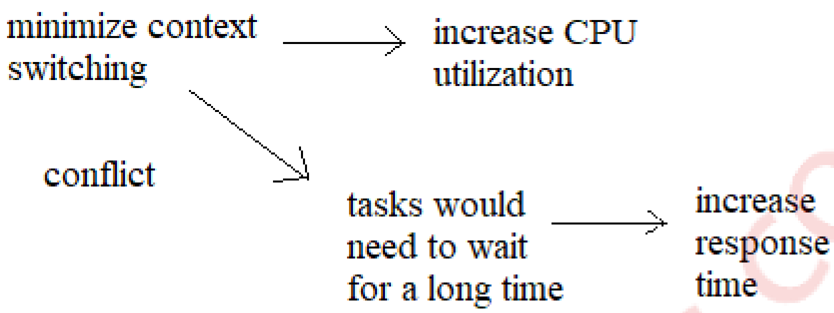
a) CPU utilization and response time

b) Average Turnaround time and maximum waiting time

➔ **CPU utilization and response time:**

- CPU utilization is increased if the overhead associated with context switching is minimized. - The context switching overheads could be lowered by performing context switches infrequently.

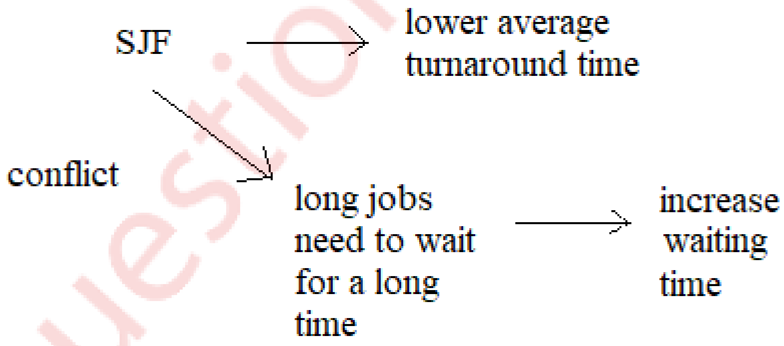
CPU utilization and response time



Average turnaround time and maximum waiting time:

- Average turnaround time is minimized by executing the shortest tasks first.
- Such a scheduling policy could, however, starve long-running tasks and thereby increase their waiting time.
- Average turnaround time and maximum waiting time SJF lower average turnaround time.

Average turnaround time and maximum waiting time



I/O device utilization and CPU utilization: CPU utilization is maximized by running long-running CPU-bound tasks without performing context switches, I/O device utilization is maximized by scheduling I/O-bound jobs as soon as they become ready to run, thereby incurring the overheads of context switches.

I/O device utilization and CPU utilization

maximum CPU utilization ← run long running CPU-bound tasks without context switching

maximize I/O device utilization ← schedule I/O-bound tasks as soon as they're ready need context switching

b. Consider the following snapshot of the system. Using Bankers Algorithm, determine whether or not system is in safe state. If yes determine the safe sequence. **10**

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

➔

Banker's Algorithm is a deadlock avoidance algorithm that checks for safe or unsafe state of a System after allocating resources to a process.

When a new process enters into system ,it must declare maximum no. of instances of each resource that it may need.After requesting operating system run banker's algorithm to check whether after allocating requested

resources,system goes into deadlock state or not. If yes then it

will deny the request of resources made by process else it allocate resources to that process.

No. of requested resources (instances of each resource) may not exceed no. of available resources in operating system and when a process completes it must release all the requested and already allocated resources.

Process	Need(Max-Allocation)			
	A	B	C	D
P0	2	1	0	3
P1	1	0	0	1
P2	0	2	0	0
P3	4	1	0	2
P4	2	1	1	3

Need matrix is calculated by subtracting Allocation Matrix from the Max matrix.

To check if system is in a safe state

P0 starts with available and proceed with the process P0

now available memory is 0301+3014=3315

Now P1 starts and P1 needs 1001

now available memory is 3315+1001=4316

now P2 starts and P2 needs 0200

now available memory is 4361+0200=4561

now p3 starts and P3 needs 4102

now available memory is 4561+4102= 8663

now P4 starts and P4 needs 2113

now available memory is 8663+2113= 10, 7, 7, 6

Available memory is 10A, 7B, 7C, 6D

The system is not in safe state

Q-4

a. Calculate number of page faults and page hits for the page replacement policies FIFO, Optimal and LRU for given reference string 6, 0, 5, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 5, 2, 0, 5, 6, 0, 5 (assuming three frame size). 10

➔ **Frame Size =3**

i) FIFO

Frame	6	0	5	2	0	3	0	4	2	3	0	3	2	5	2	0	5	6	0	5
0	6	6	6	2	2	2	2	4	4	4	0	0	0	0	0	0	0	6	6	6

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

ii) LRU

Frame	6	0	5	2	0	3	0	4	2	3	0	3	2	5	2	0	5	6	0	5
0	6	6	6	2	2	2	2	4	4	4	0	0	0	5	5	5	5	5	5	5
1		0	0	0	0	0	0	0	0	3	3	3	3	3	3	0	0	0	0	0
2			5	5	5	3	3	3	2	2	2	2	2	2	2	2	2	6	6	6
PF	Y	Y	Y	Y	-	Y	-	Y	Y	Y	Y	-	-	Y	-	Y	-	Y	-	-

iii) Optimal

Frame	6	0	5	2	0	3	0	4	2	3	0	3	2	5	2	0	5	6	0	5
0	6	6	6	2	2	2	2	2	2	2	2	2	2	2	2	2	2	6	6	6
1		0	0	0	0	0	0	4	4	4	0	0	0	0	0	0	0	0	0	0
2			5	5	5	3	3	3	3	3	3	3	3	5	5	5	5	5	5	5
PF	Y	Y	Y	Y	-	Y	-	Y	-	-	Y	-	-	Y	-	-	-	Y	-	-

b. Explain synchronization problem in detail. How counting semaphore can be used to solve readers writers problem. **10**

➔ **Synchronization:** Process Synchronization is a sharing system resources by processes in such a way that, Concurrent access to shared data is handled thereby minimizing the chance of inconsistent data.

