

Date ___ / ___ / ___



Operating System

Notes

— by Riti Kumari

Please Share And Help others ::

Operating System

dec-10

Syllabus of Operating System

1. Basic Introduction → types of OS

Process diagram ✓ (Main)
System call.

**

2. Process scheduling (CPU scheduling)

FIFO

SJF

Round Robin

3. Process synchronization → Semaphore ✓

4. Deadlocks and threads → Banker's algorithm

5. Memory management → Paging Virtual memory
Segmentation
Fragmentation
Page replacement algo ✓

6. Disk scheduling ← SCAN
CS, CSCAN
FCFS

7. Unix commands - ls

→ mkdir

→ cd

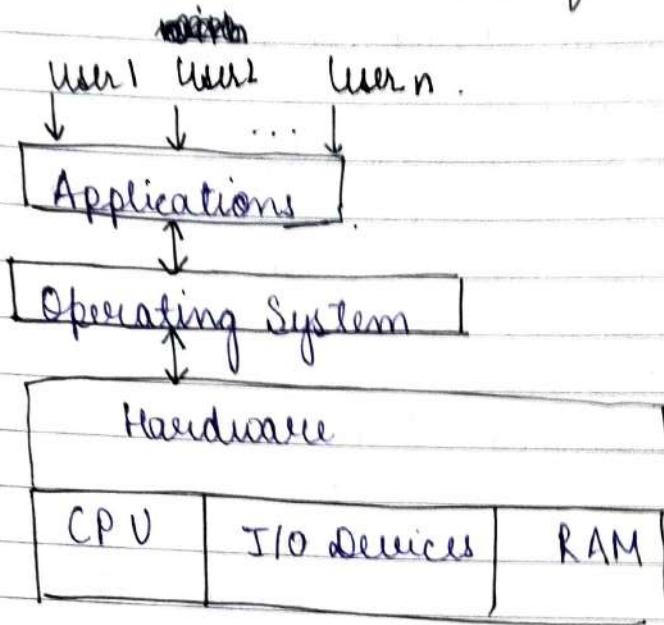
→ chmod

→ Open system call

8. File management & Security ← sequential
Random
linked

L-1.1

Introduction to OS and its function



Operating System :- It is a system software which works as an interface between users and hardware.

Job (process) • User

↓
Operating System (interface)

↓
Hardware

Eg: Windows, Macintosh (Mac), Linux

Primary goal → To provide convenience to user.

Throughput → no of tasks executed per unit time. Eg - Linux

Functionality of Operating System

- 1) Resource management (kis user ko kitne particular time ke liye hardware ko provide kروانا ہے).

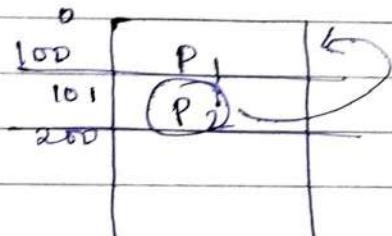
2. Process management - Managing & executing multiple process at a time. we use CPU scheduling

program ~~execute~~ process

3. Storage management \Rightarrow How to store the data permanently using file system.
Eg - Hardisk.

4. Memory management (RAM) \Rightarrow RAM is limited in system. Every process before execution comes to RAM. Allocation & deallocation takes place.

5. Security and privacy - (Password protection)
Windows uses two security protocol.



It provides security b/w the process also.

Windows (cmd)

linux (Terminal)

OS works only through system calls.
(Read, open, write)



L-1.2

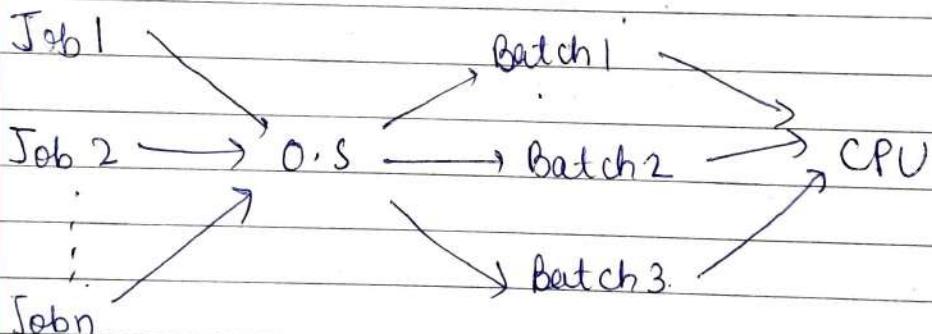
Batch Operating System

Types of Operating System

- 1) Batch OS.
- 2) Multiprogramming OS.
- 3) Multitasking OS / Time sharing
- 4) Real time OS
- 5) Distributed OS
- 6) Clustered OS
- 7) Embedded OS.

