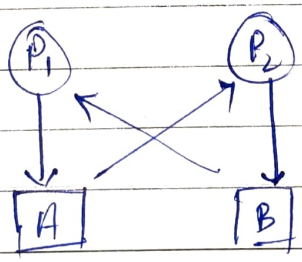
22/11/23

Date
Page No.

Deadlock

- It is defined as permanent blocking of a set of process that either compete for system resources or communicate with each other.
- A set of process is deadlock, when each process in the set is blocked awaiting an event that can only be triggered by another blocked process in the set.

P_1	P_2
Get A	Get B
Get B	Get A
Release A	Release B
Release B	Release A



Reusable

P_1	P_2
Req(D)	Req(T)
Lock(D)	Lock(T)
Req(T)	Req(D)
Lock(T)	Lock(D)
Unlock(D)	Unlock(T)
Unlock(T)	Unlock(D)

Consumable

P_1	P_2
Receive(M)	Rc(M1)
Send(M1)	Send(M)

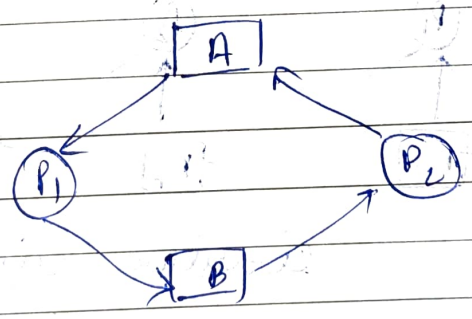
Date
Page No.

- There are 2 types of resources:
(i) Reusable resources
(ii) Consumable "

Ex of (i) are: Processors, I/O channels, main & secondary memory, devices and data structures.

Ex of (ii) are: interrupts, signals, messages and information in I/O buffers.

P_1	P_2
Get A	Get B
Get B	Get A
Release A	Release B
Release B	Release A

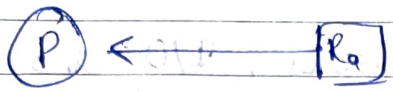


- The resource allocation graph is a directed graph that depicts a state of system resources & process with each process and each resource represented by a node.

Date
Page No.

1) 

Resource is requested.

2) 

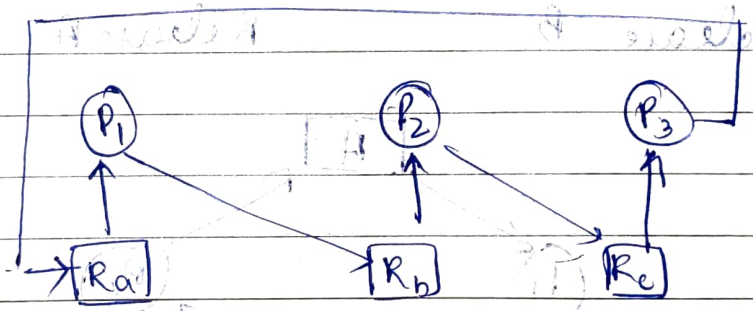
Resource is acquired or held.

Conditions

1) Mutual Exclusion
2) No preemption.
3) Hold & wait
* 4) Circular wait

] Necessary

Sure Existence



circular wait

(i) Mutual Exclusion:
Only one process may use a resource at a time. No process may access a resource unit that has been allocated to another process.

(ii) No preemption:
No resource can be forcibly removed from a process holding it.

(iii) Hold & wait:
A process may hold allocated resources while awaiting assignment of other resources.

(iv) Circular wait:
It is the closed chain of processes such that each process atleast holds one's resource needed by next process in the chain.

Strategies

1) Deadlock Prevention
2) " Avoidance
3) " Detection & Recovery
4) " Ignorance

- There is no single effective strategy that can deal with all types of deadlock but few approaches are common:

(i) Deadlock Prevention:
Disallow one of the three necessary condition for deadlock occurrence or prevent circular wait from happening

(ii) Deadlock avoidance:
Do not grant a resource if this allocat^n might lead to deadlock.

(iii) Deadlock detection & recovery:
Grant resource request when possible but periodically check for presence of deadlock and take action to recover.

(iv) Deadlock ~~Prevention~~ Ignorance:
Assume that deadlock doesn't exist.

