

Virtual Memory (chapter - 8)

Virtual memory is a memory management technique that allows a computer to run programs larger than the size of physical RAM by using a portion of the hard disk as an extension of memory.

Why virtual memory:—

- Physical RAM is limited: large programs cannot fit in RAM
- Allow execution of large program
Even if a program is bigger than RAM, it can run smoothly.
- Increase multiprocessing
more programs can run at the same time.
- provides protection & isolation
Each program process gets its own virtual address space.

Working

Virtual memory works using paging, segmentation. or a combination.

Key idea:

→ A process is divided into pages (virtual memory)

→ RAM is divided into frames (physical memory)

→ Only the required pages are loaded into RAM → this is called demand

$$\frac{m}{f} = \frac{p}{s}$$
$$\frac{m}{1} = \frac{p}{1}$$

paging

Demand paging:

Pages are brought into RAM only when needed.

→ If a process tries to access a page that

is not in RAM → page fault occurs.

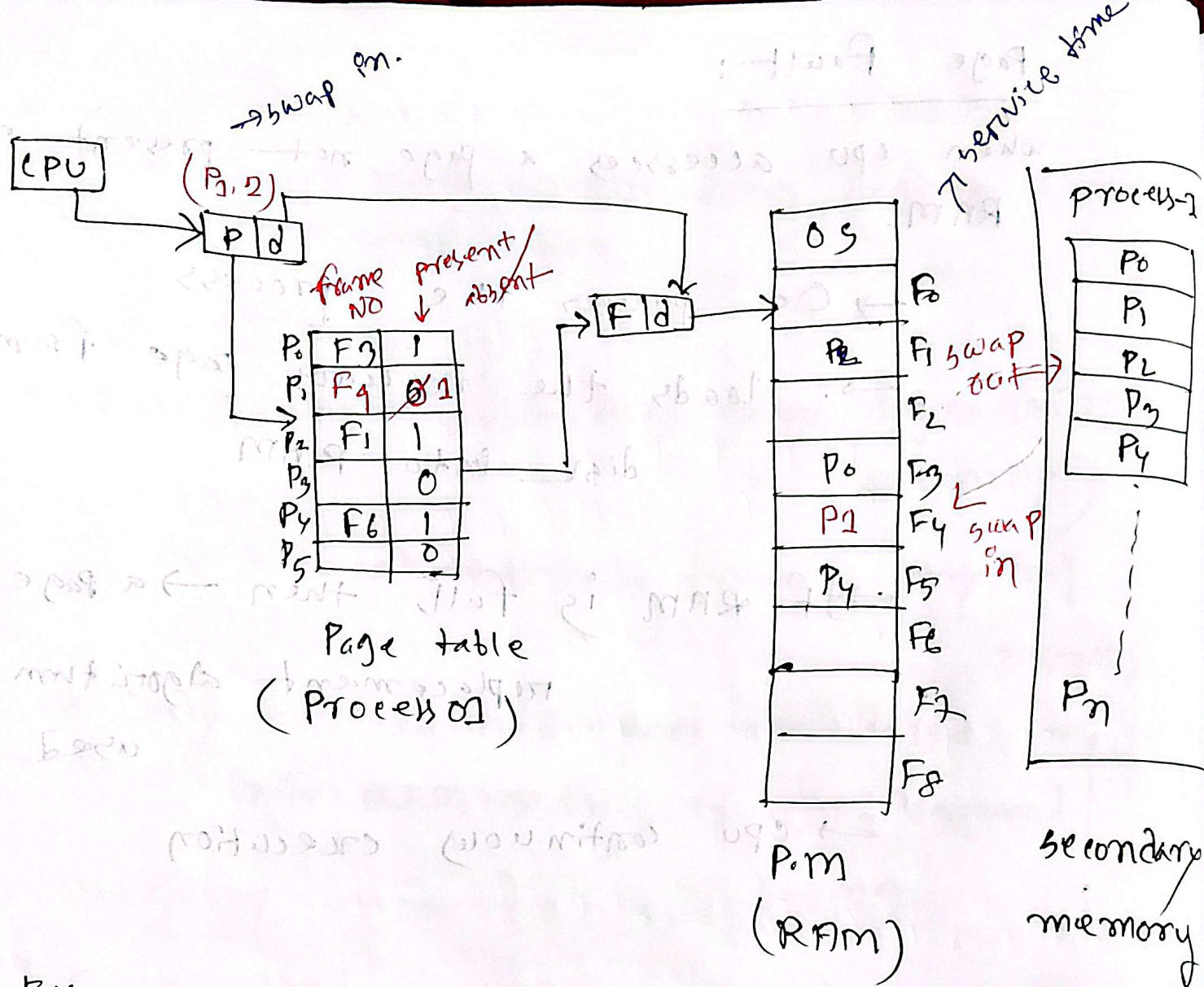
Page Fault:

when CPU accesses a page not present in RAM:

- OS pauses the process
- loads the required page from disk into RAM
- If RAM is full, then → a page replacement algorithm is used
- CPU continues execution

Algorithm:

- FIFO (First-In-First-Out)
- LRU (Least Recently Used)
- Optimal
- LFU (Least Frequently Used)
- MFU (Most Frequently Used)



Ex: memory access time = 20 ns
 page fault = 30% and page fault service time = 700 ns. Find ~~EMAT~~ EMAT (Effective Memory Access Time)

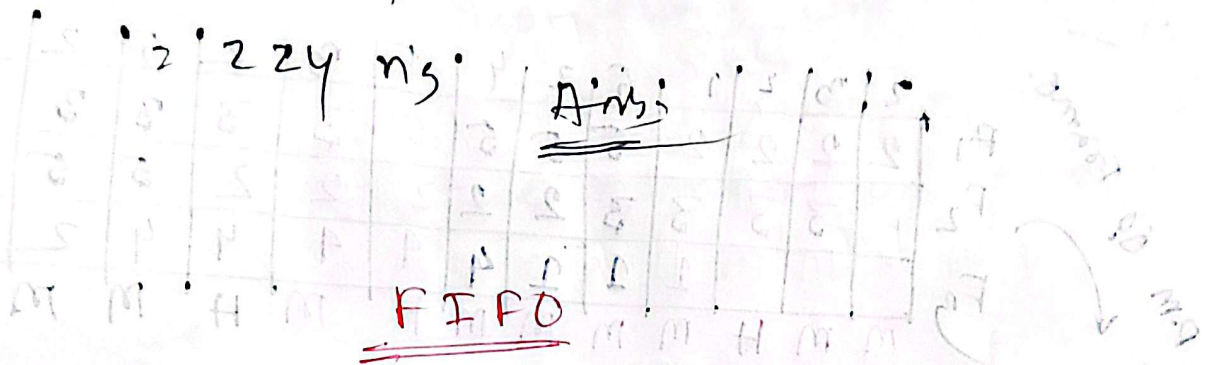
~~EMAT = EMAT~~

we know,

$$EMAT = (\text{page fault} \times \text{service time}) + (1 - \text{page fault}) \times \text{memory access time}$$

$$= (0.3 \times 700) + (1 - 0.3) \times 20$$

$$= 210 + 14$$



FIFO is the simplest page replacement algorithm. It replaces the page that has been in memory the longest (the oldest page).

Hit $\rightarrow$ Page in the memory (p.m.)

Miss $\rightarrow$ (1) Page is not in the primary memory & space is available.

(2) Page is not in the primary memory and space is not available.

string: 2, 3, 2, 1, 5, 2, 4, 5, 3, 2, 5, 2. Now find out Hit & miss ratio.

Solⁿ

P.M. Frame.

	2	3	2	1	5	2	4	5	3	2	5	2
F ₁	2	2	2	2	5	5	5	5	3	3	3	3
F ₂		3	3	3	3	2	2	2	2	2	5	5
F ₃				1	1	1	4	4	4	4	4	2
	m	m	H	m	m	m	m	H	m	H	m	m

F ₁
F ₂
F ₃

No. of Hits $\rightarrow 3$

No. of miss $\rightarrow 9$

No. of reference string = 12

$$\therefore \text{ratio of Hits} = \left(\frac{3}{12}\right) \times 100 = 25\%$$

$$\text{miss} = \left(\frac{9}{12}\right) \times 100 = 75\%$$

Belady's Anomaly

Belady's Anomaly is a phenomenon in page replacement algorithms, where increasing the number of frames (physical memory) actually increases the number of page faults.

Normally, ~~fewer~~^{we} would expect more frames $\rightarrow$ fewer page faults, but Belady's Anomaly shows this ~~not~~ is not always true.

For Frame = 3

	1	2	3	4	1	2	5	1	2	3	4	5
F ₁	1	1	1	4	4	4	5	5	5	5	5	5
F ₂		2	2	2	1	1	1	1	1	3	3	3
F ₃			3	3	3	2	2	2	2	2	4	4
	m	m	m	m	m	m	m	H	H	m	m	H

no. of hits = 2

" 4 miss = 9

no. of hit =

" 4 miss = 10

For Frame = 4

	1	2	3	4	1	2	5	1	2	3	4	5
F ₁	1	1	1	1	1	1	5	5	5	5	4	4
F ₂		2	2	2		2	2	2	1	1	1	5
F ₃			3	3	3	3	3	3	3	2	2	2
F ₄				4	4	4	4	4	4	4	3	3
	m	m	m	m	m	H	H	m	m	m	m	m

Optimal page replacement Algorithm

	2	3	2	1	5	2	4	5	3	2	5	2
F ₁	2	2	2	2	2	2	4	4	4	2	2	2
F ₂		3	3	3	3	3	3	3	3	3	3	3
F ₃				1	5	5	5	5	5	5	5	5
	M	M	H	M	M	H	M	H	H	M	H	H

H = 6
m = 6

Note: replace (page) not in cache
CPU reference string

Hit ratio = $(\frac{6}{12}) \times 100 = 50\%$
Miss ratio = $(\frac{6}{12}) \times 100 = 50\%$

→ This algorithm replace the page that will not be used for the longest period of time in the future.

→ It is called optimal because it results in the minimum number of page faults for a given reference string.

→ It is mostly theoretical because the OS cannot know the future reference string.

work:

① when a page faults occur and RAM is full

— look at all pages currently in memory

— determine which page will be used farthest in the future.

— replace that page.

② If a page is not used again, it is the

best candidate for replacement.

~~LRY~~

left of 25701
LRU

Ref. seq: 2 3 2 1 5 2 4 5 3 2 5 2

	2	3	2	1	5	2	4	5	3	2	5	2
P ₁	2	2	2	2	2	2	2	2	3	3	3	3
P ₂		3	3	3	5	5	5	5	5	5	5	5
P ₃				1	1	1	4	4	4	2	2	2
	m	m	m	m	m	H	m	H	m	m	H	H

note:
replace

miss (to replace)
refer

H = 5
m = 7

Hit ratio = $\left(\frac{5}{12} \times 100\right) = 41.6\%$

Miss = $\left(\frac{7}{12} \times 100\right) = 58.3\%$

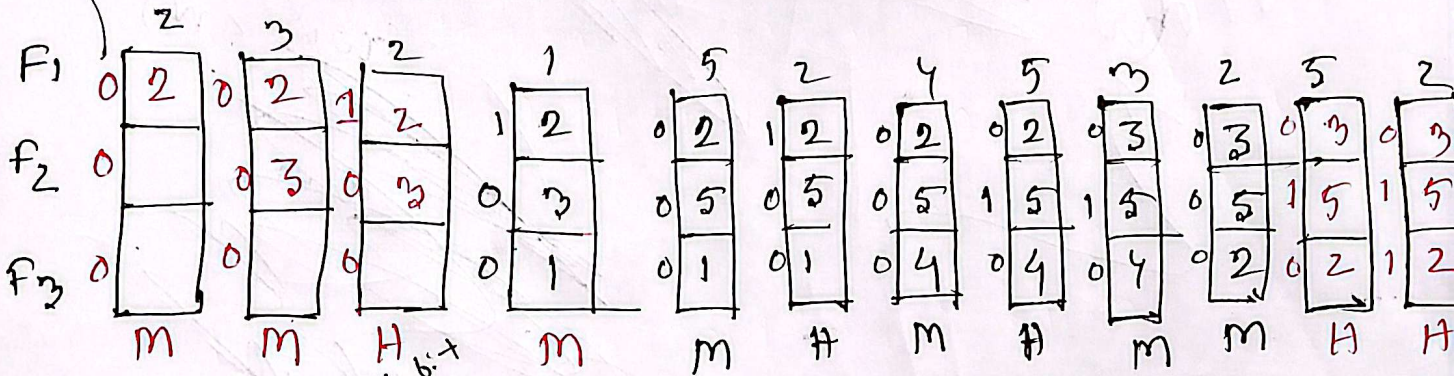
- LRU - replace the page in memory that has not been used for the longest period of time.
- LRU is a practical approximation of the optimal Algorithm, because it uses past history instead of future knowledge

work:

- ① Keep track of the order in which pages are accessed.
- ② when a page fault occurs and RAM is full:
→ Find the page that was least recently used

→ replace it with the new page.

Second chance / clock Page Replacement algorithm.



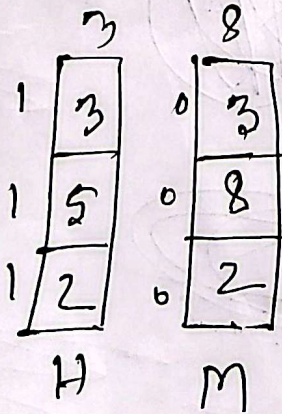
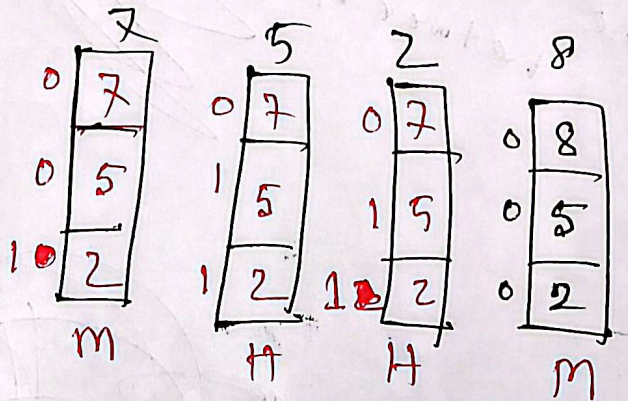
Handwritten notes:

2
7
1
2
3
4
5

Reference bit count replace NO

No. of H = 5

u u m z 7



Thrashing

cpu thrashing: cpu thrashing is a process situation where the cpu spend most of its time performing page swapping instead of executing useful instructions.