1) Batch OS

Similar (batch) kind of jobs.

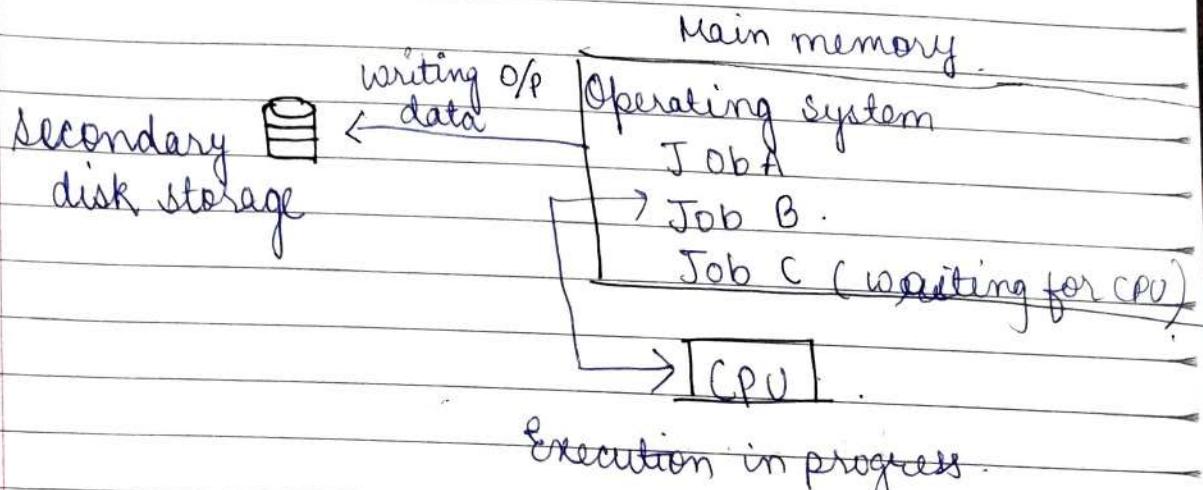


Same type of Jobs batch together and execute at a time.

Demerit: It processes one job at a time till the time CPU remains idle.

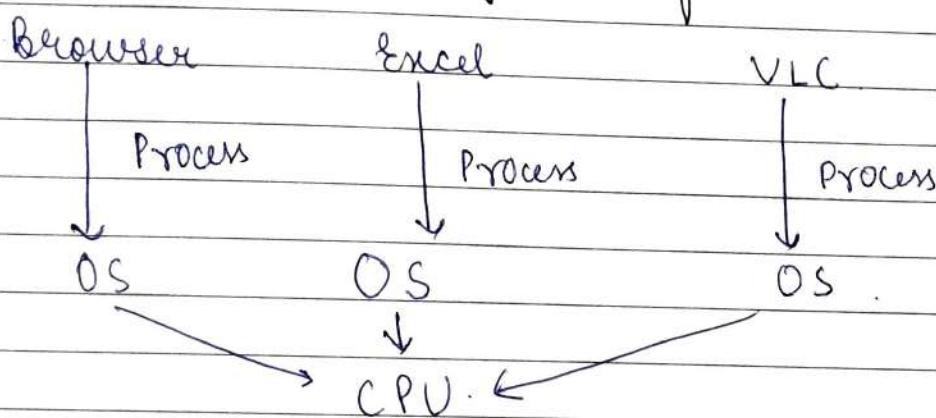


L-1.3 (Multiprogramming & Multitasking system)



Execution in progress.

Multiprogramming OS / Time sharing



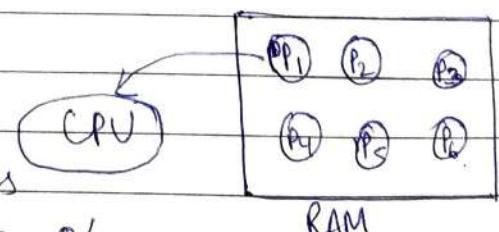
Multitasking OS

Multiprogramming OS.

Non preemptive

CPU executes a process completely until or unless process demands any I/O or O/P operation.

