FILE SYSTEM

* Allocation Methods

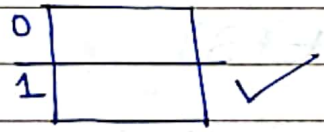
allocate the blocks to file blocks (in disk).

⇒ Space to files allocated, so that disk space utilized effectively.

1) Contiguous allocation

[Vani Notes] → / call

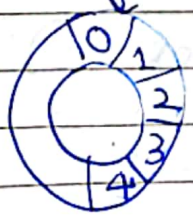
- temp



✓

- file t1 [start = 5, length = 4]

Adv: read performance is excellent.



→ need not go to another track.

∴ putting in contiguous blocks ✓

0, 1 ; 2, 3, 4 ; 5, 6, 7, 8 blocks

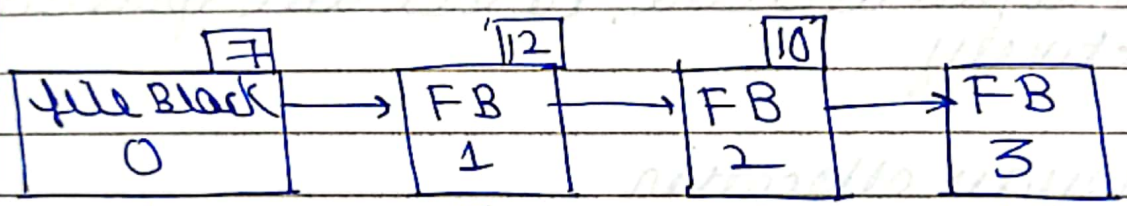
[2, 3, 4 empty].

Now, 4 blocks needed ✓

∵ not contiguous → external fragmentation.

⇒ File 3 → 2.5 B
[Internal fragmentation].
(can't be avoided in any Method)

2. Linked list Allocation →



Phys. Block = 4 → 7 → 12 → 10.

⇒ Non-contiguous.
- (Every file contains Start PB)
- Maintain pointer

→ External fragmentation avoided.

(internal fragmentation in the last block of each file).

Vimp

Disadv: · Random access is very low
(go to track, then shift track...)
· Pointer takes up few bytes.
[Earlier M → no Pointer].

* File Allocation Table

- create array
- for every disk blocks in HD, make entry

File A	....	4

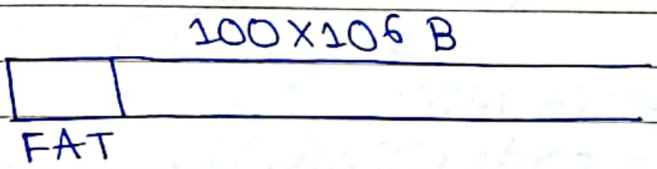
call ⟶ FAT

- Random Access is easier.
- FAT size can be too huge, to be stored in Main Memory.
  [not in Hard Disk] ⟶ in Memory DS.

* Gate 14 on FAT

HD capacity = 100×10^6 B.

- While booting, get into MM (FAT) ⟵ init in HD.



FAT Size = no. of entries × Size of each entry
data block = 1 KB

$$\Rightarrow \frac{100 \times 10^6}{10^3} \times 4 \Rightarrow 0.4 \times 10^6.$$

∴ Free Space Available = 99.6×10^6 B

∴ Total Size, Files can occupy ⟶ 99.6.

* Indexed Allocation - 1

- Contiguous: declare the size of file earlier.
- FAT solves disad. of linked Allocation, ∵ it supports random access.