The OS repeatedly exchange data between RAM & the swap Area, causing:

→ Excessive page fault

→ High disk activity

→ very low actual cpu work.

causes:

① Excessive multiprogramming: If too many process run at the same time, the available page frames in RAM become insufficient → page faults increase → swapping increases.

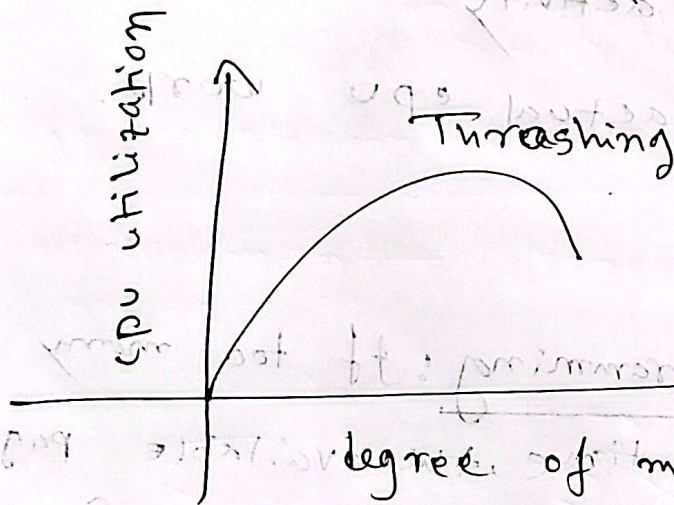
② Insuffing RAM or Page Frames: If the required working set of process cannot fit in memory, thrashing occur.

③ large working set

④ poor Page Replacement Algorithm.

Effect:

- cpu efficiency decreases
- Page fault increases suddenly
- Program take a long time to complete that's why system becomes slow, hang or freezer.
- Overall system throughput decreases.



How to detect

① High page fault: If pages fault per second rise

sharply → possible thrashing

→ low CPU utilization

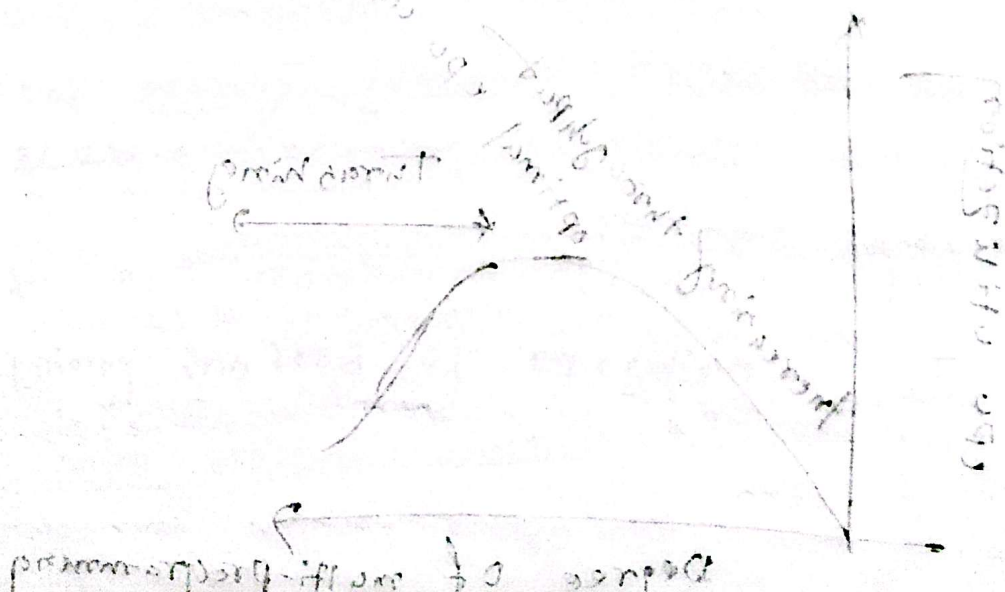
② low remainder waiting for pages to be swapped

→ high disk activity

③ Disk perform continuous read/write operation.

How to prevent/control

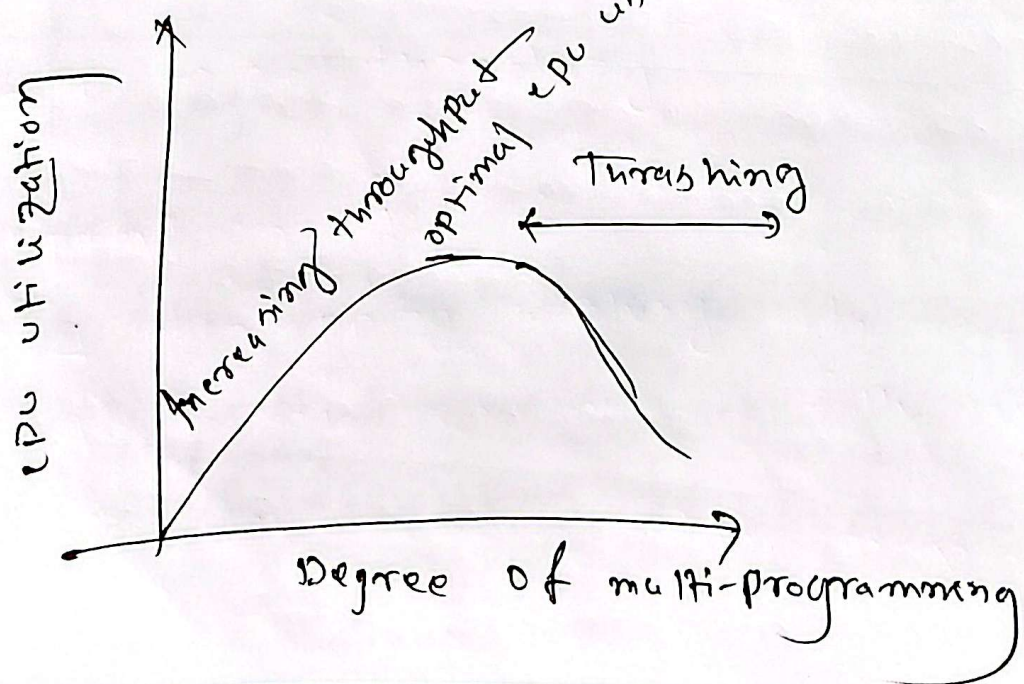
- ① use working set model
→ ensure each process has enough frames to hold its working set.
- ② limit multiprogramming
→ reduce the no. of processes to reduce memory pressure.
- ③ use better page replacement algorithm
→ like LRU, LFU or optimal.
- ④ Increase RAM.
→ adding more physical memory significantly reduces thrashing.
- ⑤ Priority based allocation.
→ Give more frames to high-priority processes.



Thrashing occurs when the OS spends more time swapping pages between main memory & disk than executing process. This leads to excessive page faults & a significant drop in CPU utilization.

The cycle works like this:

- ① High degree of multiprogramming: Too many processes are loaded into memory.
- ② Lack of frames: Each process gets fewer frames than needed.
- ③ Page replacement policy: Frequent replacement increases page faults.



This repeat cycle of low cpu utilization $\rightarrow$ more processes $\rightarrow$ more page faults is called thrashing.

locality model:

A locality is a set of pages that a program actively uses together.

For example, when a function is called, instruction local variables, & global variables define a locality.

$\rightarrow$ If allocated frames $>$ locality size, page fault stay slow.

$\rightarrow$ " " $\leq$ " " , frequent page fault occurs.

when multiple process run, their localities must fit in memory.

If total locality demand exceeds RAM, pages keep ~~swapping~~ getting swapped.

Result $\rightarrow$ Thrashing, because cpu spends time swapping instead of executing.

Handle Thrashing:

1. Working set model:-

Based on the locality model: a process uses a set of pages activity at a time.

→ If enough frames are allocated to cover the current locality few page faults.

→ If frames $<$ locality size, process will thrash

→ Working set (WSS_i) = pages ~~referre~~ referenced in the last Δ references (window size)

Total demand $D = \sum WSS_i$

cases

① if $D > m$ (m = available frames) → thrashing occurs

② If $D \leq m$ → No thrashing

Accuracy depend on Δ :

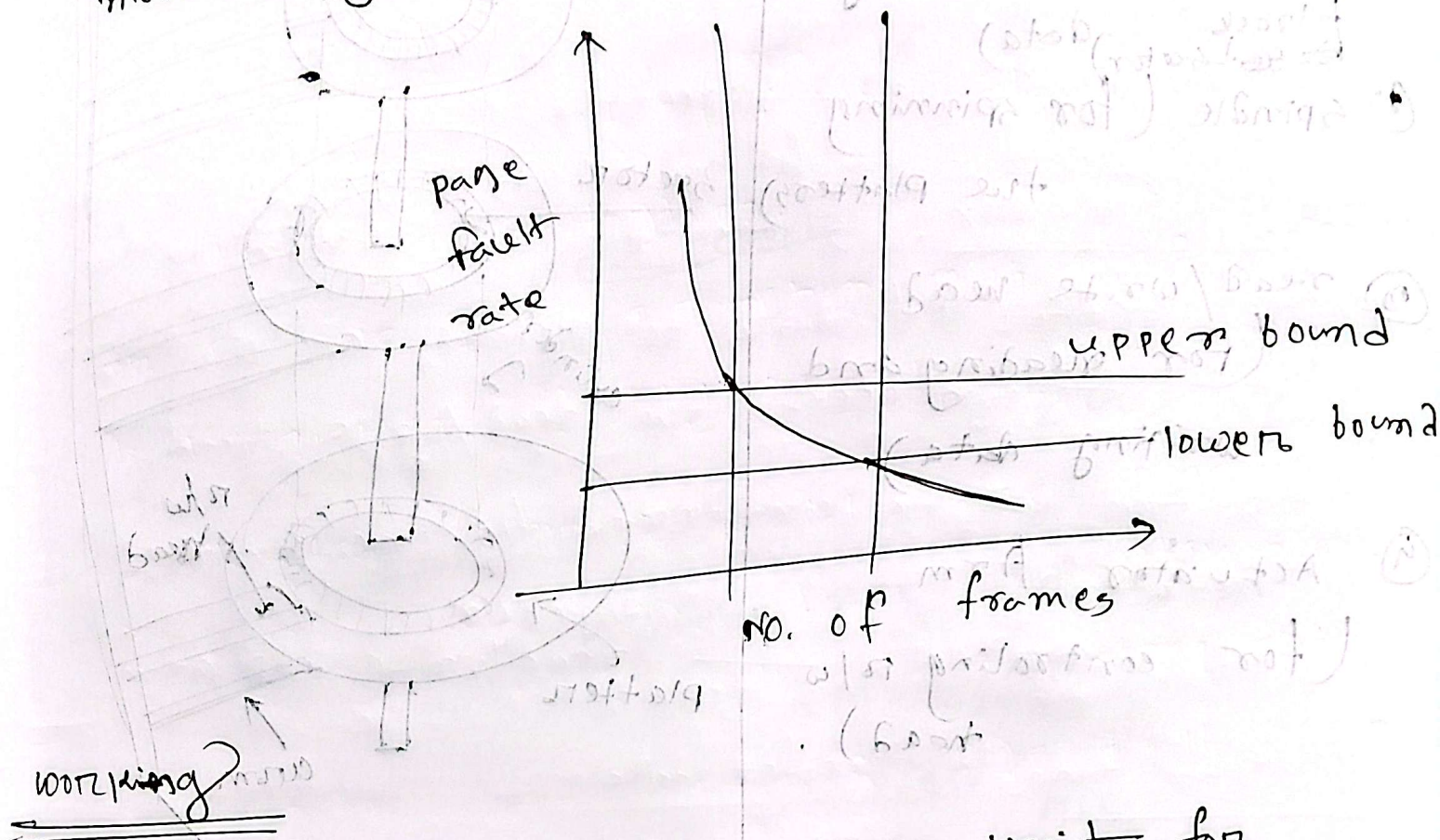
→ large Δ overlapping working sets

→ small Δ , locality may not be fully captured

→ If $\Delta = \infty$ → will encompass entire program.

2. Page Fault Frequency:

PFF is a technique to control thrashing by directly monitoring the page fault rate of processes.



→ Define an upper limit & a lower limit for acceptable page fault rate.

→ If fault rate $>$ upper limit, give more frames to the process.

