Semester Final Note (Fall2024)

Course code : CSE323

Course Title : Operating Systems

Provided by:

Name : Bishal Biswas

ID: 221-15-5394

Section: 61_I

Attention : Since this is a handwritten note, there may be minor errors like spelling mistakes or calculation mistakes or something else. For which I apologize. Those who read this note should first follow the rules & instructions of their Course teacher first and then if they want, they can follow this note additionally.

Planned effort never goes in vain.

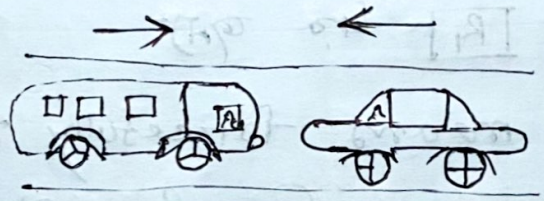
(Best of Luck)

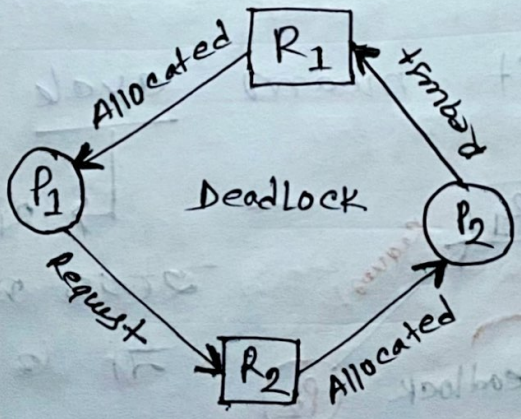
Semister final

Dead Lock

→ Deadlock occurs when two or more Processes are stuck, waiting for each other to release resources & none of them can continue.

Ex: It's like a traffic jam where no car can move because every car is blocking another.





P1 / P2 → Processes

R1 / R2 → Resources

↑

Not Shareable

Deadlock occurs 4 necessary conditions

(i) Mutual Exclusion → যে Resource যে Process এর জন্য বরাদ্দ কেবল একক কেবল মাত্র সেই Process তা Access করতে পারবে অন্য কোন Process পারবে না।

(ii) Hold & wait

(iii) NO Preemption

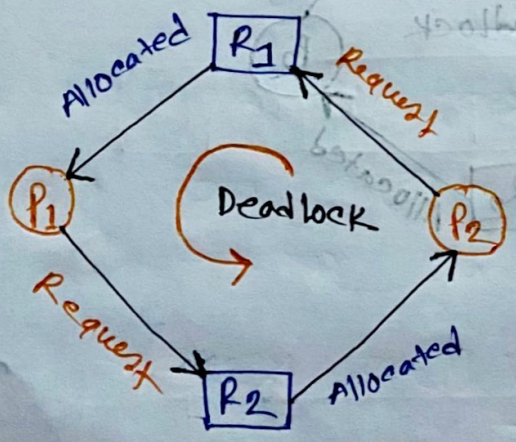
(iv) circular wait or cycle

(P1) hold করে রেখেছে [R1] কে এবং wait করতেছে [R2] এর জন্য। (P2) holding রেখেছে [R2] কে এবং wait করতেছে [R1] এর জন্য।

→ Preemption means forcefully কোন Process কে stop করে দেওয়া যদি Preemption occur হয় তা Deadlock হবে না। তাই deadlock Non-Preemptive হবে।

→ circular wait means cycle থাকা বুঝায়

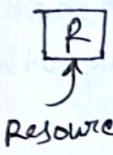
→ থাকলে Deadlock হতেও পারে আবার না ও হতে পারে।



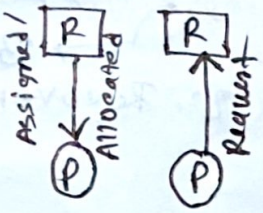
Resource allocation Graph in Deadlock

Graph

(i) vertex



(ii) Edge



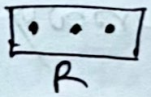
Resource:

(i) Single-Instant : Which has only one instant. [যার কোন কপি থাকবে না]

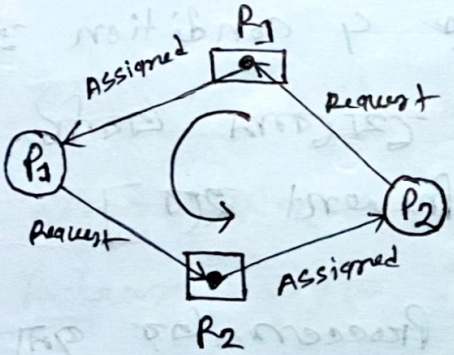
Ex:

(ii) Multiple-Instant : Which has multiple instantance [যার অনেক কপি থাকতে পারে।

Ex:



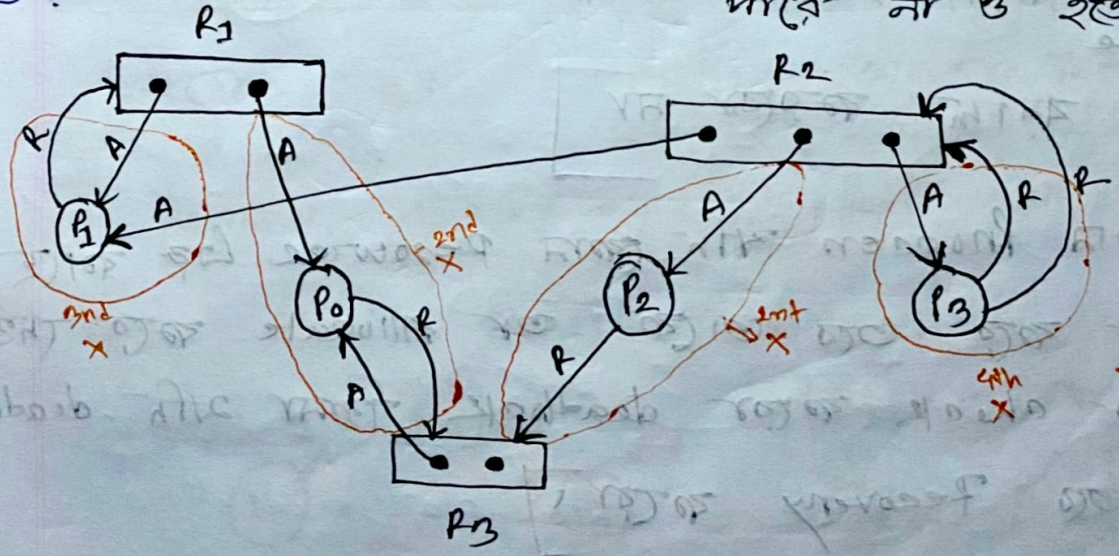
(i) Single-Instant:



→ Deadlock

→ Single instance ক্ষেত্রে cycle হলেই Deadlock but multiple instance ক্ষেত্রে check করা লাগবে → হতেও পারে না ও হতে পারে।

(ii) Multiple-Instant:



→ cycle exist
→ NO Deadlock

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

Deadlock handling method :

(i) → Deadlock : Ignorance

(ii) → Deadlock Prevention **

(iii) → Deadlock Avoidance ** → Safe & unsafe state
→ Resource allocation graph
→ Banker's algorithm

(iv) → Deadlock detection & Recovery

→ [যদি deadlock occur হয় simply ignore করে OS. ↳ তার OS performance ও বেশি focus করে।]

→ [Necessary deadlock এর 4 condition হয় Remove করে দেও অথবা যেকোন একটি condition false করে দেও তাহলে Prevent হবে।]

→ [কোন Resource কে কোন Process এর জন্য বরাদ্দ করার সময় check করবে deadlock হয় না Allocate হলে বরাদ্দ করবে না]

→ [কোন Process যদি কোন Resource এর জন্য Request করে তবে তাকে তা Allocate করে দেও তারপর check করে deadlock, যদি না হয় তবে Recovery করো।]

Deadlock Prevention :

→ Need to remove all conditions or at least one condition to prevent the deadlock

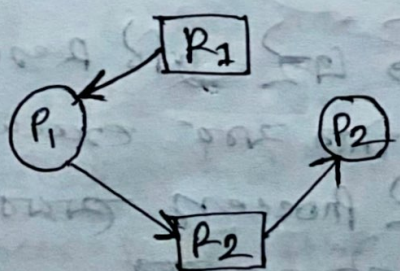
[Necessary 4টি condition এর সবগুলো / যেকোন একটি- Remove করতে হবে]

(i) Remove mutual Exclusion : Resource গুলো সব সবার জন্য উন্মুক্ত করে দিতে হবে।

(ii) Remove hold & wait : New Resource কে request করার আগে পূর্বের বরাদ্দ Resource কে Release করতে হবে।

(iii) Do Preemptive : Priority based → যার Priority আগে তাকে execution আগে করতে হবে।

(iv) Remove cycle/ circular wait : Order maintain করে cycle Remove করতে হবে



R_1 → Increasing order

R_2

$P_1 \rightarrow R_1$

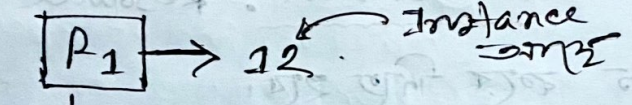
$P_2 \rightarrow R_2$

Note: Increasing order এর $P_2 \rightarrow \times R_1$

Safe & Unsafe State Deadlock Avoidance :

Safe state : If the system can successfully allocate to each process by giving resources then it will be called as safe state

[যদি system প্রত্যেক প্রসেসকে resource দিয়ে allocate করতে পারে তবে তা safe state]



Process	Required Resource	allocated Resource	Resource Needed
P_1	6	2	4
P_2	11	5	6
P_3	5	3	2
		total (10)	

(*) Available Resources = (Total Instant/Resource − Allocated Resource)

= 12 − 10

= 2

↳ এই 2টি Resource system কোন একটি process কে দিয়ে তার execution আগে complete হবে এবং সেই process যতগুলো resource hold করে রেখেছে তারা উন্মুক্ত হবে।

$P_3 \rightarrow P_1 \rightarrow P_2$

P	Available R.
P_3	5
P_1	7
P_2	12

① → ② hold : 6+1 = 7
② → ③ hold : 11+1 = 12

unsafe state : If the system can't avoid the deadlock by allocating resources to each process then it will be called as unsafe state.

