Que→3) Explain process and thread.

Ans:→ Process :→ processes are basically the program that are dispatched from ready state & are scheduled in the CPU for execution. PCB (Process Control Block) holds the concept of process. A process can create other processes which are known as child processes. The process takes more time to terminate and it is isolated means it does not share the memory with any other process. It have following states :→ new, ready, running, waiting, terminated & suspended.

Thread :→ Thread is the segment of a process which means a process can have multiple threads & these multiple threads are contained within a process. A thread has three states :→ Running, Ready & Blocked.

The thread takes less time to terminate as compared to the process but unlike the process, threads do not isolate.

Degree of Multiprogramming :→ The degree of multiprogramming describes the maximum number of processes that a single-processor system can accomodate efficiently. There are some of the factors affecting the degree of multiprogramming such as the primary factor is the amount of memory available to be allocated to executing processes. Job Scheduler manages DOM.

* Number of process available in ready state known as Degree of multiprogramming.

Dispatcher :→ A dispatcher is a special program that comes into play after the scheduler. When the short term scheduler selects from the ready queue, the dispatcher performs the task of allocating the selected process to the CPU. Switching of CPU from one process to the other is called context switching.

Que:-) Multilevel Queue Scheduling & Multilevel feedback queue scheduling, explain.

Ans:→ Multilevel Queue (MLQ) Scheduling :→ It may happen that processes in the ready queue can be divided into different classes where each class has its own scheduling needs. for example, a common division is a foreground (interactive) process & a background (batch) process. These two classes have different scheduling needs. For this kind of situation Multilevel Queue Scheduling is used.

Advantages:→

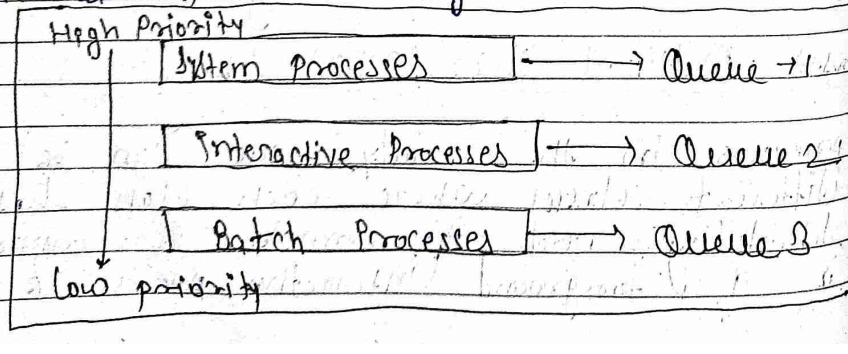
* The processes are permanently assigned to the queue, so it has advantage of low scheduling overhead.

Disadvantages:→ * Some processes may starve for CPU is some higher priority queues are never become empty.

* It is inflexible in Nature.

Ready Queue is divided into separate queues for each class of processes, for example, let us take three different types of processes System processes, interactive processes & Batch processes. All three processes have their own queue. Now look at the below figure.

Diagram:-



(i) System Processes:→ The CPU itself has it's own process to run which is generally termed as System process.

(ii) Interactive processes:→ An interactive process is a type of process in which there should be same type of interaction.

(iii) Batch Processes:→ Batch processing is generally a technique in the Operating System that collects the programs and data together in the form of the Batch before the processing starts.

Multilevel Feedback Queue (MLFQ) Scheduling:→

Multilevel feedback Queue Scheduling is like multi-level Queue Scheduling but in this processes can move between the queues. And thus, much more efficient than multilevel queue scheduling.

Advantages:→

* It is more flexible.
* it allows different processes to move between different queues.
* It prevents starvation by moving a process that waits too long for the lower priority queue to

the higher priority queue.

Disadvantages:→

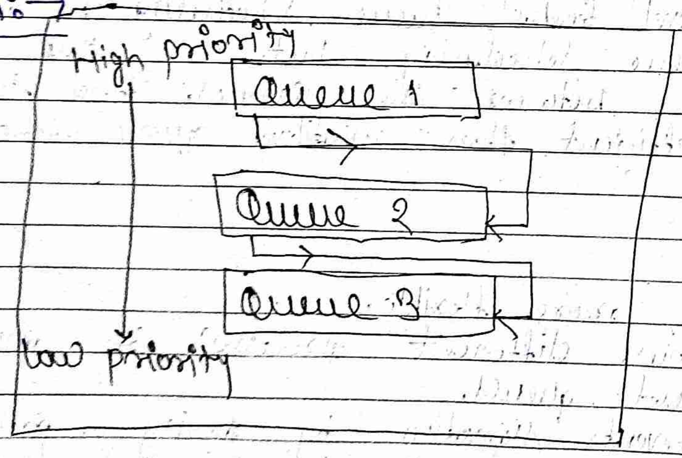
* For the selection of the best scheduler, it requires some other means to select the value.

* It produces more CPU overheads.

* It is most complex algorithm.

MLFQ Scheduling:→ However, allows a process to move between queues. MLFQ keeps analyzing the behavior (time of execution) of processes and according to which it changes its priority.

Diagram:→



Deadlock:→ Deadlock is a situation where each of the computer process waits for a resource which is being assigned to some another process. In this situation, none of the process get executed since the resource it needs, is held by some other process which is also waiting for some other resource to be relesed.

Reason for Deadlock:→

(i) Circular Waiting:→ When the two people refuse to retreat and wait for each other to retreat so that they can complete their task, it is called circular wait.

(ii) Hold & wait:→ When two people refuse to retreat and hold their ground it is called holding.

(iii) No Preemption:→ For resolving the deadlock one can simply cancel one of the processes for other to continue. But operating system doesn't do so. It allocates the ~~processes~~ resources to the processes for as much time as is needed until the task is completed. Hence, there is no temporarily reallocation of the resources.

Date ____
Page ____

(12) Mutual Exclusion (non Shareable) :→ when two people meet in the landings, they can't just walk through because there is space only for one person. This condition allows only one person (or process) to use the step between them (or the resources) is the first condition necessary for the occurrence of the deadlock.

Que:→ ① identify deadlock present or not.

	Required			Allocation			Available (work)		
	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3
P_1	2	1	1	1	1	1	1	1	1
P_2	1	2	1	1	0	1			
P_3	2	1	1	1	1	1			

Solution:→ Need Matrix[ ] = Required[ ] (max[ ]) − Allocation[ ]

Need Matrix :→ Calculation

	R_1	R_2	R_3
P_1	1	0	0
P_2	0	2	0
P_3	1	0	0

Available (work)

R_1	R_2	R_3
1	1	1
2	2	2
3	2	3
4	3	4

Date ____
Page ____

⇒ Step for process P_1 = Work[ ] > need[ ]
= P_1 will execute.
Work[ ] = Work[ ] + Allocation[ ]
= [1 1 1] + [1 1 1]
= [2 2 2]

⇒ Step for process P_2 = Work[ ] > need[ ]
= P_2 will execute
Work[ ] = Work[ ] + Allocation[ ]
= [2 2 2] + [1 0 1]
= [3 2 3]

⇒ Step for process P_3 = Work[ ] > need[ ]
= P_3 will execute.
Work[ ] = Work[ ] + Allocation
= [3 2 3] + [1 1 1]
= [4 3 4]

order (pattern) = P_1 P_2 P_3
instints :→ 4, 3, 4

So, Deadlock is not present.

Que:→ ② :> identify Deadlock is present or not.

Process	Allocation			request			available		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	0	0	0	0	0	0
P_1	2	0	0	2	0	2			
P_2	3	0	3	0	0	0			
P_3	2	1	1	1	0	0			
P_4	0	0	2	0	0	2			

Solution :- ∵ request is given.
∴ We don't need to calculate need matrix. Need matrix = request.

need matrix =

	A	B	C
P_0	0	0	0
P_1	2	0	2
P_2	0	0	0
P_3	1	0	0
P_4	0	0	2

Available (work)

A	B	C
0	0	0
0	1	0
3	1	3
5	2	4
5	2	6
7	2	6

⇒ Step for process P_0 = Work[] = need[]
P_0 will execute
Work[] = Work[] + Allocation[]
= [0 0 0] + [0 1 0]
= [0 1 0]

⇒ Step for process P_2 = Work[] < need[]
P_2 will not execute.

⇒ Step for process P_2 = Work[] > need[]
= P_2 will execute
Work[] = Work[] + Allocation[]
= [0 1 0] + [3 0 3]
= [3 1 3]

⇒ Step for process P_3 = Work[] > need[]
= P_3 will execute
Work[] = Work[] + Allocation[]
= [3 1 3] + [2 1 1]
= ~~[4 1 3]~~ [5 2 4]

⇒ Step for process P_4 = Work[] > need[]
= P_4 will execute.
Work[] = Work[] + Allocation[]
= ~~[4 1 3] + [0 0 2]~~
= ~~[4 1 5]~~
= [5 2 4] + [0 0 2]
= [5 2 6]

Switch to step for process P_1.

⇒ Step for process P_1 = Work[] > need[]
= P_1 will execute
Work[] = Work[] + Allocation[]
= [5 2 6] + [2 0 0]
= [7 2 6]

order :→ P_0, P_2, P_3, P_4, P_1
instints :→ 7, 2, 6
Deadlock is not present.