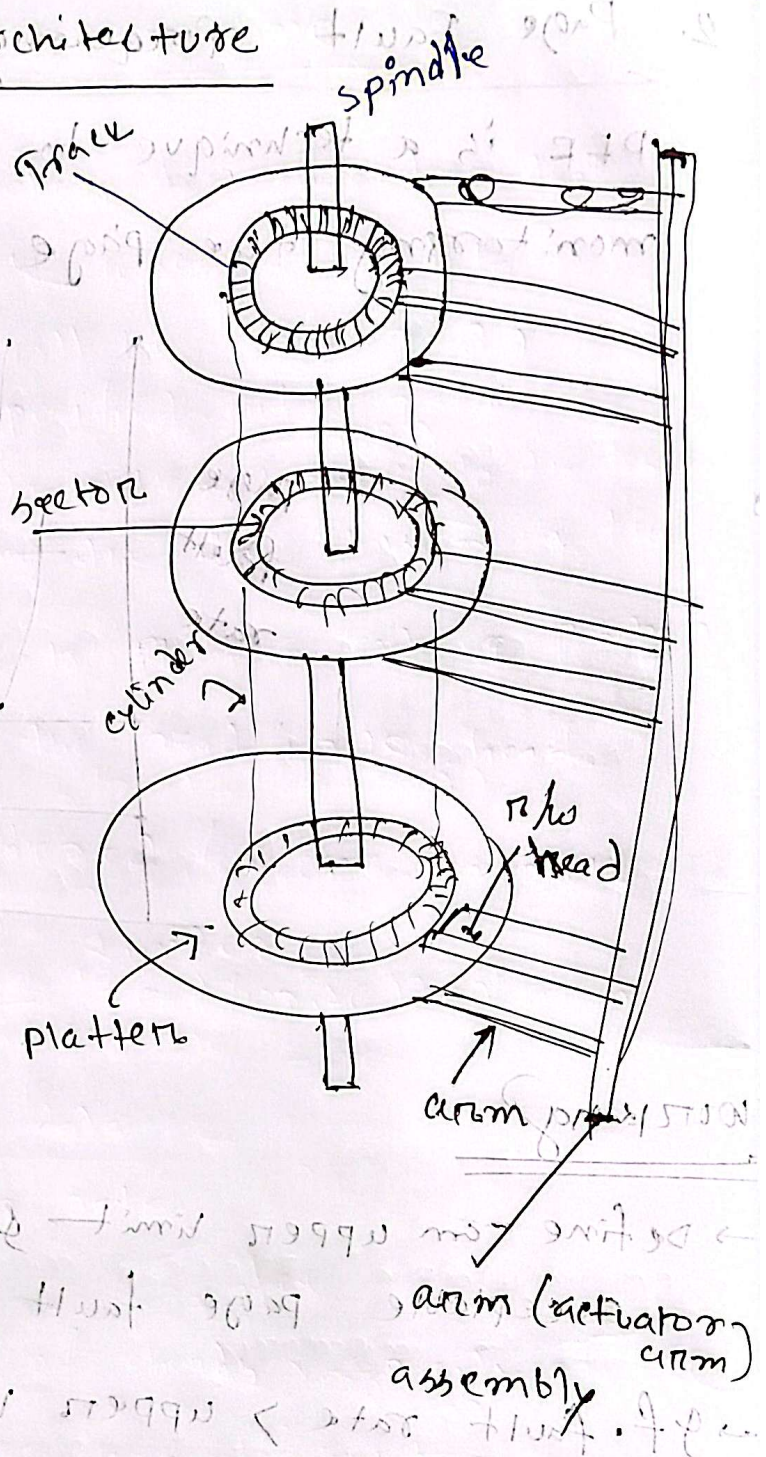
→ If fault rate $<$ lower limit, take away frames.

→ If no free frames are available suspend some processes & reallocate frames.

Disk Architecture

Component

- ① Platter (for storing track & sector data)
- ② Spindle (for spinning the platters)
- ③ read/write head (for reading and writing data)
- ④ Actuator Arm (for controlling r/w head).



* A hard disk has with 4 platters that has 2048 tracks for each platter & 1024 sectors for each track & 512 byte sector. calculate total capacity.

For one platters = 2 surface $\times$ tracks $\times$ sector $\times$ data-size

$$= 2 \times 2048 \times 1024 \times 512 \text{ byte}$$

$$= 2^1 \times 2^{11} \times 2^{10} \times 2^9$$

$$= 2^{31} \text{ byte} = 2 \text{ GB}$$

$$= \frac{2^{31}}{2^{30}} \text{ GB} = 2 \text{ GB}$$

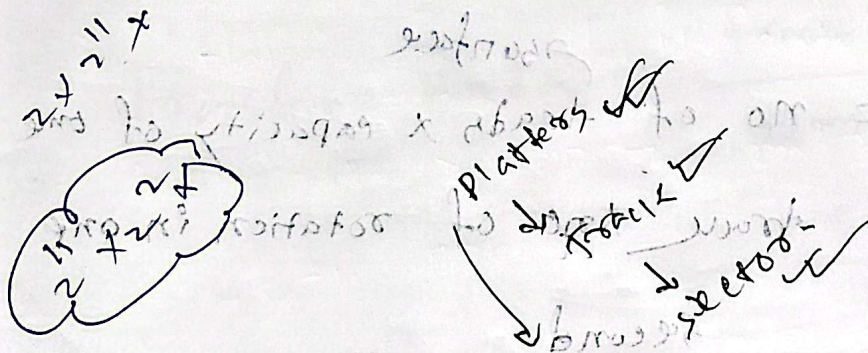
$$= 2 \text{ GB}$$

For platters need capacity

$$= 2(4 \times 2) \text{ GB} = 8 \text{ GB}$$

$$= 8 \text{ GB}$$

Ans:



For platters need capacity

time to move head to the target Disk access time
seek time: The time taken to locate the disk arm to a specified track where the data is to be read/write (avg time will be considered)

read/write head $\rightarrow$ specified data track

Rotational latency: is the time taken by the desired sector of disk to rotate into a position so that it can access the read/write head.

(avg time will be considered for one complete rotation) $\rightarrow$ read/write head arm

Transfer time: $\frac{\text{transfer data}}{\text{transfer rate}}$

transfer rate = no. of ^{surface} heads $\times$ capacity of one track $\times$ no. of rotation in one second.

disk access time = seek time + rotational latency + transfer time

W. there transferring of data $\rightarrow$ 512 byte, Avg seek time 4ms, Data transfer rate 40KB/s. Disk rotation 6000Rpm & controller overhead 0.7ms. Now find out disk access time.

50/7

here data rotation $\rightarrow$ 6000 Rpm

6000 rotate in 60 second

$$\Rightarrow 1 \text{ rev} = \frac{60}{6000} \text{ sec.}$$

or $1 \text{ rev} = 0.01 \text{ sec}$

(20121, 00, 5F1, PP, 410 ms, 88)

or $1 \text{ rev} = 0.01 \text{ sec}$

$$\therefore \text{rotational latency} = \frac{10}{2} = 5 \text{ ms} \checkmark$$

transfer time = $\frac{\text{transfer data}}{\text{transfer rate}}$

$$= \frac{512 \text{ B}}{40 \text{ KB/s}} = \frac{512 \text{ B}}{40 \times 2^{10} \text{ B/s}} = 12.5 \text{ ms}$$

$$\text{D.A.T} = 4 + 5 + 12.5 + 0.7 = 21.9 \text{ ms. Ans}$$

$\therefore$ Disk Access Time (D.A.T) = A. seek time + rotational latency + transfer time + controller overhead

Algorithm

① FCFS

② SSTF

③ SCAN

④ LOOK

⑤ C-SCAN

⑥ C-LOOK

① FCFS (First come First serve)

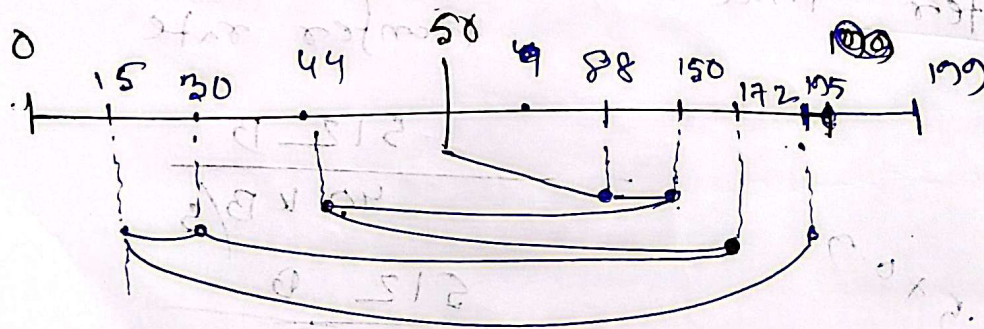
A disk contain 200 tracks (0-199)

Request queue contain track no

(88, 150, 44, 172, 30, 15, 105)

current position of read/write = 50

→ calculate total no. of track movement.



$$= (150 - 50) + (150 - 44) + (172 - 44) + (172 - 15) + (105 - 15)$$

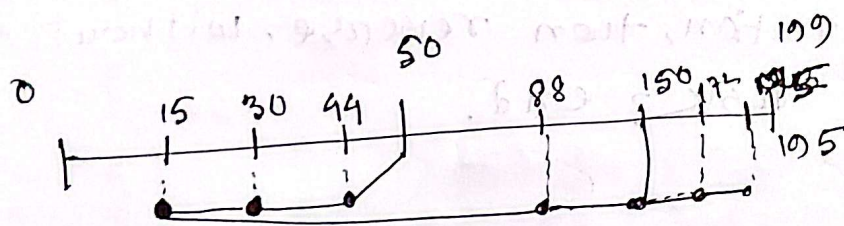
$$= 671$$

Ans:

$$= 671$$

Ans:

⑪ SSTF (Shortest seek Time First)
 schedules the disk request closest to the current head position first to minimize seek time.



$$\begin{array}{r} 50 \\ - 44 \\ \hline 6 \end{array}, \begin{array}{r} 44 \\ - 30 \\ \hline 14 \end{array}$$

$$= (50 - 15) + (195 - 15)$$

= 215

Ans:

88, 150, 44, 172, 30, 15, 195

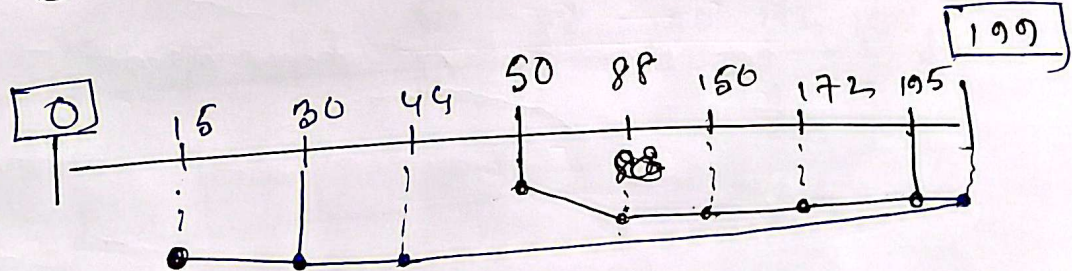
⑫

SCAN: head move the head in one direction serving servicing all requests until it reaches the end, then reverse direction, like an elevator.

#

88, 150, 44, 172, 30, 15, 195

Increasing order



$$(199 - 50) + ((199 - 15) = 337$$

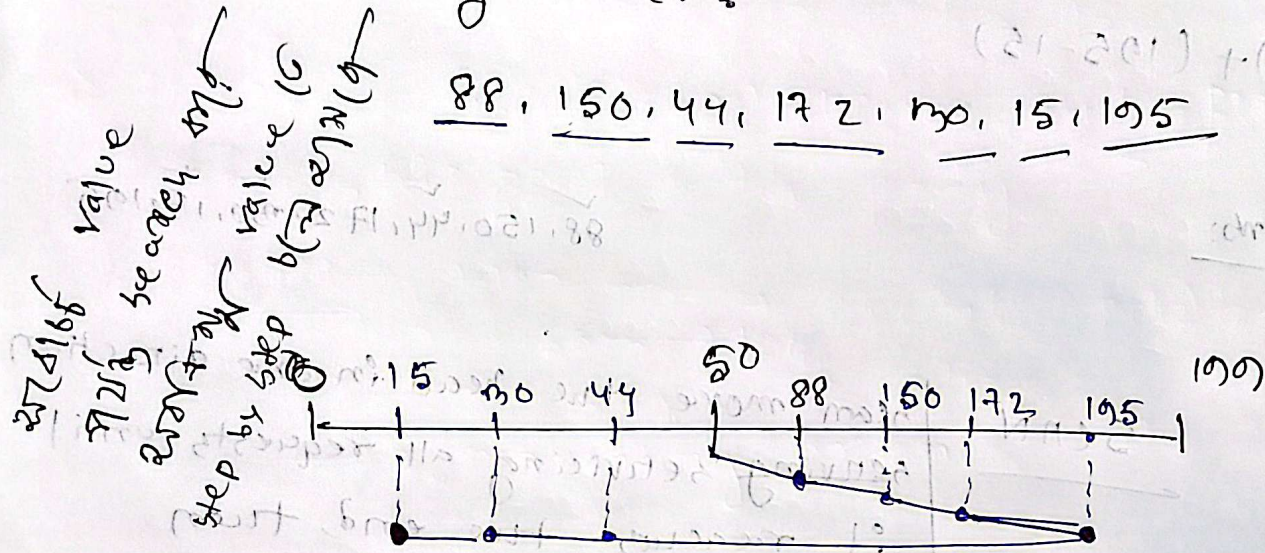
$$\text{total track movement} = (199 - 50) + (199 - 15)$$

$$= 337$$

(iv) look: moves the disk head in one direction servicing requests until the last request in that direction, then reverse, without going to the disk's end.