Process	Required resource	allocated resource	Resource Needed
P_1	9	2	7
P_2	11	5	6
P_3	5	3	2

total : 10

instantaneous

$P_1 \rightarrow 12$

deadlock

Available Resources : 12 − 10 = 2

$P_3 \rightarrow P_2 \rightarrow P_1$

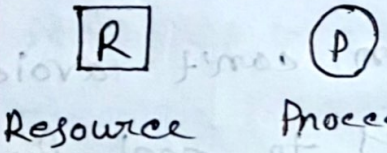
deadlock

P	Available
P_3	5
P_2	
P_1	

P_2 need 6 resource but available 5. So full execution not complete therefore P_1 also can't do execution it's a deadlock.

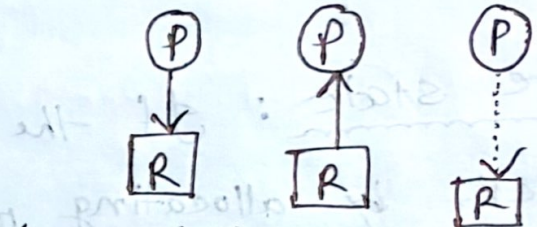
⊞ Resource allocation Graph Deadlock Avoidance:

(i) Vertex:



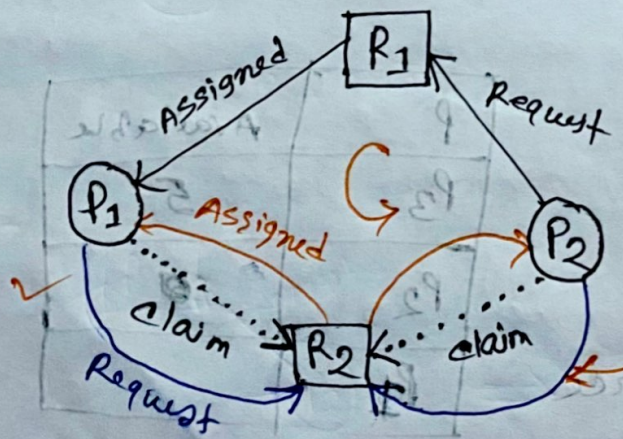
Resource Process

(ii) Edge:



(Request edge) (Allocate/Assigned edge) (claim edge)

Procedure: কোন Process তার প্রয়োজনীয় Resource কে claim করে রাখবে। System check করবে যে Resource টেই Process কে দিলে তা safe না unsafe state এ যায়। safe হলে deadlock occur হবে না কিন্তু unsafe হলে deadlock occur হতে পারে অথবা না ও হতে পারে।



→ single instant জন্য কাজ করে

Note: P_2 কে R_2 দিলে cycle occur হয় তাই system P_2 কে R_2 দিবে না।

Banker's Algorithm Deadlock Avoidance :

→ Multiple Instance এর জন্য ব্যবহার করে।
→ মানে copy

Ex : Here,

$R_1 = 10,\ R_2 = 5,\ R_3 = 7$

Process	Max need			Allocated Resource			current Resource need			Available Resource		
	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3
P_0	7	5	3	0	1	0	7	4	3	3	3	2
P_1	3	2	2	2	0	0	1	2	2	5	3	2
P_2	9	0	2	3	0	2	6	0	0	7	4	3
P_3	2	2	2	2	1	1	0	1	1	7	4	5
P_4	4	3	3	0	0	2	4	3	1	10	4	7
total -				7	2	5				free →	10 (R_1)	5 (R_2)	7 (R_3)

current Resource Need = Max need − Allocated Resource

Available Resource = Instance − total allocated resource

$R_1 = 10 - 7 = 3$

$R_2 = 5 - 2 = 3$

$R_3 = 7 - 5 = 2$

Safe scan → $P_1 \rightarrow P_3 \rightarrow P_4 \rightarrow P_2 \rightarrow P_0$

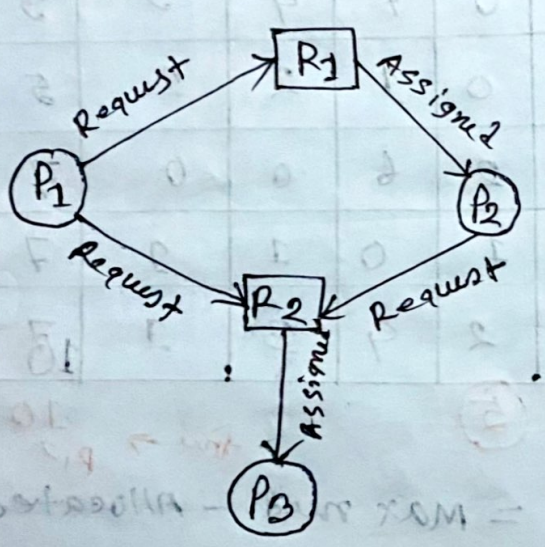
Available resource Increase = Available Resource + Allocated Resource (এর আগের scan থেকে)

Deadlock Detection & Recovery

Deadlock Detection

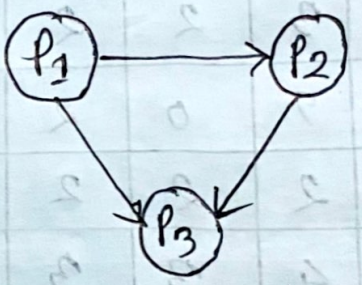
- Wait-for Graph (single instance)
- Banker's algorithm (multiple instance) — Also called safety algorithm

Example of wait-for Graph:



(Resource allocation graph)

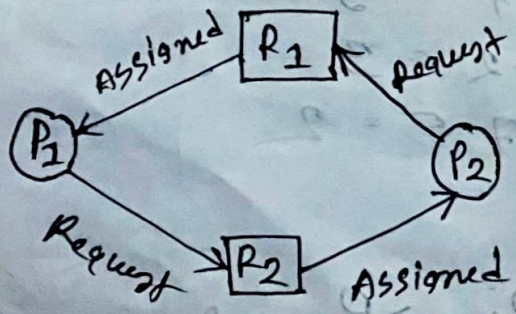
⇒



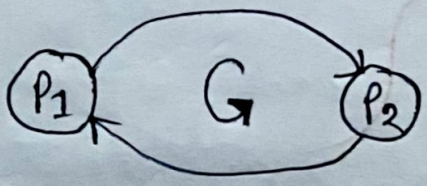
→ NO cycle
→ NO Deadlock

(wait for graph)

Ex 2:



⇒



→ cycle occur
→ Deadlock occur.

Banker's Algorithm:

Update Available = old available + Allocation (of process)

Ex: A = 7, B = 2, C = 6

current resource need

Process	Allocation			Request			Available R.		
	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	0	1	0
P_2	3	0	3	0	0	0	3	1	3
P_3	2	1	1	1	0	0	5	2	4
P_4	0	0	2	0	0	2	5	2	6
							(7)	(2)	(6)

→ All [R] available
→ No deadlock

∴ safe scan ⇒ $P_0 \rightarrow P_2 \rightarrow P_3 \rightarrow P_4 \rightarrow P_1$

Ex: A = 7, B = 2, C = 6

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	0	1	0
P_2	3	0	3	0	0	1			
P_3	2	1	1	1	0	0			
P_4	0	0	2	0	0	2			

Deadlock occur

Pain doesn't compareable that is unsafe state

safe scan ⇒ P_0 → deadlock → unsafe state

Deadlock Recovery

1] Process Termination :

(i) Abort all deadlock Processes.

(ii) Abort one Process at a time untill the deadlock cycle is eliminated.

↳ একবার কয়েক Process কে eliminate করে check করে algorithm apply করে deadlock আছে কিনা, → costly

2] Resource Preemption :

(i) selecting a victim : Which resources and Process are to be Prempted? As in Process termination, we must determine the order of Preemption to minimize the cost.

(ii) Roll back : Rollback the Process which was selecting as a victim

→ Roll back the Process to some safe state.

→ Abort the Process and the restart it.

2. Suppose, in an office, we have a set of resource types, R = {R1, R2, R3} and a set of processes, P = {P1, P2, P3, P4, P5}. **R1, R2, and R3** have **3, 2, and 1** instances respectively.

- P1 requests for 1 instance of R3
- P1 is holding 2 instances of R2
- P2 requests for 1 instance of R2
- P2 is holding 1 instance of R1
- P3 is holding 1 instance of R3
- P3 requests 1 instance of R1
- P4 is holding 1 instance of R1
- P5 is holding 1 instance of R1
- P5 requests 1 instance of R2

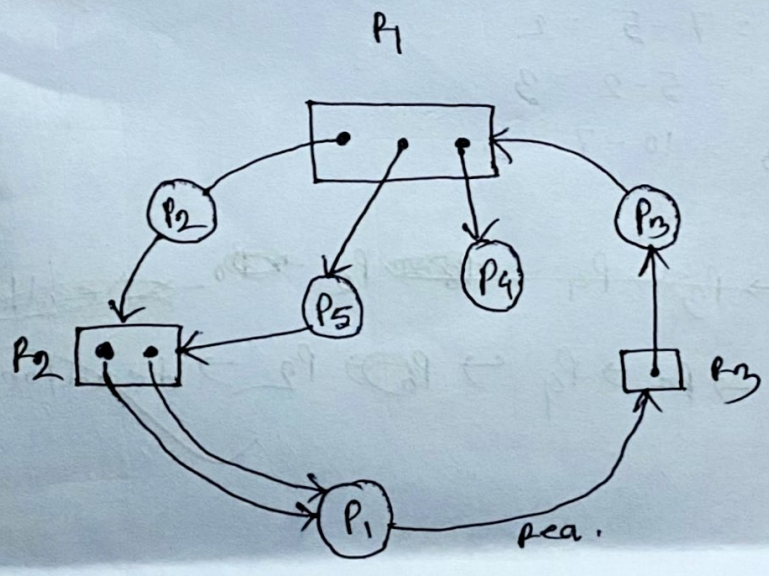
Construct a resource allocation graph for the above scenario. **Mention** the number of cycles found and **identify** whether there is a deadlock or not. [5]

Given,

$R_1 = 3$ instance

$R_2 = 2$ instance

$R_3 = 1$ instance



→ 2 cycle detected

→ but there are no dead lock

→ cycle 1: $P_1 \rightarrow R_3 \rightarrow P_3 \rightarrow R_1 \rightarrow P_5 \rightarrow R_2 \rightarrow P_1$

→ cycle 2: $P_1 \rightarrow R_3 \rightarrow P_3 \rightarrow R_1 \rightarrow P_2 \rightarrow R_2 \rightarrow P_1$

(iii) Starvation : It may happen that the same process is always picked as a victim. As a result the process never completes its task we must ensure that process can be picked as a victim only a (small) finite number of time.