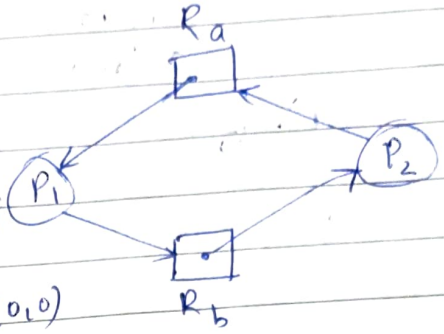
24/4/23

DOS

Date
Page No.

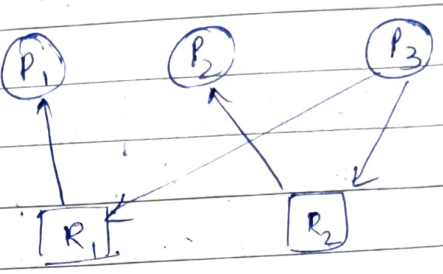
Circular Wait

	Allocate		Req	
	R_a	R_b	R_a	R_b
P_1	1	0	0	1
P_2	0	1	1	0



Availability = (0,0)

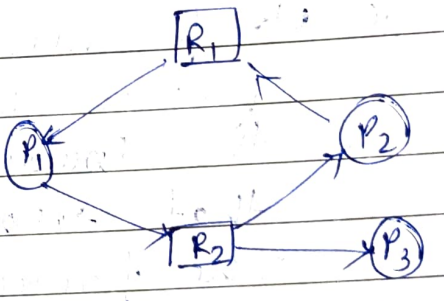
	Allocation		Req.	
	R_1	R_2	R_1	R_2
P_1	1	0	0	0
P_2	0	1	0	0
P_3	0	0	1	1



Availibity = (0,0)
+ 1,0
1,0
+ 0,1
1,1

$P_1 \rightarrow P_2 \rightarrow P_3$
safe seq.

	Allocation		Req.	
	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

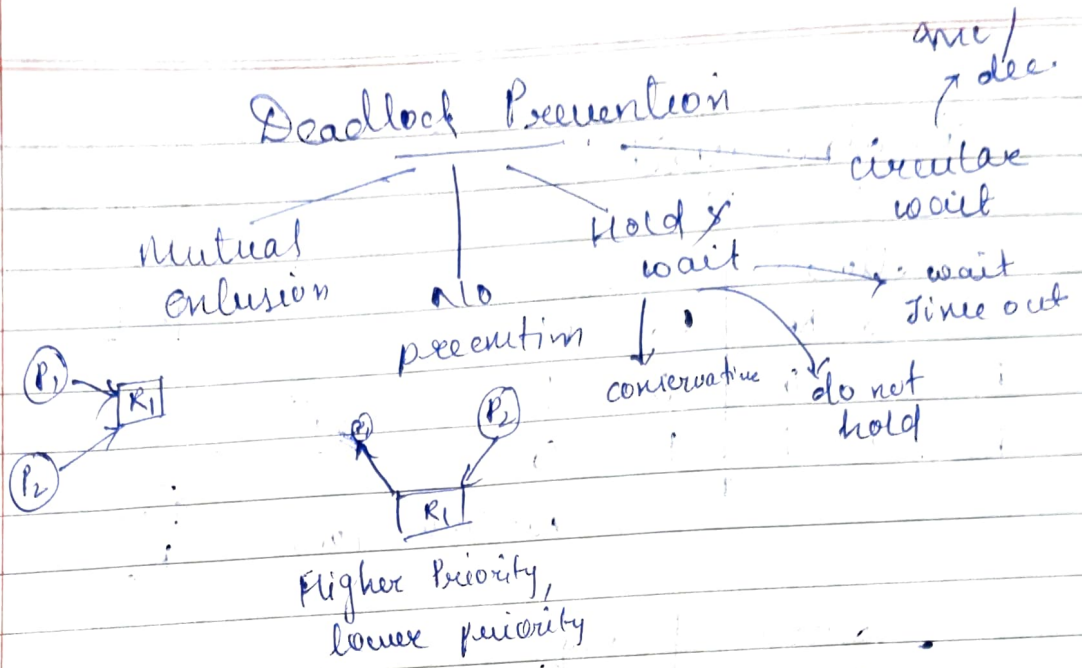


Avail = (0,0)
(0,1)
0,1
1,0
1,1

$P_3 \rightarrow P_1 \rightarrow P_2$
safe seq.

Date
Page No.