→ Increasing order:

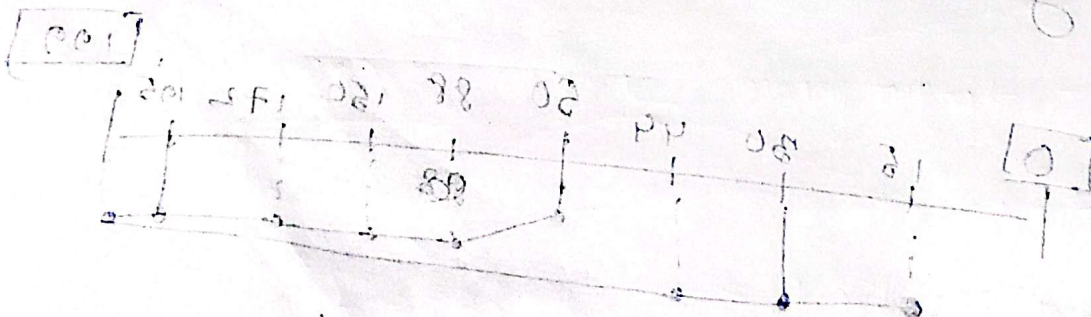
88, 150, 44, 172, 130, 15, 195



total track movement = $(195 - 50) + (195 - 15)$

= ~~220~~ 325

Ans



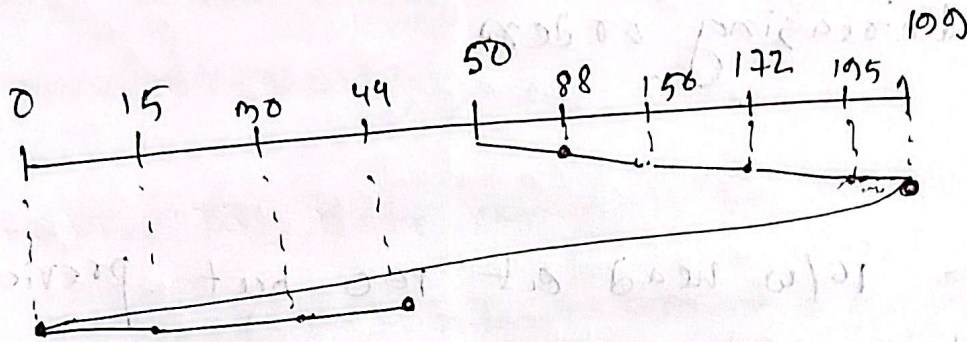
total = $(8 - 001) + (002 - 001)$

$(002 - 001) + (003 - 002) = \dots$

② C-SCAN (Increasing order)

(88, 150, 44, 172, 30, 15, 105)

last of file
with 0 (or) first
with

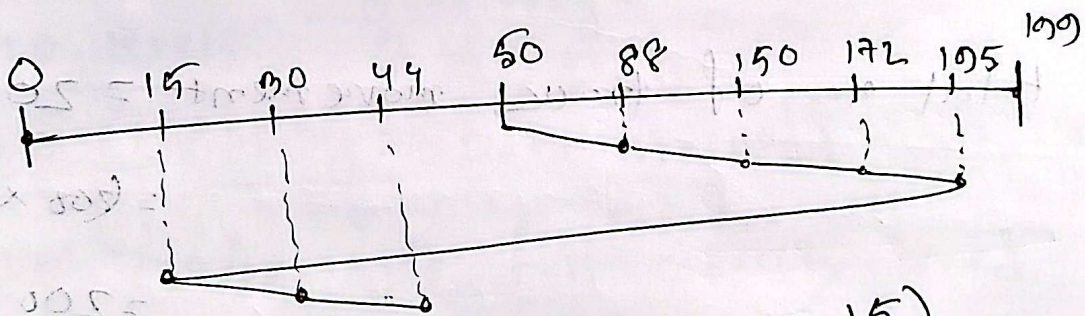


$$\text{total track movement} = (100 - 50) + (100 - 0) + (44 - 0)$$

= 194

③ C-LOOK (Increasing order)

(88, 150, 44, 172, 30, 15, 105)



$$\text{total track} = (105 - 15) + (44 - 15)$$

$$= 134$$

= 134

Ans:

order

⇒ current R/W head is at 50 but previous request was at 75

$\swarrow$ decreasing order
 50 ← 75 order

→ decreasing order.

⇒ current R/W head at 100 but previous request 50.

$(0-001) + (002-001) =$ 50 → 100 total
 (0-001) +

→ increasing order.

→ If R/W head takes 2ms to move from one track to another then calculate the total time taken.

⇒ total no. of track movement = 200
 $= (200 \times 1) \text{ ms}$
 $= 200 \text{ ms}$

$(21-001) + (002-001) =$ total
 $100 =$
 100

Ans:

RAID

RAID is a storage technology that combines multiple physical disks into a single logical unit to improve performance, reliability or both.

If one drive fails, data can still be recovered from others RAID (Redundant array of Independent

or Inexpensive disks)

Key Goals of RAID:

- ① Data redundancy - protects against disk failure
- ② performance Improvement: Faster read/writes by splitting data across disks
- ③ Fault Tolerance
- ④ Increase storage capacity.

Main RAID levels

→ three main techniques

- ① striping
- ② mirroring
- ③ parity

① RAID-0 - striping: Data striped across multiple disks

→ High performance, full capacity used

→ No redundancy, failure of one disk → data lost

RAID 1 — mirror

→ Data mirrored ^{copying same data to multiple disk}

→ High reliability, easy to recover ^{realign more disk space}

→ 50% storage efficiency

→ 100% capacity overhead.

RAID 5: Block level striping with parity

→ Data striped with parity ^{storing error correcting}

→ Fault tolerance & decent performance

→ can survive one disk failure

→ minimum 3 disks, slower write due to parity.

RAID 6: striping with double parity

→ like RAID 5, but with double parity

→ can survive 2 disk failure.

→ slower write; minimum 4 disks.

RAID 10 (H0)

→ combination of mirroring & striping

→ High speed & reliability

→ Expensive, needs at least 4 disks.

use: server, database system, cloud storage, data center

⑧

11b

①

Given,

rotational speed = 5200 rpm

$$= \frac{5200}{60}$$

Now,

Q. 5200 rotation in 60 sec

$$1 \text{ " } \frac{60}{5200} \text{ sec.}$$

$$(1 - \frac{1}{2}) \times 8 = \text{avg} \text{ wait } = 11.538 \text{ ms.}$$

$$\therefore \text{avg rotational delay} = \frac{11.538}{2} \text{ ms} = 5.769 \text{ ms}$$

Ans 2

⑪

Time to read 4MB file.

Q Determine the no. of sectors in the file

$$\text{File size} = 4 \text{ MB} = 2^2 \times 2^{20} \text{ bytes} = 2^{22} \text{ bytes}$$

Each sector hold 512 bytes

$$= \text{No. of sectors} = \frac{2^{22}}{512}$$

$$= 8192 \text{ sectors.}$$

$$\text{No. of tracks} = \frac{8192}{128}$$

$$= 64 \text{ tracks}$$

Each track
has 128 sectors

Calculate total time

seek time: Moving between ~~track~~ adjacent tracks takes 3ms per track.

$$\therefore \text{Total seek time} = 3 \times (64 - 1) \\ = 189 \text{ ms}$$

Rotational delay

average rotational delay per track is ^{5.760}~~6050~~ ms

$$\therefore \text{total rotational delay} = (5.760 \times 64) \text{ ms} \\ = 368.64 \text{ ms}$$

Data transfer time:

$$\text{Time to read one sector} = \frac{11.538}{128} \text{ ms}$$

$$\therefore \text{total time to read} = (8192 \times \frac{11.538}{128}) \text{ ms} \\ = 738.432 \text{ ms}$$

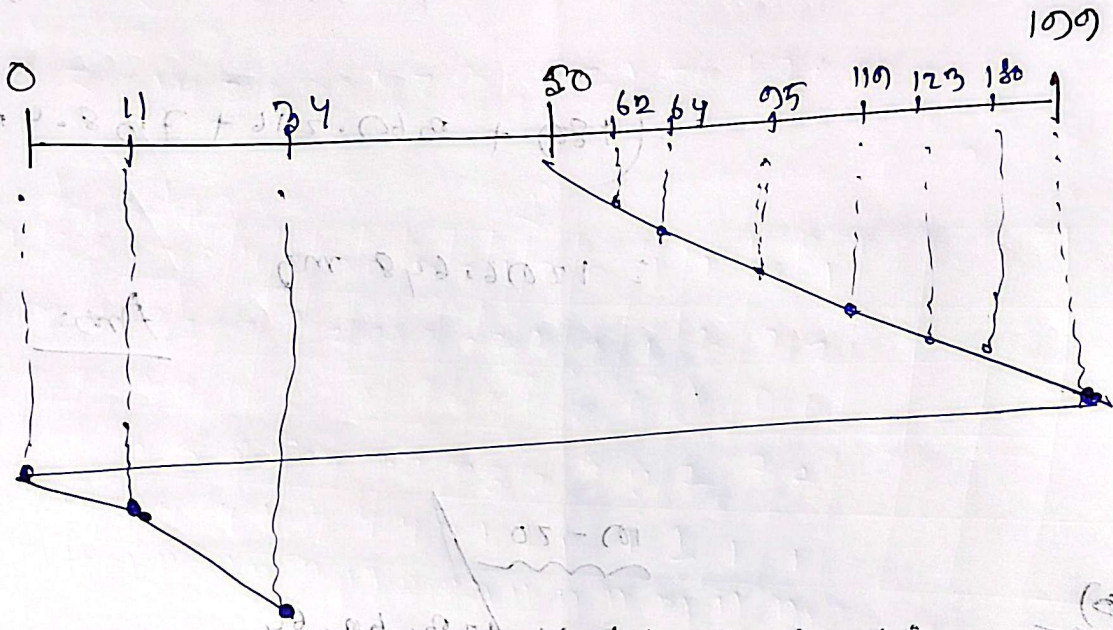
8 (b)

10-20

(i)

C-SCAN

05, 180, 34, 110, 11, 123, 62, 64



total movement $\rightarrow$ track

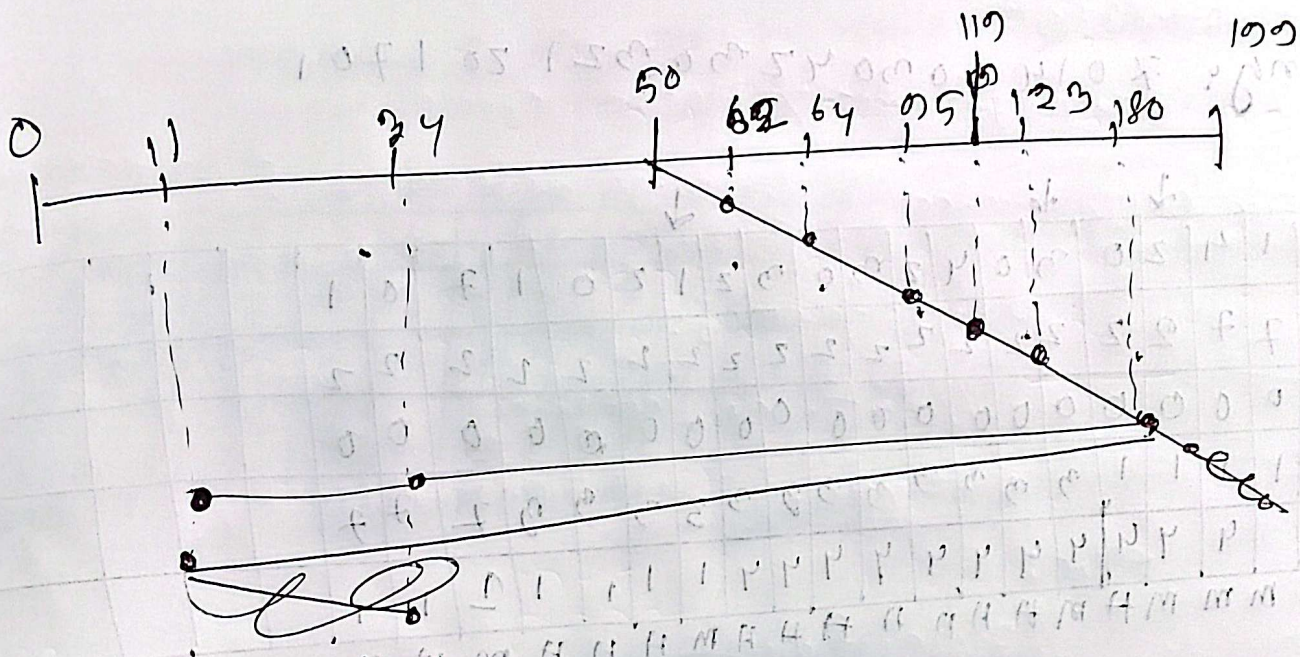
$$= (180 - 50) + (180 - 0) + (34 - 0)$$

$$= 382$$

Ans

⑪ LOOK

75, 180, 34, 110, 11, 123, 42, 64



total movement = $(180 - 50) + (180 - 11)$

8. 11 (ethanol) 2000 (1000)

- 322 fms:

$\rho = 2 \text{ mass} \cdot \overline{7.5}$

1 13 18 10
~~6 9 10~~ 11 15 17 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

① WBA (Woot + 1000) 6000

It is found
in the forest

21-22

8(a)

Optimal page replacement algorithm

String: 7 0 1 4 2 0 3 0 4 2 3 0 3 2 1 2 0

String: 7 0 1 4 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

	7	0	1	4	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1				
F ₁	7	7	7	7	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2				
F ₂		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
F ₃			1	1	1	1	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3				
F ₄				4	4	4	4	4	4	4	4	4	4	4	1	1	1	1	1	1	1				
	m	m	m	m	m	H	m	H	H	H	H	H	H	H	M	H	H	H	M	H	H				

(11-01) (00-00) total Hit = 13

total miss (Page Faults) = 8

20-21

7(a)

if frame = 3

String: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

① MRU (most recently used)

algorithm:

on reference P:

if P in frames

mark P as most-recently used

else

if frames has empty slot

load P into empty slot

अनुसंधान
अनुसंधान

else

find page x that is most-recently-used

replace x with p

mark p as most-recently used.

	7	0	1	2	0	3	0	4	2	3	0	3	0	3	2	1
F_1	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
F_2		0	0	0	0	0	0	4	4	4	4	4	4	4	4	4
F_3			1	2	2	2	2	2	2	3	0	3	0	3	2	1
	m	m	m	m	H	M	M	M	H	m	m	m	m	m	m	m

Page fault = 14

(ii) Optimal page replacement

	7	0	1	2	0	3	0	4	2	3	0	3	0	3	2	1
F_1	7	7	7	2	2	3	3	3	3	3	3	3	3	3	3	1
F_2		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
F_3			1	1	1	1	1	4	2	2	2	2	2	2	2	2
	m	m	m	m	H	M	M	M	M	H	H	H	H	H	H	m

Page fault = 8

Algorithm: For each page p at position i in reference string

④ second change (clock) page replacement.

Algorithm

Initialize:

frames = empty slot:

For each index i & page reference P in reference string

if P in frames:

// hit, do nothing

continue.

else:

// page fault

if frame has empty slot:

add P to frames

else:

For each page x in frames:

find the next occurrence index of x after i

if x does not appear again:

choose x for replacement

break

choose the page whose next use is farthest

replace that chosen page with P .

(11)

algorithms

second chance / clock

initialize pointer = 0

for each page p in reference-string:if p in frames:set reference-bit(p) = 1

else:

loop reference-bit(pointer) = 0

replace frame at pointer with p

set reference-bit(pointer) = 1

pointer = (pointer + 1) mod frame-size

break

else:

set reference-bit(pointer) = 0

pointer = (pointer + 1) mod frame-size

	7	0	1	2	0	3	0	4	2	3	0	3	0	3	2	11		
F_1	0	7	0	7	0	2	0	2	0	4	0	4	0	0	1	0	0	
F_2	0	0	0	0	1	0	0	1	0	0	0	2	0	2	0	2	2	
F_3	0	0	0	1	0	1	0	3	0	3	0	3	1	3	1	3	2	3
	<u>m</u>	<u>m</u>	<u>m</u>	<u>m</u>	H	<u>M</u>	H	<u>m</u>	<u>M</u>	H	<u>M</u>	H	H	H	H	H	H	H

total fault = 9

F_1	0	0
F_2	0	2
F_3	1	3

→ virtual memory is a memory management technique in a computer system that allows

How virtual memory empower computing

① Runs larger programs than RAM size

↳ Ex: If RAM = 4 GB, program requiring 8 GB can still run using virtual memory.