Memory Management in Operating system

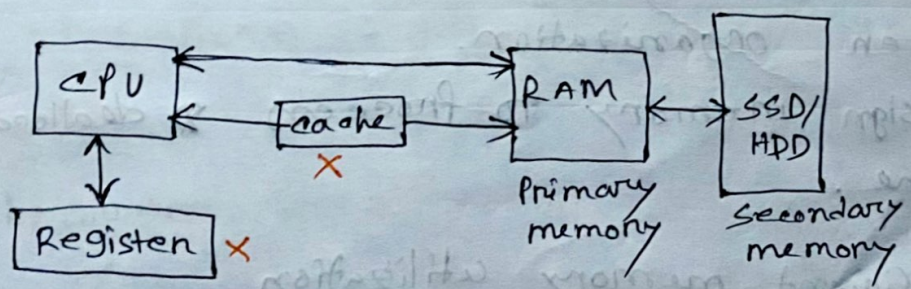
=> It is the process of organizing & controlling a computer's memory.

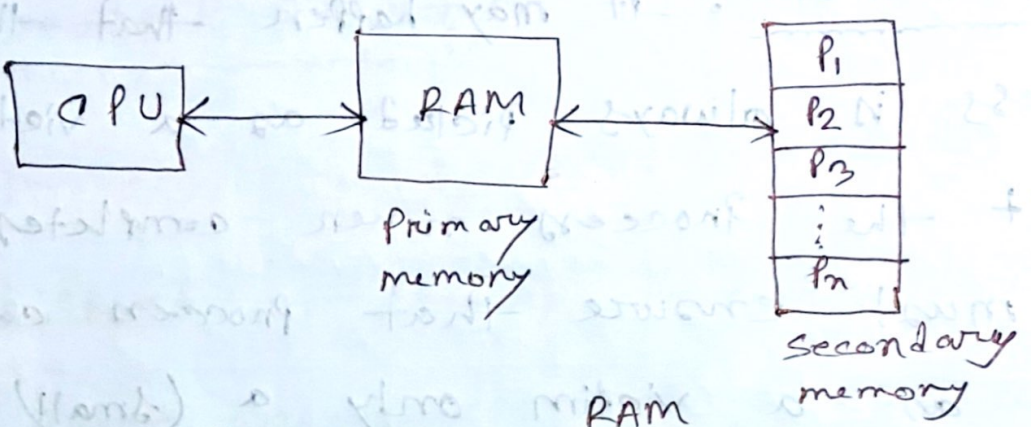
Three criteria :

(i) memory size → বড় হলে ভালো

(ii) Access time → কম হলে ভালো

(iii) Per unit cost → কম হলে ভালো





(*) যত বেশি Process RAM তে আনা যায় CPU utilization তত বেশি হবে।

(*) কোন Process টিকে RAM আনা লাগবে এবং তার জন্য কতটুকু space বরাদ্দ এই কাজ OS করে।

(*) memory management function টি primary memory অর্থাৎ RAM কে manage করে থাকে।

I. Function & Goals of memory management system:

Functions:

- keep track of which parts of memory are use by whom.
- Divide memory into manageable sections for better organization.
- Assign memory to Process & deallocated when done.

Goals:

- Efficient memory utilization
- Distribute memory fairly among Process
- Prevent Processes from interfering with each other's memory.

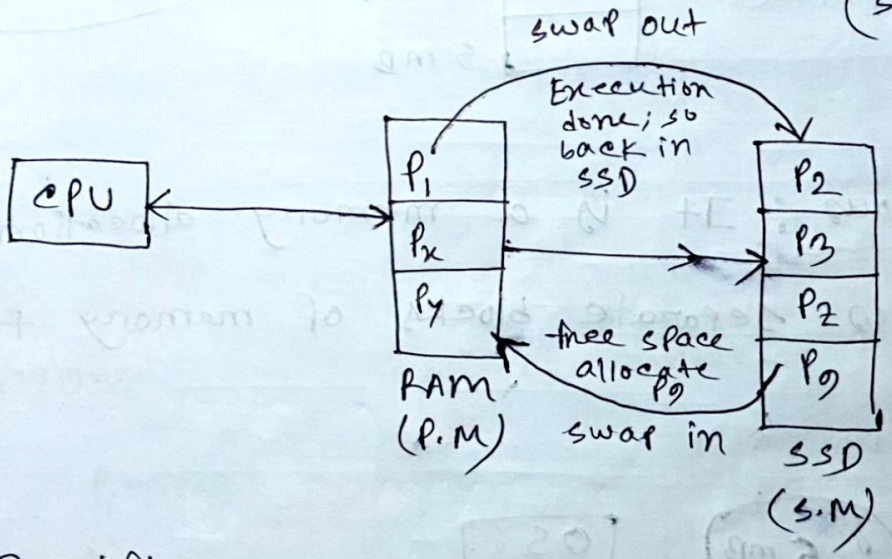
[+] Swaping in memory management system:

⇒ Proccess are temporarily moved from the primary memory (RAM) to a secondary memory (SSD/HDD), to free up space for other proccess.

(RAM → SSD)

Two term:

- → swap out: Proccess send primary memory to secondary memory
- → swap In: Proccess send secondary memory to primary memory (SSD → RAM)



Benefits

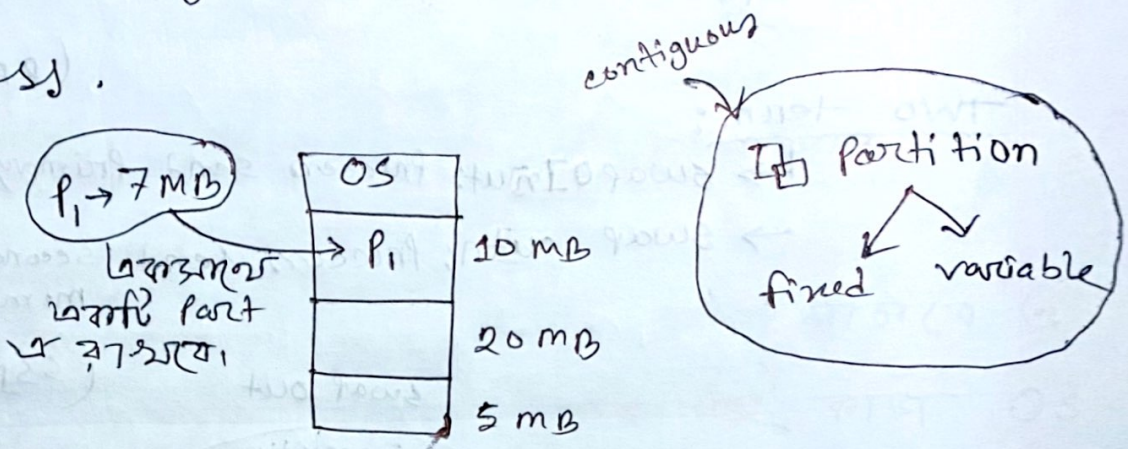
- → Allow multitasking by sharing memory between multiple Proccess.
- → support larger Program than Physical memory.

Drawback:

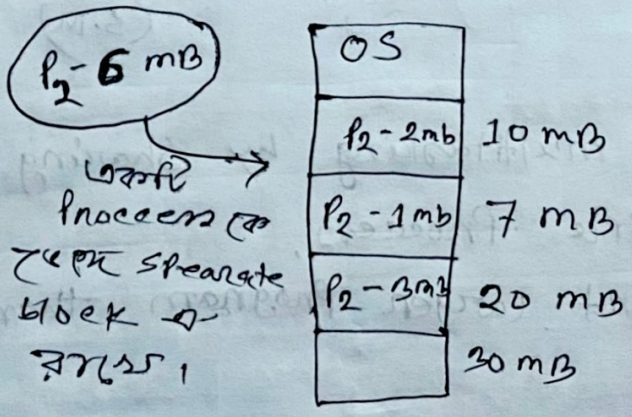
- → slower Performance due to dis I/O operation (known as "swap thrasing").
- → Increase load on secondary storage.

Contiguous & Non-contiguous memory allocation :

Contiguous : It is a memory allocation method that allocates a single contiguous section of memory to a Process.



Non-contiguous : It is a memory allocation method that allocates separate blocks of memory to a Process.



→ Paging
→ segmentation
→ Non-contiguous.

⊡ fixed size partitioning

↳ contiguous memory allocation

⇒ The memory is divided into fixed sized blocks or partitions at the start of program execution

- → Each partition can hold one process.
- → Memory is divided into Equal-sized blocks.
- → Each blocks size can be same or different.

Internal fragmentation: When process size is smaller than the partition size.

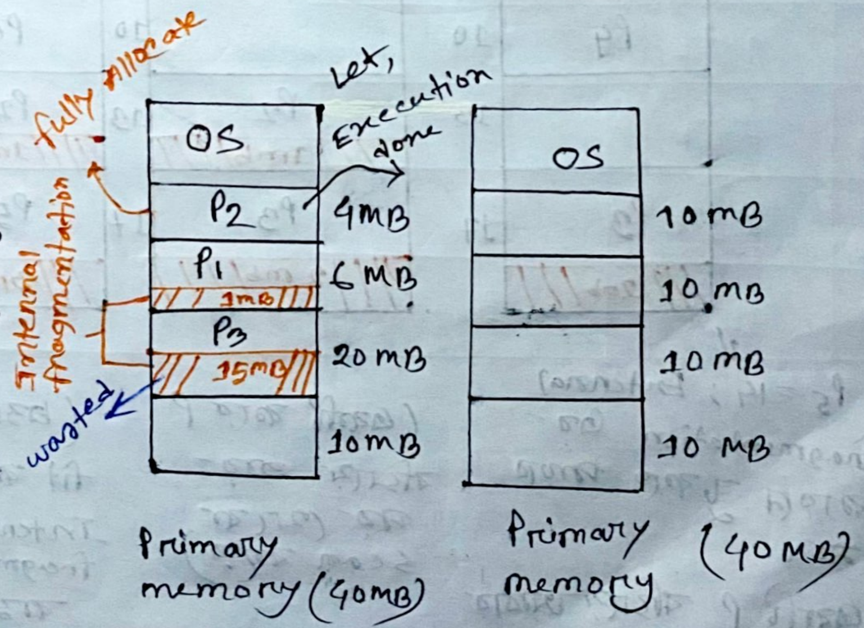
If a process doesn't use the full space in its assigned partition, the remaining space is wasted.