Deadlock Prevention



deadlock prevention : we try to fail one of the four necessary 'condit" for the deadlock to ~~to~~ occur!

(i) Mutual exclusion:
We cannot fail / disallow this condition. Some resources such as file may allow multiple accesses for reads but only exclusive acces for write.

(ii) No preemtion: If a higher priority process want some resource, then the lower priority process holding that resource must release it. The processes which are in waiting state must be selected as the next process for holding the resource instead of the process in running state.

(iii) Hold & wait:
(i) Conservative: process is allowed to start execution if & only if

-it has acquired all the resources. However it is not practical and less efficient approach

2) Do not hold: Process will acquire only desired resources but before making any fresh requests it must release all the resources that it is currently holding.

3) Wait timeout: we place a maxm time upto which a process can wait, after the timeout the process must release all the holding resources.

(iv) Circular wait:
This condition can be prevented by defining a linear ordering of resource types. Either the order can be increasing / decreasing.

Deadlock avoidance:-

It allows the 3 neccessary conditions but makes choices to ensure that the deadlock point is never reached. There are 2 approaches:

(i) Do not start a process if its demand might lead to deadlock.

(ii) Do not grant an incremental resource request to a process if this allocatn might lead to deadlock.

28/11/23

Date
Page No.

Deadlock Avoidance.

Banker's Algorithm:

Q)

A = 10
B = 5
C = 7

Process	Allocation			Max Need			Remaining			Available		
	A	B	C	A	B	C	A	B	C	A	B	C
P_1	0	1	0	7	5	3	7	4	3	3	3	2
P_2	2	0	0	3	2	2	1	2	2	+2	0	0
P_3	3	0	2	9	0	2	6	0	0	5	3	2
P_4	2	1	1	4	2	2	2	1	1	+2	1	1
P_5	0	0	2	5	3	3	5	3	1	7	4	3
	7	2	5							+0	0	2
										7	4	5
										+0	1	0
										7	5	5
										3	0	2
										10	5	7

$P_2 \rightarrow P_4 \rightarrow P_5 \rightarrow P_1 \rightarrow P_3$

Q)

A = 8
B = 4
C = 6

Process	Allocation			Max Need			Remaining			Available		
	A	B	C	A	B	C	A	B	C	A	B	C
P_1	1	0	1	4	3	1	3	3	0	3	3	0
P_2	1	1	2	2	1	4	1	0	2	1	0	1
P_3	1	0	3	1	3	3	0	3	0	4	3	1
P_4	2	0	0	5	4	1	3	4	1	1	3	3
	5	1	6							5	3	4
										1	1	2
										6	4	6
										2	0	0
										8	4	6

$P_1 \rightarrow P_3 \rightarrow P_2 \rightarrow P_4$

Q)

R = Resources

Process	Allocation	Max Need	Remaining (Max − Allocation)	Available
P_1	3	8	5	2
P_2	4	9	5	+1
P_3	2	5	3	3
P_4	1	8	2	+2
	10			5
				+3
				8
				+4
				12

$P_4 \rightarrow P_3 \rightarrow P_1 \rightarrow P_2$

↳ safe sequence.

Q)

Process	Allocat^n A	B	C	Max Need A	B	C	Remaining A	B	C
P_1	6	1	2	7	8	3	1	7	1
P_2	1	2	3	1	5	4	0	3	1
P_3	0	4	2	2	6	9	2	2	7
P_4	1	1	1	2	1	3	1	0	2
	8	8	8						

$A = 10$
$B = 9$
$C = 12$

Available A	B	C
2	1	4
1	1	1
3	2	5

$P_4 \rightarrow$
unsafe seq.

Q)

Process	Allocat^n A	B	C	Max Need A	B	C	Remaining A	B	C
P_1	1	0	0	3	2	2	2	2	2
P_2	6	1	2	6	1	3	0	0	1
P_3	2	1	1	3	1	4	1	0	3
P_4	0	0	2	4	2	2	4	2	0
	9	2	5						

$A = 9$
$B = 3$
$C = 6$

Available A	B	C
0	1	1
+ 6	1	2
6	2	3
+ 2	1	1
8	3	4
0	0	2
8	3	6
1	0	0
9	3	6

$P_2 \rightarrow P_3 \rightarrow P_4 \rightarrow P_1$
Safe seq.