② provides separate address spaces programs.

→ Ensure memory protection — one program cannot overwrite another's memory.

→ Increase system security.

③ Enables efficient multitasking

→ multiple program simultaneously even if all cannot fit entirely in RAM

→ The OS swaps pages in & out as needed

④ reduce memory fragmentation

→ logical addresses are separate from physical addresses → program don't need continuous physical memory.

⑤ support Demand paging

only the required pages are loaded into RAM
→ Improve performance

2(c) system calls are the interface between a user program and the OS.

purpose

① Allow user programs to request OS services
(like I/O, protocol control, memory allocation)

② provide control access to hardware — Program
don't directly access devices.

③ Ensure protection & security — OS
mediates all sensitive operations.

Ex: → read(), write() → file I/O

→ fork() → create new process

→ malloc() → memory allocation (internally
may use system calls)

✓ System program — ~~purpose~~ are software tools provided by the OS to perform common system-related tasks.

Purpose:

→ provides a user-friendly interface to OS services.

→ Assist in program development & execution

→ Facilitate file management, device management & system monitoring.

Ex:

File manipulation : CP, mv, rm

Status & monitoring : ps, top

tools : compilers, assemblers, debuggers

Handling a Page table fault:-

Demand Paging: Demand paging is a virtual memory management technique in which pages of a process are loaded into main memory only when they are needed during execution.

How it work

1. Program start execution
→ only essential pages are loaded
2. When the CPU references a page not in memory, a page fault occurs
3. The OS then
→ checks if the page reference is valid
→ finds a free frame in physical memory
→ read the required page from disk into RAM
→ update the page table.
4. CPU retrieves the instruction after the page is loaded

Page fault Handling

7(1) 21/22

1. CPU detect a page fault.
2. OS check validity of the address
(If invalid $\rightarrow$ terminate process)
3. If valid, find a free frame.
4. swap page into frame via scheduled disk operation.
5. + Reset tables to indicate page now in memory set validation bit = 1
6. Restart the instruction that caused the page fault.

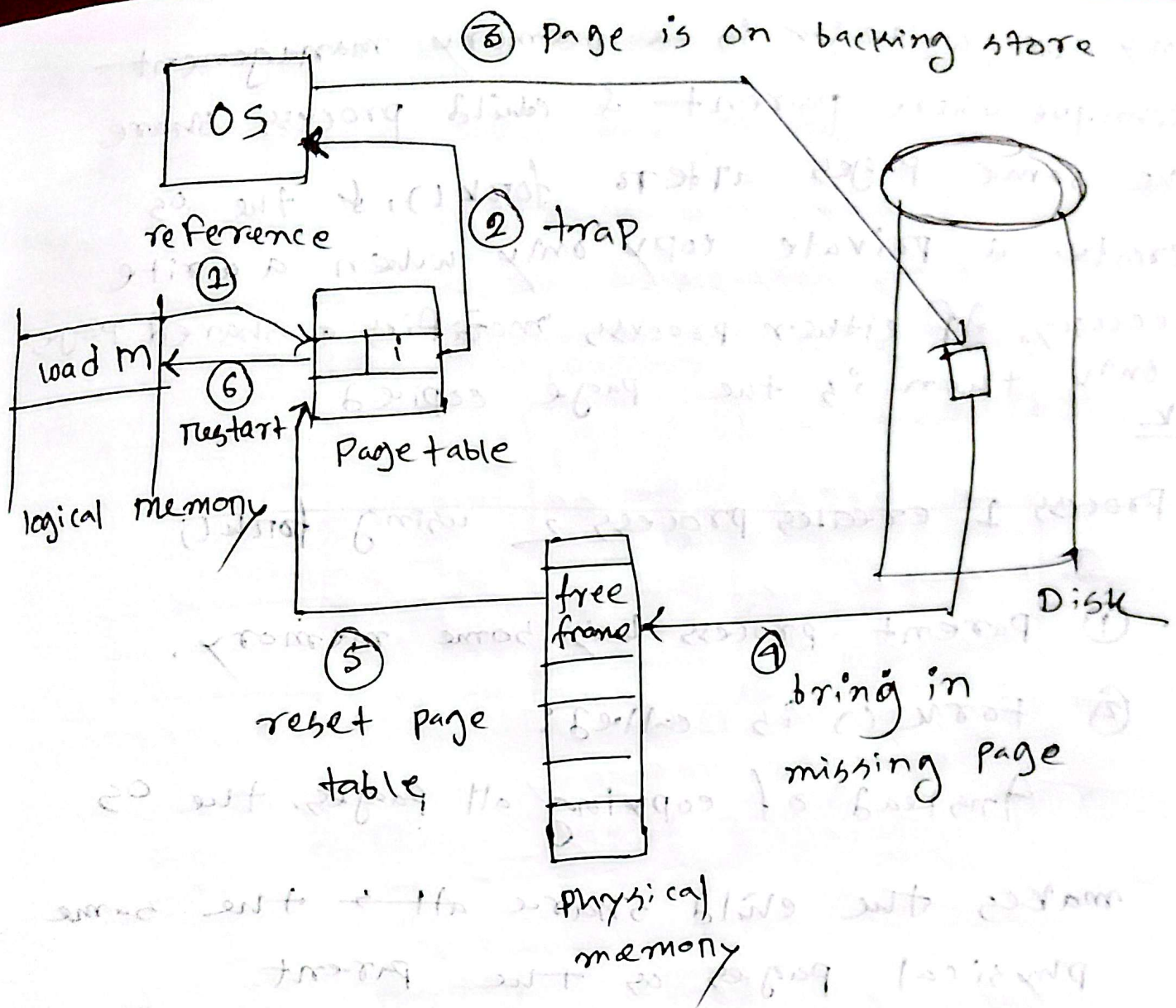


Fig: Page fault Handling.

copy-on-write is a memory management technique where parent & child process share the same pages after `fork()`, & the OS creates a private copy only when a write occurs. If either process modifies a shared page, only then is the page copied.

~~① Process 1 creates process 2 using `fork()`~~

① Parent process has some memory.

② `fork()` is called.

Instead of copying all pages, the OS makes the child share the same physical pages as the parent.

③ shared pages are marked 'read-only' — both processes can read, but cannot write.

④ If either process tries to write
→ A page fault occurs (write fault)
→ OS detect it is a cow page

⑤ OS creates a private copy of that page.
→ Only writing process gets the new page.

⑥ page table are updated

→ the process now has its own writable copy

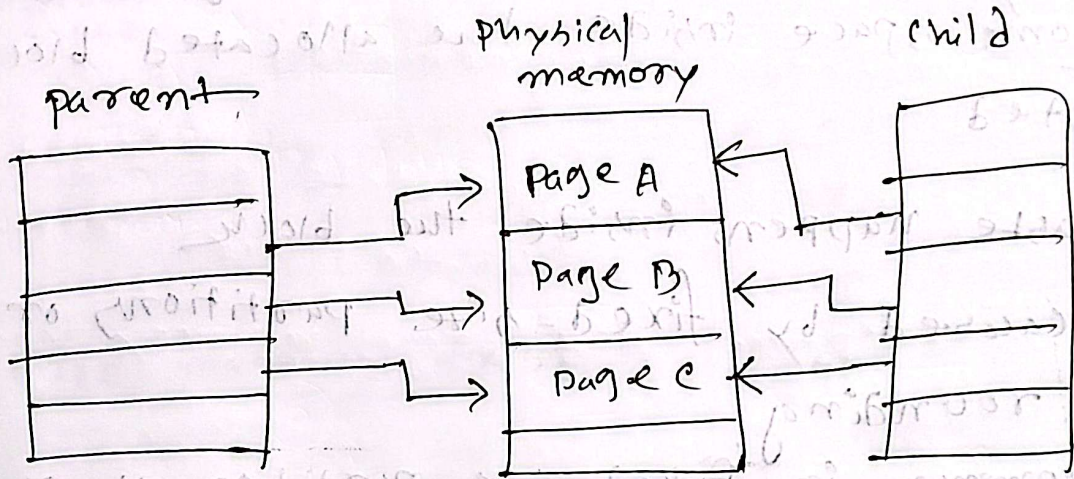


Fig: Before parent modifies page C.

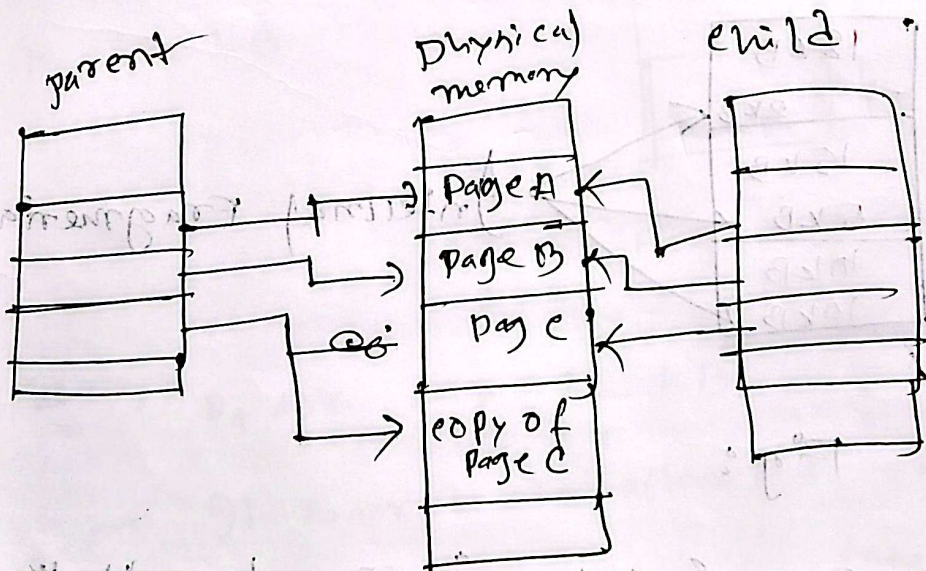


Fig: After parent modifies page C.

6(c)

Internal Fragmentation occurs when allocated memory is larger than the requested memory. So some space inside the allocated block is wasted.

- waste happens inside the block
- caused by fixed-size partitions or block rounding
- common in fixed-size partition allocation

Ex: requested memory = 18 KB, 15 KB, 10 KB
 Allocated block = 20 KB

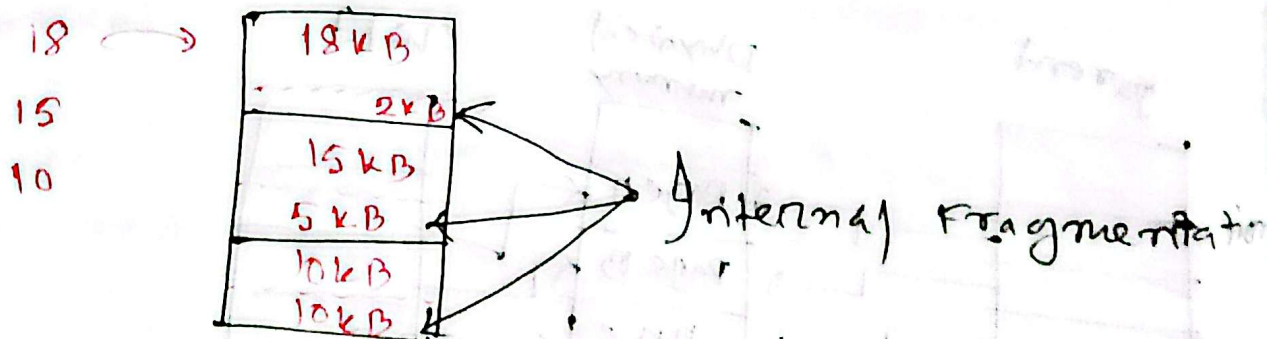


Fig:

Here, 2 KB, 5 KB & 10 KB inside the blocks is wasted → Internal Fragmentation.

External Fragmentation occurs when free memory is available, but it's divided into small non-contiguous holes, so a large block cannot be allocated. unused space that is left between the fragments of non-contiguous memory.

→ waste happens outside allocated blocks.

→ caused by variable size allocations.

→ common in variable size partition allocation.

Ex: Free memory exist (total 40 KB), but scattered

$P_1 = 5 \text{ KB}$

$P_2 = 8 \text{ KB}$

$P_3 = 10 \text{ KB}$

$P_4 = 12 \text{ KB}$

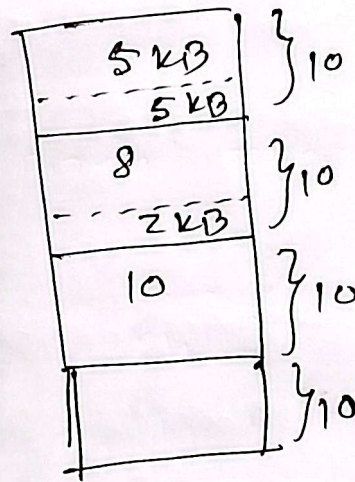


Fig: External

here, P_4 ask for 12 KB,

→ It cannot be allocated, even though

$$\text{total free} = 5 + 2 + 10$$

$$= 17 \text{ KB}$$

→ Because memory is not contiguous.

① First-fit

Pros: ① First allocation

② simple to implement

cons: ① moderate fragmentation

② small holes accumulate at the beginning
(wasted space)

③ No optimal for large memory request

② Best Fit



10KB
8KB
10KB
5KB
0KB

Free space

Free space: 10KB, 8KB, 10KB, 5KB, 0KB

At request of 15KB, no allocation is possible as no free space is available.

Therefore, Best Fit is not efficient.

First - Fit	Best - Fit	Worst - Fit
→ pick the first free block that is large enough → Fastest speed → stop as soon as a fit is found → moderate fragmentation → small hole → memory utilization: average → simple implementation → Time complexity $O(n)$	→ pick the smallest block that fits. → slower → search all free blocks → highest external fragmentation → many tiny holes → poor → complex → $O(n)$ worst-case every time	→ pick the largest block available → slower than first but faster than best → search all free blocks → lower fragmentation → medium size holes → Better than Best - Fit → simple - moderate → $O(n)$

Address binding is the process of mapping of a ~~log~~ logical (virtual) address to a physical & memory address.