Que:→ Explain Banker's algorithm.

Ans.⇒ Banker's algorithm:→ The banker's algorithm is a resource allocation and deadlock avoidance algorithm that tests for safety by simulating the allocation for predetermined maximum possible amounts of all resources, then makes an "s-state" check to test for possible activities before deciding whether allocation should be allowed to continue.

Banker's algorithm is named so because it is used in banking system to check whether loan can be sanctioned to a person or not. It is also known as deadlock avoidance algorithm or deadlock detection in the operating system.

Resource Allocation Graph (RAG):→ The resource allocation graph is the pictorial representation of the state of a system. As its name suggest, the resource allocation graph is the complete information about all the processes which are holding some resources or waiting for some resources.

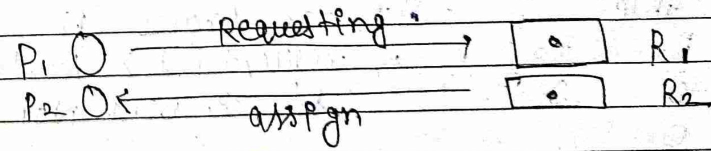
It also contains the information about all the instances of all the resources whether they are available or being used by the processes.

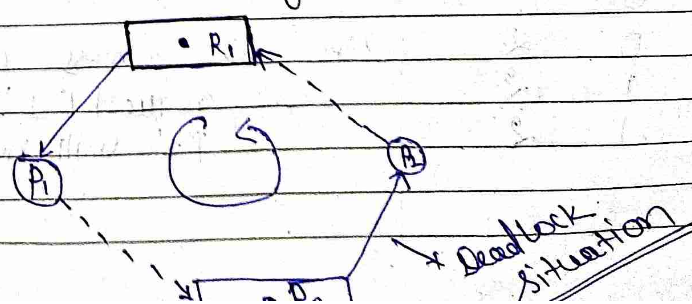
In Resource allocation graph, the process is represented by a circle while the Resource is represented by a rectangle.

Resource allocation graph is explained to us what is the state of the system in terms of processes & 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 using RAG, but then you might not be able to know that by looking at the table.

Symbols:→

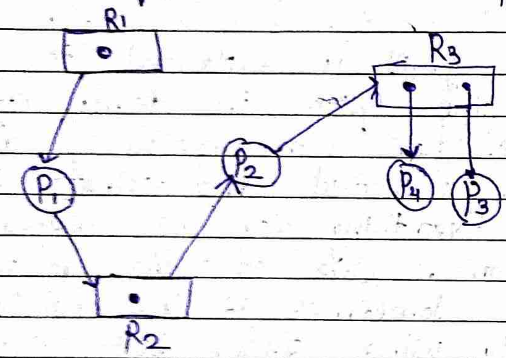
○ ⟶ Process.
▭ ⟶ Resource.
[•] ⟶ Single instance.
[• • •] ⟶ Multi instance.
⇌ ⟶ Arrow.





Deadlock situation

Que → Using given Resource allocation Graph (RAG) draw RAT (Resource allocation Table) & identify all safe Sequence if it is possible.



Solution:→

Process	Allocation R_1	R_2	R_3	Request R_1	R_2	R_3
P_1	1	0	0	0	1	0
P_2	0	1	0	0	0	1
P_3	0	0	1	0	0	0
P_4	0	0	1	0	0	0

Available

R_1	R_2	R_3
0	0	0
0	0	1
0	0	2
0	1	2
1	1	2

Step for Process P_1
⇒ request[ ] > Available[ ]
so, P_1 will Not execute

Step for process P_2
⇒ request[ ] > Available[ ]
P_2 will not execute

Step for process P_3
⇒ request[ ] = Available[ ]
⇒ P_3 will execute
⇒ Available[ ] = Available[ ] + Allocation[ ]
⇒ = [0 0 0] + [0 0 1]
⇒ Available = [0 0 1]

Step for process P_4
⇒ request[ ] < Available[ ]
⇒ P_4 will execute.
⇒ Available[ ] = Available[ ] + Allocation[ ]
= [0 0 1] + [0 0 1]
⇒ Available = [0 0 2]

Switch to process P_1
⇒ Request[ ] > Available[ ]
P_1 will Not execute.

Switch to process P_0
⇒ request[ ] < Available[ ]
P_0 will execute.
⇒ Available[ ] = Available[ ] + Allo[ ]
= [0 0 2] + [0 1 0]
⇒ Available = [0 1 2]

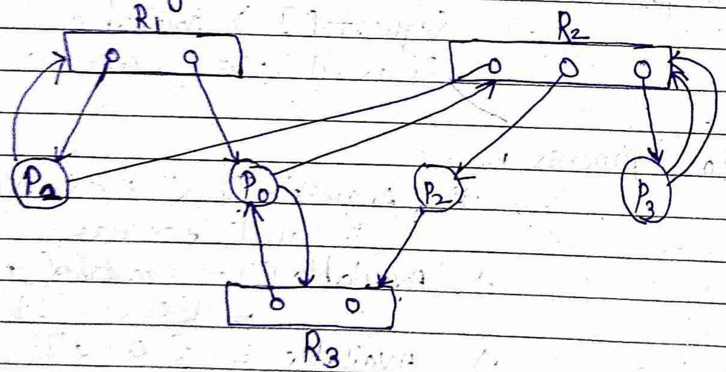
Switch to process P_1
⇒ request[ ] < Available[ ]
P_1 will execute.
⇒ Available[ ] = Available[ ] + Allocation[ ]
= [0 1 2] + [1 0 0]
⇒ Available = [1 1 2)

So, Here Deadlock is not present.
And possible safe sequences are.

⇒ $P_3 \rightarrow P_4 — P_2 — P_1$
⇒ $P_3 \leftarrow P_2 — P_1 — P_4$
⇒ $P_3 — P_2 — P_4 — P_1$
⇒ $P_4 — P_3 — P_2 — P_1$
⇒ $P_4 — P_2 — P_3 — P_1$
⇒ $P_4 — P_2 — P_1 — P_3$

instincts :→ for ($P_3 — P_4 — P_2 — P_1$)
⇒ 1 1 2.

Que :→ Consider the resource allocation graph in the figure :→



Find if the system is in a deadlock state otherwise find a safe sequence.

Solution :→

Process	Allocation			request			Available		
	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3
P_0	1	0	1	0	1	1	0	0	1
P_1	1	1	0	1	0	0	0	1	1
P_2	0	1	0	0	0	1	1	1	2
P_3	0	1	0	0	2	0	2	2	2
							2	3	2

Step for process P_0
⇒ Request [ ] > Available [ ].
P_0 will Not execute.

Step for process P_1
⇒ Request [ ] > Available [ ]
P_1 will Not execute.

Step for process P_2.
⇒ Request [ ] = Available [ ].
P_2 will execute
⇒ Available [ ] = Available + Allocation [ ]
= [0 0 1] + [0 1 0]
⇒ Available [ ] = [0 1 1]

Step for process P_3.
⇒ Request [ ] > Available [ ]
⇒ P_3 will Not execute.

Switch to process P_0.

$\Rightarrow$ Request [] = Available[]

P_0 will execute.

$\Rightarrow$ Available [] = Available[] + Allo[]

= [0 1 1] + [1 0 1]

$\Rightarrow$ Available[] = [1 1 2]

Switch to process P_1.

$\Rightarrow$ request [] < Available[]

P_1 will execute.

$\Rightarrow$ Available [] = Available[] + Allo[]

= [1 1 2] + [1 1 0]

$\Rightarrow$ Available[] = [2 2 2]

Switch to process P_3

$\Rightarrow$ request [] < Available[]

$\Rightarrow$ P_3 will execute.

$\Rightarrow$ Available[] = Available[] + Allo[]

= [2 2 2] + [0 1 0]

$\Rightarrow$ Available[] = [2 3 2]

So, Here the system is in a safe state

Safe Sequence:→ $P_2 - P_0 - P_1 - P_3$

instints:→ [2 3 2]

Que:→ A system is having 3 user process P_1, P_2, P_3 where P_1 requires 21 unit of resource R. P_2 requires 31 unit of resource R. P_3 requires 41 unit of resource R. What is minimum number of R that ensure no Deadlock.

Solution:→

Process	require unit
P_1	21
P_2	31
P_3	41

P_1 require 21 unit to execute. After execution P_1 relese unit & P_2 use it and execute, similarly P_3 do the same

Minimum Unit require :→

20 + 1 = 21
30 = 30
40 = 40
= 91

91 unit require to ensure no Deadlock.

Que:→ A system is having 10 user process, each requires 3 unit of resource R. find minimum No of R such that no deadlock occur.

Solution:->

Process	require
P_1	3
P_2	3
P_3	3
P_4	3
⋮	⋮
P_{10}	3

after execution processes ~~relese~~ release instints, so that next process will execute.

P_1	2+1	= 3
P_2	2	= 2
P_3	2	= 2
⋮	⋮	⋮
P_{10}	2	= 2
		21

Minimum No require = 21 unit.

Que:-> If there are six unit of resource R in the system & process in the system requires 2 unit of resources (R1). Then How many process can be present at maximum so that No Deadlock will occur.