Idleness should be removed

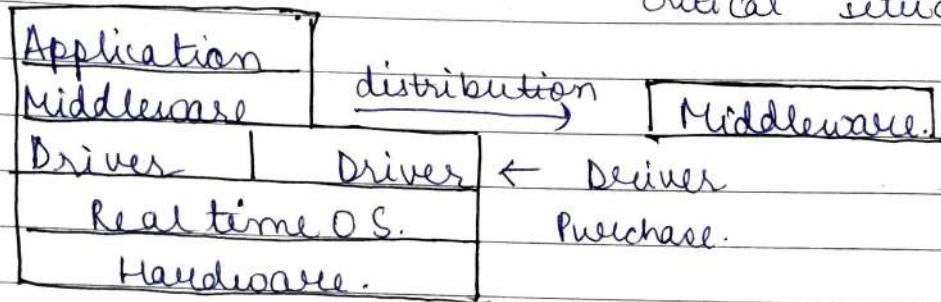


Multitasking / time sharing - Provides particular time for every process. If its complete its great but if not it moves to other process P₂ & schedule P₁ for further.

Response time is reduced for every process. Idleness is not there for CPU.

Dec 1.4 Types of OS (Real time)

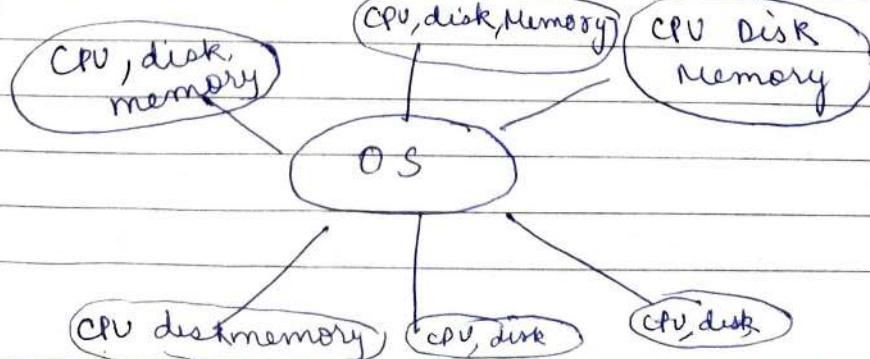
Real time OS → Hard (Restriction on time is more)
soft Eg - In gaming no critical situation



Immediate output

- i) No delays

Distributed → Processing environment is distributed all over the world. Every process has its own environment.

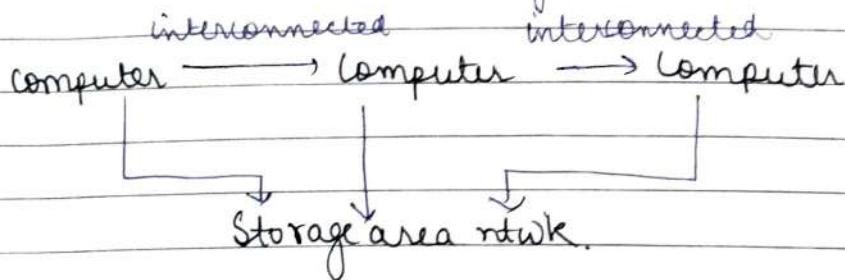


It uses multiple central processors to serve multiple real-time applications & multiple users using telephone wires.

Clustered OS. → ~~processors~~ Clustered OS share storage and are closely linked via LAN or a faster interconnection.

Group of connected computers

* fault tolerance, scalability

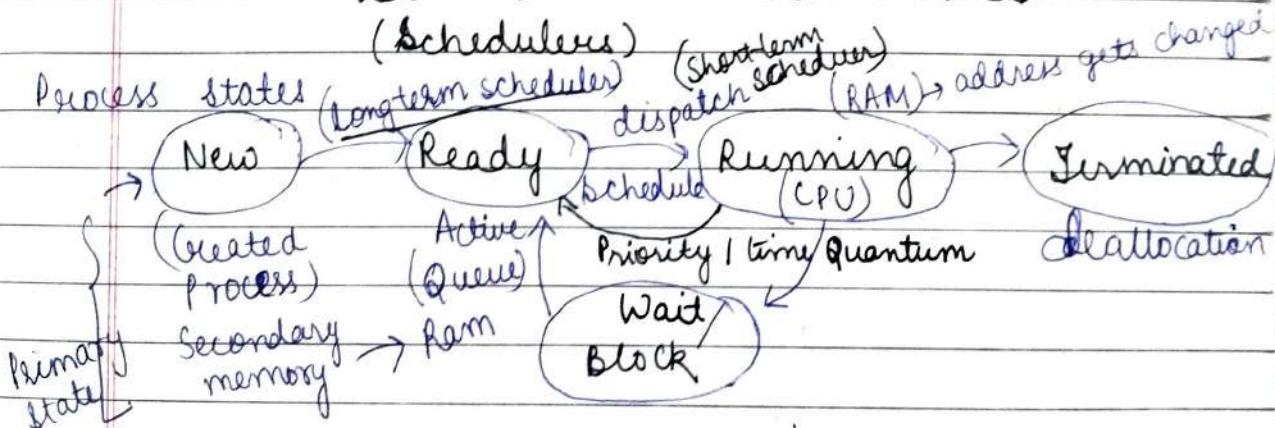


Embedded OS - It is a special purpose computer designed to perform one or few dedicated functions with real time computing. Works on a fixed functionality.