Program use logical address, but memory hardware uses physical addresses - so the OS must bind (map) one to the another.

logical address $\rightarrow$ physical address mapping

$\therefore$ Address binding

Why need

① A program is written using logical addresses.

② Physical RAM uses physical addresses.

③ CPU must access correct memory location

④ OS needs binding for ~~relat~~ relocation,

protection, & memory management.

Address binding ensures: -

- $\rightarrow$ correct memory access $\rightarrow$ efficient memory usage
- $\rightarrow$ relocation of program
- $\rightarrow$ protection btw processes

Types

① compile time:

- Binding occurs ~~compile time~~ during compilation.
- compiler converts logical address to physical addresses directly.

characteristics

- physical address fixed before execution
- No relocation possible.
- very simple, but not flexible.

② Ex: A compiler decides that the program starts physical address 1000.

If the instruction ~~has~~ use address 50, The compiler converts it to $1000 + 50 = 1050$ during compilation. (decided at compile time)

② load - Time

Binding occurs when the program is loaded into memory.

loader maps logical $\rightarrow$ physical address at load time.

character
 $\rightarrow$ program can be loaded in any free memory block.

$\rightarrow$ After loading, relocation is not possible.

$\rightarrow$ more flexible than compile-time.

001 : 228652 10101, 0000 0000 0000

0012 - 001 + 0008 = 00005 1010101

when loading, The OS places the

Ex: Program starts at physical address 4000.

Instruction uses logical address 40. so loader maps it to $4000 + 40 = 4040$
(decided at load time)

③ Execution-Time (run time)

- Binding occurs while the program is running
- The program can be moved from one memory location to another during execution.
- Hardware requires MMU (Memory Management Unit)

Characteristics

- most flexible binding
- used in paging, segmentation, virtual memory
- CPU generates logical address → MMU converts it to physical address dynamically

Ex: Initially,

MMU base = 8000, logical address = 100

$$\text{Physical address} = 8000 + 100 = 8100$$

Later OS moves the program to a new location

base = 12000 & same logical = 100 becomes

$$12000 + 100 = 12100$$

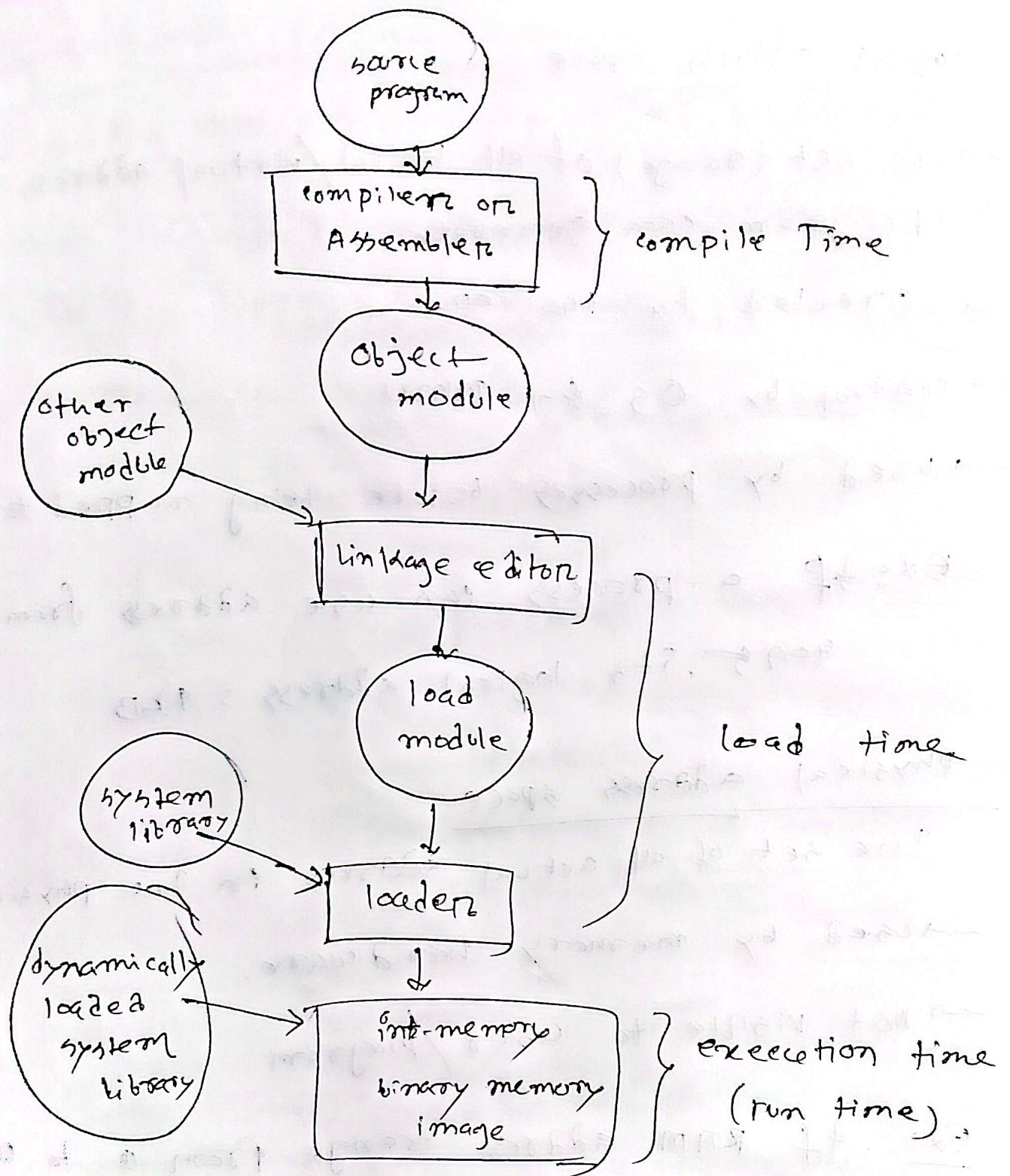


Fig. Address binding

Logical address space

- The set (range) of all logical/virtual address that program can generate.
 - created by the CPU
 - control by OS & MMU
 - used by processes before being mapped to RAM
- Ex: If a process can use address from 0 to 4095, its logical address = 1 KB

Physical address space

- The set of all actual address in the physical memory
- used by memory hardware.
 - Not visible to user/program
- Ex: If RAM address range from 0 to 65535 then physical address space = 64 KB.

memory management

Segmentation is a technique where a program is divided into logical units called segments.

Each segment represents a meaningful part of the program. Such as

A segment is a logical unit such as

main program

procedure

function

method

object

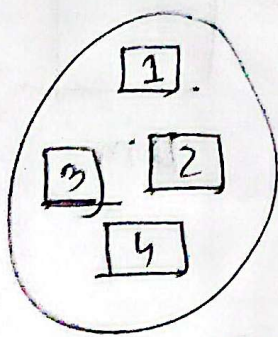
local variables, global variables

common block

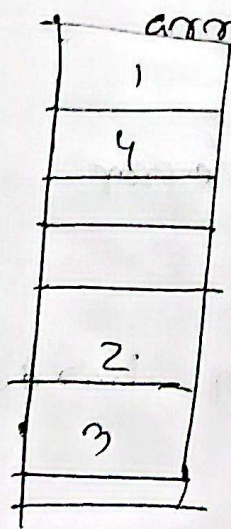
stack

symbol table

arrays



user space



~~array~~

p.m. space

① logical address = (segment no., offset)

(2, 100) → Access offset 100 inside segment 2

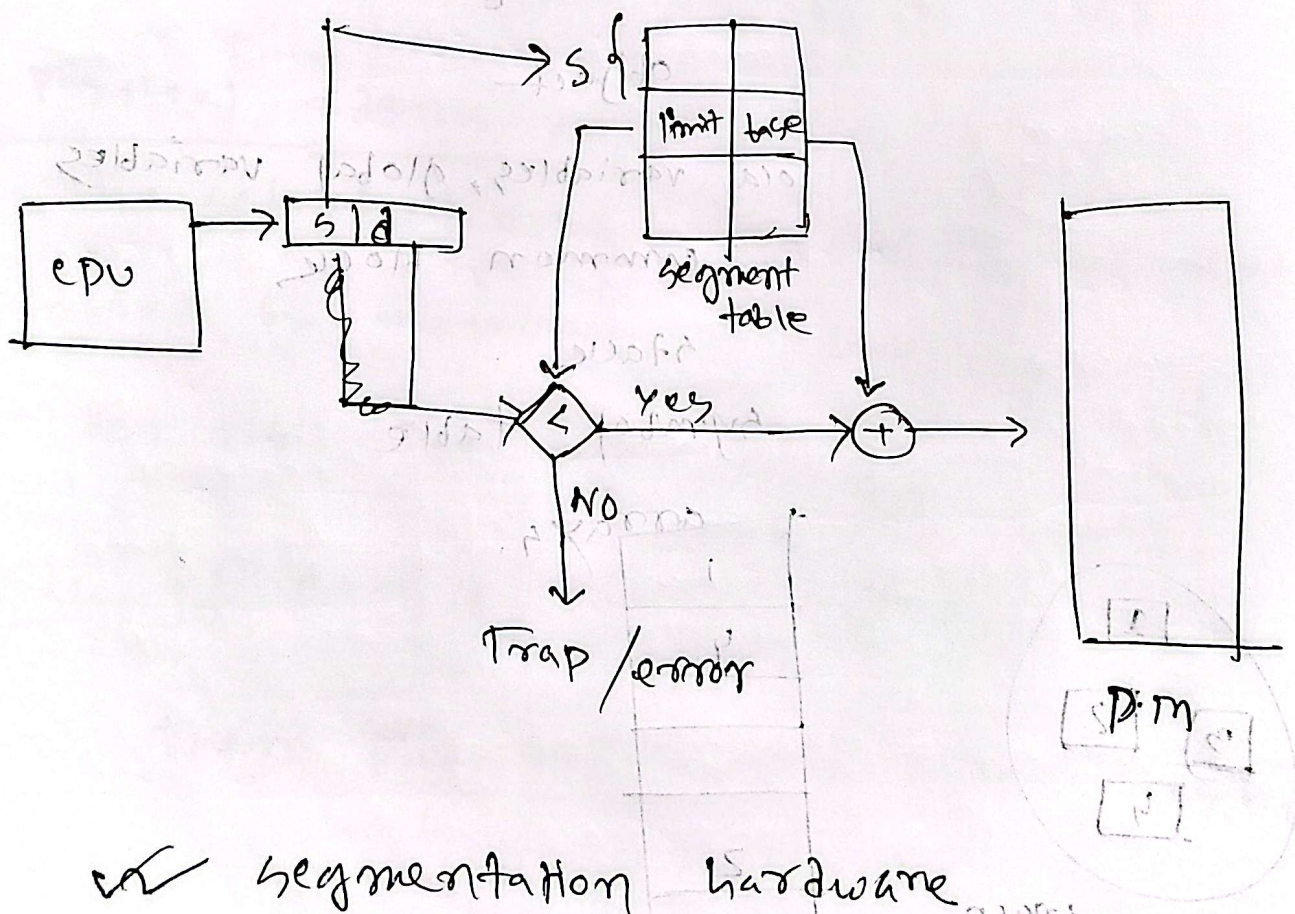
→ segment table (base, limit)

② Physical address = Base + offset

segment 2 → Base = 10000

Offset = 10000

∴ P.A. = 10000 + 10000



✓ Segmentation hardware

- Darggin

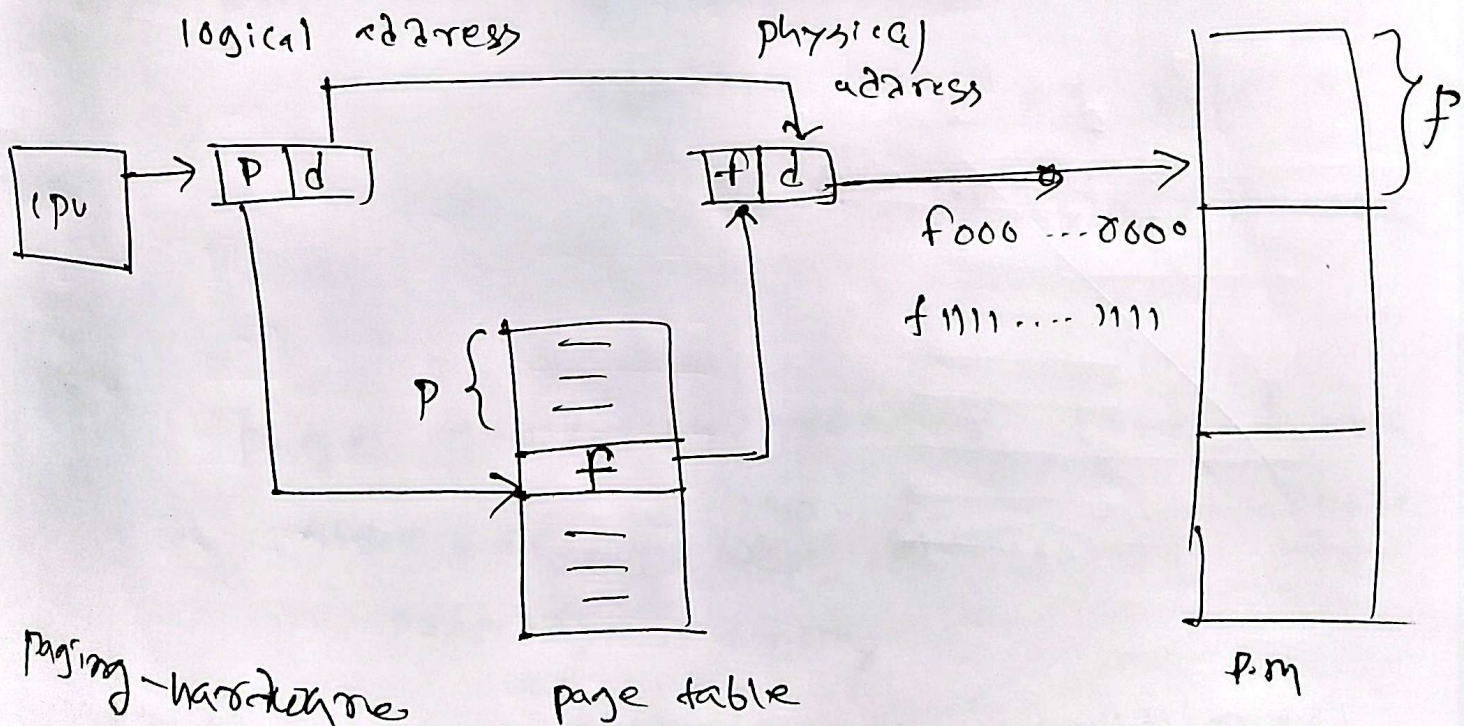
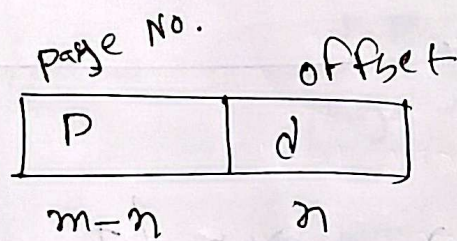
Paging (a No. 279)

logical address = page no. + page offset

Page Number $\rightarrow$ index into page table

page table $\rightarrow$ frame Number in physical memory

physical address = Frame no. * Frame size + page offset.