⇒ FAT ⟶ MM ↑

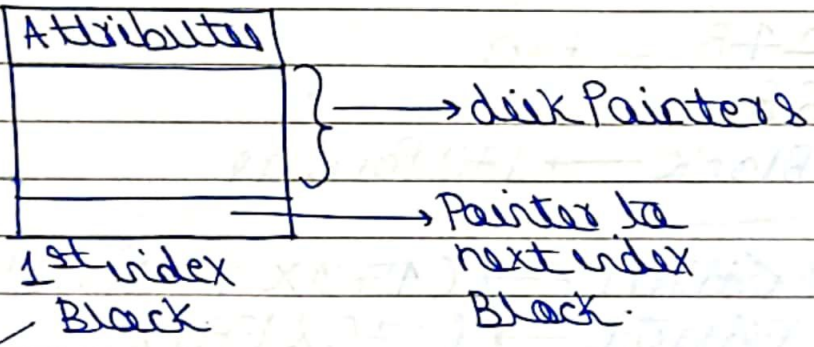
* For every file maintain 1 special block, called index Block / I-Node.
  - What are all the Physical Blocks, where the file is Present.

- File Inode
  File A 10. (go to node 10; 10 shall all the Block no.s (3, 4, 5, 8))
- need not bring FAT in MM. (just bring one node (Inode)).

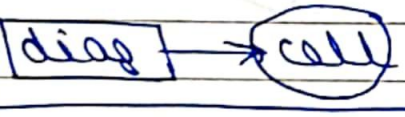
* Indexed Allocation - 2

- If index block (very less no of blocks allocated to file) : index block → internal fragmentation.
- If too many blocks, lot of entries in the index Block will be too many.

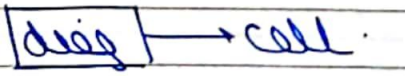
∴ we can maintain many index Blocks.

1) 

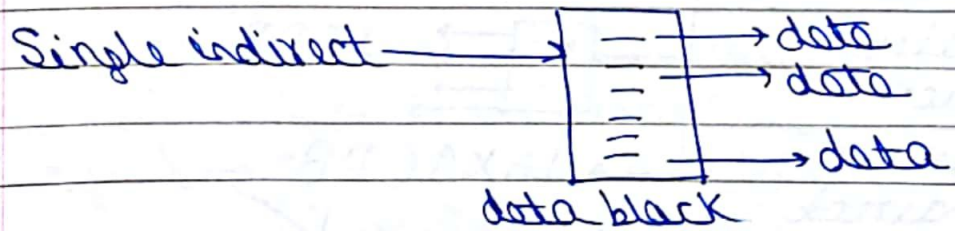
2) Multi level Index.



* Indexed Allocation - 3 → Extended indexed Allocation.



- We can increase exponentially, no. of data blocks a file can have.



GATE QS - 2004

- 10 direct block Pointer → 10 data blocks
- 1 single indirect pointer → 170 data Blocks

Size of each Pointer = 6B
Disk block Size = 1KB

$$\frac{1024B}{6B} = 170.$$

∴ in 1 Block ⟶ 170 Pointers

- 1 double indirect ⟶ (170 × 170) data Blocks.
- 1 triple indirect ⟶ (170 × 170 × 170) data blocks.

∴ A file can have max of:

$$[10 + 170 + 170^2 + 170^3] * 1024B$$

(Ans is wrong in Gate)

Gate 2012 on Indexed Allocation

direct Block	→ 8 data Blocks
indirect Block	→ 16 DB.
doubly indirect	→ 16 × 16 DB

blocks.

7

$\Rightarrow \frac{128B}{8B} \Rightarrow 16$

16 Pointers.

∴ 16 data blocks.

∴ Total no. of data blocks = $(8 + 16 + 16^2) * 128$.

Max File Size.

$8(1+2+32) \times 128$.

$\Rightarrow$ 35KB

* DISK SCHEDULING ALGORITHMS

- Process → CPU burst
  → i/o burst (r/w). [access HD].
- Requests to HD will pile up. How to process these requests?

T-10 —tym→ T-100

- To Serice all requests.
  OS Main Purpose is that HD service all my request in an optimum way.

- Seek time: Moving r/w Head to the appraciate cylinder/track.

eg: A → cylinder 1, B → cylinder 2
(Save Seek time).

HD → Slow.
⇒ [Algo reduce Seek time].

* Disk Scheduling: If queue of requests, which request to be served next.