$P_1 = 5$ MB

$P_2 = 4$ MB

$P_3 = 15$ MB

$P_4 = 13$ MB



External fragmentation: free memory is available but it is scattered in small, non-contiguous blocks.

Ex: P_4 13 MB এর কোনো block এ যেতে পারছে না মেমোরি থাকা সত্ত্বেও

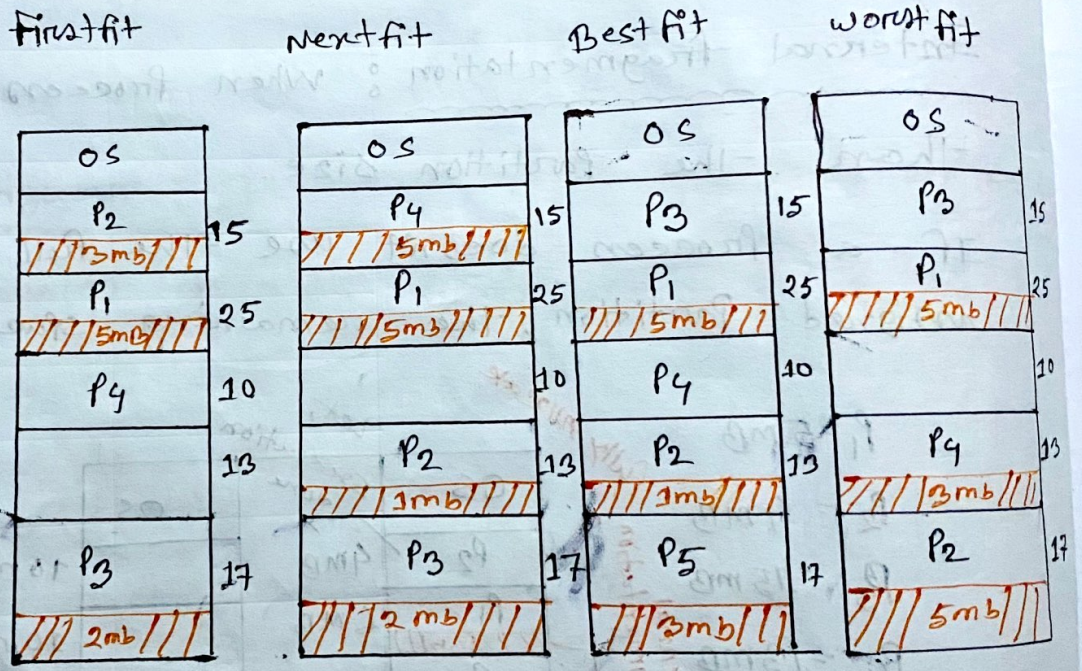
fixed size partitioning first fit, Next fit, Best fit, worst fit :

→ কোন Algorithm এ Internal fragmentation কম হয় তার বের করবো।

(i) first fit
(ii) Next fit
(iii) Best fit
(iv) worst fit

} Algorithm

$P_1 = 20$
$P_2 = 12$
$P_3 = 15$
$P_4 = 10$
$P_5 = 14$



$P_5 = 14$; External fragmentation এর কারণে চুকতে পারছে না।
(একটি P বসিয়ে আবার প্রথম থেকে scan করে)

(একটি করে P বসিয়ে তার পর থেকে scan করে)

(এভাবে fit করে গেলে Internal fragmentation কম হয়)

(এভাবে fit করে গেলে Internal fragmentation বেশি হয়)

[Best fit Algorithm fixed size Partition এর জন্য একদম Perfect]

Variable size Partition in contiguous memory allocation

⇒ Memory is divided into Partitions of variable sizes to allocate to Processes dynamically.

Key features:

→ Memory is divided into partition only when needed.

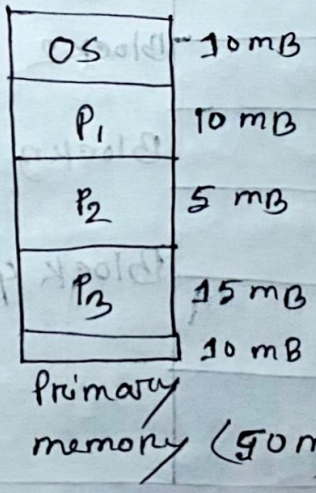
→ The size of each partition matches the memory requirements of the process being loaded.

→ Remove Internal fragmentation

→ May lead to external fragmentation over time.

→ when a Process finishes, its partition is free for re-use.

$P_1 = 10$
$P_2 = 5$
$P_3 = 15$



Note: External fragmentation occur হলে Block গুলোকে merge করে তার সমাধান করা যায়।

*** A system uses fixed Partitioning to allocate memory to Jobs using the Best-Fit memory allocation method. The memory blocks & Job sizes are listed bellow. your task is to determine the memory allocation & analyze the results.

Job	Size (Kb)	TAT (ms)	Memory Block	Size (KB)
Job 1	90K	4	Block 1	40K
Job 2	20K	2	Block 2	100K
Job 3	50K	3	Block 3	60K
Job 4	15K	1	Block 4	120K
Job 5	75K	3		
Job 6	25K	2		
Job 7	60K	3		

⇒ Best fit :

Step 1 :

Block	size	Job	TAT	
1	40	Job2	2	1
2	100	Job 1	4	3
3	60	Job 3	3	2
4	120	Job 4	1	0

Step 2 :

Block	size	Job	TAT	
1	40	Job2	2	0
2	100	Job 1	4	2
3	60	Job 3	3	1
4	120	Job 5	3	2

Step 3 :

Block	size	Job	TAT	
1	40	Job 6	2	1
2	100	Job 1	4	1
3	60	Job 3	3	0
4	120	Job 5	3	1

Step 4 :

Block	size	Job	TAT	
1	40	Job 6	2	0
2	100	Job 1	4	0
3	60	Job 7	3	2
4	120	Job 5	3	0

Step 5 :

Block	size	Job	TAT	
1	40			
2	100			
3	60	Job 7	3	1
4	120			

Step 6 :

Block	size	Job	TAT	
1	40			
2	100			
3	60	Job 7	3	0
4	120			

Step 7 :

Block	size	Job	TAT
1	40		
2	100		
3	60		
4	120		

Memory is fully free

Paging (Non contiguous memory allocation)

Paging is classic example of non-contiguous memory allocation where the logical memory of a process split into parts (page) that can be stored in physical memory location.

Key concept of Paging:

(i) Page: A fixed-size block of logical memory

(ii) Frame: A fixed block of physical memory

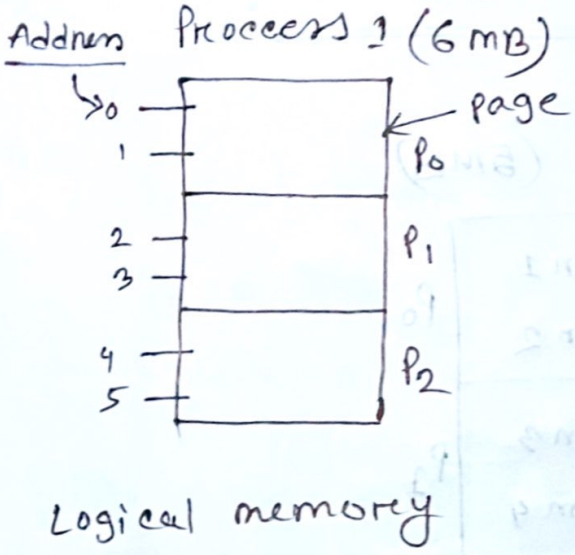
(iii) Page Table: A data structure maintained by OS to keep track of mapping between a process's pages & physical memory frames.

Advantages:

→ Remove external fragmentation
→ Efficient use of memory
→ Simplifies memory management

Disadvantages:

→ Internal fragmentation
→ Overhead of address translation
→ Increase complexity.



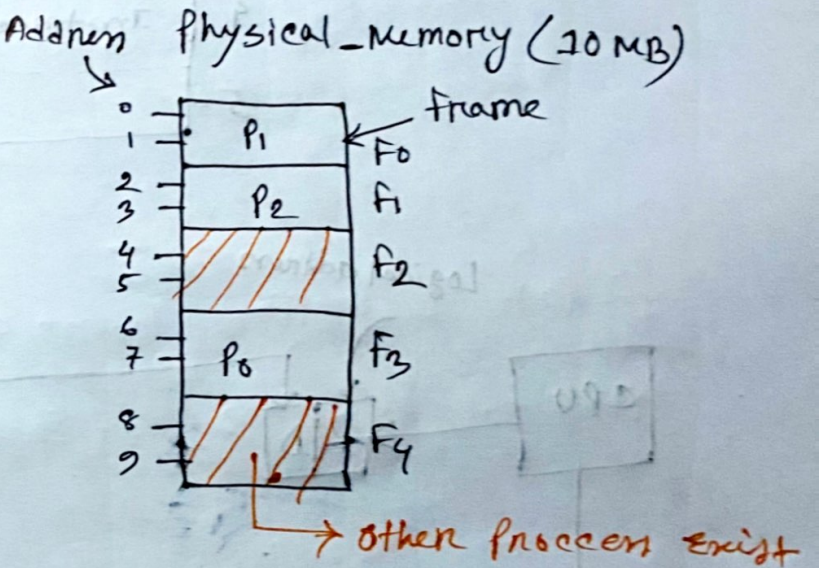
Let,

Proccess 6MB (size)