Paging - hardware

page table

P.M.

multi level Paging

logical address space = 64 byte, page size = 2 byte

physical address = 2^{10} byte.

Now,

$$L.A.S = 64 \text{ byte} = 2^6 \text{ byte}$$

$$\text{page size} = 2^2 \text{ byte}$$

$$P.A.S = 2^{10} \text{ byte}$$

$$\text{No. of page} = \frac{L.A.S}{\text{Page size}} = \frac{2^6}{2^2} = 2^4$$

$$\text{No. of frame} = \frac{P.A.S}{\text{page size}} = \frac{2^{10}}{2^2} = 2^8$$

$$\text{size of the page table} = 2^4 \times 8$$

$$= 2^4 \times 8 \text{ byte}$$

$$= 2^4 \text{ byte} > 2^2 \text{ byte}$$

Page table is greater than frame size. ~~That's why~~ ^{so} we cannot ~~use~~ ^{allocate} it in primary memory.

That's why we are use multilevel paging.

multi level

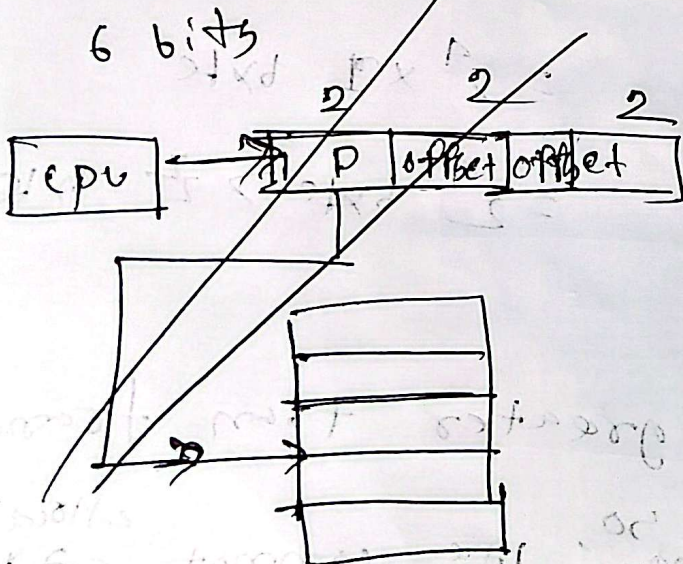
size of the page table = 2^4 byte

$$\text{No. of Page} = \frac{2^4}{2^2}$$

2 new page table size = $2^2 \times 8$
= $2^2 \times 1$ byte

Now, $2^2 > 2^2$

i. Now, allocate this page table in pm



Q. 6 (a)

Total memory Access ~~Time~~ = 1500

TLB Hits = 1200

TLB misses = 250

$$\text{TLB Hit Rate} = \frac{1200}{1500} = 80\%$$

$$\text{TLB miss rate} = \frac{250}{1500} = 16.67\%$$

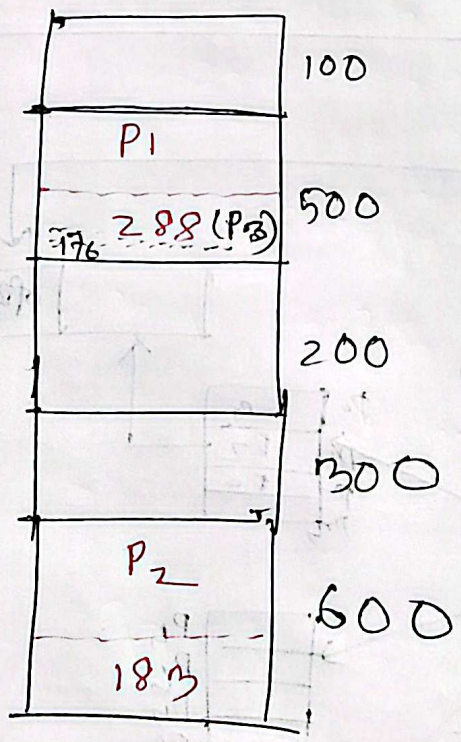
$P_1 = 212$ ✓

$P_2 = 417$ ✓

$P_3 = 112$ ✓

$P_4 = 426$

First Fit



426 K must wait

Best Fit

ϕ	100
P_2	500
8m	
P_3	200
88	
P_1	300
88	
P_4	600
174	

worst Fit

	100
P_2	500
53	
	200
P_3	300
188	
P_1	600
(P_3) 388	296

below P_5 of P_4 must waiting

$\therefore$ Best fit turns out the best

and it is better to

6
(a)

we know,

$$EAT = \text{hit} (T_{TLB} + T_{pm}) + \text{miss} (T_{TLB} + 2 T_{pm})$$

$$\Rightarrow EA = 0.8 (25 + 150) + 0.2 (25 + 2 \times 150)$$

$$= 205$$

Ans:

$$T_{TLB} = 25 \text{ nsec}$$

$$\text{hit rate} = 80\% = 0.8$$

$$\text{miss} = 1 - 0.8$$

$$= 0.2$$

$$T_{pm} = 150 \text{ nsec}$$

b) → donec) Given,

logical address space = 64 page

page size = 1024 words,

physical memory = 32 frame

① now, offset bit, b .

page size = 1024 words

$$= 2^{10}$$

∴ offset bits = 10 bits

page no. bits

Number of page = 64

$$= 2^6$$

$\therefore$ Page number bits = 6

$\therefore$ total logical address bit = page number

bit + offset bits

$$= 6 + 10$$

Ans

⑪

Number of frame

$$= 32$$

$$= 2^5$$

Frame number bits = 5

$\therefore$ Physical address = 5 + 10

$$= 15 \text{ bits}$$

Ans

7(c)

	logical address	length	base address + offset	physical address
i	0 430	430 < 600	210 + 430	
ii	1 10			
iii	2 500			
iv	3 400			

↓ offset

Segment

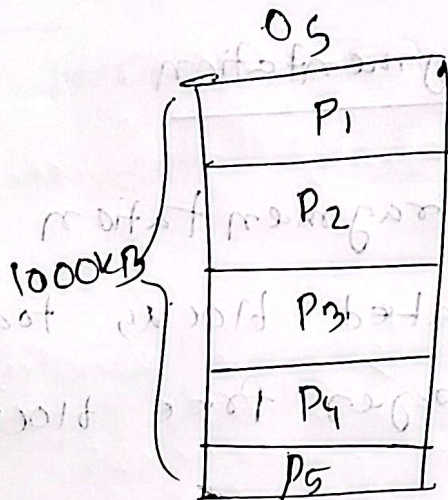
No.

if the offset > length, it indicates an address trap (segment violation).
as the logical address is outside the bounds of the segment.

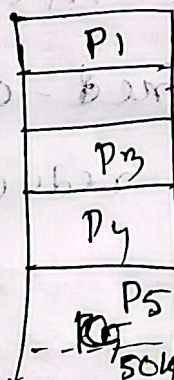
logical address		segment base	segment length	offset < length	physical address = offset + base
segment No.	offset				
0	430	210	600	430 < 600	$430 + 210 = 640$
1	10	2300	14	10 < 14	$2300 + 10 = 2310$
2	500	010	100	500 > 100	Trap (segment violation)
3	400	1327	58006	400 < 58006	$400 + 1327 = 1727$

6. (b)

Data memory size = 1000 KB

processes : $P_1 = 200$, $P_2 = 300$, $P_3 = 100$, $P_4 = 250$, $P_5 = 150$ Step-1 First Fit allocation:

No fragmentation yet.

Step-2: Fragmentation, when P_2 finishes & $P_6 (350)$ arrives. $\rightarrow$ external $\rightarrow$ For $P_5 (100KB) \rightarrow$ internal $\rightarrow P_2$'s 300 KB free $\rightarrow$ too small for350 KB $\rightarrow$ cannot allocate $\therefore P_2 (350)$ do not allocate.

here occur two fragmentation.

External : total free memory $(300+50) = 350$
but P_2 cannot fit.

Internal : P_5 needs only 100KB but
gets 150 KB $\rightarrow$ 50 KB wasted.

How minimize Fragmentation

① compaction / defragmentation
 $\rightarrow$ moves allocated blocks together
 $\rightarrow$ creates larger free block

② Best fit allocation
 $\rightarrow$ allocate smallest block that fit
 $\rightarrow$ reduce leftover fragments.

③ Advanced technique:
 $\rightarrow$ paging / segmentation / Buddy system
 $\rightarrow$ allocate fixed or variable size
block to reduce.

paging

- Divide memory into fixed size pages (pages)
- " process " same " "
- Eliminate external fragmentation

segmentation

- allocate variable-size segments
- allow logical separation (code, data, stack)

Buddy system

- Allocate block in power of 2 sizes
- easy coalescing of free blocks
- Balance external & internal fragmentation

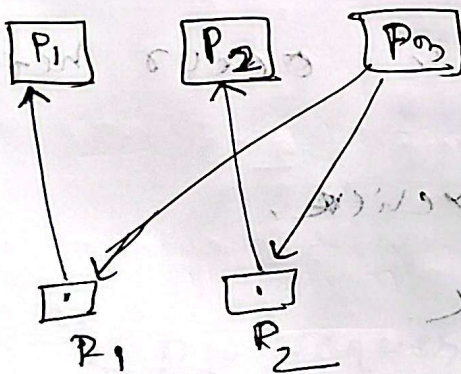
slab allocation

- pre-allocate memory to kernel object
- eliminates fragmentation for frequently used object

Deadlock

A deadlock is a situation in which two or more processes (or threads) are ~~perfor~~ permanently blocked because each of them is waiting for a resource that is held by another. As a result none of the processes can proceed.

① single instance



Process	Allocated		request	
	R ₁	R ₂	R ₁	P ₂
P ₁	1	0	0	0
P ₂	0	1	0	0
P ₃	0	0	1	1

check availability (R_1, R_2)
 $(0, 0)$

→ For P_1 , availability and request $(0, 0)$
so, it can be terminate

→ Now, availability = $(1, 0)$

old available = 0,0

New μ = 1,0

New total = 1,0

For process P_2 , there is no request, so it can be terminated.

new available = 0,1

old μ = 1,0

new μ = 1,1

P_3 need availability, need 1,1, which is true, so no deadlock occur here

there is no cycle & Acyclic,

NO dead lock.

$P_1 \rightarrow P_2 \rightarrow P_3$

wait

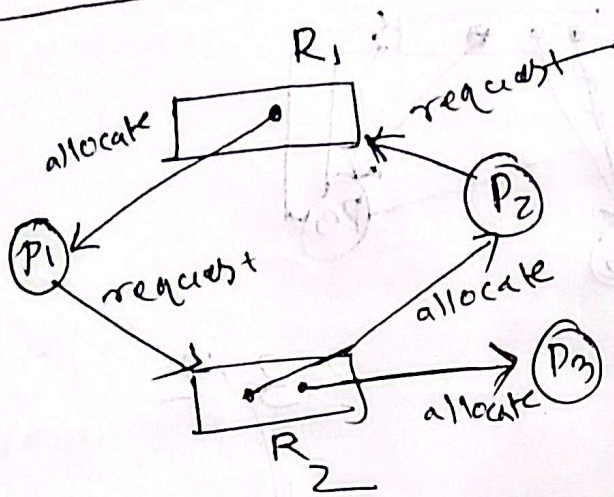
Finite

starvation

Infinite

Dead lock

multi-instance



process No	Allocated		Request	
	R_1	R_2	R_1	R_2
P_1	1	0	0	1
P_2	0	1	1	0
P_3	0	1	0	0

→ check deadlock

①. Now check availability $\rightarrow (0, 0)$

at first P_1, P_2 do not terminate,

but P_3 will be terminated.