Hard drives → Slowest Part of PC. ← Vimp.
∵ [When Process gets suspended blocked I/O, it is swapped onto HD, Queue of Process get piled up, & one by one, they get served].
MTS

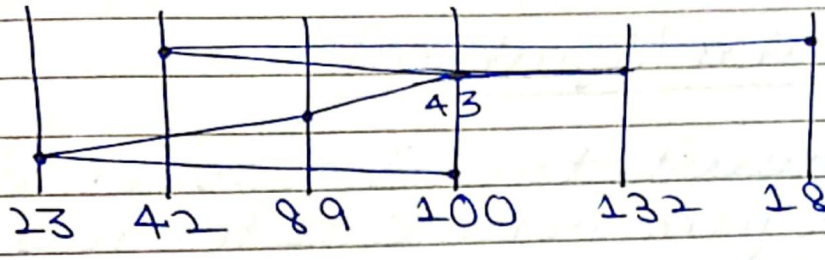
1) FCFS Scheduling

- Simplest
- whatever req comes, scheduled first.
- No ordering of work Queue.
- No Starvation. (every request is serviced).

eg. 23, 89, 132, 142, 187.
init. disk head initially at 100.

(all concentric tracks → cylinder).

How many cylinders we need to cross? (R/W Head moment).



* 77+66+43+90+145 = 421 cylinders Crossed

2) SSTF Scheduling → Shortest Seek time First

- @100 ; 110 , 101 (VIMP)

first.

∵ less no. of cylinders crossed to schedule 101 cylinder.

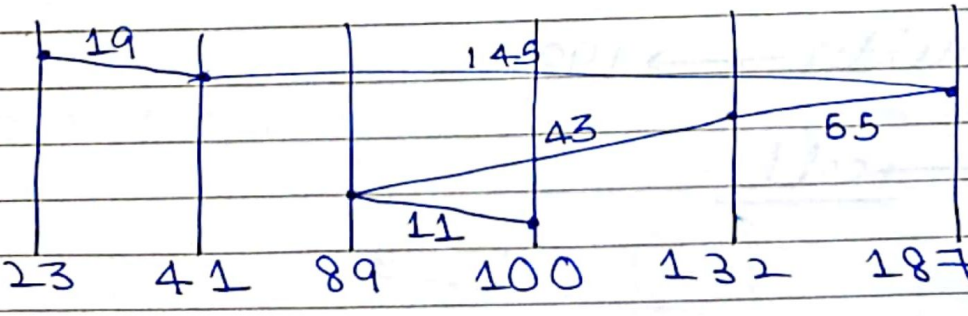
⇒ Like SJF ; least movement of the disk arm from its current

⇒ Reduces total Seek time.

⇒ Starvation is Possible.

⇒ Switching direction may slow this

[100 101 200
102 X
101
102
⋮
⋮
Starvation.

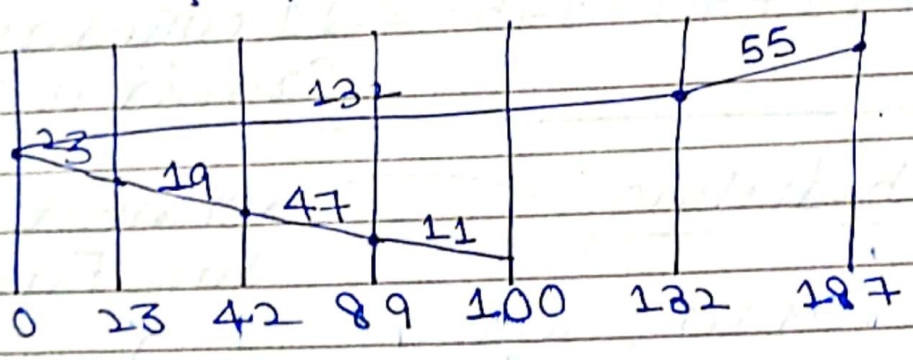


⇒ (273) ∵ R/W Head Movement ↓

∵ Seek time ∝ no. of cylinders Crossed.

* SCAN (Elevator Algorithm)

- going from outside to inside servicing req. and then back from the inside to the outside servicing requests.



