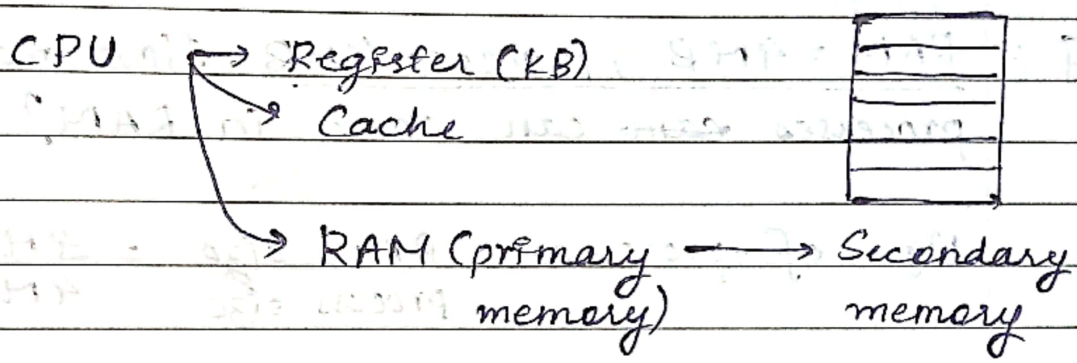
Memory Management & Degree of Multiprogramming

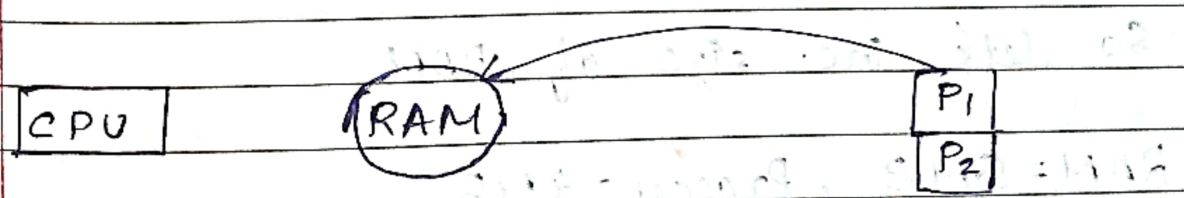
Memory management: Method of managing primary memory

"Goal": Effecient utilization of memory

Method of managing primary memory



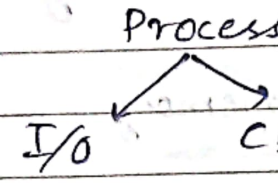
Secondary memory is not directly connected with CPU, ∴ its a bit slower than primary memory, also its size is very large as compared to ~~RAM~~ primary memory.



Multiprogramming: Whenever we are bringing programs from secondary memory to primary mem. than don't bring one process in the RAM, bring more than one processes in RAM.

More processes in RAM, degree of multi-program increases, so CPU utilization increases.

When we bring a process P, & if it wants to have some i/o operation it leaves CPU & CPU is idle & performance decreases:



Q: RAM = 4MB, process = 4MB. How many processes can come in RAM?

$$\text{No. of processes} = \frac{\text{RAM size}}{\text{process size}} = \frac{4MB}{4MB} = 1$$

K = time for i/o operation (70%)
CPU = (1-K) (30%)

Here the CPU utilization is less.

So let's inc. size of RAM

RAM = 8MB, Process = 4MB

$$n = \frac{8}{4} - 2 \qquad K^2 = \text{i/o operation}$$

$$\text{CPU util.} = (1-K^2) = 1-(0.7)^2 = 76\%$$

→ Inc. the no. of processes in the RAM.
→ OS needs to pay attention to allocation & deallocation

Memory Management techniques

(Continuous mem. add. alloc.)
Contiguous

(Non-cont. mem. addr. allocat.)
Non-Contiguous

→ Paging
→ Multilevel paging
→ Inverted paging
→ Segmentation
→ Segmented paging

Fixed partition (Static) (Make fixed slots)

Variable partition (Dynamic) (provide at runtime)

Ready state of process

CPU utilization (effecfiency)

~~Internal fragmentation~~
or
Fixed size partitioning

Whenever we start our PC, first of all OS is mounted on the RAM & the rest space on the RAM is used for allocation of processes.