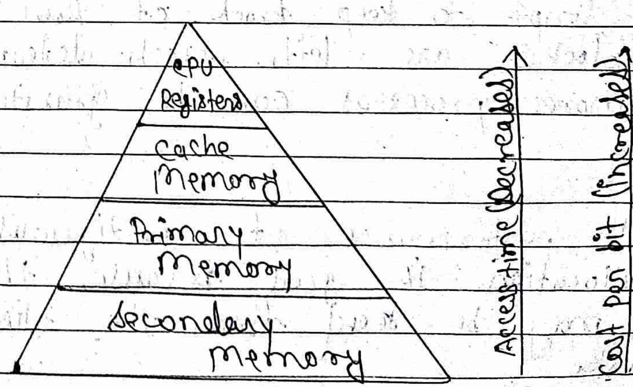
Solution:->

~~Que~~ ~~Proc~~ P_1 requires 2 instints (unit).
So 4 unit remains.

P_1	⟶	2
P_2	⟶	1
P_3	⟶	1
P_4	⟶	1
P_5	⟶	1
total	⟶	6

Max process present for no Deadlock :-> 5 process

Memory Management:-> In a multiprogramming computer the operating system resides in a part of memory & the rest is used by multiple processes. The task of subdividing the memory among processes is called memory management. Memory Management is a method in the operating system to manage operations between main memory & disk during process execution. The main aim of M.M. is to achieve efficient utilization of memory.



Register — Cache — Primary — Secondary

↓ fastest .... ↓ slowest

Two ways of Memory Allocation :→

(I) Contiguous memory Allocation.

(II) Non-Contiguous Memory Allocation.

(I) Contiguous Memory Allocation :→ It is the type of Memory allocation method. When a process requests the memory, a ~~sign~~ single contiguous section of memory blocks is allotted depending on it's requirements.

It is completed by partitioning the memory into fixed-sized partitions and assigning every partition to a ~~sign~~ single process.

Advantages :→

(I) It is simple to keep track of how many memory blocks are left, which determines how many more processes can be granted memory space.

(II) The read performance of contiguous memory Allocation is good because the complete file may be read from the disk in a single task.

(III) The Contiguous memory Allocation is simple to set up & performs well.

Disadvantages :→

(I) Fragmentation isn't a problem because every new file may be written to the end of the disk after the previous one.

(II) When generating a new file, it must know its eventual size to select the appropriate hole size.

(III) When the disk is filled up, it would be necessary to ~~compare~~ compress or reuse the spare space in the holes.

Algorithms for memory Allocation :→

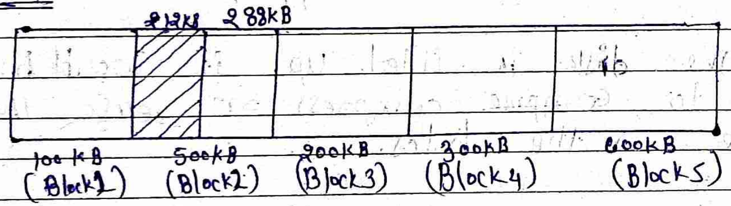
(I) First fit :→ In this method, first job claims the first available memory with space more than or equal to its size. The operating system doesn't search for appropriate partition but just allocate the job to the nearest memory partition avaible with sufficient size.

Advantage :→ It is fast in processing. As the processor allocates the nearest available memory partition to the job.

Disadvantage:→ It wastes a lot of memory. The processor ignores if the size of partition allocated to the Job is very large as compare to the size of job or not.

Que:→ Given memory partition of 100 KB, 500KB, 200KB, 300KB, 600KB (in order). How would FCFS (first fit) algorithm place process of 212 KB, 417 KB, 112 KB, 426 KB (in order).

Solution:→



Step for process P_1 (212 KB):→

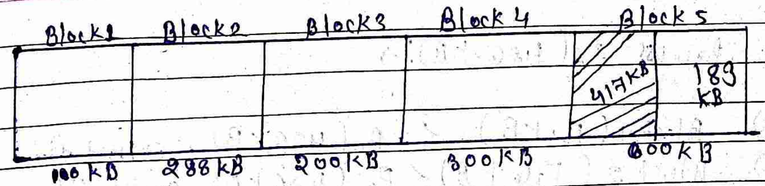
⇒ All Blocks 1-5 are available.

⇒ P_1 checks for Block 1, size(100KB) < size P_1 (212KB), can't fit.

⇒ P_1 checks for Block 2, size (500KB) > size P_1 (212 KB), P_1 goes to Block 2.

⇒ remain Block 2 size = 500 KB − 212 KB

⇒ remain size = 288KB.



Step for process P_2 (417KB):→

⇒ Block 1 size (100KB) < Process P_2 (417 KB), can't fit.

⇒ Block 2 size (288 KB) < P_2 (417 KB), can't fit.

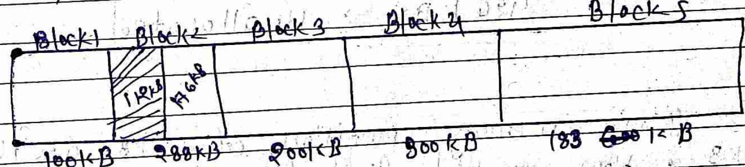
⇒ Block 3 size (200KB) < P_2 (417KB), can't fit.

⇒ Block 4 size (300KB) < P_2 (417KB), can't fit

⇒ Block 5 size (600KB) > P_2 (417 KB), P_2 allocated to Block 5

⇒ remain size = Block5 size (600KB) − P_2 size (417)

⇒ remain size = 183 KB



Step for process P_3 (112KB):→

⇒ Block1 (100KB) < P_3 (112 KB), can't fit.

⇒ Block 2 (288KB) > P_3 (112KB), P_3 allocated to Block 2.

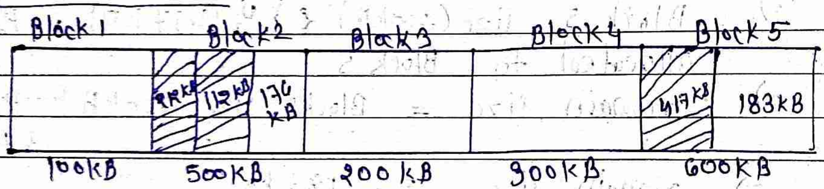
⇒ remain size = Block (288) − P_3 (112 KB)

= 176 KB

Step for Process P_4 (426 KB):→

⇒ Block 1 (100KB) < P_4 (426KB), can't fit.
⇒ Block 2 (176 KB) < P_4 (426KB), can't fit.
⇒ Block 3 (200 KB) < P_4 (426 KB), can't fit.
⇒ Block 4 (300KB) < P_4 (426 KB), can't fit.
⇒ Block 5 (183 KB) < P_4 (426KB), can't fit.
⇒ So P_4 is unallocated.

Final chart:→

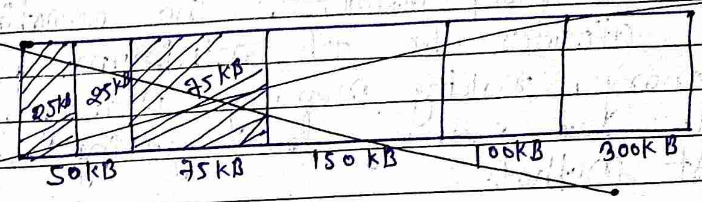


426 KB ⟶ unallocated

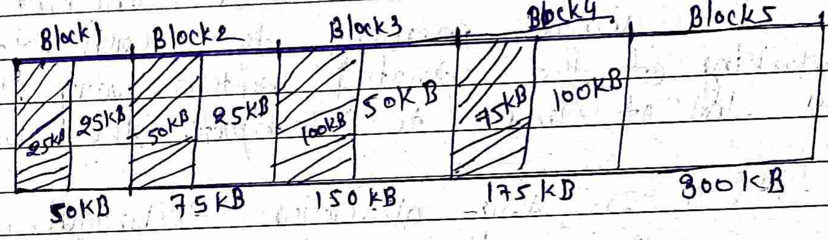
Que:→ given memory partition of 25 KB, 50 KB, 100 KB, 75KB. How would first fit algorithm place process of

Que:→ given memory partition of 50 KB, 75 KB, 150 KB, 175 KB, 300 KB. How would first fit algorith place processes of 25 KB, 50 KB, 100 KB, 75 KB.

Solution:→



Solution:→



Process	Allocated
P_1 (25KB)	Block 1
P_2 (50KB)	Block 2
P_3 (100KB)	Block 3
P_4 (75KB)	Block 4

(ii) Best fit:→ This method keeps the free/busy list in order by size - smallest to largest. In this method, the operating system first searches the whole of the memory according to the size of the given job and allocates it to the closest-fitting free partition in the memory, making it able to use memory efficiently. Here the jobs are in the order from smallest job to largest job.

Advantages:→ memory efficient. The operating system allocates the job minimum possible space in the memory, making memory management very efficient. To save memory from getting wasted, it is the best method.

Disadvantages:→ It is a slow process. Checking the whole memory for each job makes the working of the operating system very slow. It takes a lot of time to complete the work.