∴ 287

* C-SCAN

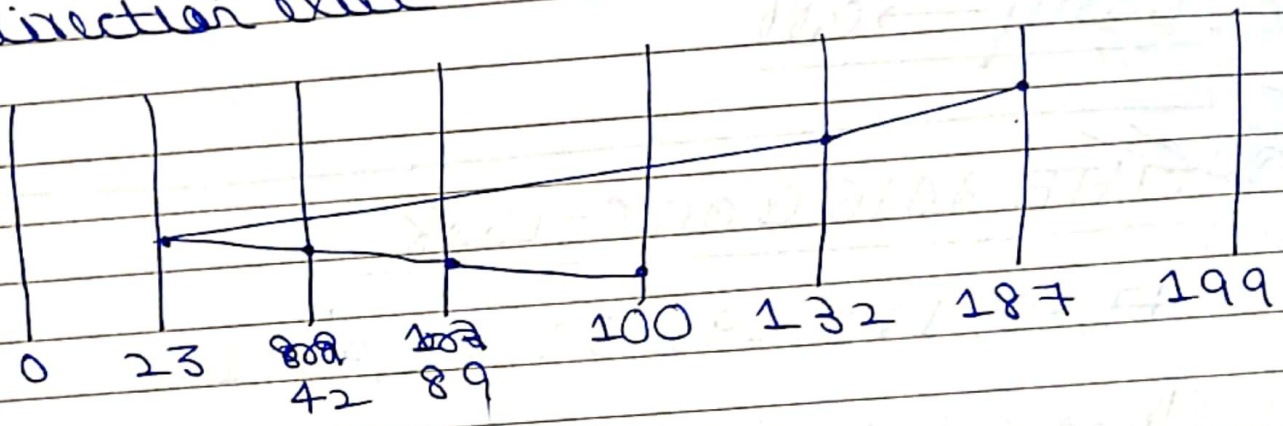
- moves inwards servicing requests until it reaches the innermost cylinder, then jumps to the outside cylinder of the disk without servicing any requests.

Last cylinder ⟶ 199.

disq ⟶ call.

* LOOK

Like SCAN, but stops moving inwards (or outwards), when no more req in that direction exist.

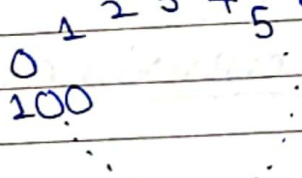


∴ 241 Cylinders

* C-LOOK

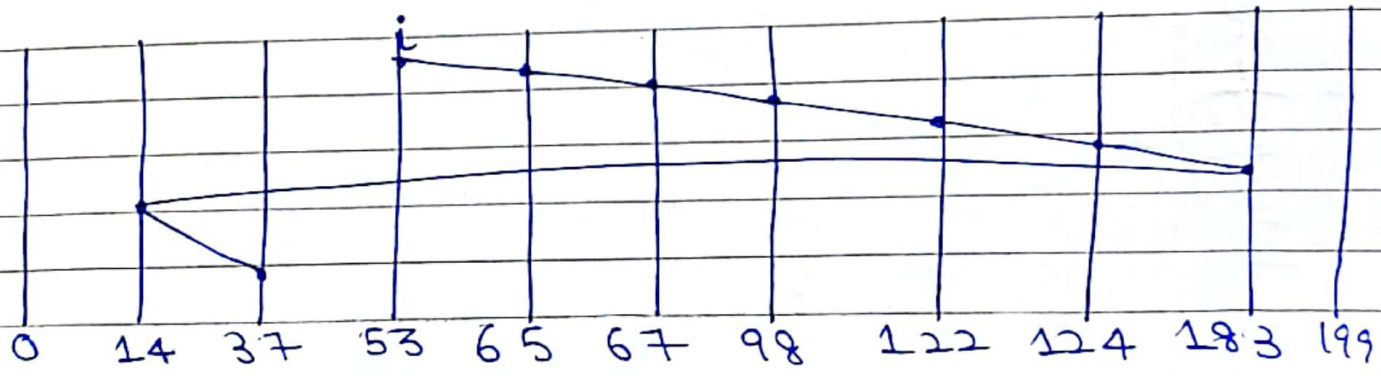
[go to the highest cyclin to lowest]

↳ circular → the way they are serviced.