- Each process executes its own operations on shared variables sequentially, as specified by its own program. Nevertheless, different processes may execute their operations on the same shared variable concurrently. That is, operation executions of different processes may overlap, and they may affect one another.

- Each operation on a shared variable, when executed indivisibly, transforms the variable from one consistent value to another. However, when the operations arc executed concurrently on a shared variable, the consistency of its values may not the guaranteed.
- The behaviours of operation executions on shared variables must, be predictable for effective inter-process communication.
- Thus, operation executions on shared variables may need to be coordinated to ensure their consistency semantics.
- Coordination of accesses to shared variables is called synchronization.
- A synchronization solution coordinates accesses from processes to shared variables. where all accesses to shared variables are channeled through access controllers
- The controllers do the coordination. Most operating systems implement a few different synchronization schemes for process coordination purposes. Each scheme supports a set of primitives. The primitives are used when it is absolutely necessary to have orderly executions of operations on shared variables in a particular manner.
- While defining the reader/writer’s problem, It is assumed that, many processes only read the file(readers) and many write to the file(writers). File is shared among a many number of processes. The conditions that must be satisfied are as follows
  1) Simultaneously reading of the file is allowed to many readers.
  2) Writing to the file is allowed to only one writer at any given point of time.
  3) Readers are not allowed to read the file while writer is writing to the file.
- in this solution, the first reader accesses the file by performing a down operation on the semaphore file. Other readers only increment a counter, read count. When readers finish the reading counter is decremented.
- when last one end by performing an up on the semaphore, permitting a blocked writer, if there is one, to write. suppose that while a reader is reading a file, another reader comes along. Since having two readers at the same time is not a trouble, the second readers and later readers can also be allowed if they come.
- After this assumes that a writer wants to perform a write operation on the file. The writer can’t be allowed to write the file, since writer is suspended. The writer will suspend until no reader is reading the file. If new reader arrives continuously with short interval and perform reading, the writer will never obtain the access to the file.

Algorithm

Writer process

```
While(TRUE)
{
Wait(w)
# perform write operation
Signal(w)
}
```

Reader Process

```
While(TRUE)
{
Wait(m)
Read_count ++;                //reader enters
If (Read_count == 1)
Wait(w)                       //blocks writer
Signal(m)
# perform read operation
Wait(m)
Read_count - -;
If (Read_count == 0)          //No reader
Signal(w)                     //Writer can write
Signal(m)
}
```

Q-5

a. Given memory partitions of 150k,500k,200k,300k,550k(in order) how would each of the first fit, best fit and worst fit algorithm places the processes of 220k,430k,110k,425k(in order).Evaluate, which algorithm makes most efficient use of memory? 10

➔ **- First Fit**

- 220 K is put in 500 K partition.

- 430 K is put in 550 K partition.

- 110 K is put in 150 K partition.

- 425 K must wait.

- Best Fit

- 220 K is put in 300 K partition.

- 430 K is put in 500 K partition.

- 110 K is put in 150 K partition.

- 425 K is put in 550 K partition.

- Worst Fit

- 220 K is put in 550 K partition.

- 430 K is put in 500 K partition.

- 110 K is put in 330 K partition. (550 K – 220 K)

- 425 K must wait.

In this example, Best Fit turns out to be the best.

b. Suppose that a disk drive has 5000 cylinders, numbered 0 to 4999. The drive is currently serving a request at cylinder 143, and the previous request was at cylinder 125. The queue of pending requests in FIFO is ordered as 80, 1470, 913, 1777, 948, 1022, 1750,130. What is the total distance that the disk arm moves for following by applying following algorithms? **10**

1. FCFS 2. SSTF 3. LOOK 4. SCAN

➔

1. FCFS: The FCFS schedule is

143, 80 1470, 913, 1777, 948, 1022, 1750, 130

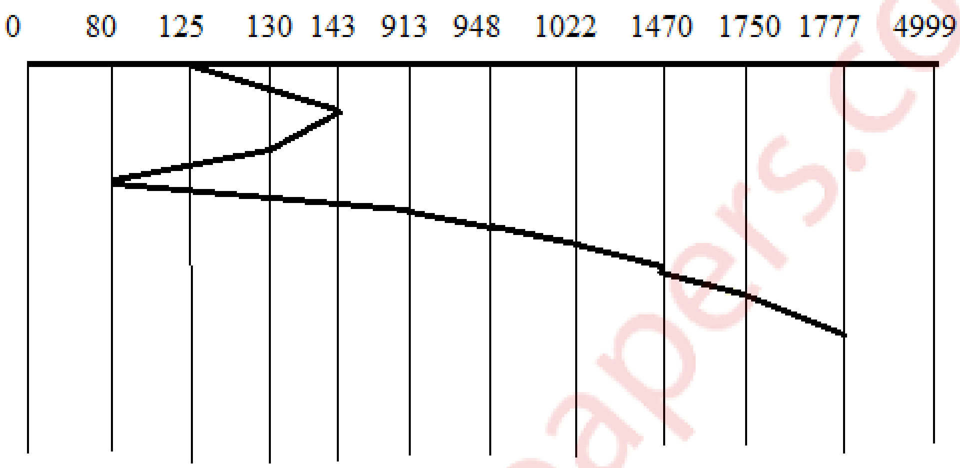
It gives (143-80) =63, (1470-80)=1390, (1470-913)=557, (1777-913)=864, (1777-948)=829, (1022-948)=74, (1750-1022)=728, (1750-80)=1670