Q) 9 tape drives

Process	Allocat^n	Max Need	Remaining	Available
P_1	3	7	4	2
P_2	1	6	5	+ 3
P_3	3	5	2	5
	7			+ 3
				8
				1
				9

$P_3 \rightarrow P_1 \rightarrow P_2$
or
$P_3 \rightarrow P_2 \rightarrow P_1$
$\rightarrow$ Safe seq.

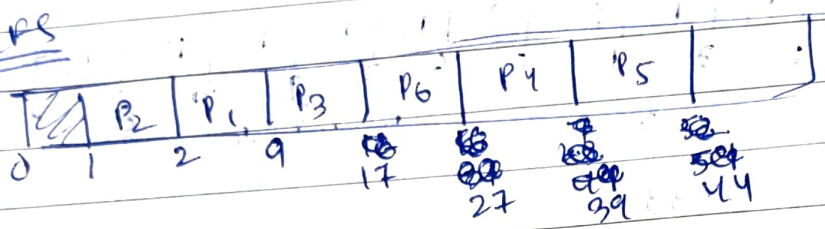
Date
Page No.

FCFS

①

Px	AT	BT	CT	TAT	WT
P_1	2	7	9	7	0
P_2	1	1	2	1	0
P_3	3	8	17	14	6
P_4	7	12	39	32	20
P_5	9	5	44	35	30
P_6	4	10	27	23	13

FCFS

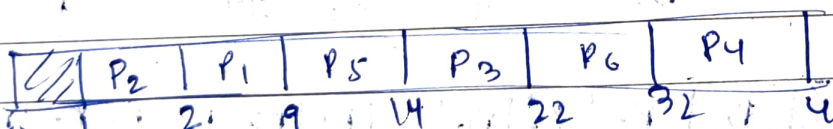


P4 P5 P6

avg TAT = 18.67

SJF

Px	AT	BT	CT	TAT	WT
P_1	2	7	9	7	0
P_2	1	1	2	1	0
P_3	3	8	22	19	11
P_4	7	12	44	37	25
P_5	9	5	14	5	0
P_6	4	10	32	28	18



Avg TAT = 16.166

Avg WT = 9

29/11/23

Date
Page No.

Deadlock Detection & Recovery

$P_3 \rightarrow P_1 \rightarrow P_2$

Q)

Process	Allocation	Remaining	Available
P_1	0 1 1 1 0	0 1 0 0 1	0 0 0 0 1
P_2	1 1 0 0 0	0 0 1 0 1	+ 0 0 0 1 0
P_3	0 0 0 1 0	0 0 0 0 1	0 0 0 1 1
P_4	0 0 0 0 0	1 0 1 0 1	
	2 1 1 2 0		

$R = 2\ 1\ 1\ 2\ 1$

$P_3 \rightarrow$

Deadlock state

four ways to recover deadlock state:

(i) → Abort all process
(ii) Successively preempt process.
(iii) Successively preempt resources.
(iv) Backup and restart.

- Once deadlock is detected, there are few approaches for the system to recover.

(i) Abort all deadlock process.
(ii) Backup each deadlock process to some prev. defined checkpoint and restart all process. The ~~risk~~ risk in this is the original deadlock may reoccur
(iii) Successively abort deadlock process until deadlock no longer exists.
(iv) Successively preempt resources until deadlock no longer exists.

19/12/23

Date
Page No.

Memory Management

Requirements

1) Relocation
2) Protection
3) Sharing
4) Logical Org.
5) Physical Org.

1) **Relocation** :- Once a program is loaded in the MM and swapped out, it cannot be specified at what location, it will be swapped in. Therefore, the OS needs to relocate the process to a different area of memory.
- The processor h/w and OS must be able to translate the memory references found in code of program.

2) **Protection** :-
- Every process should be protected against unwanted ~~inf~~ interference by other process whether accidental / intentional.