Points to remember:

1) No. of partitions are fixed
2) Size of each partition may or may not be same.

A process cannot break itself in 2 or more parts to stay in diff. memory locations

3) Contiguous allocation so spanning is not allowed. i.e. a process can only stay in the partition if its size is less than or equal to part. We can put process in any partition taking into consideration its size should be less than partition size.

Internal fragmentation

→ Occurs when memory is distributed in fixed size blocks.

→ If the memory allocated to the process is slightly larger than the memory demanded, then the difference between allocated & demanded memory is called internal fragmentation

External fragmentation

→ It occurs when there is suffecient quantity of area within the memory to satisfy the memory request of a method. However, the process' memory request cannot be fulfilled because of the memory offered is in a non-contiguous manner.

Limitations

1) Internal fragmentation
2) Limit in process size
3) Limitation in degree of multi-programming. We can't bring more & more processes
4) External fragmentation

Advantages

1) Easy to Implement
2)
3)

Variable size partitioning

→ Initially the RAM is empty (i.e. has no processes, only has OS)
→ Whenever a new process comes we grant him space as per the demand of the process

OS	
P_1	2MB
P_2	4MB
P_4	8MB

Advantages

→ No Internal fragmentation
→ No limitation on no. of processes
→ There is no limitation on the process size

Disadvantage

For e.g. this is the current scenario & a new process of 8MB comes

OS		
P_1	2MB	Now the new process can't come in the RAM ∵ we don't have a partition of 8MB
Hole	4MB	
P_3	8MB	
Hole	4MB	
P_5	2MB	

Disadvantages:

1) External ~~frag~~ fragmentation occuss. We can remove it using compaction, but it also has it's own disadvantages.

Compaction: Technique to collect all the free memory present in form of fragments into one large chunk of free memory, which can be used to run other processes.

Disadvantages:

→ We need to stop the running process to move it from one location to other

→ ~~We need~~ Moving a process from one location to the other takes a lot of time

⇒

2) Allocation / De-allocation is difficult

3) Lot of holes are created

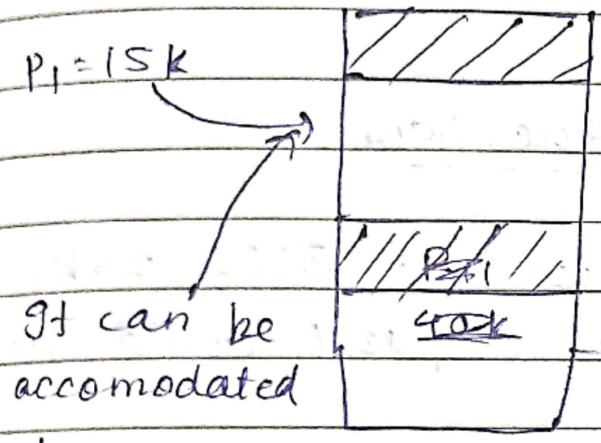
First fit, Next-fit, Best-fit, Worst-fit

→ First fit: Allocate the first hole that is long enough

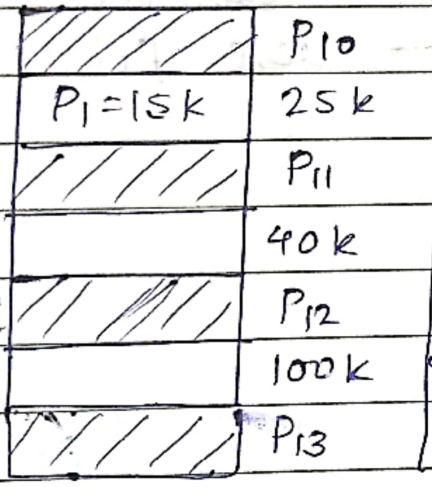
→ Next fit: Same as first fit, but start search always from last allocated hole

Best fit : Allocate the smallest hole that is big enough

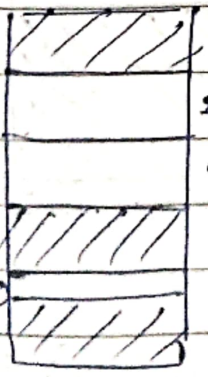
Worst fit : Allocate the largest hole



~~find the best~~ First fit
(find the first ~~best fit loc^n~~ loc^n having size greater than or equal to process)