Total head movements are:

63+1390+557+864+829+74+728+1670= 6125 Cylinders

2. SSTF: The SSTF schedule is

143, 130, 80, 913, 948, 1022, 1470, 1750, 1777.



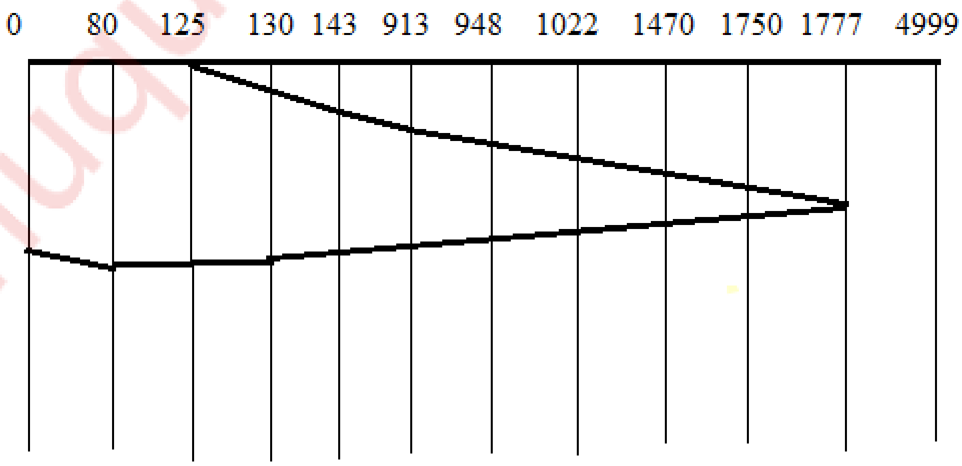
It gives (143-130) = 13, (130-80) = 50, (913-80)= 833, (948-913)=35, (1022-948)=74, (1470-1022)=448, (1750-1470)=280, (1777-1750)=27

Total head movements are:

13+50+833+35+74+448+280+27=1760 Cylinders

3. LOOK: The LOOK schedule is

143, 913, 948, 1022, 1470, 1750, 1777, 130, 80



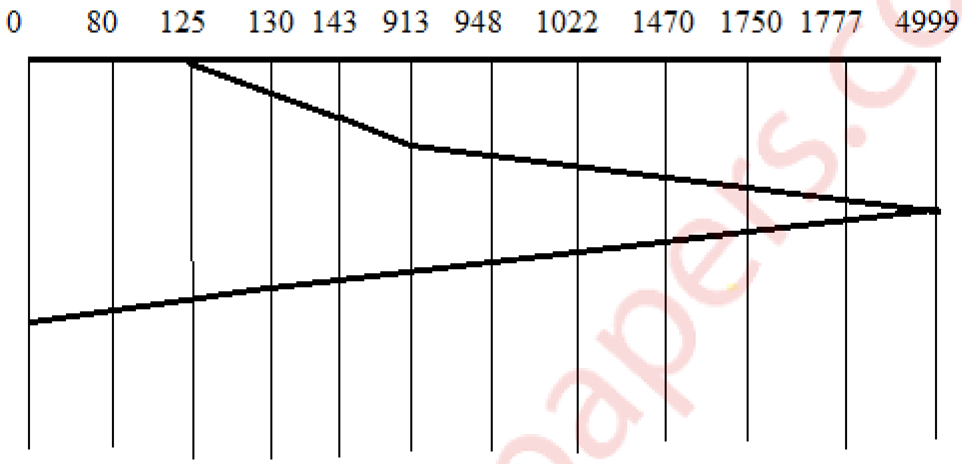
It gives (913-143)=770, (948-913)=35, (1022-948)=74, (1470-1022)=448**,** (1750-1470)=280, (1777-1750)= 27, (1777-130)=1647, (130-80)=47

Total head movements are:

770+35+74+448+280+27+1647+50=3331 Cylinders

4. SCAN: The SCAN schedule is

143, 913, 948, 1022, 1470, 1750, 1777, 4999, 130, 80.



It gives (913-143)=70, (948-913)=35, (1022-948)=74, (1470-1022)=448, (1750-1470)=280, (1777-1750)=27, (4999-1777)=3222, (4999-130)=4869, (130-80)=50

Total head movements are:

770+35+74+448+280+27+3222+4869+50 =9775 Cylinders

Q-6

Write short notes on: (any two) **20**

a. Linux Virtual File system

➔ 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) <u>I-node Object</u> – An individual file is represented by I-node Object.
    2) <u>File Object</u> – An open file is represented by file object.
    3) <u>Superblock Object</u> – An entire file system is represented by a Superblock Object.