98, 183, 37, 122, 14, 124, 65, 67

-53



(direction will be given in Q).

if not, assume, it will go to higher no.'s

Theory → call

* GATE 2016 Q on C-look

47, 38, 121, 191, 87, 11, 92, 10.

Head is init at 63.

dirⁿ → moving towards larger cylinder numbers.

[0-199] → total cylinders

diag → call → go till 191, return to 10.
⇒ 346 cylinder mov

0 10 11 38 47 63 ...

DEADLOCK

Q Gate 2006.

$P_1 \longrightarrow$ holds x_1.
$P_2 \longrightarrow$ holds x_2.

(All Processes have taken up all resources)

- Only 2 Processes are done.
(Rest need additional resources).

Sol? R = 9

P_1	P_2	P_3	
5	4	6	← requirement (max).
3	4	2	← allocated. (9) ∵ all are already allocated.
2	0	4	← need (request)

P_2 doesn't need any resource, complete its exec, release every res.
(4)

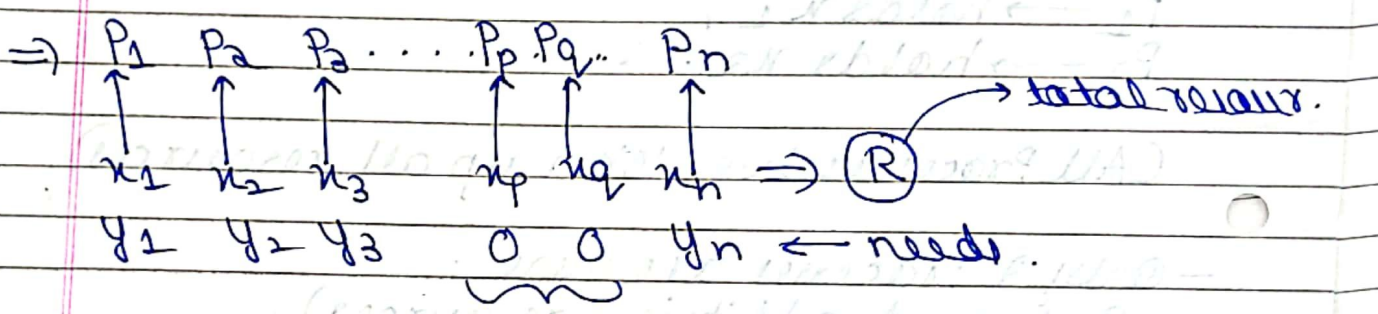
Available → 0.

⇒ if satisfy any 1 need. (if 1✗)
⇒ if proc after exec, release resources, & using those resources, should be able to satisfy the need of atleast 1 process ⟶ if yes,

there is a chance of no deadlock.

⇒ Necessary condn:

Sufficient condn:→ If all needs in total

⇒ 

$(x_p + x_q)$ → Avail resour.

⇒ we should be able to satisfy the need of atleast 1 Process

$$(x_p + x_q) \geq \min_{k \neq p,q} y_k$$

↳ if not true, Deadlock (not sufficient).

$$(x_p + x_q) \geq \sum_{i \neq p,q} y_i$$

↳ sufficient condition.

PROCESS SYNCRONIZATION

[BARRIER PROBLEM]

- stay there, & wait for others.
- (cricket, playground, wall).
  (wait till 11 people arrives.
- [Busy wait till Process arrived is 3].

a) if count-S: inconsistency in var. can occur.