Page size 2 MB

$$\text{NO. of page} = \frac{\text{Proccess\_size}}{\text{Page size}}$$

$$= \frac{6}{2} = 3$$



P.M_size = 10MB

fram_size = Page time

= 2 MB

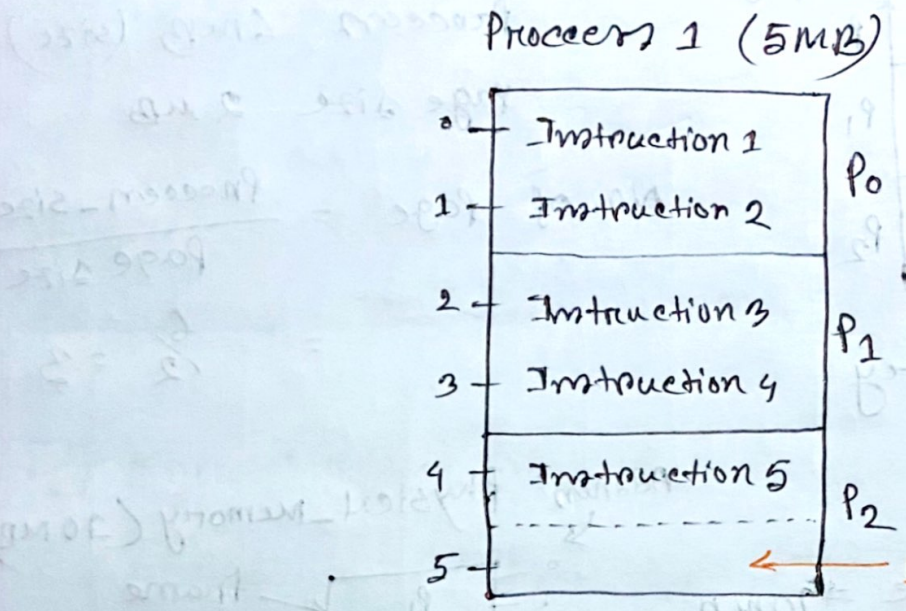
NO. of frame = 10/2

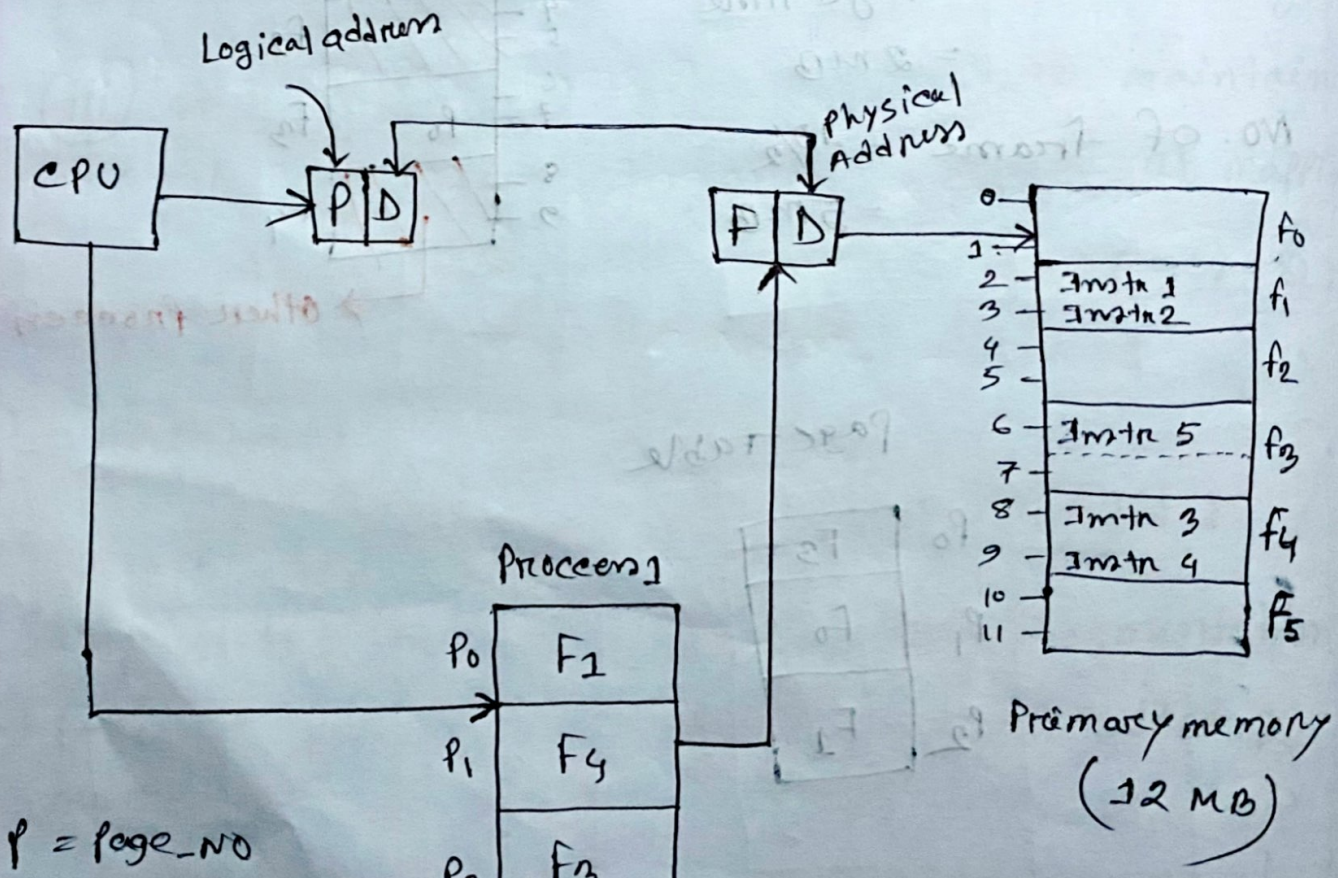
= 5MB

Page Table

P0	F3
P1	F0
P2	F1

*** Paging Hardware Architecture :





p = page_no

d = offset

f = frame_no

Logical Address to physical Address :

- Logical Address
- page size
- frame no

} Question এ-
- যেমন থাকবে
অথবা figure আকারে যেমন থাকবে।

সূত্র :

(i) page_no (p) = Logical address / page_size

(ii) offset (d) = Logical Address % page_size

(iii) Physical Address = (frame_no * page_size) + offset

(iv) Physical Address = (Base address of frame + offset)
↳ figure এরকম easy হয়

Example :

From the bellow given details, calculate the physical address :

page size : 4 kb (1kb = 2^{10} bytes)

Logical address : 8197

page table :

pg0 → frame 3

pg1 → frame 5

pg2 → frame 9

Solution :

Page NO = Logical address / page_size

$= \frac{8197}{4096}$

$= 2$ (integer part)

Page size = 4 KB

$= 4 \times 2^{10}$

= 4096 byte

Page offset (d) = Logical address % page_size

= 8197 % 4096

= 5

∴ Page 2 mapped - frame 9

Physical Address = (frame NO * page_size) + page offset

= (9 × 4096) + 5

= 36869

2^{10} byte = 1 Kb

2^{20} byte = 1 MB

2^{30} byte = 1 GB

2^{40} byte = 1 TB

Ex: Here, a Proccess size is 8 MB & physical address space is 16 MB. If the page size is 4KB then calculate total num. of page & total num. of frame from the given information.

⇒ Given,

Proccess size = 8 MB = 8×2^{20} =

Physical Address = 16 MB $\times 2^{20}$ =

Page size = 4 Kb $\times 2^{10}$ = 4096

$$\therefore \text{No. of page} = \frac{\text{Proccess size}}{\text{page size}} = \frac{8 \times 2^{20}}{4096} = 2048$$

$$\therefore \text{No. of frame} = \frac{\text{Physical Address}}{\text{frame size}} = \frac{16 \times 2^{20}}{4096}$$

$$= 4096$$

[framesize = page size]

Ex: Logical Address = 23 bits = 2^{23} bytes

Physical memory = 16 MB

frame size = 4 Kb

$$\therefore \text{Total no. Process} = \frac{2^{23}}{4 \times 2^{10}} = 2048$$

$$\therefore \text{Total no frame} = \frac{16 \times 2^{20}}{4 \times 2^{10}} = 4096$$

☐ segmentation : segmentation refers to the process of dividing a large entity into smaller, more manageable & meaningful parts,
→ memory management technique.

A segment is a logical unit such as :

→ main program
→ function
→ method
→ stack
→ array
→ object & so on.

How segmentation reduce paging limitations?

=> Paging limitations are:

(i) Internal waste

(ii) complex mapping between logical & physical memory

(iii) without special hardware like TLBs, access will be slow

(iv) There are no logical grouping

(v) Internal fragmentation occur.

Advantages of segmentations:

→ logical grouping occur between function, array, random blocks

→ No waste of space

→ Give user friendly view to see Program

→ segment can grow & shrink easily because of its flexible size.

→ Allow specific sharing (like func.) between Program.

Difference between paging & segmentation

Paging	Segmentation
→ Divide memory into fixed size	→ Divide memory into variable size
→ use page table to map physical & Logical address	→ use a segment table for mapping address
→ No external fragmentation	→ Can be external fragmentation
→ Internal fragmentation exist	→ less internal fragmentation, as segment vary
→ faster access due to uniform page size (TLB's)	→ Slower access due to varied size.

Virtual Memory

- Page Replacement algorithm:
  - → FIFO (First In First out)
  - → Optimal
  - → LRU (Least Recently used)

- F = Page fault / Page miss
  H = Page Hit

* Hit Ratio = $\frac{\text{Total no. of Hit}}{\text{Total num. of string}} \times 100$

* fault Ratio = $\frac{\text{Total no. of fault}}{\text{Total num. of string}} \times 100$
  ↳ miss ratio

FIFO :

Ex: use 3 frame & Reference string are:

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

⇒

String	7	0	1	2	0	3	0	4	2	3	0	3	0	3	2	1	2	0	1	7	0	1
frame 1	7	7	7	2	2	2	2	4	4	4	0	0	0	0	0	0	0	0	0	7	7	7
frame 2		0	0	0	0	3	3	3	2	2	2	2	2	2	2	1	1	1	1	1	0	0
frame 3			1	1	1	1	0	0	0	3	3	3	3	3	3	3	2	2	2	2	2	1
Result	F	F	F	F	H	F	F	F	F	F	F	H	H	H	H	F	F	H	H	F	F	F

Total fault = 15

Total Hit = 7

Total string = 22

∴ Hit Ratio = $\frac{7}{22} \times 100 = 31.82\%$

∴ fault Ratio = $\frac{15}{22} \times 100 = 68.18\%$

[4] Optimal Algorithm :

checking → (Left to Right)

Ex : A system uses 3 page frames for storing process pages in main memory. It uses the optimal page replacement policy. Assume that all the page frames are initially empty. what is the total num. of page fault & Hit that will occur while processing the page reference string given bellow -

4, 7, 6, 1, 7, 6, 1, 2, 7, 2

also calculate Hit & Miss Ratio.