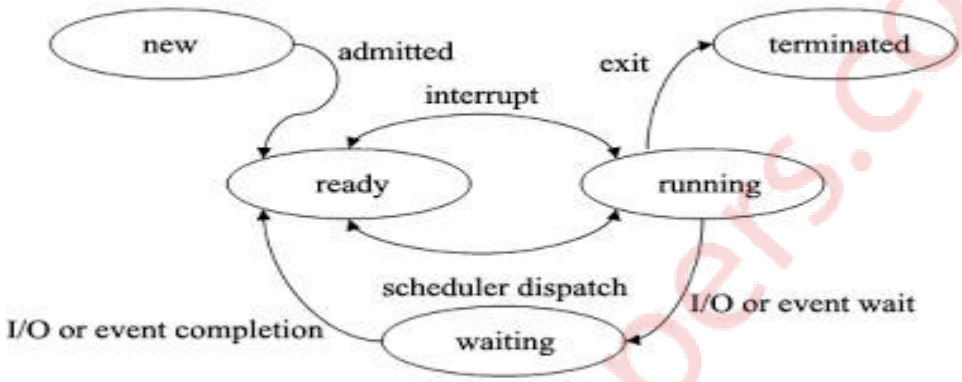
4) <u>Dentry Object</u> – An individual directory entry is represented by Dentry object.

- 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.

b. Process State transition

➔



- 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. Terminated state: 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. System Calls

➔

1) The interface between OS and user programs is defined by the set of system calls that the operating system offers. System call is the call for the operating system to perform some task on behalf of the user's program. Therefore, system calls make up the interface between processes and the operating system.
2) The system calls are functions used in the kernel itself. From programmer's point view, the system call is a normal C function call.
3) Due to system call, the code is executed in the kernel so that there must be a mechanism to change the process mode from user mode to kernel mode.

System call is categorized in five groups.

Sr. No.	Group	Examples
1	Process control	End, abort, load, execute, create process, get process attributes, set process attributes, wait for time, wait event, allocate and free memory.
2	Device Manipulation	Request device, release device, read, write, reposition, get device attributes, set device attributes, logically attach or detach devices.
3	Communications	Create, delete communication connection, send, receive message, transfer status information, attach or detach devices.
4	File Manipulation	Create file, delete file, open, close, read, write, reposition, get file attributes, set file attributes.
5	Information Maintenance	Get time or date, set time or date, get system data, set system data, get process, file, or device attributes, set process, file or device attributes.

Some examples of System Calls

Open()	A program initializes access to a file in a file system using the open system call.
Read()	A program that needs to access data from a file stored in a file system uses the read system call.
Write()	It writes data from a buffer declared by the user to a given device, maybe a file. This is primary way to output data from a program by directly using a system call.
Exec()	exec is a functionality of an operating system that runs an executable file in the context of an already existing process, replacing the previous executable
Fork()	fork is an operation whereby a process creates a copy of itself. Fork is the primary method of process creation on Unix-like operating systems.

Operating System
(DEC 2019)

Q.P. Code – 58567

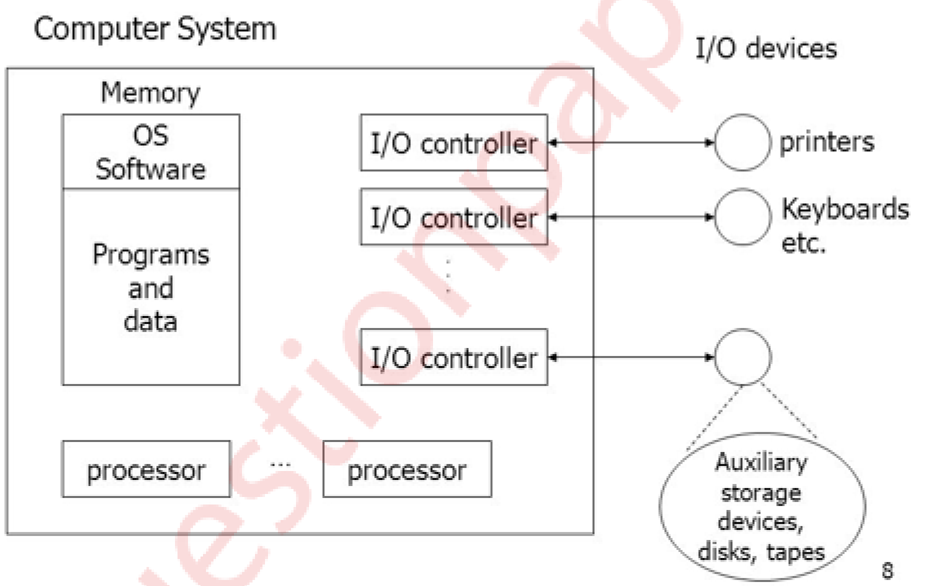
Q 1

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.