availability $(0, 1)$

P_1 request $(0, 1) \rightarrow$ so it will possible

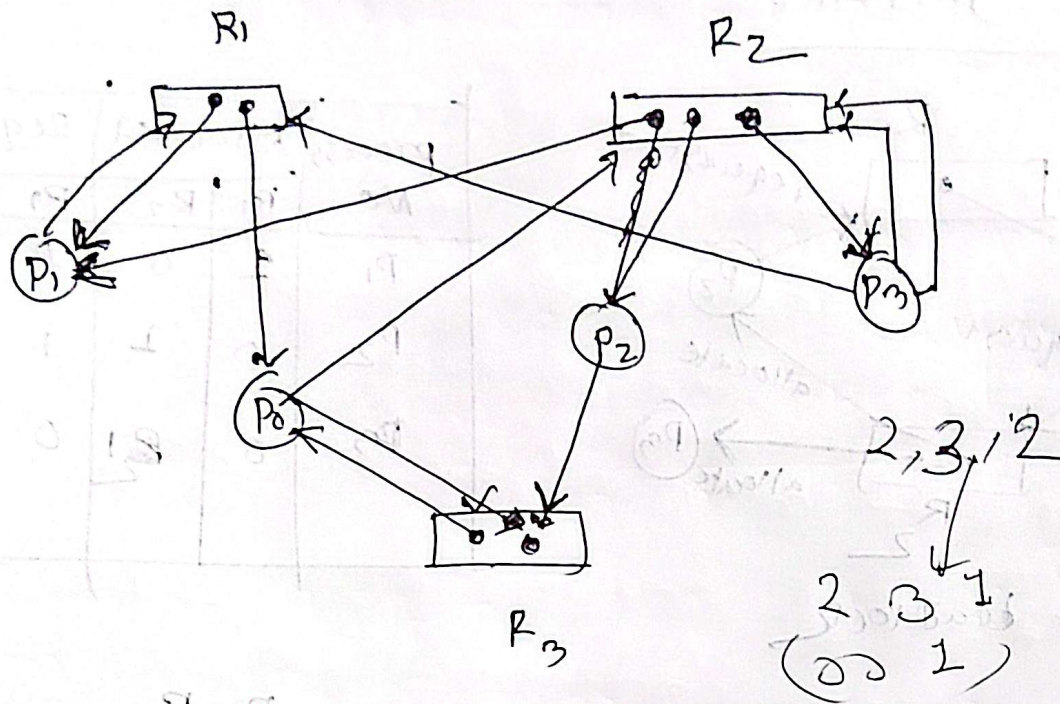
$(0, 1)$
 $(1, 0)$
 $\hline 1, 1$

availability $> 1, 1$

now, P_2 will be terminated. once

availability $= (1, 1)$

$\therefore$ NO deadlock. $P_3 \rightarrow P_1 \rightarrow P_2$



Process	Allocated			request		
	R ₁	R ₂	R ₃	R ₁	R ₂	R ₃
P ₀	1	0	1	0	1	1
P ₁	1	1	0	1	0	0
P ₂	0	1	0	0	0	1
P ₃	0	1	0	1	2	0

2 if outgoing 2 2 (2 2)

Now check deadlock occurs

availability (0, 0, 1)

P₀ (0, 1, 1), P₁ (1, 0, 0), P₂ are not full (full) request. But

P₂ (0, 0, 1) are full (full) request with available (0, 0, 1)

~~∴ available~~

P_2 terminated, $\rightarrow$ total available (2, 2, 0)

0, 0, 1

0, 1, 0

0, 1, 1

New availability = 0, 1, 1

Now, P_0 are terminated.

$\therefore$ new avail =

0, 1, 1

1, 0, 1

0, 0, 0, 0

(1, 1, 2)

~~similarly P_1, P_3 are both are~~

Now P_1 will terminate

new avail = (1, 1, 2 + 1, 1, 0)

= (2, 2, 2)

P_3 will terminate

new avail = (2, 2, 2 + 0, 1, 0)

= (2, 3, 2)

$P_2 \rightarrow P_0 \rightarrow P_1 \rightarrow P_3$

No dead lock

→ single instance → circular wait (Always deadlock)

→ multi-instance → circular wait (Not always deadlock)

P_1, P_3, P_4, P_2, P_0

5	3	2
7	4	3
7	4	5
10	4	2
10	5	2

18-10-2020

5 (10) Three examples of deadlock that are not related to computer system.

① Traffic deadlock at a four-way

Intersection:

Imagine 4 cars coming from four directions

→ each car enters the intersection

→ each car blocks the next car's path

→ No car can move forward

→ everyone is waiting for someone else to move first

② Dining philosophers - real kitchen

→ chef need two knives to cook

→ each chef picks one knife & waits for the second

deadlock

mutual exclusion: knife shared used by one

Hold & wait: each hold one knife, waiting for another

No preemption: can't force a chef to drop his knife.

circular wait: chef 1 → 2 → 3 → 4 → 5 → 1 waiting cycle.

③ Narrow bridge - single-lane Tunnel

→ Two cars enter from opposite side & block each other

deadlock

ME: Only one car can use bridge

H & W: Each car hold part of bridge waiting

No preem: can't force a car to reverse

Circular wait: A wait for B, B for A.

④ Two friends borrow a pen
→ A has notebook, B has pen. each ^{want to} ~~other~~ other item
deadlock

MF pen/notebook used by one.

HW each hold one item,

ND! A pen can't be taken forcefully

EW! A wait for B, B for A

Ex

21-22

3 (c)

① Resource management
process

- Has its own memory space (code, data, stack)
- Has its own PCB
- Heavy weight (creating or switching process is expensive)

Thread

- share same memory & resources of the process (global variables, files, heap)
- Has its own stack & register set
- lightweight (thread creation and switching fast)

2. Resource Execution

process

→ processes run independently

→ one process does not automatically know what another is doing

→ IPC is slower (shared memory, message passing)

Thread

→ Thread inside the same process

→ execute concurrently

→ They can communicate easily by sharing memory

→ If one thread changes data, other

threads can see it

process	thread
→ program in execution	→ smallest execution unit inside a process
→ Own memory	→ Shares memory
→ communication slow	→ fast
→ crash may not affect other	→ Thread crash may kill whole process
→ High cost	→ low
→ High switching cost	→ low

3(b) Deadlock prevention method

① prevent mutual exclusion (not always possible)

→ make resource sharable.

→ If a resource can be read/written shared,
no deadlock happens

limitation

most resources (printer, locks, etc) must be mutually exclusive

② prevent hold & wait

→ Ensure a process does not hold one resource while waiting for another.

→ The process must release all currently held resources before requesting new

ones.

limit

→ low resource utilization

→ processes may wait long, even if

most resources are idle.

③ prevent No preemption

→ Forcefully take away resources if needed

rule • If a process holding some resources request another that is unavailable
→ It must release all currently held resources

• These resources go back to the system

④ prevent circular wait

→ Impose a fixed ordering of resource acquisition

→ assign each resource a number

→ a process must request resources in

(increasing (or decreasing) order.

ex: If: $R_1 = 1$
 $R_2 = 2$
 $R_3 = 3$

A process holding R_2 cannot request

advans R_1 . Thus circular wait impossible,
simple & effective.

5 (a)

(i) ~~mut~~

(1) mutual exclusion (yes)

→ printers are non sharable resources

→ only one student can use a printer at a time.

(2) Hold & wait (yes)

→ Alice holds printer A while waiting for B

→ Bob u u B u A

(3) No preemption (yes)

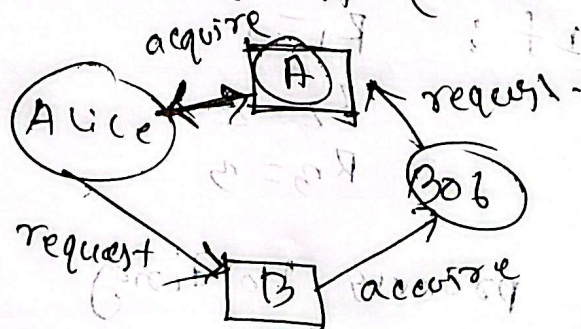
→ system explicitly states! does not preempt resources.

→ neither student printer can be taken away.

(4) circular wait (yes)

→ Alice (wait for printer B (held on Bob))

→ Bob u u A (u u Alice)



(iv) modified system

let assume,

(R) Resources = 2 printer: (A, B)

max need = each student needs at most 2 for printers to complete.

Alice $\rightarrow$ A

Bob $\rightarrow$ B

process	max need	Allocated	current need	Available
initial state				2
Alice A	2	0	2	
Bob B	2	0	2	

① Alice request 1 printer
 check $1 \leq 2$ & $1 \leq 2$
 (request $\leq$ need, request $\leq$ available)

process	max need	Allocated	current	avail
A	2	1	1	1
B	2	0	2	

safe state A $\rightarrow$ B

done

② Bob request 1 printer

check $1 \leq 2$ and $1 \leq 2$

		Max	Alloc	Need	avail
Alice	A	2	1	1	0
	B	2	1	1	

unsafe $\rightarrow$ no process can finish

③ Alice request one more printer

check $1 \leq 1$ and but $1 \leq 0$ (false)

$\therefore$ Alice wait (blocked)

deadlock occur the system

correct safe path

① Alice request 2 printers

check $2 \leq 2$ & $2 \leq 2$

		Max	Alloc	Need	avail
	A	2	2	0	0
	B	2	0	2	

safe sequence Alice $\rightarrow$ Bob

② Grant request, Alice releases finishes, releases resources,

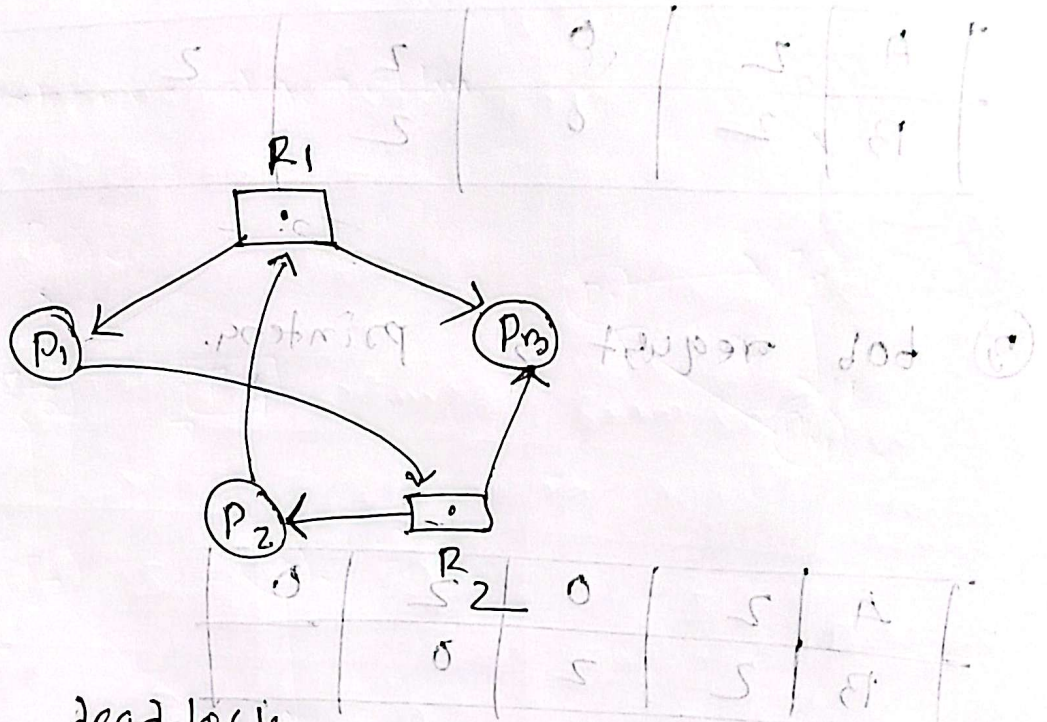
A	2	0	2	2
B	2	0	2	

③ bob request 2 printers.

A	2	0	2	0
B	2	2	0	

safe sequence = Bob $\rightarrow$ Alice

5

Q RAG Graphnow check deadlock

① availability $A = (0, 0)$
 now, $P_1(0, 2) \rightarrow \text{request}$ and $P_2(1, 0) \rightarrow \text{request}$
 which is not match $A(0, 0)$.

$\therefore P_1, P_2$ do not terminated

now $P_3(0, 0) \rightarrow \text{request}$ and match $A(0, 0)$

$\therefore P_3$ will terminated &
 total $A(1, 1)$

(11) now, $P_1(0,1) \rightarrow$ request and match

$A(1,1)$,

$\therefore P_1$ terminates and availability

$A(1+1,1) \Rightarrow A(2,1)$

$P_3 \rightarrow P_2$

(11)

$P_2(1,0) \rightarrow$ request

& match $A(2,1)$

$\therefore P_2$ terminate & update

$A(2,1+1) \Rightarrow A(2,2)$

$\therefore P_3 \rightarrow P_1 \rightarrow P_2$

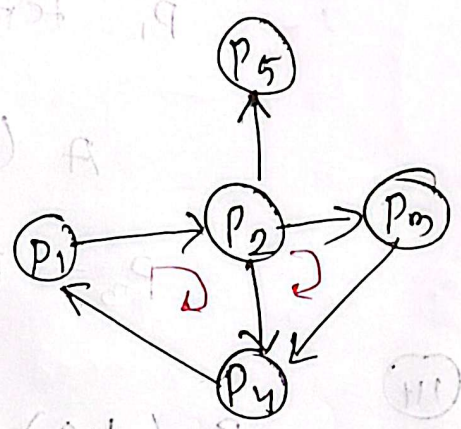
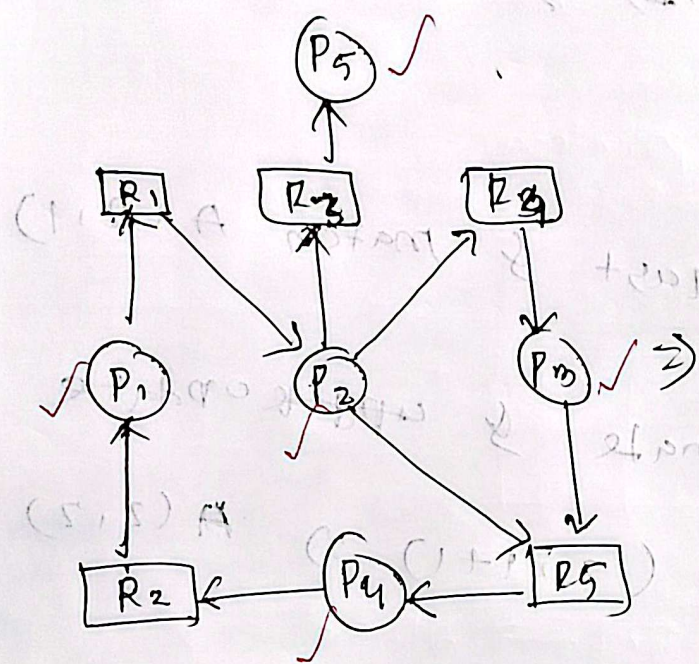
No deadlock occur.

5
①

18-19

two common deadlock detection me -

- ① wait-for graph or res. RAGs
- ② Banker's algorithm



wait for graph

RAGs

here cycle occurs, — a deadlock occurs.

$P_1 \rightarrow P_2 \rightarrow P_3 \rightarrow P_4 \rightarrow P_1$ are suffering to the deadlock, but P_5 not