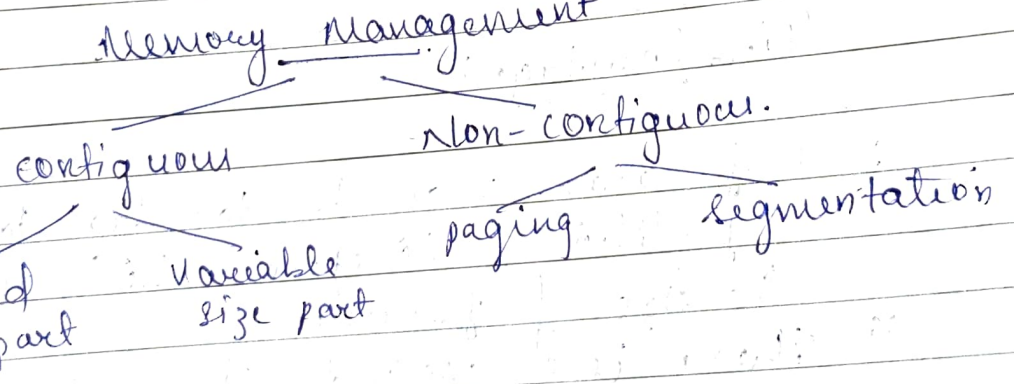
3) **Sharing**:
- If many ~~prog.~~ processes are executing the same prog. it is benificial to allow each process to access the same copy of the program instead of having a seperate copy.

4) Logical Org:

- Most prog. are organised into modules which contain data that may be modified. These modules can be written and compiled independently with all references from one module to another.
- The tool that is used is segmentation.

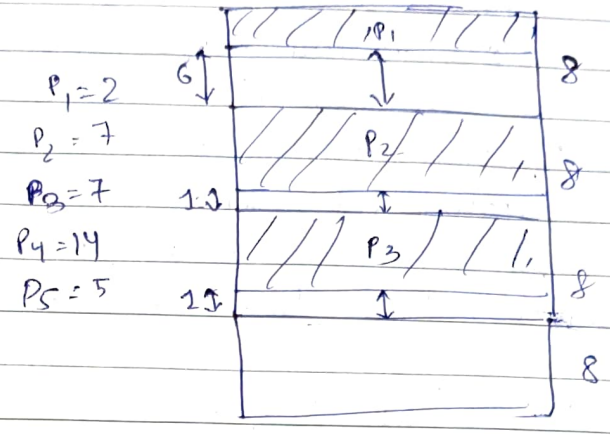
5) Physical Org:

- The MM may be insufficient for a prog. In this case overlaying can be used where prog. and data are organised in such a way that various modules can be assigned the same region of memory.
- There is a main prog., that is responsible for switching the modules, in & out as needed.



Date
Page No.

Fixed-size part.



- The OS occupies some fixed portion of the ram and the rest of main memory is available for use by multiple processes.
- Here the memory is partitioned into regions with fixed boundaries.
- It is a contiguous memory management scheme where we cannot combine the regions.

Disadvantages:

(i) A program may be too big to fit into a partition. Therefore this scheme limits the size of process.

(ii) There is a wasted space internal to a partition due to the fact, that the block of data loaded is smaller than the partition which is known as internal fragmentation.

(iii) Even after having enough space, we cannot accomodate a new process which is known as external fragmentation.

(iv) The no. of processes that can be loaded in MM are limited, i.e. the degree of multiprogramming is limited.

Fixed Size — Unequal partitions

- This solves the problems upto some extent but still consist of above disadvantages.

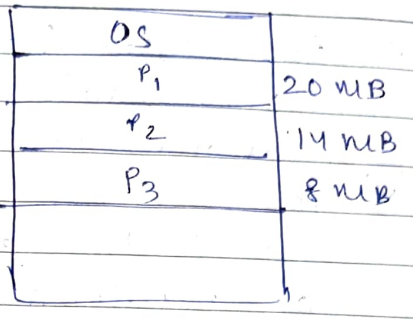
- An unequal size partitions a scheduling queue is needed for each partition to hold swapped out processes meant for that partition, but it is not benificial as even after having empty space and smaller processes could have been assigned to it. So, we maintain a single queue for all the partitions and when it is the time to load a process into MM, the smallest available partition that will hold the process is selected.

20/12/23

Date
Page No.

Dynamic/variable Partitioning

20, 14, 8.

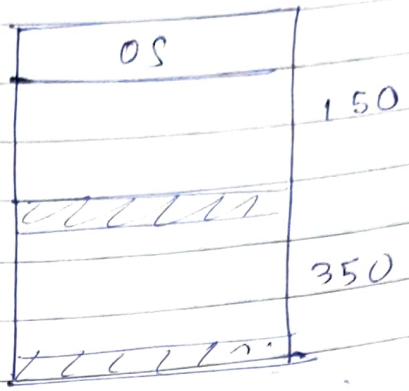


- Here, the space is allocated to the process dynamically on their request. There are not static partitions made before hand.
- It solves the problem of fixed partitioning but still has few disadv:
  (i) External fragmentation
  (ii) Allocation & reallocation is complex.