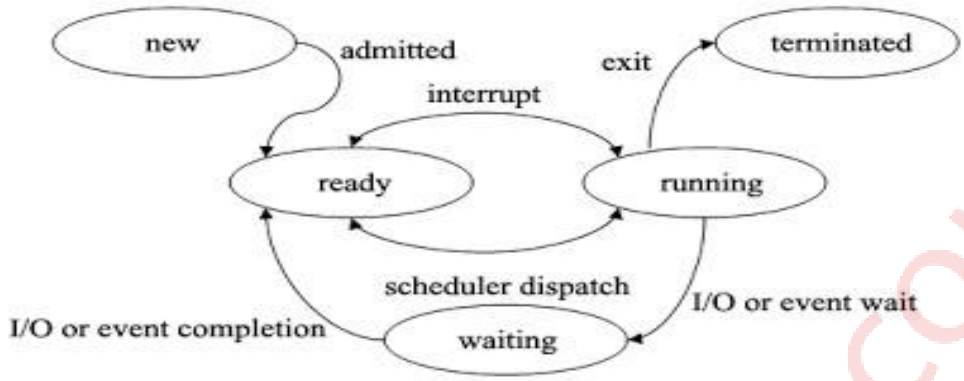
- In this role, the operating system keeps track of the status of each resource, and decides who gets a resource, for how long and when.

- 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.

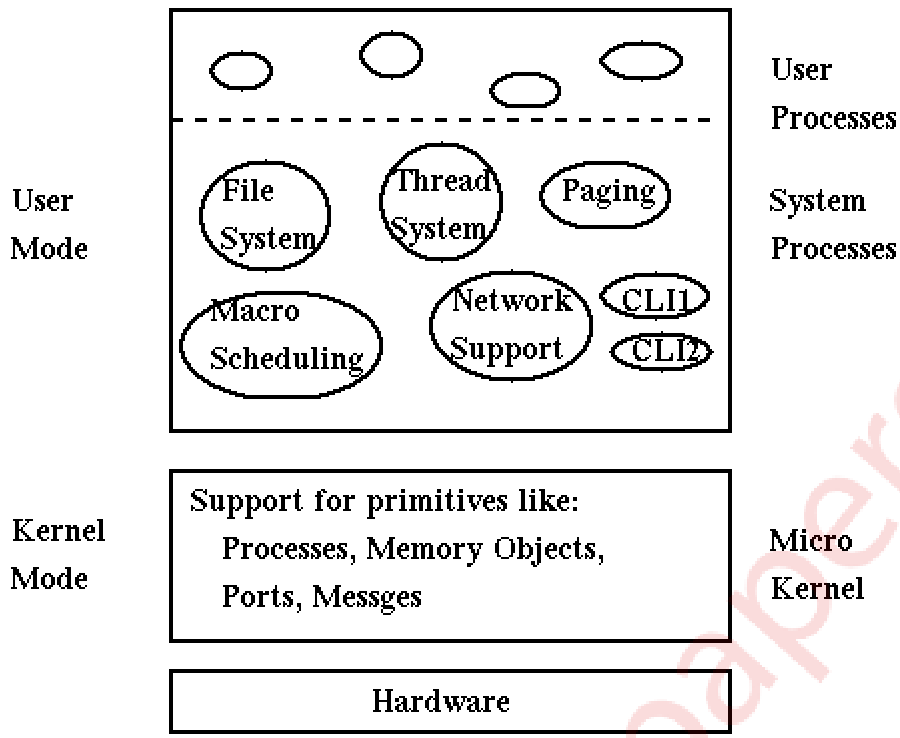
2. Ready state: When the process is ready to execute but he is waiting for the CPU to execute then this is called as the ready state. After completion of the input and output the process will be on ready state means the process will wait for the processor to execute.

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

	P1	P3	P4	P2	P5	
0	1	8	9	11	16	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)
```

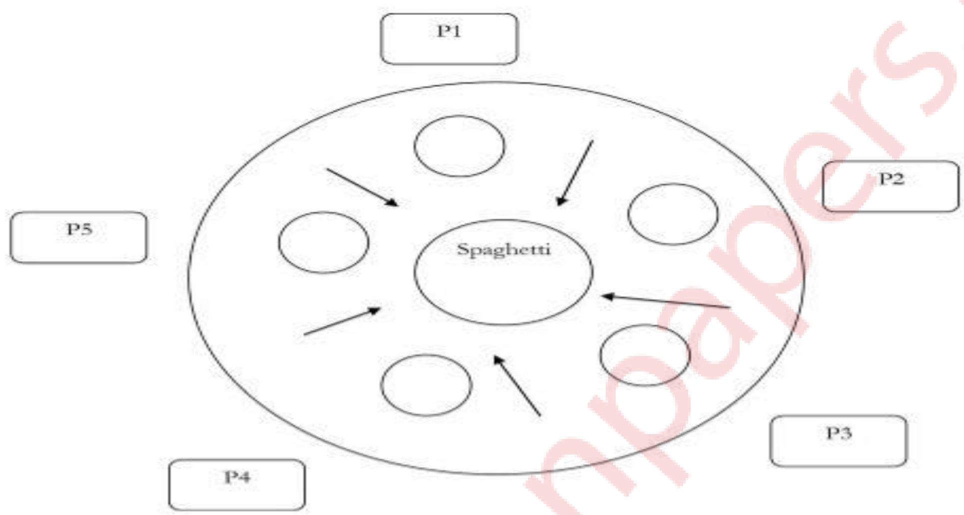
The signal operation on semaphore S

```
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);

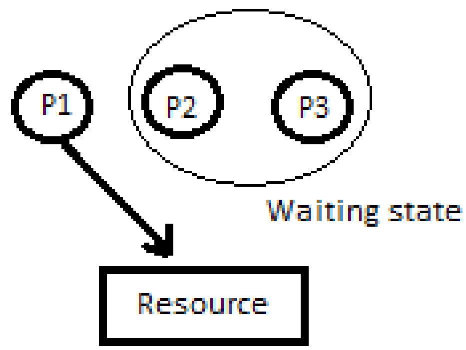
b) What do you understand by a deadlock? Explain deadlock avoidance method. 10M