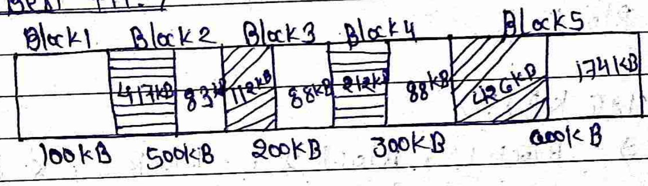
③ Worst fit:→ In this allocation technique, the process traverses the whole memory & always search for the largest hole/partition, and then the process is placed in that hole/partition. It is a slow process because it has to traverse the entire memory to search the largest hole.

Advantages:→ Since the process chooses the largest hole/partition, therefore there will be large internal fragmentation. Now, this internal fragmentation will be quite big so that other small processes can also be placed in that leftover partition.

Disadvantages:→ It is a slow process because it traverses all the partition in the memory & then selects the largest partition among all the partitions, which is a time-consuming process.

Que:→ given memory partition of 100 KB, 500KB, 200KB, 300KB, 600KB (in order). How would Bf, Wf algorithm place process of 212 KB, 417 KB, 112 KB, 426 KB (in order).

Solution:→ Best fit:→



Process P_1 (212 KB):→

⇒ Block 1 & Block 3 are not available as their size is less than 212KB.

⇒ Block 2 Memory wastage (Remain) :→ 500 − 212 = 288 KB

⇒ Block 4 memory wastage :→ 300 − 212 = 88 KB

⇒ Block 5 memory wastage ⇒ 600 − 212 = 388KB

⇒ Thus Block 4 is best for P_1, so P_1 placed in block 4.

Process P_2 (417 KB):→

⇒ Block 1, Block 3, Block 4 are not available as their size is ~~the~~ less than 417 KB

⇒ Block 2 Memory Remain :→ 500 − 417 = 83KB

⇒ Block 5 Memory Remain :→ 600 − 417 = 183

⇒ thus Block 2 is best. P_1 Placed in Block 2.

Process P_3 (112 KB)

⇒ Block 1, Block 2 & Block 4 are not available as their size is less than 112 KB.

⇒ Block 3 memory remain ⇒ 200 − 112 = 88 KB

⇒ Block 5 memory Remain ⇒ 600 − 112 = 488 KB

⇒ thus Block 3 is best, P_3 placed in Block 3.

Process P_4 (426 KB) :→

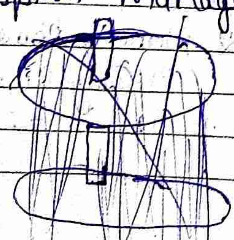
⇒ Block 1, Block 2, Block 3, & Block are not available

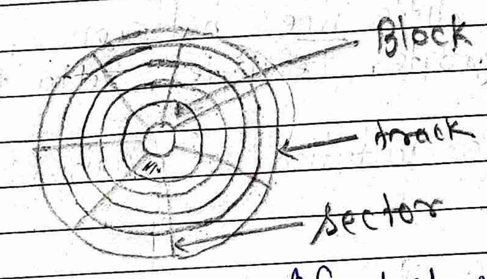
⇒ Block 5 memory Remain ⇒ 600 − 426 = 174 KB

⇒ P_3 Placed in Block 5.

Disk Management :→ The operating system is responsible for various operations of Disk management. Mordern operating systems are constantly growing their range of services and add-ons, and all operating systems implement four essential operating system administration functions. These functions are as follows :→

1. Process management.
2. Memory management.
3. File & Disk Management
4. I/O System management.





* Number of tracks × No Of Sectors = Blocks

Que :→ 10 disk are attached with a Spindle. It is divided into 10 tracks & 5 sectors. Plates of the disks are double sided, each Block of Disk can hold 1 GB of memory. identify capacity.

Solution :→ For 1 Disks
⇒ tracks × sectors = Blocks
⇒ 10 × 5 = 50 Blocks
disks are double sided, so No of Blocks = 2 × 50 = 100 Blocks

100 Blocks in 1 disk.

So,
for 10 Disk,
No of Blocks are = 100 × 10 = 1000 Blocks

Here, every Block capacity is 1 GB of memory
so total capacity = 1000 GB

A.

Que :→ What is difference between , Byte, Kilo B, MegaByte, GigaByte, TeraByte, Zetta Byte.

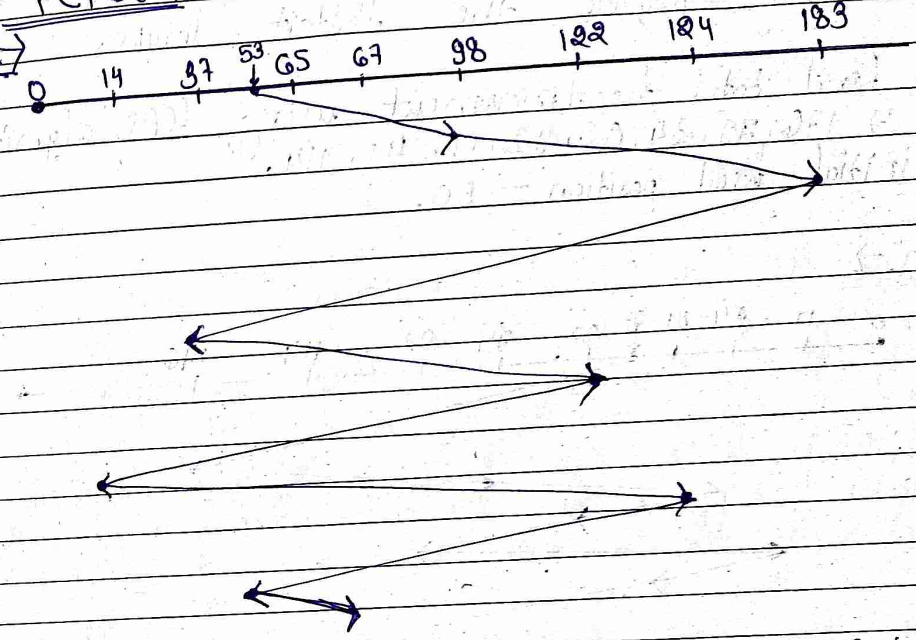
Solution:→ Tabular Representation of various memory sizes.

Name	Equal to	Size (in Bytes)
Bit (b)	1 Bit	1/8
Nibble	4 Bits	1/2 (rare)
Byte (B)	8 Bits	1
Kilobyte (KB)	1024 Bytes	1024
Megabyte (MB)	1024 KB	1,048,576
Gigabyte (GB)	1024 MB	1,073,741,824
Terabyte (TB)	1024 GB	1,099,511,627,776
Petabyte (PB)	1024 TB	1,125,899,906,842,624
Exabyte (EB)	1024 PB	1,152,921,504,606,846,976
Zettabyte (ZB)	1024 EB	1,180,591,620,717,411, 303,424
Yottabyte (YB)	1024 ZB	1,208,925,819,614,629, 174,706,176

Que :→ By using FCFS algorithm find out total head movement :→ 98, 183, 37, 122, 14, 124, 65, 67, head starts :→ 53.

Ex :→ FCFS :→

Solution :→



total head-movement (cylinder) = (98-53) + (183-98) + (183-37) + (122-37) + (122-14) + (124-14) + (124-65) + (67-65)

= 45 + 85 + 146 + 85 + 108 + 110 + 59 + 2

= 640 Ans.

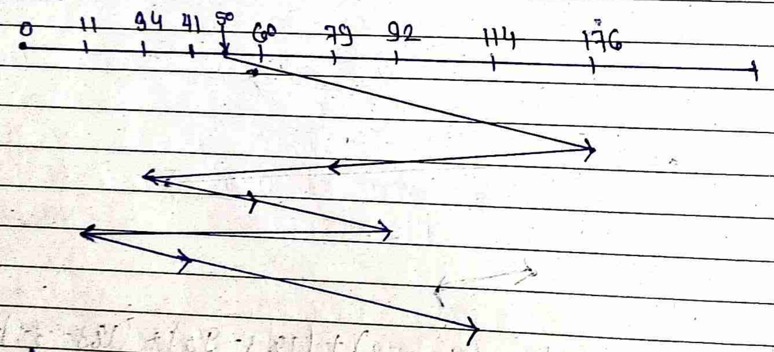
Disk Scheduling Algorithms :→

① First come first serve (FCFS) :→ FCFS is the simplest disk scheduling algorithm.

As the name suggests, this algorithm entertains requests in the order they arrive in the disk queue. The algorithm looks very fair and there is no starvation (all requests are serviced sequentially) but generally, it does not provide the fastest service.

Que:- find total head movement using FCFS algorithm.
:- 176, 79, 34, 60, 92, 11, 41, 114.
initial head position → 50.