⇒

Data	4	7	6	1	7	6	1	2	7	2
frame 1	4	4	4	1	1	1	1	2	2	2
frame 2		7	7	7	7	7	7	7	7	7
frame 3			6	6	6	6	6	6	6	6
Result	F	F	F	F	H	H	H	F	H	H

Total fault = 5 ; Miss Ratio = $5/10 \times 100 = 50\%$

Total Hit = 5 ; Hit Ratio = $5/10 \times 100 = 50\%$

Least Recently used (LRU) :

check ← left — Right

Ex: you can use three frame given string are

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

⇒

Data	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
frame 1	7	7	7	2	2	2	2	4	4	4	0	0	0	1	1	1	1	1	1	1
frame 2		0	0	0	0	0	0	0	0	3	3	3	3	3	3	0	0	0	0	0
frame 3			1	1	1	3	3	3	2	2	2	2	2	2	2	2	2	7	7	7
result	F	F	F	F	H	F	H	F	F	F	F	H	H	F	H	F	H	F	H	H

Fault Ratio = $12/20 \times 100 = 60\%$

Hit Ratio = $8/20 \times 100 = 40\%$

Disk scheduling Algorithms

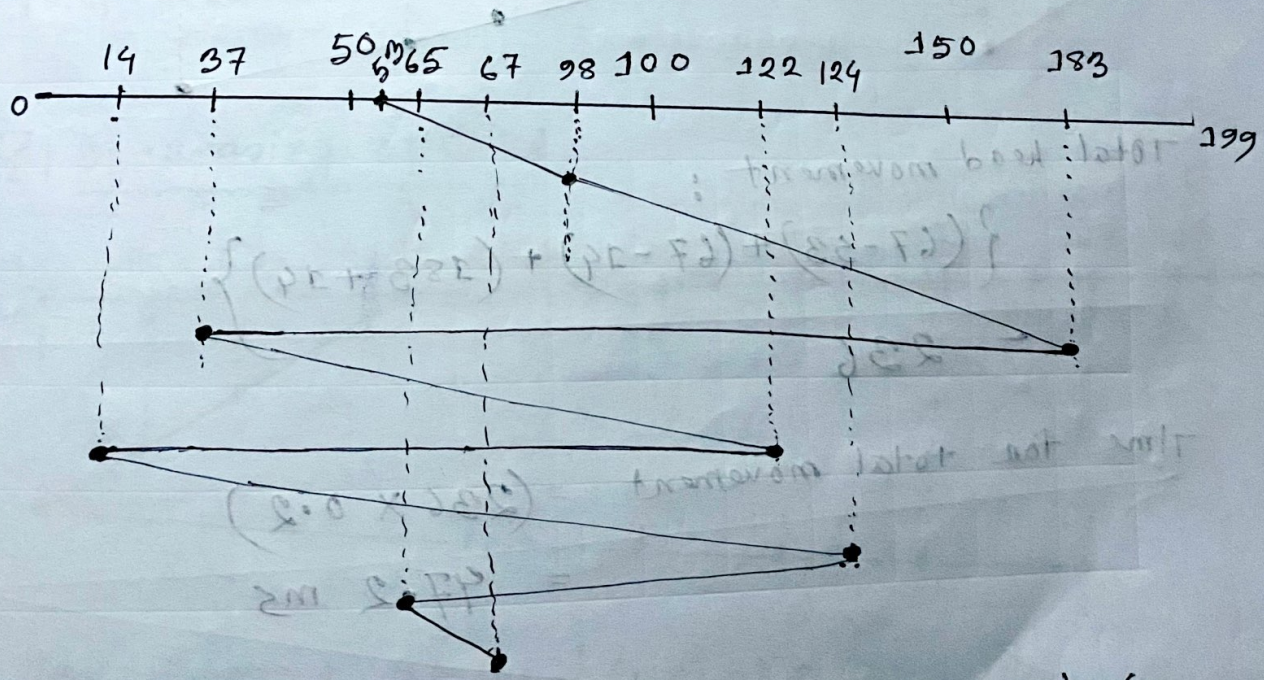
- FCFS (first come first serve)
- SSTF (shortest seek time first)
- SCAN
- C-SCAN
- C-Look

FCFS :

Starting Point = 53

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

Range : 0 – 199



Total head movement = {(183-53) + (183-37) + (122-37) + (122-14) + (124-14) + (124-65) + (67-65)}

= 640

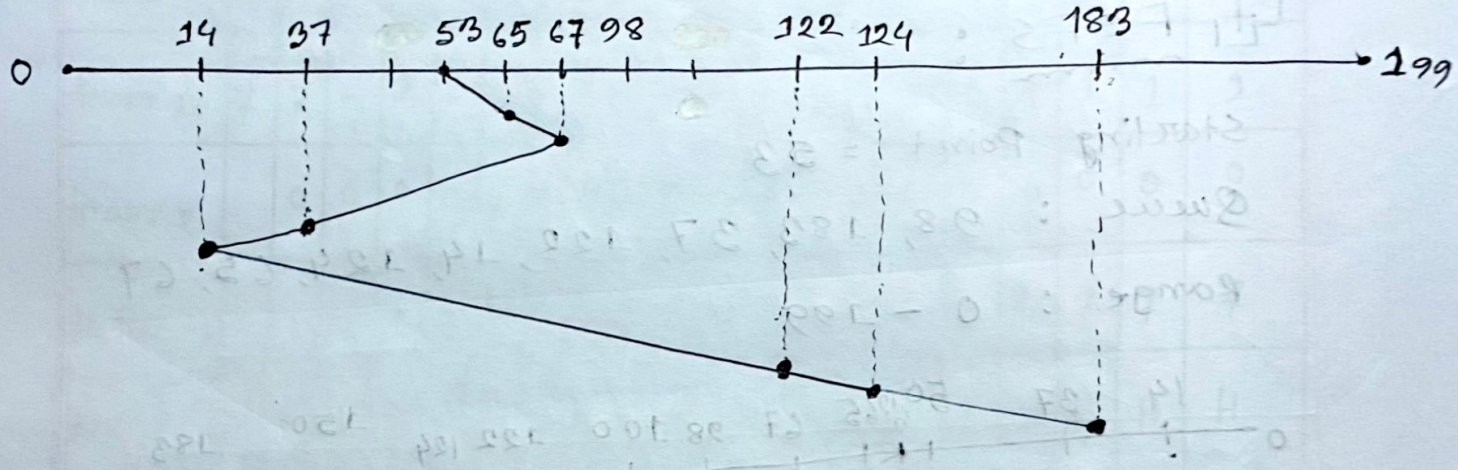
SSTF :

→ Starting Point এর দুপাশের compare করে minimum Point distance এর দিকে যাবে।

Starting Point : 53 , Range : 0 - 199

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

one head movement need 0.2 msec.



Total head movement :

$$\{(67-53)+(67-14)+(183+14)\}$$

$$= 236$$

Time for total movement $= (236 \times 0.2)$

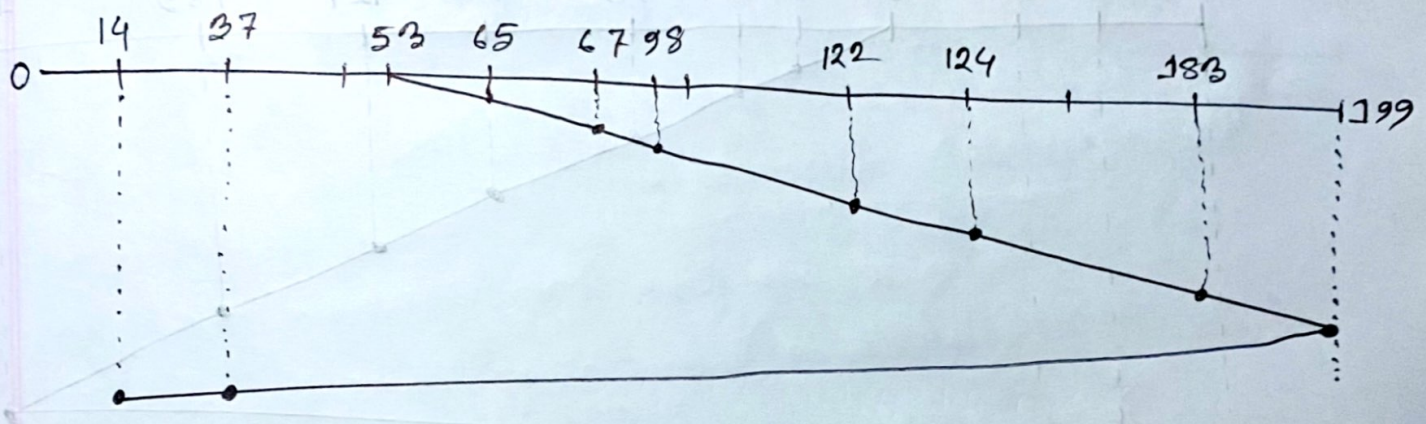
$$= 47.2 \text{ ms}$$

Scan :

Increasing order :