➔ Deadlock:

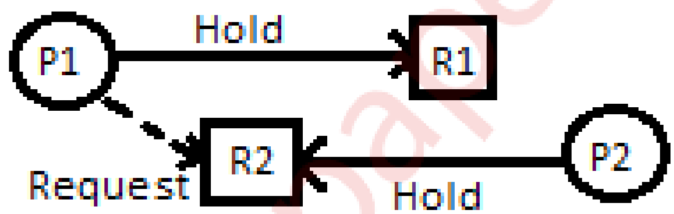
* We know that processes need different resources in order to complete the execution.
* So in a multiprogramming environment, many processes may compete for a multiple Number of resources.
* In a system, resources are finite. So with finite number of resources, it is not possible to fulfill the resource 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:

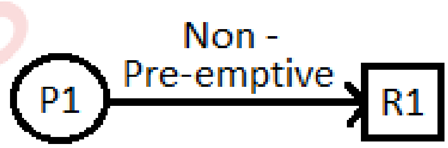
- **Mutual Exclusion**
    - A resource at a time can only be used by one process.
    - If another process is requesting for the same resource, then it must be delayed until that resource is released.



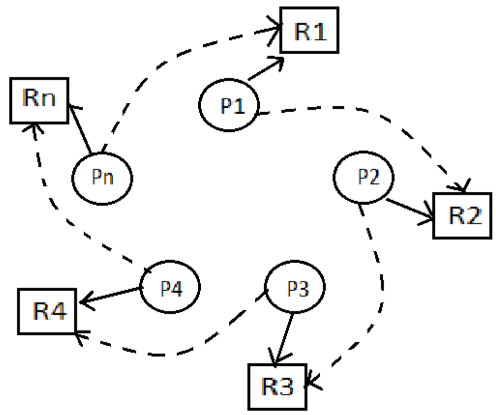
- **Hold and Wait**
  - A process is holding a resource and waiting to acquire additional resources that are currently being held by other processes.



- **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.

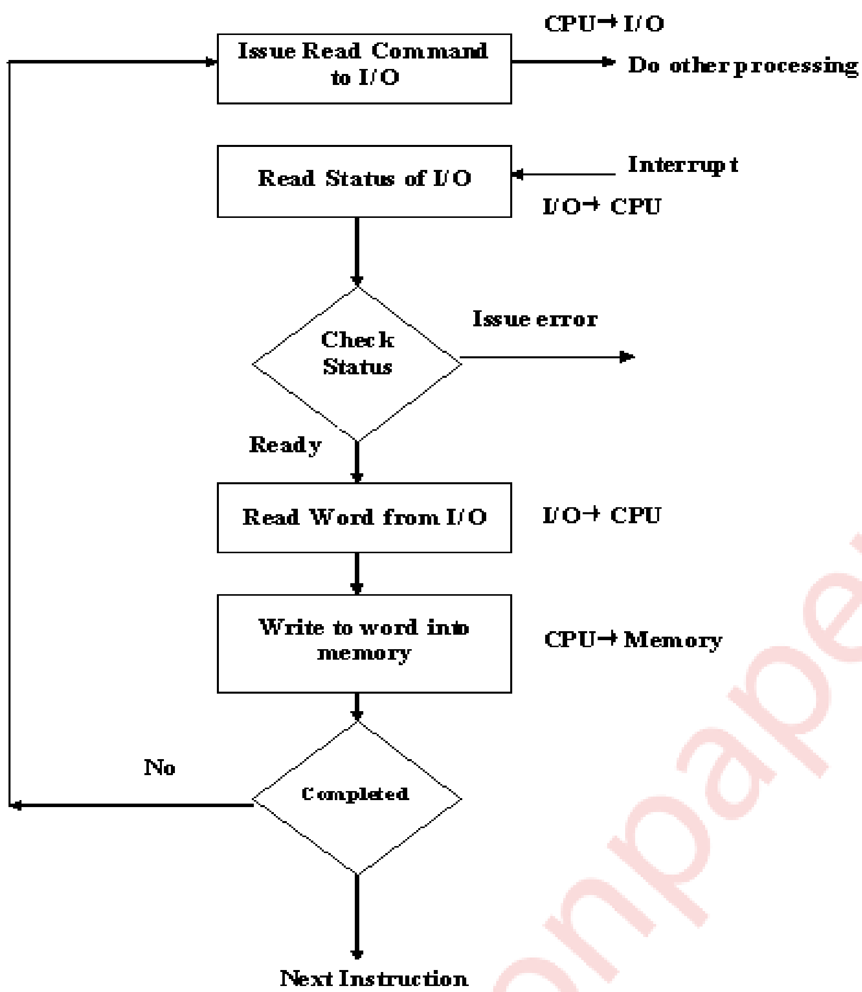
- **Design Issues**

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)