→ count gets lost.
[2:20]

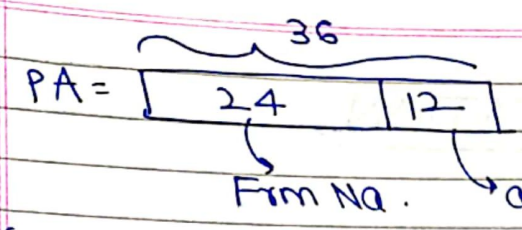
Q 4.51. (Gate 2008)

PA = 36 bits; MM Size = 2^{36}.
LA = 32 bits, Process Size = 2^{32}.

· 3 level Paging.

no. of bits in PTE for frame no. → Q.
(

⇒ in all the PTs: same
[no. of bits need to identify a frame will be same].



(B) 24, 24, 24.

=0.

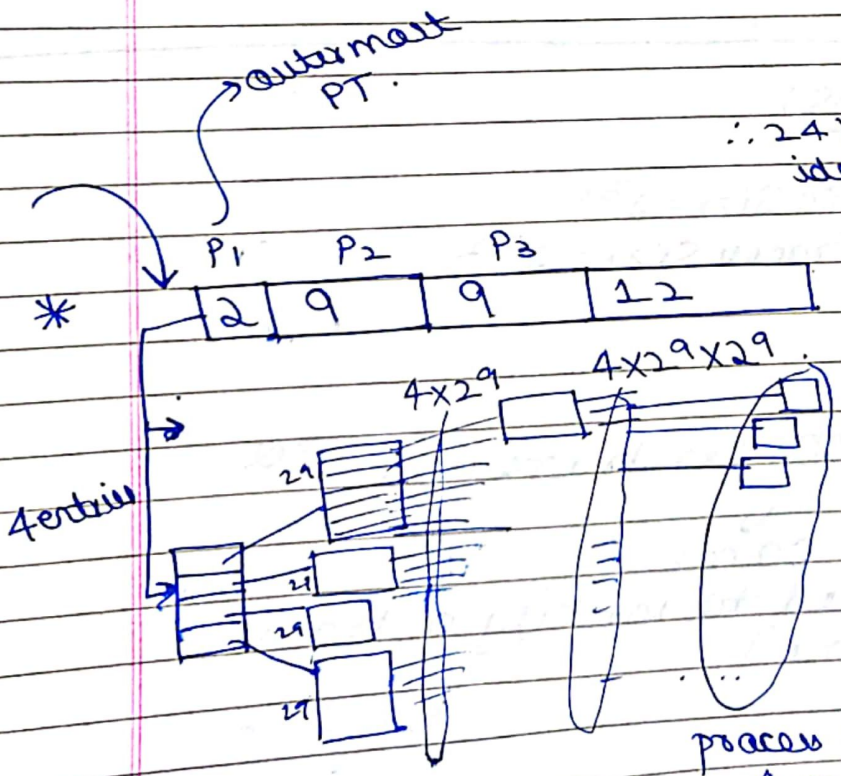
[All the levels will contain same no. of bits needed to access frame no]

- MM / P. Add Space } $= 2^{36}$ B.
- Page Size = 4KB
  $= 2^{12}$ B.

∴ No. of frames = $\frac{\text{PAS}}{\text{frame Size}}$

$\Rightarrow 2^{24}$.

∴ 24 bits needed to identify each frame.



Pg Size

∴ $4 \times 29 \times 29 \times 2^{12}$

$\Rightarrow 2^{32}$.

process contains $4 \times 29 \times 29$ pages.

Q.4.39

TLB.HR = 90%.
m = 150.
TLB Access time = 0.

TLB & Page fault.

- To finish 1 instr^n, how much time?

$$EAIT = \frac{CPU + 2*(EMAT)}{100 + 2*(EMAT)}$$

[EMAT = (VA → PA) + Access the byte from PA.]

$\Rightarrow t + (1-P_t)(2*m) + [m + P_f * PS]$

$\Rightarrow 0 + 0.1 \times 300$

$\Rightarrow$ 30ns.

$150 + \left(\frac{1}{104} \times 8\right)$

$\therefore 30 + 150 + 800$

$\Rightarrow$ 980nsec.

$\therefore$ EAIT = $100 + 2 \times 980 \Rightarrow$ 2060 nsec.