Solution:-



total head movement:- (176 − 50) + (176 − 79) + (79 − 34) + (60 − 34) + (92 − 60) + (92 − 11) + (41 − 11) + (114 − 41)

= 126 + 97 + 45 + 26 + 32 + 81 + 30 + [illegible]

= 510 A

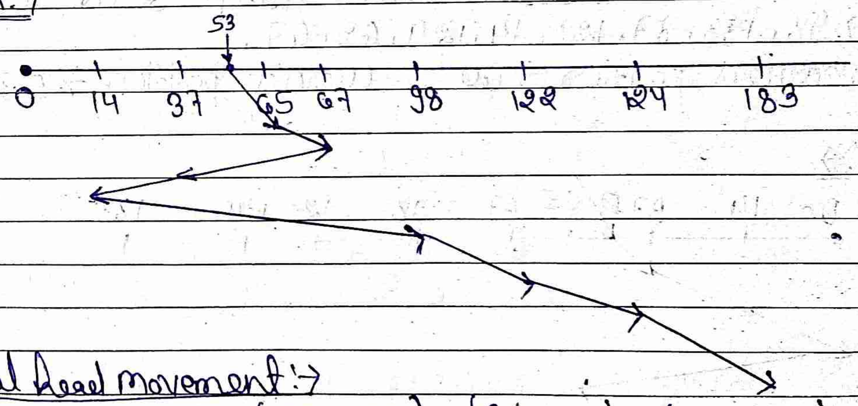
(ii) Shortest seek time first:- (SSTF):- SSTF is abbreviation of Shortest Seek time first which is a disk scheduling algorithm.

It selects the request which is closest to the current head position before moving the head away to service other requests. This is done by selecting the request which has the least seek time from the current head position.

SSTF scheduling priority is given to those processes which have the shortest seek, even if these requests are not the first ones in the queue. To implement this, the seek time of every request is calculated in advance in the queue and then requests are scheduled according to their seek time.

Que:- find total headmovement using SSTF algorithm
:- 98, 183, 37, 122, 14, 124, 65, 67, starting → 53.

Solution:-



total head movement:-
= (67 − 53) + (67 − 14) + (183 − 14)
= 14 + 53 + 169
= 236 A

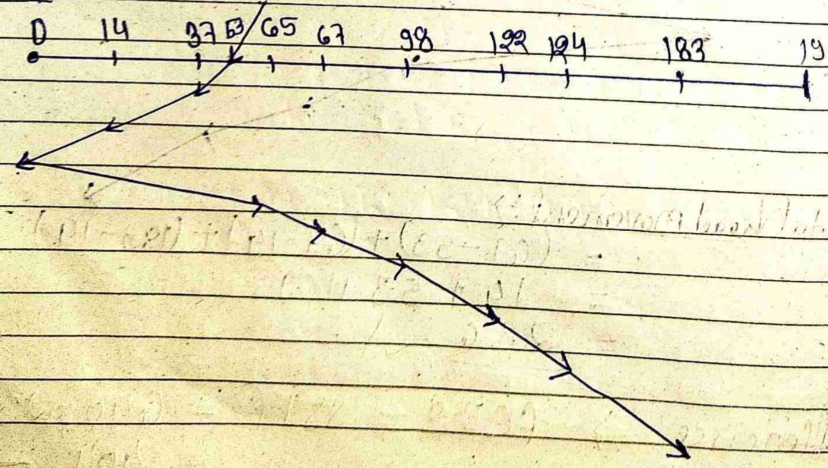
Difference ⇒ FCFS − SSTF = 640 − 236
= 404 A.

③ SCAN (Elevator) Disk Scheduling algorithm:→

In SCAN disk Scheduling algorithm, head starts from one end of the disk and moves towards the other end, servicing requests in between one by one & reach the other end. Then the direction of the head is reversed & the process continues as head continuously scan back & forth to access the disk. So, this algorithm works as an elevator & hence also known as the elevator algorithm. As a result, the requests at the midrange are serviced more and those arriving behind the disk arm will have to wait.

Que:→ Find total headmovement using SCAN algorithm
→ 98, 183, 37, 122, 14, 124, 65, 67.
previous position → 60, current position → 53

Solution:→



Total headmovement:→ (53−37) + (37−14) + (14−0) + (65−0) + (67−65) + (98−67) + (122−98) + (124−122) + (183−124)
= 16 + 23 + 14 + 65 + 2 + 31 + 24 + 2 + 59
= 236 Ans

④ C-SCAN (Circular Elevator) Disk scheduling algorithm:→

The circular scan scheduling algorithm is a modified version of the SCAN disk scheduling algorithm that deals with the inefficiency of the SCAN algorithm by servicing the requests more uniformly. Like SCAN, C-SCAN moves the head from one end servicing all the requests to the other end. However, as soon as the head reaches the other end, it immediately returns to the begining of the disk without servicing any requests on the return trip & starts servicing again once reaches the begining. This is also known as the 'circular Elevator Algorithm' as it essentially treats the cylinders as a circular list that wraps around from the final cylinder to the first one.

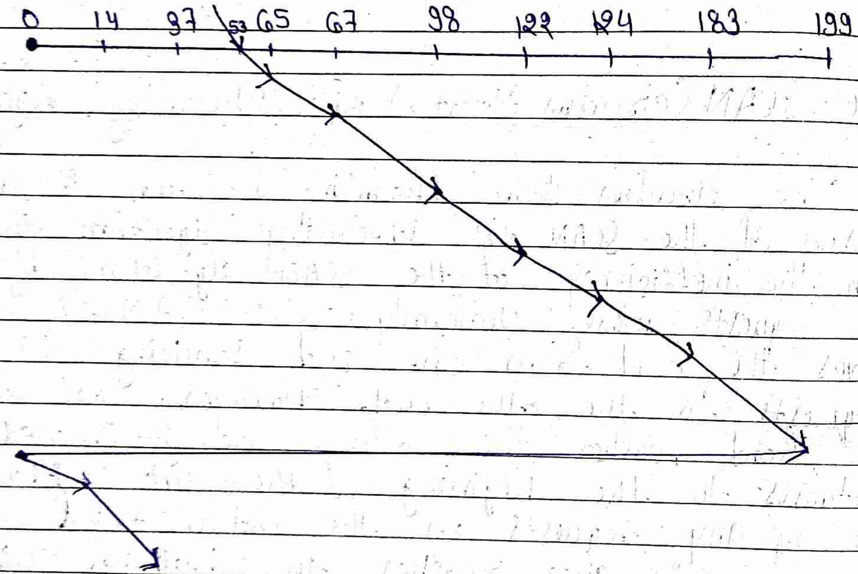
Advantages:→

* Works well with moderate to heavy loads.
* It provides better response time & uniform waiting time.

Que:→ Find total headmovement using C-SCAN algorithm:→ 98, 183, 37, 122, 14, 124, 65, 67

previous → 40, current position → 53

Solution:→



total headmovement:→ (65−53)+(67−65)+(98−67)+(122−98) + (124−122)+(183−124)+(199−183)+ +(199−0)+(14−0)+(37−14)

= 12+2+31+24+2+59+16+199+14+23

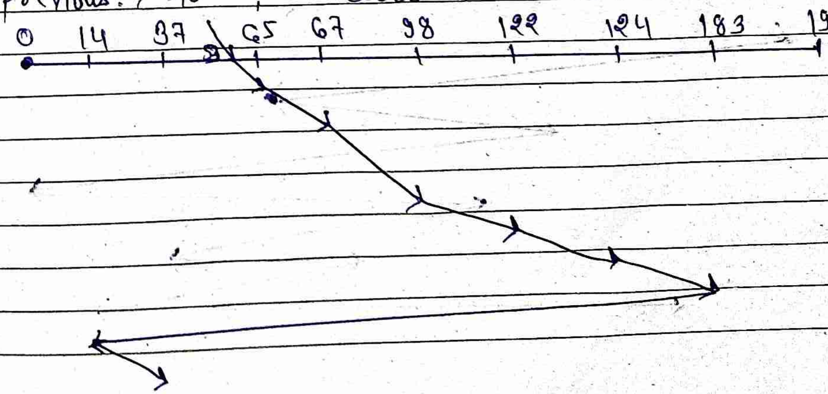
= 382 A.

⑤ C-Look Disk Scheduling algorithm:→ C-Look (Circular Look) Disk Scheduling Algorithm is an enhanced version of both SCAN as well as Look disk Scheduling algorithms. This algorithm also uses the idea of wrapping the tracks as a circular cylinder as C-SCAN algorithm but the seek time is better than C-SCAN algorithm. We know that C-SCAN is used to avoid Starvation & Services all the requests more uniformly, the same goes for C-look. In this algorithm, the head services requests only in one direction (either left or right) until all the requests in this algorithm are not serviced & then jumps back to the farthest request on the other direction & service the remaining requests which gives a better uniform servicing as well as avoids wasting seek time for going till the end of the disk.