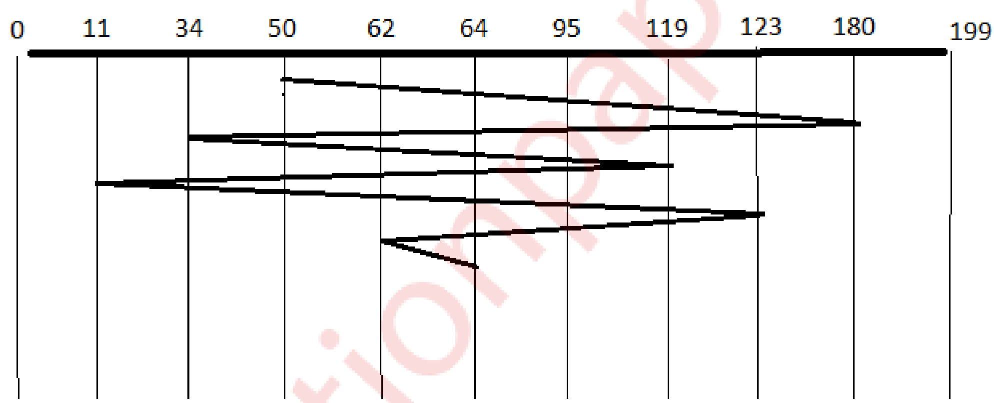
    5) LOOK

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):

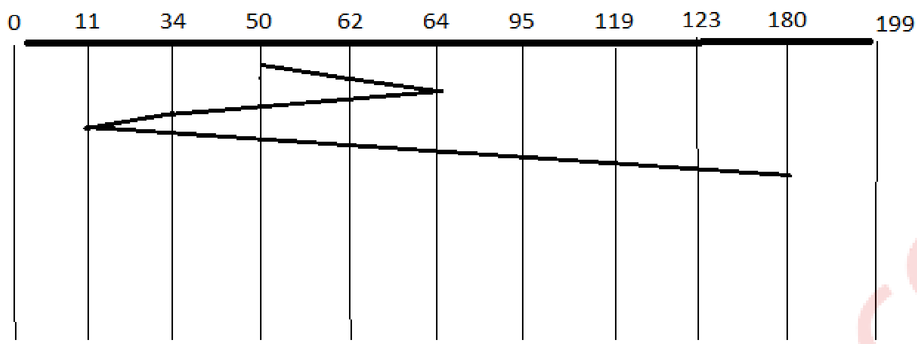
All incoming requests are placed at the end of the queue. Whatever number that is next in the queue will be the next number served. Using this algorithm doesn't provide the best results. To determine the number of head movements you would simply find the number of tracks it took to move from one request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

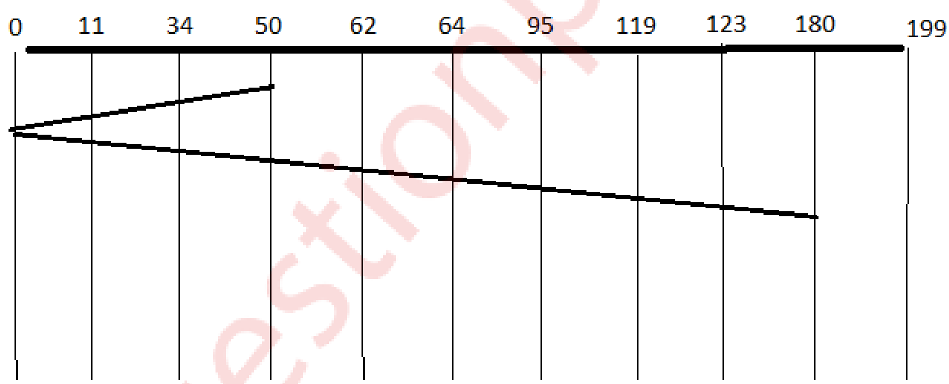
In this case request is serviced according to next shortest distance. Starting at 50, the next shortest distance would be 62 instead of 34 since it is only 12 tracks away from 62 and 16 tracks away from 34. The process would continue until all the process are taken care of. For example, the next case 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):

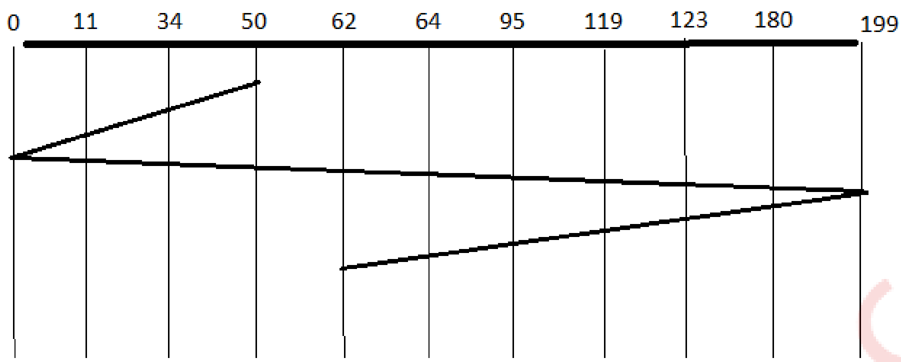
This approach works like an elevator does. It scans down towards the nearest end and then when it hits the bottom it scans up servicing the requests that it didn't get going down. If a request comes in after it has been scanned it will not be serviced until the process comes back down or moves back up.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

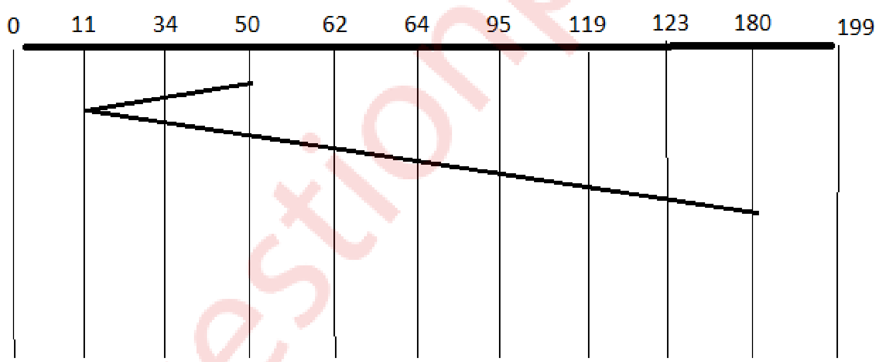
Circular scanning works just like the elevator to some extent. It begins its scan toward the nearest end and works it way all the way to the end of the system. Once it hits the bottom or top it jumps to the other end and moves in the same direction. Keep in mind that the huge jump doesn't count as a 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.