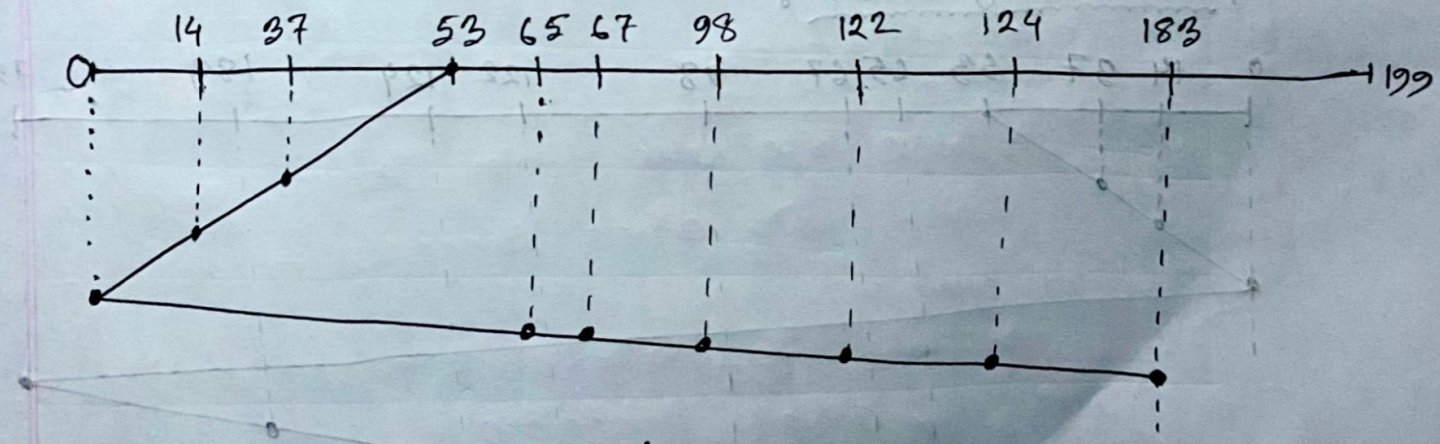
Starting Point : 53
Range : 0 - 199

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



Total Head movement : $\{(199-53)+(199-14)\} = 331$

Decreasing order :



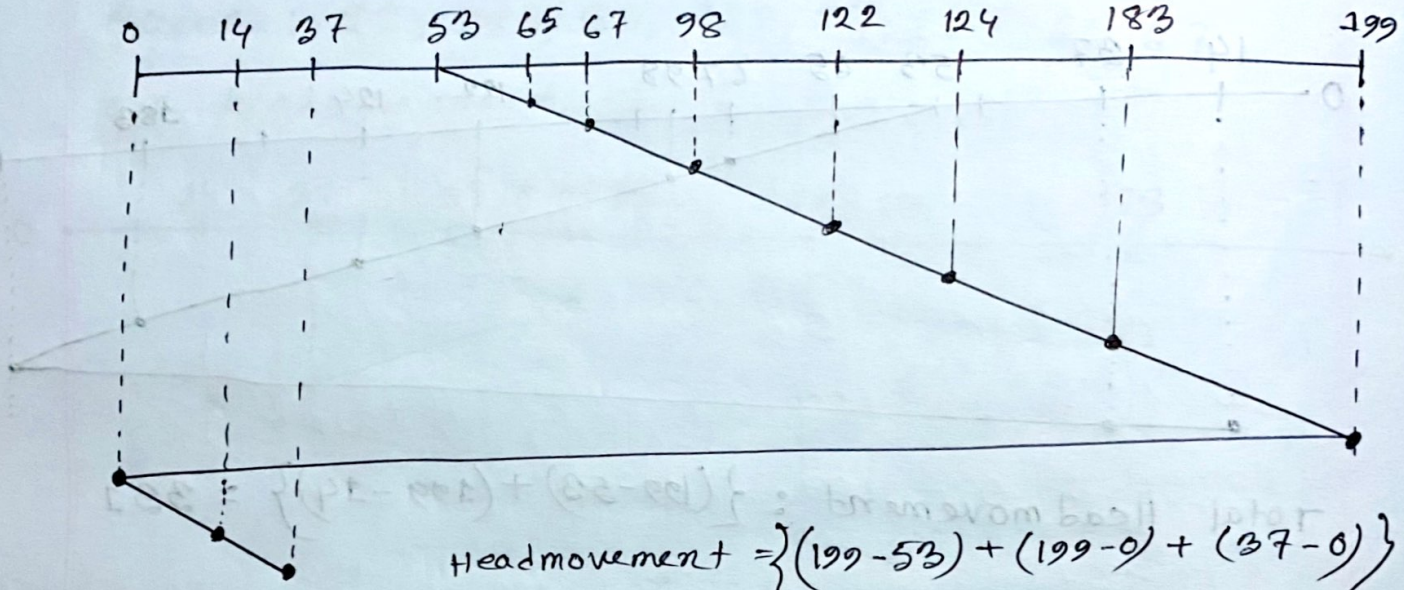
Head movement total $= \{(183-0)+(53-0)\}$

$= 236$

C-scan :

Increasing Order : Starting Point = 53
Range : 0 - 199

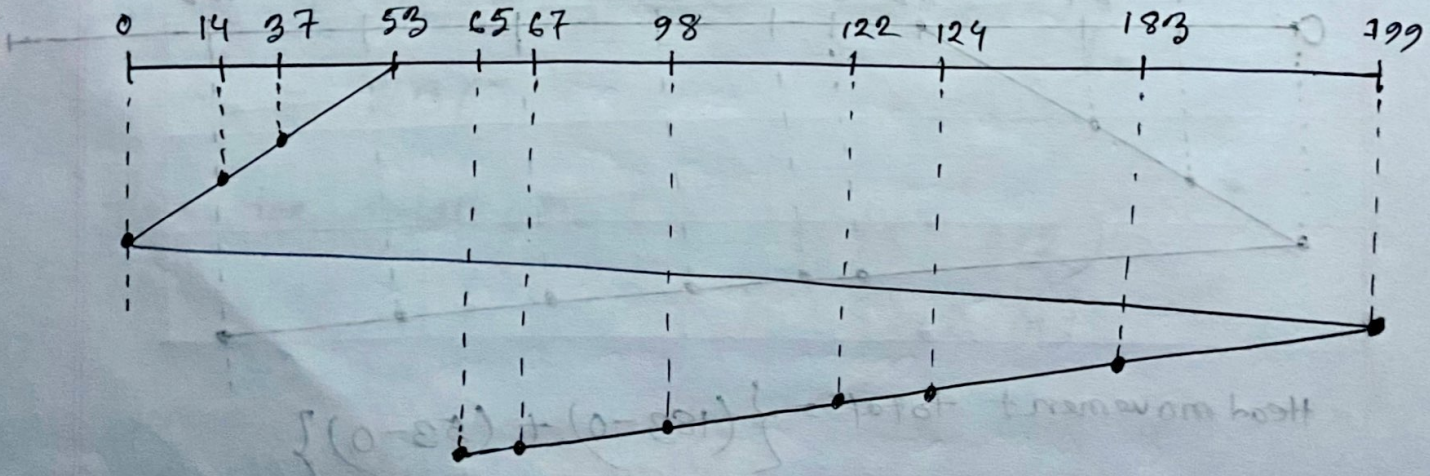
reference : 183, 98, 37, 122, 14, 124, 65, 67



Headmovement = {(199-53) + (199-0) + (37-0)}

= 382

Decreasing Order :



Headmovement = {(53-0) + (199-0) + (199-65)}

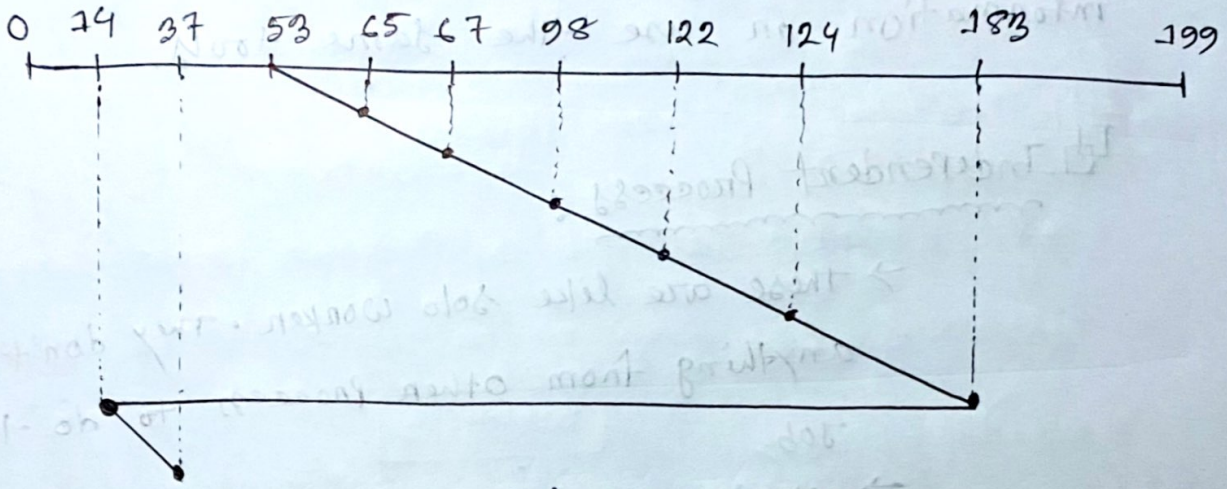
= 386

C-Look :

Increasing order : Starting Point = 53

Range : 0 - 199

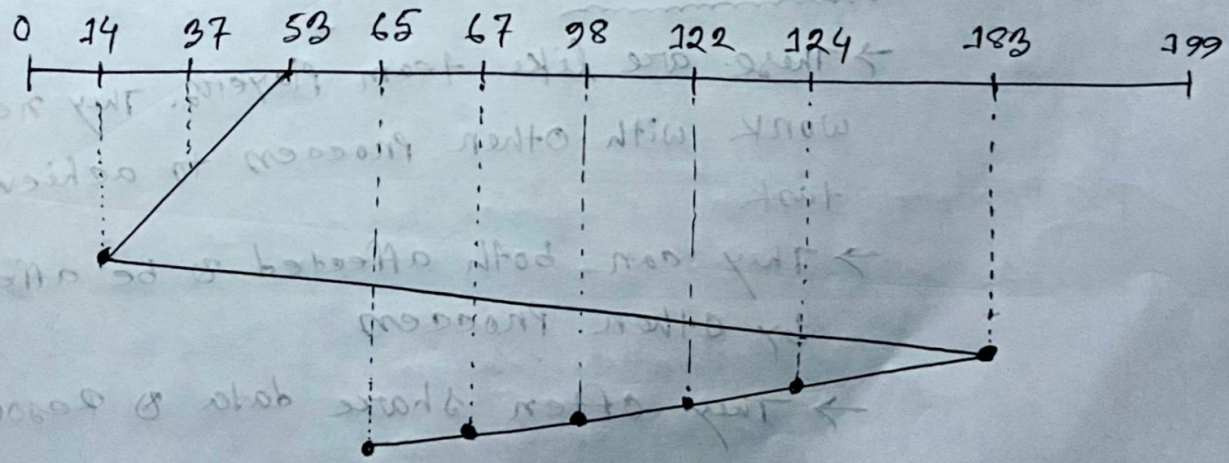
Reference Queue : 98, 183, 37, 122, 14, 124, 65, 67



Headmovement = {(183-53) + (183-14) + (37-14)}

= 322

Decreasing order :



Total Traverse = {(53-14) + (183-14) + (183-65)}

= 326

Process Synchronization

⇒ Process synchronization is a set of rules that helps different programs (called "process") work together nicely when they need to share the same information or use the same tools.