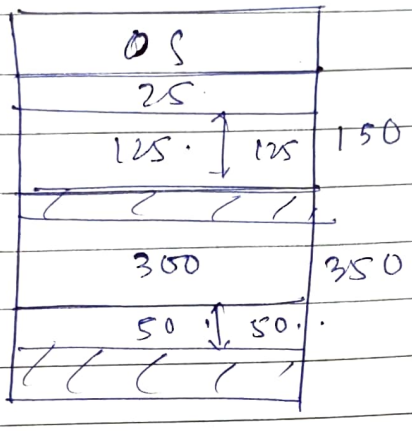
- Compaction solves the problem of internal fragmentation by combining the free space; but it is not feasible as we need to copy million of bits in order to achieve this.

1) First fit
2) Next fit
3) Best fit
4) Worst fit

300, 25, 125, 50.



first fit:



Buddy system:

- In this system memory is allocated based on the powers of 2.
- The memory is further split in powers of 2 if any process has size less than the current memory available.

3/1/24

Date
Page No.

Virtual Memory

Replacement process

FIFO | LRU | optimal

FIFO

Q. 1, 3, 0, 3, 5, 6, 3 → 3 frame memory

3			0	0	0	0	3
2		3	3	3	3	6	6
1	1	1	1	1	5	5	5
	M	M	M	H	M	M	M

Miss Ratio = $\frac{6}{7} \times 100$

Hit ratio = $\frac{1}{7} \times 100$

Q. 4, 7, 6, 1, 7, 6, 1, 2, 7, 2

3			6	6	6	6	6	7	7
2		7	7	7	7	7	7	2	2
1	4	4	4	1	1	1	1	1	1
	M	M	M	M	H	H	H	M	H

Miss - 6

Hit - 4

Date
Page No.

a) 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 1, 2, 0.

3			1	1	1	1	0	0	0	3	3	3	3	2	2
2		0	0	0	0	3	3	3	2	2	2	2	1	1	1
1	7	7	7	2	2	2	2	4	4	4	0	0	0	0	0
	m	m	m	m	h	m	m	m	m	m	m	h	m	m	h

Hit - 3
Miss - 12

Optimal

a) 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

			2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
		1	1	1	1	1	4	4	4	4	4	4	4	4	4	4	1
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	7	7	7	7	3	3	3	3	3	3	3	3	3	3	3	3	3
m	m	m	m	h	m	h	m	h	h	h	h	h	h	m			

2	2	2	2	2	2
1	1	1	1	1	1
0	0	0	0	0	0
3	3	3	7	7	7
h	h	h	m	h	h

3 frames.

9) 6, 1, 1, 2, 0, 3, 4, 6, 0, 2, 1, 2, 1, 2, 0, 3, 1, 4, 0

3				2	2	3	4	4	4	4	1	1	1
2		1	1	1	0	0	0	0	0	0	0	0	0
1	6	6	6	6	6	6	6	6	6	2	2	2	2
	M	M	H	M	M	M	M	H	H	M	M	H	H

3	1	1	1	1	1	1
2	0	0	0	0	0	0
1	2	2	3	3	4	4
	H	H	M	H	M	H

9 → hit
10 → miss

LRU

8) 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

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

1	2	2	2	2	2	2	2
3	1	1	1	1	1	1	1
2	0	0	0	0	0	0	0
1	3	3	3	3	7	7	7
	M	H	H	H	M	H	H

Date
Page No.

5/1/24

9) 6, 1, 1, 2, 0, 3, 4, 6, 0, 2, 1, 2, 1, 2, 0, 3, 2, 1, 4, 0

3				2	2	2	4	4	4	2	2	2	2	2	2
2		1	1	1	x	3	3	3	0	0	0	0	0	0	0
1	6	6	6	6	0	0	0	6	6	6	1	1	1	1	1
	M	M	H	M	M	M	M	M	M	M	M	H	H	H	H

3	2	2	2	2	0
2	0	0	1	1	1
1	3	3	3	4	4
	M	H	M	M	M

6 - bit
14 - bits
} x
7 - bit
13 - bits

5/1/24

Date
Page No.