Que:→ find total headmovement using C-look algorithm.
:→ 98, 183, 37, 122, 14, 124, 65, 67,

previous:→ 40, current :→ 53

Solution:→



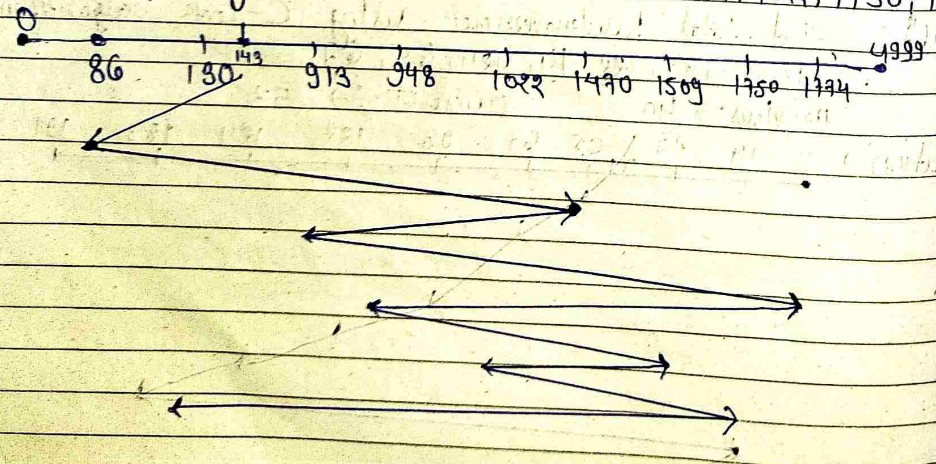
total head movement :→ (65 − 53) + (67 − 65) + (98 − 67) + (122 − 98) + (124 − 122) + (183 − 124) + (183 − 14) + (37 − 11)

= 12 + 2 + 31 + 24 + 2 + 59 + 169 + 26
= 325 Ans.

Ques :→ Disk driver has 5000 cylinder number as 0 to 4999. the head currently serving request at cylinder 143 & the previous request was cylinder 125. The queue pending request is, 86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130.
(i) FCFS (ii) SSTF, (iii) SCAN (iv) C-SCAN (v) C-look.

Solution :→

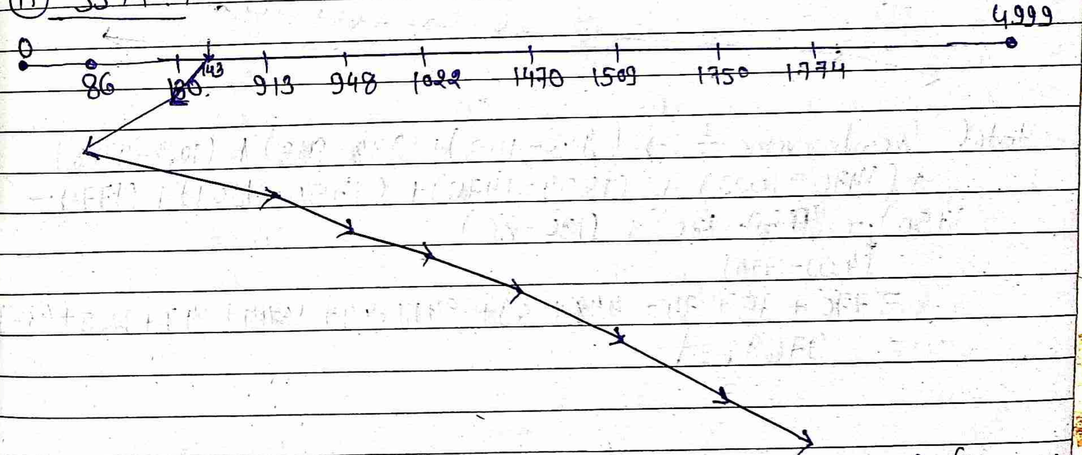
FCFS :→ c.r. = 143, p.r. = 125
pending :→ 86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130.



total head movement :→ (143 − 86) + (1470 − 86) + (1470 − 913) + (1774 − 913) + (1774 − 948) + (1509 − 948) + (1509 − 1022) + (1750 − 1022) + (1750 − 130)

= 57 + 1384 + 557 + 861 + 826 + 561 + 487 + 728 + 1620
= 7081 Ans.

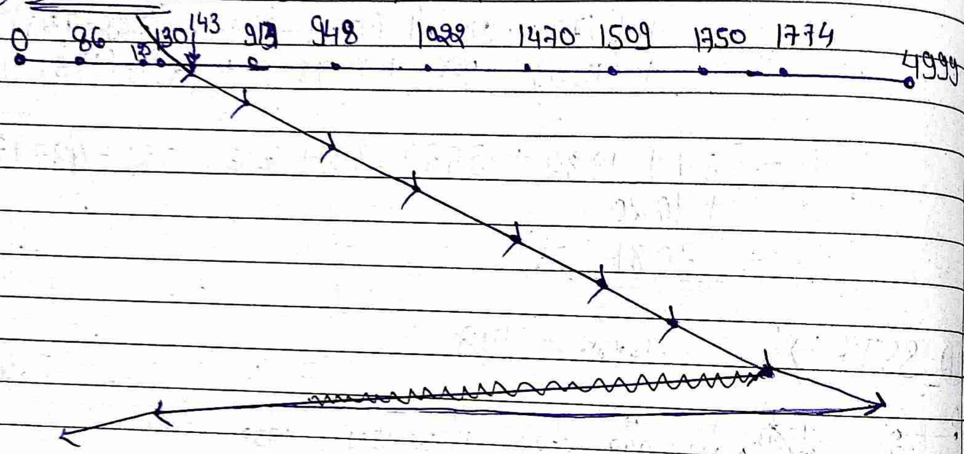
(ii) SSTF :→ C.r. = 143



total head movement :→ (143 − 130) + (130 − 86) + (913 − 86) + (948 − 913) + (1022 − 948) + (1470 − 1022) + (1509 − 1470) + (1750 − 1509) + (1774 − 1750)

= 13 + 44 + 827 + 35 + 76 + 448 + 39 + 241 + 24
= 1745 Ans.

(iii) SCAN :→ C.r. = 143 , P.r. = 125

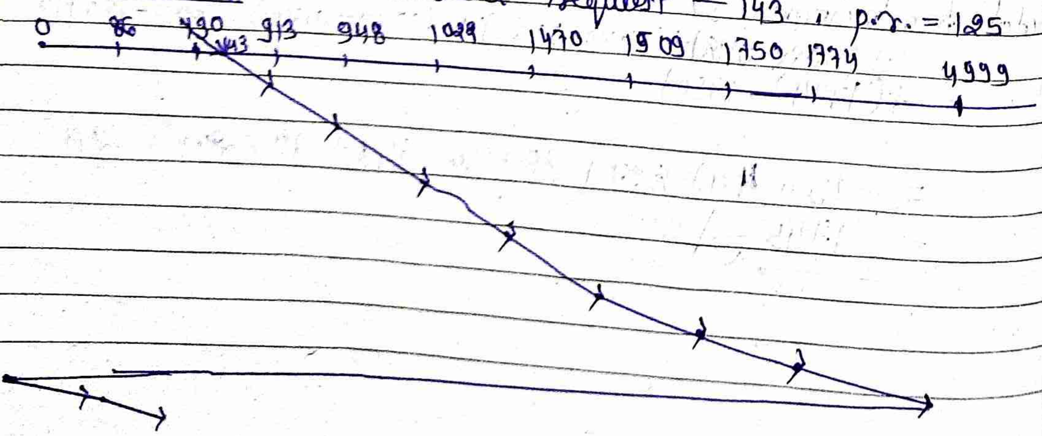


total Headmovement :→ (913 − 143) + (948 − 913) + (1022 − 948) + (1470 − 1022) + (1509 − 1470) + (1750 − 1509) + (1774 − 1750) + ~~(4999 − 130)~~ (4999 − 1774) + (130 − 86)

= 770 + 35 + 74 + 448 + 39 + 241 + 24 + ~~1644~~ 44 + 3225 + 4869

= 9769. A.

(iv) C-SCAN :→ Current request − 143 , p.r. = 125

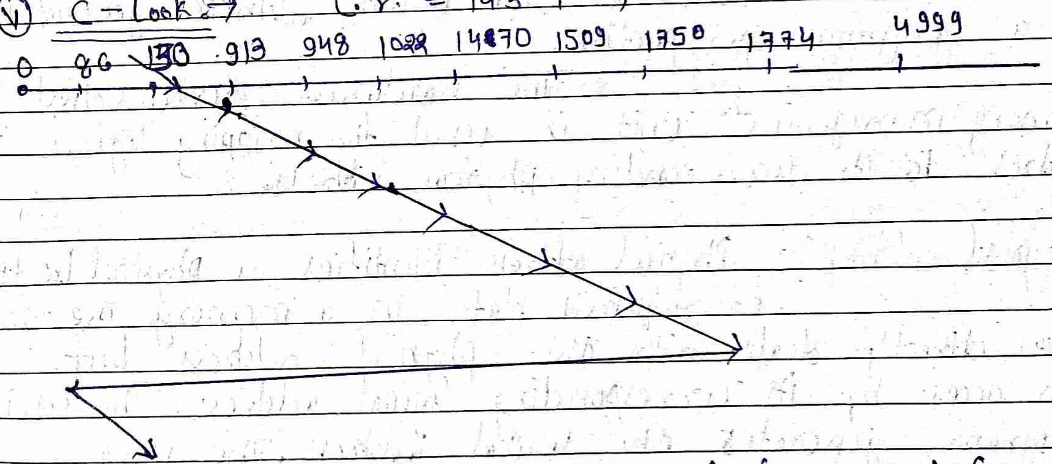


Total headmovement :→ (913 − 143) + (948 − 913) + (1022 − 948) + (1470 − 1022) + (1509 − 1470) + (1750 − 1509) + (1774 − 1750) + (4999 − 1774) + (4999 − 0) + (86 − 0) + (130 − 86)

= 770 + 35 + 74 + 448 + 39 + 241 + 24 + ~~144~~ + 3225 + 4999 + 86 + 44

= 9985 A.