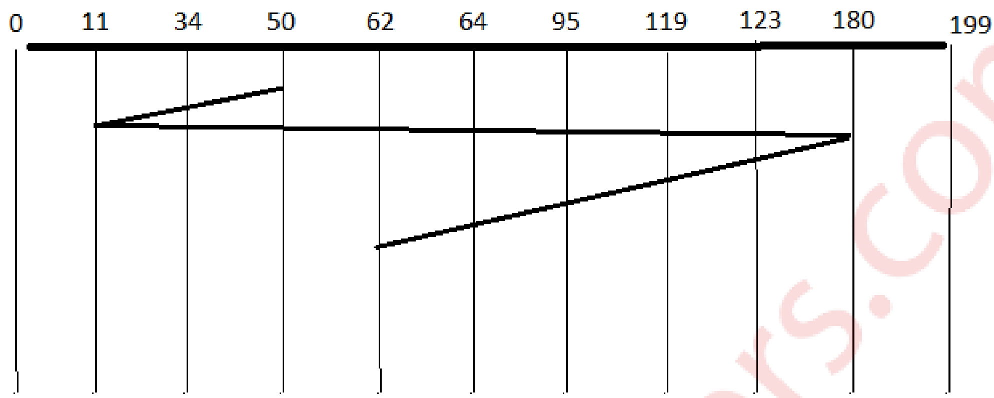


Total head moment = (50-34)+(34-11)+(62-11)+(64-62)+(95-64)+(119-95)+

(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**

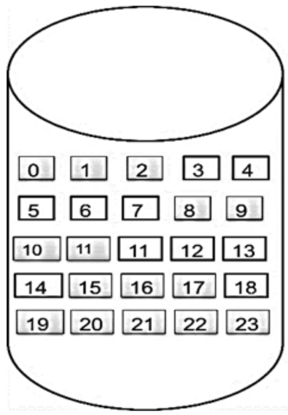
Q6

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) **<u>Contiguous Allocation</u>:** 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.

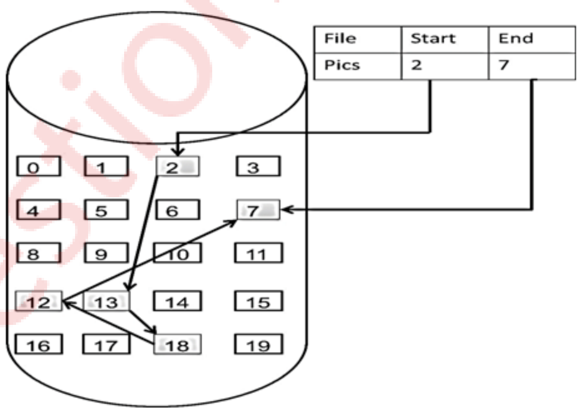


File	Start	Length
A	0	3
Hello	8	4
Marks	15	3
Photos	19	5

<u>Advantage</u>:
• Contiguous allocation is easy to implement.

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

2) **<u>Linked List Allocation</u>:** 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.



<u>Advantage</u>:
• Reading file sequentially is simple.

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

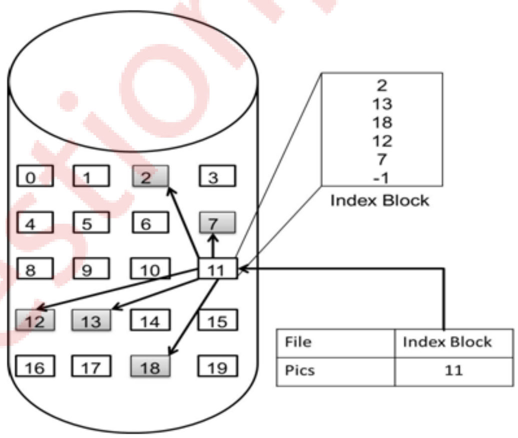
3) **<u>Linked List allocation using table in memory</u>:** Each block needs to store pointer information; therefore, entire block is not fully used to store file content. This limitation can be overcome by keeping pointer information in table which always remains in memory. Refer Linked list fig for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

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

<u>Advantage</u>:
• Random access is much easier

<u>Disadvantage</u>:
• Whole table must be in memory all the time to make it work.

4) **<u>Indexed allocation:</u>** 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.



<u>Advantage</u>:
• Indexed allocation supports random access.

<u>Disadvantage</u>:
• The pointer overhead of index block is more with compare to the pointer overhead of linked allocation.

5) **<u>I-nodes</u>**: 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) <u>I-node Object</u> – An individual file is represented by I-node Object.
  2) <u>File Object</u> – An open file is represented by file object.
  3) <u>Superblock Object</u> – An entire file system is represented by a Superblock Object.
  4) <u>Dentry Object</u> – 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.