Frame Allocation

- Some specific frames are allocated for the process. frame allocat" technique tell us how many frames are allocated to the process.
- Min requirement is the pages required to execute 1 inst" & max requirement is loading the whole process into MM.
- The frame allocat" should lie bet" min and max requirement.

$$min \leq frames \leq max$$

Frame allocation ways

- Equal
- weighted
- Dynamic / Priority Based.

- There are three ways for the allocat" of frames:

(i) Equal frame allocat": Here we just divide the frames with the total no. of processes without considering the weights of each process.
This leads to inequality (i) as some processes may allocate all of their pages
(ii) It does not consider the priority of the process.

(ii) Weighted frame allocat":
Here the weights of each process (no. of pages) are taken into consideration and then respective frames are allocated.

(iii) Dynamic/Priority Based:
→ The frames are allocated acc. to the demand at the runtime if a higher priority process wants to allocate its frames, then the frames of lower priority process are swapped out.

Disk Scheduling

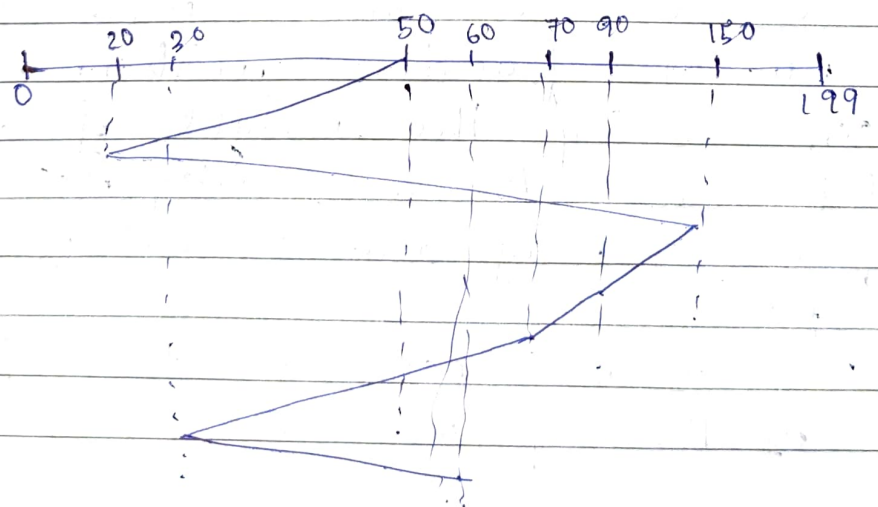
FCFS SSTF SCAN C-SCAN LOOK C-LOOK

FCFS.

Q 20, 150, 90, 70, 30, 60

No. of Tracks (0-199) = 200

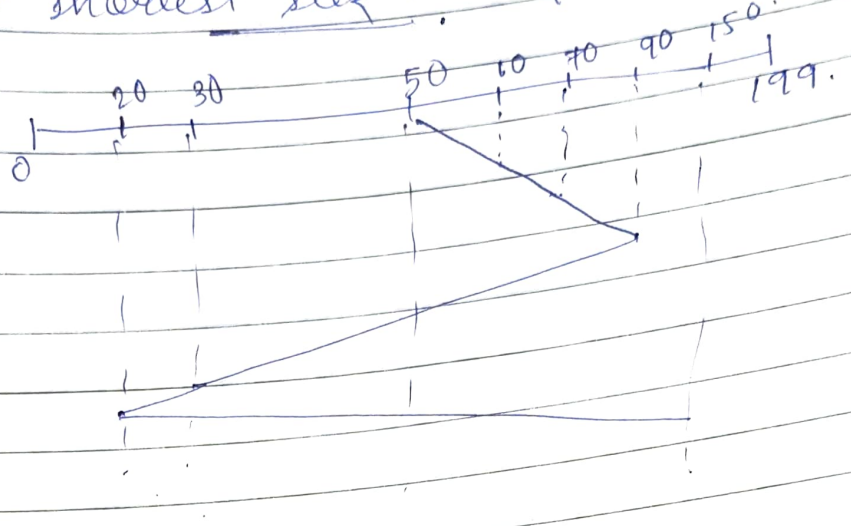
current R/W head = 50.



Seek time:

$(50-20)+(150-20)+(150-30)+(60-30)$

SSTF → shortest seek time first.



Seek time:

$(90-50)+(90-20)+(150-20)$