(v) C − Look :→ C.r. = 143 , P.r. = 125



total Headmovement :→ (913 − 143) + (948 − 913) + (1022 − 948) + (1470 − 1022) + (1509 − 1470) + (1750 − 1509) + (1774 − 1750) + (1774 − 86) + (130 − 86)

= 770 + 35 + 74 + 448 + 39 + 241 + 24 + 1688 + 44

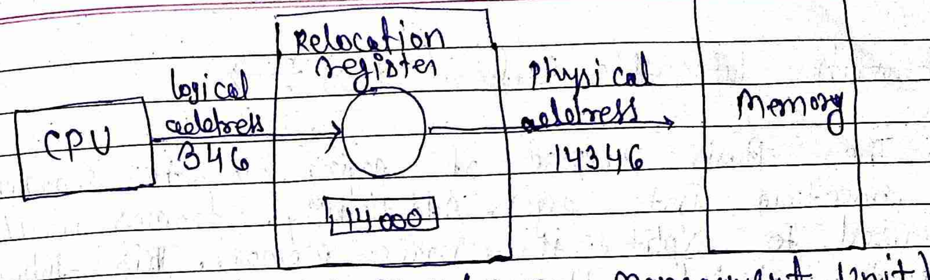
= 3363 A

Logical & Physical Address :→

Logical Address :→ logical address is generated by CPU while a program is running. The logical address is virtual address as it does not exist physically therefore, it is also known as virtual address. This address is used as a reference to access the physical memory location by CPU. The term logical address space is used the set of all logical addresses generated by a program's perspective.

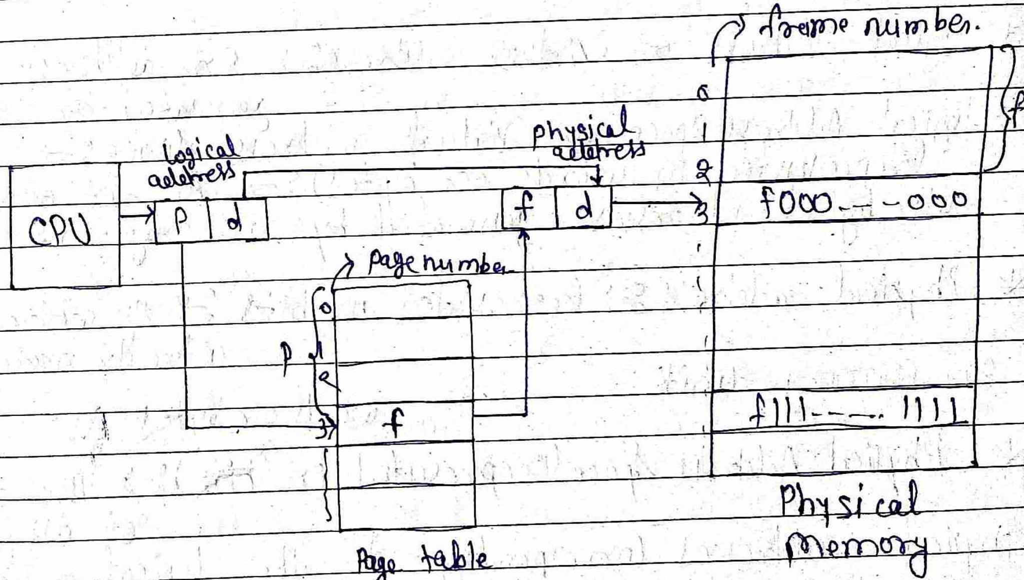
The hardware device called memory management unit is used for mapping logical address to its corresponding physical address.

Physical address :→ Physical address identifies a physical location of required data in a memory. The user never directly deals with the physical address but can access by its corresponding logical address. The user program generates the logical address. ~~The user program generates the logical~~ and think that the program is running in this logical address but the program needs physical memory for its execution. Therefore, the logical address must be mapped to the physical address by MMU before they are used. The term Physical address space is used for all physical addresses corresponding to the logical addresses in a logical address space.



MMU (Memory Management Unit)

Paging :→



Paging is a memory management scheme that eliminates the need for contiguous allocation of physical memory. "The process of retrieving processes in the form of pages from the secondary storage into the main memory is known as paging."

Translation Lookaside Buffer (TLB) :-

The Basic purpose of paging is to separate each procedure into pages. Additionally, frames will be used to split the main memory. This scheme permits the physical address space of a process to be non-contiguous.

Important terminologies :-

* Logical Address or Virtual address :- An address generated by CPU.

* Logical Address Space or Virtual address Space :- (represented in words or bytes) :- The set of all logical addresses generated by a program.

* Physical address :- (represented in bits) :- An address actually available on memory unit.

* Physical Address Space (represented in words or bytes) :- The set of all physical addresses corresponding to the logical addresses.

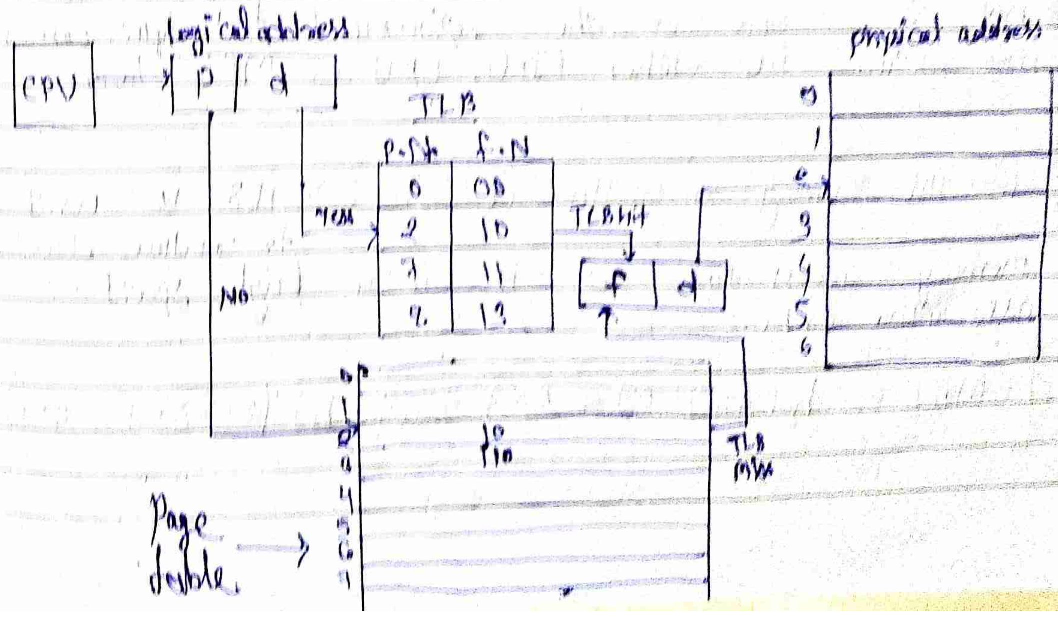
Page number :- Number of bits required to represent the pages in logical address space or page number.

Frame number :- Number of bits required to represent the frame of physical address space or frame number.

Translation look aside buffer (TLB) :- A TLB can be defined as a memory cache which can be used to reduce the time taken to access the page table again and again.

It is a memory cache which is closer to the CPU & the time taken by CPU to access TLB is lesser than that taken to access the main memory.

In other word, we can say that TLB is faster & smaller than the main memory but cheaper & bigger than the register.



Steps in TLB hit:→

(i) CPU generates virtual (logical) address.
(ii) It is checked in TLB (Present).
(iii) Corresponding frame number is retrived, which now tells where in the main momery page lies.

Steps in TLB Miss:→

(i) CPU generates virtual (logical) address.
(ii) It is checked in TLB (Not present).
(iii) Now page number is matched to page table residing in the main memory (assuming page table contains all PTE).
(iv) corresponding frame number is retrieved, which now tells where in the main memory page lies.
(v) The TLB is updated with new PTE (if space is not there, one of the replacement technique comes into picture i.e. either FIFO, LRU or MFU etc.)

Effective memory access time (EMAT):→ TLB is used to reduce effective memory access time as it is a high speed associative cache.

$$EMAT = h(AT(TLB) + AM) + (1-h)(AT(TLB) + 2AM)$$

(single level paging)

Que:→ Find effective access time. TLB Access time = 20 ms, Hit ratio = 80% Memory Access time = 100ms,

Solution:→

given:→
* Number of levels of page table = 1
* TLB Access time = 20ms
* Main memory Access time = 100 ms.
* TLB Hit Ratio = 80% = 0.8

* single level Paging = (1+1)
* Double level = (2+1)
* three level = (3+1)

Calculating Effective Access time:→

$EAT = h(AT(TLB) + AM) + (1-h)(AT(TLB) + 2AM)$

$= 0.8(20 + 100) + (1-0.8)(20 + 2 \times 100)$
$= 0.8 \times 120 + 0.2 \times 220$
$= 96.0 + 44.0$
$= 140\ ms$
$= 140000\ ns$ Ans.