Independent Process:

→ These are like solo workers. They don't need anything from other process to do their job

→ Their execution doesn't affect or get affected by any other process running in the system

Co-operating Process:

→ These are like team players. They need to work with other process to achieve a task

→ They can both affected & be affected by other process

→ They often share data & Resources.

Example :

Synchronized way :

counter = 5 → 6 → update

```
P1 ( )
{
  read (counter);
  counter ++;
  write (counter);
}
```

```
P2 ( )
{
  read (counter)
  counter ++;
  write (counter)
}
```

P1 → 6

P2 → 7

Asynchronized way : occur for sharable resources → race condition

counter = 5

```
P1 ( )
{
  read (counter); ← 5   (PCB store)
  → Preemption   (context switch)
  counter ++;
  write (counter);
}
```

```
P2 ( )
{
  read (counter); → 5
  counter ++; → 6
  write (counter);
}
```

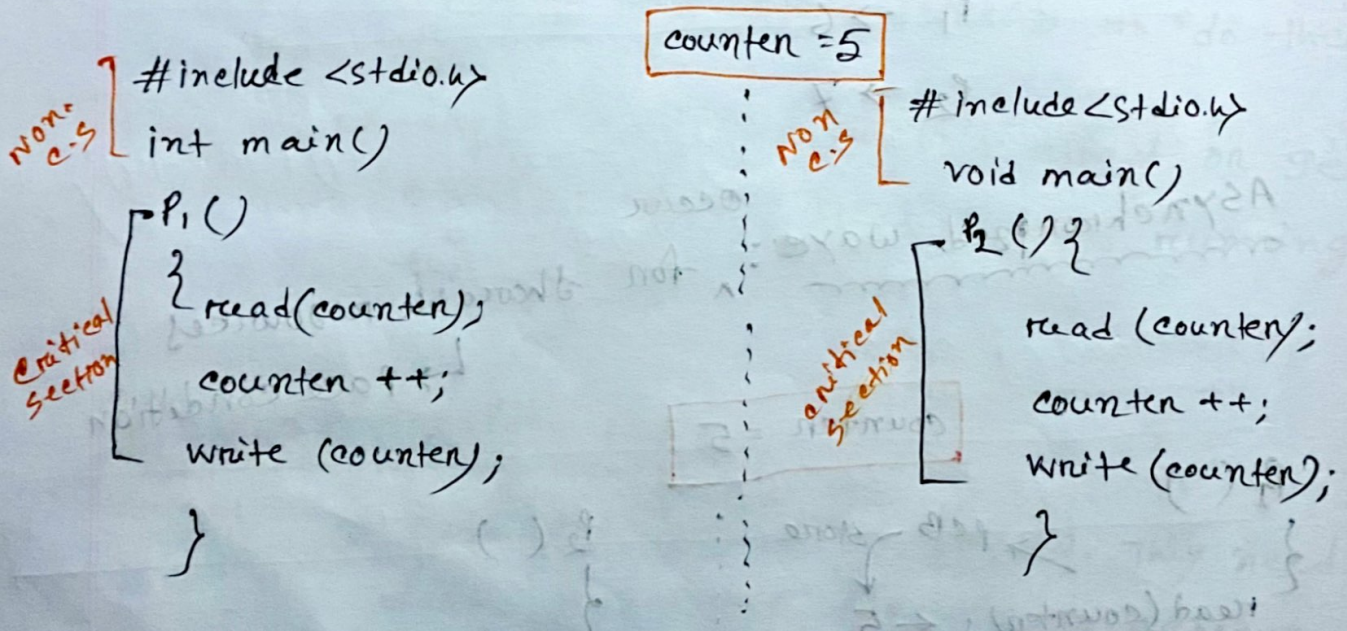
P2 = 6

P1 = 6

Race condition : A race condition occurs when two or more Process can access shared data and they try to change it at the same time.

Critical section Problem :

→ A specific part of a program where shared resources (like data on hardware) are accessed.



→ critical section ↳ direct Execution হয় না।
↳ আগের ধাপে

→ Entry section
→ Critical section
→ Exit section

Critical Section & Synchronization & techniques:

Three condition:

- Mutual exclusion
- Progress
- Bounded wait

Mutual exclusion: only one process can be in the critical section at a time.

Progress: If no process is currently use critical section but some process want to use it, only the process that are not doing unrelated work can help decide who goes next.

Bounded waiting: There must be a limit on how a process has to wait to enter the critical section.

Common Synchronization Problem:

- Race condition
- Deadlock
- Starvation

[+] Semaphore : It is an integer variable that is used to solve the critical problem by using two atomic operations "wait" & "signal" that are used in process synchronization.

Entry section → wait

critical section → full execution

Exit section → signal

S = 1 0 1

```
wait(s)
{
while(s<=0);
  s=s-1;
}
```

```
signal(s);
{
  s=s+1;
}
```

```
do{
   wait(s)
   critical section
   signal(s)
  }
while(true)
```

Producer - consumer Problem :

Producer : Add item to a buffer

consumer : Removes item from the buffer

Synchronization is need to Ensure :

→ The Producer doesn't items add when the buffer is full

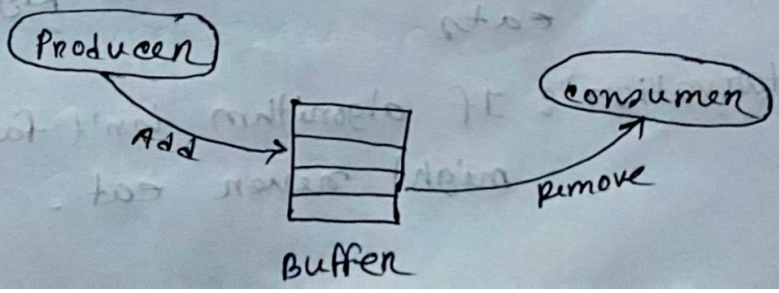
→ The consumer doesn't remove item when the buffer is empty.

Initialize :

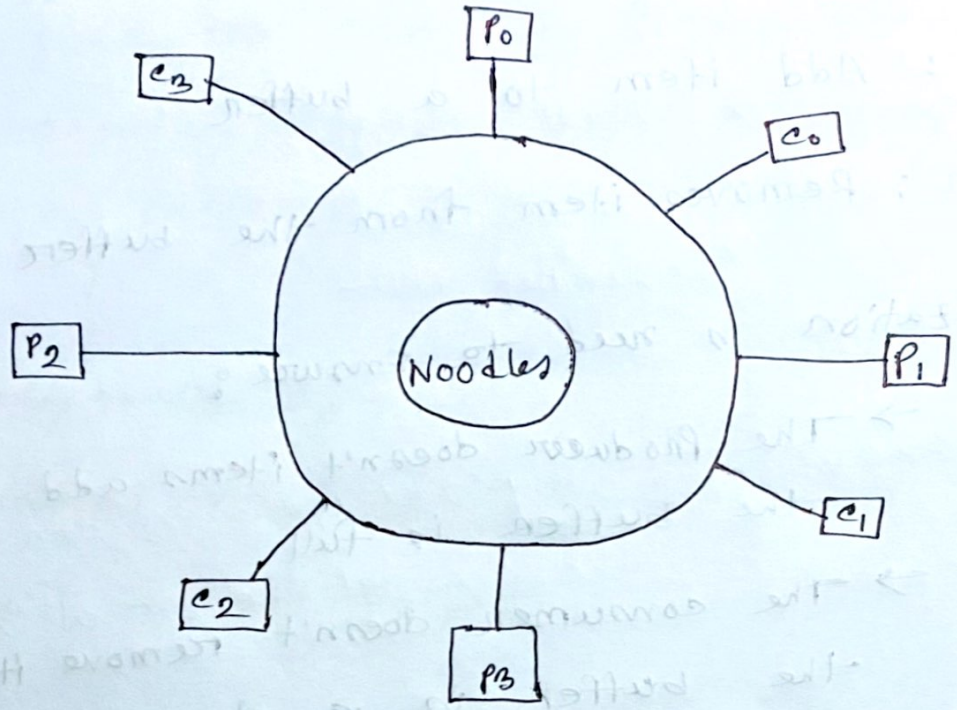
```
buffer = []
max_size = 5
```

```
Producer(){
    If buffer is not full:
        item = create_item()
        buffer.Append(item)
    else:
        wait()
}
```

```
consumer(){
    If buffer is full:
        item = buffer.Pop(0)
    else:
        wait()
}
```



Dining Philosopher Problem :



C_0 = chopstick

P_0 = Philosopher

→ 4 philosopher

→ need 8 chopstic to eat Noodles

→ 4 chopstick exist in Dining

→ Philosopher will thinking & Eating.

Challenges :

(i) Deadlock : If every philosopher picks up the chopstick on their left at the same time, no one can pick up the chopstick on their Right, so no one can eats

(ii) Starvation : If algorithm isn't fair, some philosopher might never eat.

solutions:

→ Pick chopstick only if both are available (left, Right)

→ If both not available then wait

→ After eating, put down both forks to allow others to eat.

Pseudocode :

```
void philosopher (noodels, chopstick)
{ while (true)
  { thinking ();
    wait (chopstic [i]);   # left pick
    wait (chopstic [(i+1) % 4]); # Right pick  > chopstick
    eat ();
    signal (chopstic [i]);   # Putdow left
    signal (chopstic [(i+1) % 4]); # Putdow Right  > chopstick
  }
}
```