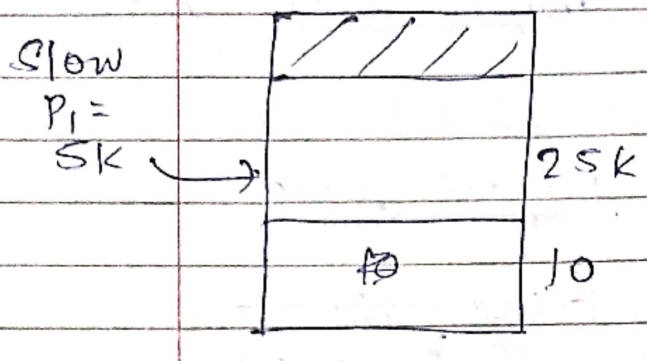
Next fit
Place a pointer at last allocated proc. & start searching for the next first ~~best~~ fit location



Best fit :
Search the entire list for hole with min. internal fragmentation

Worst fit

Search the entire list for the biggest hole among each partition

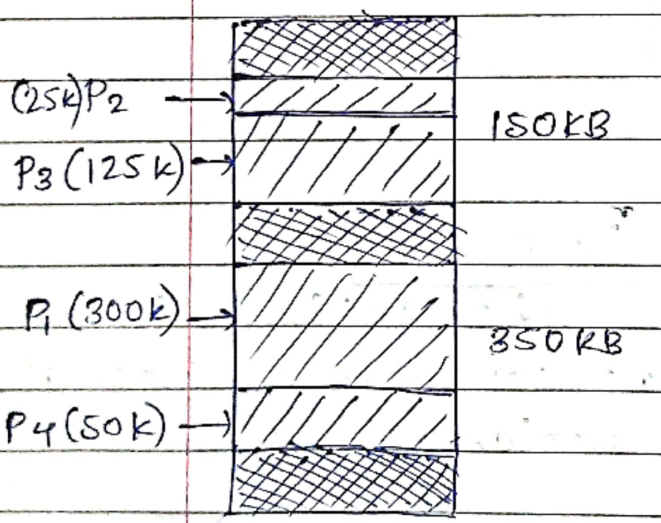


Gate Questions

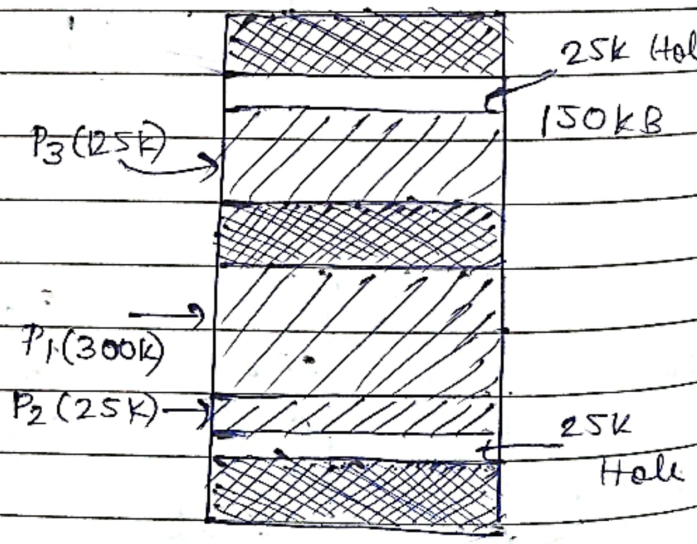
Q Request from processes are 300k, 25k, 125k, 50k resp. The above request could be satisfied by:

1) Best fit but not Worst fit
2) First -11- -11- Best fit
3) Both
4) None

First fit



Best fit

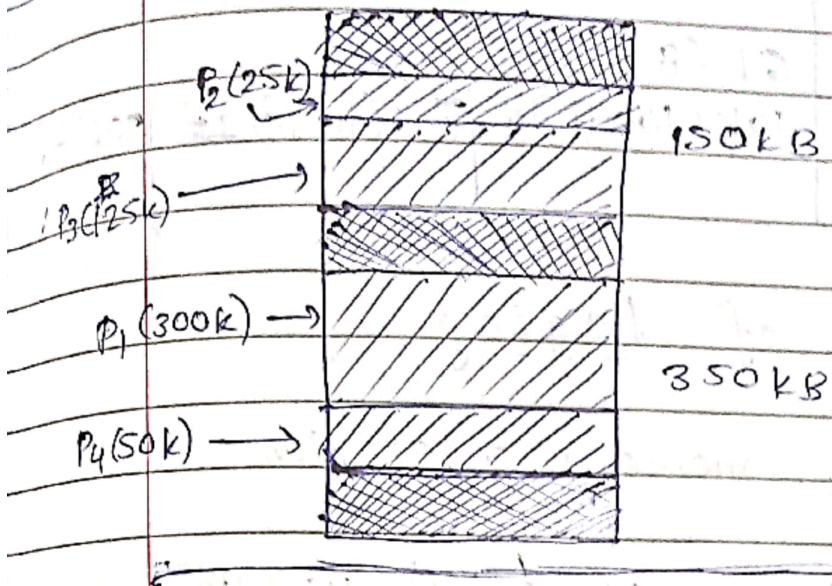


External fragmentation

P_4 can't come

PAGE NO.:

Worst fit



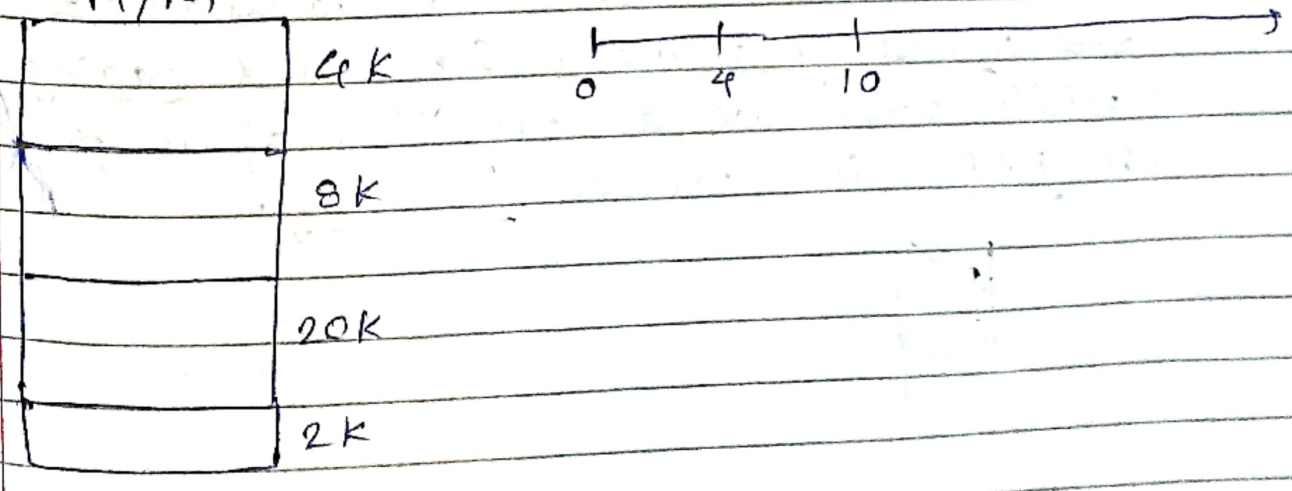
Q: 'BEST FIT'

Req. no. / Req. size / Usage time	Req. no.	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8
	Req. size	2k	14k	3k	5k	6k	10k	7k	20k
	Usage time	4	10	2	8	4	1	8	6

Calculate the time at which J_7 will be completed. 19

a) 17 b) ~~19~~ c) 20 d) 37

M/M



4K	3K(0–2) J_3			
8K	6K(0–8) J_4	6K(8–12) J_5		
20K	14(0–10) J_2	10K(10–11) J_6	7K(11–19) J_7	20K(19–25) J_8
2K	2K(0–4) J_1			