1 milisecond = 10^6 nanosecond.
1 micro = 10^3 ns
1 mili = 10^3 ms

Que:→ Find EMAT. Hit ratio = 98%, TLB Access time = 20ns, memory Access time = 100ns, calculate EAT. (single level)

Solution:→
* TLB Access time:→ 20ns
* Hit ratio → 98% = .98
* Main Memory access time = 100ns

Calculation :→

$EMAT = h(AT(TLB) + AM) + (1-h)(AT(TLB) + (1+1) AM)$

$= 0.98(20 + 100) + 0.02(20 + 2 \times 100)$

$= 0.98 \times 120 + 0.02 \times 220$

$= 117.6 + 4.4$

$= 122$ ns Ans.

Que :→ Find effective memory access time. TLB Access time = 20 ns, M.M. access time = 100 ns. Hit ratio = 90%.

Solution :→ given :→

⇒ * Hit ratio = 90% = 0.9

⇒ * M.M. Access time = 100 ns

⇒ * TLB Access time = 20 ns

Calculation :→

$EMAT = h(AT(TLB) + AM) + (1-h)(AT(TLB) + (1+1) AM)$

$= 0.9(20 + 100) + 0.1(20 + 2 \times 100)$

⇒ $0.9 \times 120 + 0.1 \times 220$

⇒ $108.0 + 22.0$

⇒ 130 ns Ans.

Que :→ find TLB Access time. M.M.A.T. = 100 ns, Hit ratio = 60%, EMAT = 160 ns

Solution :→ given that :→

* EMAT = 160 ns
* Hit ratio = 60% or 0.6
* MM Access time = 100 ns

let TLB Access time is = x ns

Calculation :→

$EMAT = h(AT(TLB) + AM) + (1-h)(AT(TLB) + (1+1) AM)$

⇒ $160 = 0.6(x + 100) + (0.4)(x + 2 \times 100)$

⇒ $160 = 0.6x + 60 + 0.4x + 80$

⇒ $160 = x + 140$

⇒ $x = 20$

⇒ TLB access time is = 20 ns. Ans.

Page Replacement Algorithms:→ The page replacement algorithm decides which page is to be replaced. The process of replacement is sometimes called swap out or write to disk. page replacement is done when the request page is not found in the main memory (page fault).

Page fault:→ A page fault happens when a running program accesses a memory page that is mapped into the virtual memory but not loaded in physical memory. Since actual physical memory is much smaller than virtual memory, page fault happens.

Page replacement algorithms:→

① **first in first out (FIFO):→** In this algorithm, a queue is maintained. The page which is assigned the frame first will be replaced first. In other words, the page which resides at the rare end of the queue will be replaced on the every page fault.

Que:→ The given frame sequence are 4 3 2 1 4 3 5 4 3 2 1 5. identify page fault using FIFO.
frame → 3

Solution:→

	4	3	2	1	4	3	5	4	3	2	1	5
f_1	4	4	4	1	1	1	5	H	H	5	5	H
f_2		3	3	3	4	4	4	I	I	2	2	I
f_3			2	2	2	3	3	T	T	3	1	T
	*	*	*	*	*	*	*	✓	✓	*	*	✓

page fault = 9 , Hit = 3

(ii) **Least recent used (LRU) page replacement algorithm:→** This algorithm replaces the page which has not been refered for a long time. This algorithm is just opposit to ~~optimal~~ optimal page replacement algorithm

Que:→ The given frame sequence are 4 3 2 1 4 3 5 4 3 2 1 5. Identify page fault using LRU. frame → 3

Solution:→

	4	3	2	1	4	3	5	4	3	2	1	5
f_1	4	4	4	1	1	1	5	H	H	2	2	2
f_2		3	3	3	4	4	4	I	I	4	1	1
f_3			2	2	2	3	3	T	T	3	3	5
	*	*	*	*	*	*	*	✓	✓	*	*	*

page fault :→ 10
Hit :→ 2
Hit ratio :→ $\frac{2}{12} = \frac{1}{6}$, Miss ratio $= 1 - \frac{2}{12}$
$= \frac{10}{12} = \boxed{\frac{5}{6}}$

(iii) Optimal Page replacement Algorithms :→ This algorithms replaces the page which will not be referred for so long in future. Although it can not be practically implementable but it can be used as a benchmark. other algorithms are compared to this in terms of optimality.

Que:→ The given frame sequence are 4 3 2 1 4 3 5 4 3 2 1 5. identify Page fault using Optimal. frame → 3

Solution:→

4	3	2	1	4	3	5	4	3	2	1	5
4	4	4	4	H	H	4	H	H	2	2	H
	3	3	3	I	I	3	I	I	3	1	I
		2	1	T	T	5	T	T	5	5	T
*	*	*	*	✓	✓	*	✓	✓	*	*	✓

Page fault = 7
Hit = 5   Hit ratio = 5/12

Que:→ The given frame sequence are 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1 identify Page fault. using

(i) FIFO (ii) LRU (iii) optimal
frame → 3.

Solution:→ (i) FIFO:→

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	2	H	2	2	4	4	4	0	H	H	0	0	H	H	7	7	7
	0	0	0	I	3	3	3	2	2	2	I	I	1	1	I	I	1	0	0
		1	1	T	1	0	0	0	3	3	T	T	3	2	T	T	2	2	1
*	*	*	*	✓	*	*	*	*	*	*	✓	✓	*	*	✓	✓	*	*	*

Page fault = 15
Hit = 5 , Hit ratio = 5/20 = 1/4

(ii) LRU:→

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	2	H	2	H	4	4	4	0	H	H	1	H	1	H	1	H	H
	0	0	0	I	0	I	0	0	3	3	I	I	3	I	0	I	0	I	I
		1	1	T	3	T	3	2	2	2	T	T	2	T	2	T	7	T	T
*	*	*	*	✓	*	✓	*	*	*	*	✓	✓	*	✓	*	✓	*	✓	✓

page fault :→ 12.
Hit → 8 , Hit ratio = 8/20 = 2/5

(iii) Optimal:→

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	2	H	2	H	2	H	H	2	H	H	2	H	H	H	7	H	H
	0	0	0	I	0	I	4	I	I	0	I	I	0	I	I	I	0	I	I
		1	1	T	3	T	3	T	T	3	T	T	1	T	T	T	1	T	T
*	*	*	*	✓	*	✓	*	✓	✓	*	✓	✓	*	✓	✓	✓	*	✓	✓

Page fault = 9 , Hit = 11
Hit Ratio = 11/20

Belady's Anomaly :→ In the case of LRU & optimal page replacement algorithms It is seen that the number of page faults will be reduced if we increase the number of frames. However, Belady found that, in FIFO page replacement algorithm, the number of page faults will get increased with the increment in number of frames.

This is the strange behaviour shown by FIFO algorithm in some of the cases. This is an Anomaly called as Belady's Anomaly.

ex:→ The reference string is given as 0 1 5 3 0 1 4 0 1 5 3 4. Let's analyze the behaviour of FIFO algorithm in two case.

Case :→ 1 : Number of frames :→ 3

0	1	5	3	0	1	4	0	1	5	3	4
0	0	0	3	3	3	4	H	H	4	4	H
	1	1	1	0	0	0	I	I	5	5	I
		5	5	5	1	1	T	T	1	3	T
*	*	*	*	*	*	*	✓	*	*	*	✓

page fault = 9

Case :→ 2 : No of frame → 4

0	1	5	3	0	1	4	0	1	5	3	4
0	0	0	0	H	H	4	4	4	4	3	3
	1	1	1	I	I	1	0	0	0	0	4
		5	5	T	T	5	5	1	1	1	1
			3	✓		3	3	3	5	5	5
*	*	*	*	✓	✓	*	*	*	*	*	*

page fault :→ 10

Therefore, in this example the number of page fault is increasing by increasing the number of frames hence this differs from Belady Anomaly.

Fragmentation :→ Fragmentation is an unwanted problem in the operating system in which the processes are loaded & unloaded from memory, & free memory space is fragmented. Processes can't be assigned to memory blocks due to their small size, and the memory blocks stay unused. It is also necessary to understand that as programs are loaded & unloaded (Deleted) from memory, they generate free space or a hole in the memory. These small blocks cannot be alloted to new arriving processes, resulting in inefficient memory use.

"As the process is loaded & unloaded from memory, these areas are fragmented into small pieces of memory that cannot be allocated to incoming processes. It is called fragmentation."

Types of fragmentation :→

(i) Internal fragmentation
(ii) External fragmentation.

Date ______
Page ______

Difference between internal & External fragmentation

internal fragmentation	External fragmentation
(i) memory blocks square measure appointed to process.	(i) Memory blocks square measure appointed to the method.
(ii) Internal fragmentation happens ~~whe~~ when the method or process is smaller than memory.	(ii) External fragmentation happens when the process is removed.
(iii) The Solution of internal fragmentation is the best fit block.	(iii) The Solution to external fragmentation is compacting & paging.
(iv) Internal ~~fra~~ fragmentation occurs when memory is divided into fixed-sized ~~partition~~ partition.	(iv) External fragmentation occurs when memory is divided into variable size partitions based on the size of processes.
(v) The difference b/w memory allocation & required space or memory is called internal fragmentation.	(v) The unused spaces formed between non-contiguous memory fragments are too small to serve a new process, which is called External fragmentation.
(vi) It occurs in worst fit memory allocation method.	(vi) It occurs in best fit & first fit memory allocation method.