CPU

I/O → Processor → O/P

Dec-1.5 Process states in OS



long term scheduler - bring more & more process in ready state.

LTS - long term scheduler

STS - short term scheduler.

Non preemptive.

Preemptive.

Non preemptive - Running process isn't stopped at middle.

Preemptive - Running process stopped at middle.



Syllabus of OS.

1. Basic Introduction : what is O/S, Goals, need, system calls, diffⁿ OS.
2. Process M/g : what is process, Algo - FIFO, SJF, Preemptive, Non preemptive, Round Robin
3. Deadlock - Banker Algo
4. Concurrency control - Semaphore, msg passing, race condⁿ.
5. Memory - Paging, Segmentation, Fragmentation, virtual memory, Page Replacement.
6. Disk M/g - C-cook, Algo, FCFS, SFCF, look
7. File mlg : NIFS, FAT 32, Seg., Random
8. Unix System : commands.
9. Case study : MSDOS
10. Security - Attacks, firewalls, exceptions



L-1.1

Operating system acts as an interface between user and hardware. If there is no OS user has to write a program to interact with hardware.

Primary goal - Its primary goal is to provide convenience to the user.

user user user user



Applications



Operating System



Hardware

CPU I/O devices Ram

Throughput - No of tasks executed per unit time.

Eg - Linux

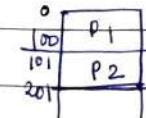
Functionalities of OS.

1) Resource management - In case of multi-user managing, OS acts as key to provide hardware among the users for a given time. When many users are sending request at a time so we use OS so that there is no load on system.

2. Storage/Process management \rightarrow Performing multiple process at a time, OS uses CPU scheduling algorithms for processing & executing in a proper & efficient way
3. Storage management (Hard disk) - How to store data in hardisk using file system management is done by OS.
4. Memory management (RAM) - In RAM we have limited no of size. Every process which gets executed goes in RAM there is allocation & deallocation done by OS in RAM after the process is done.

5. Security & privacy
It provides security & privacy between the process such that it doesn't get blocked. Password protection

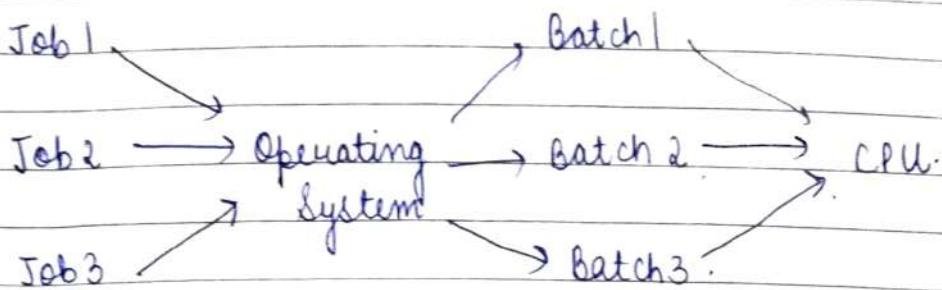
Windows (cmd)	}	System calls
Linux (terminal)		



1.2. Types of OS

- 1) Batch OS
- 2) Multiprogramming
- 3) Multitasking
- 4) Real time OS
- 5) distributed
- 6) Clustered
- 7) Embedded

1) Batch OS

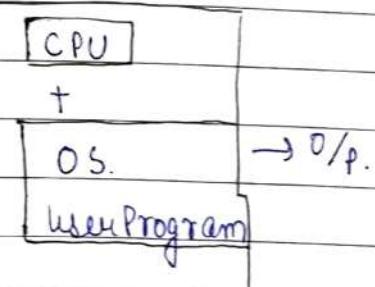


~~before~~

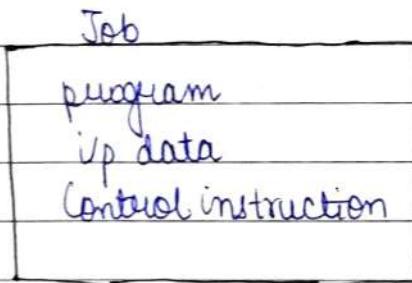
before batch OS.. we used mainframe computers

- 1) Common I/p & O/p devices

were card readers & I/p
tape drives.



- 2) User prepare a job which consist of the program I/p data & control instructions.

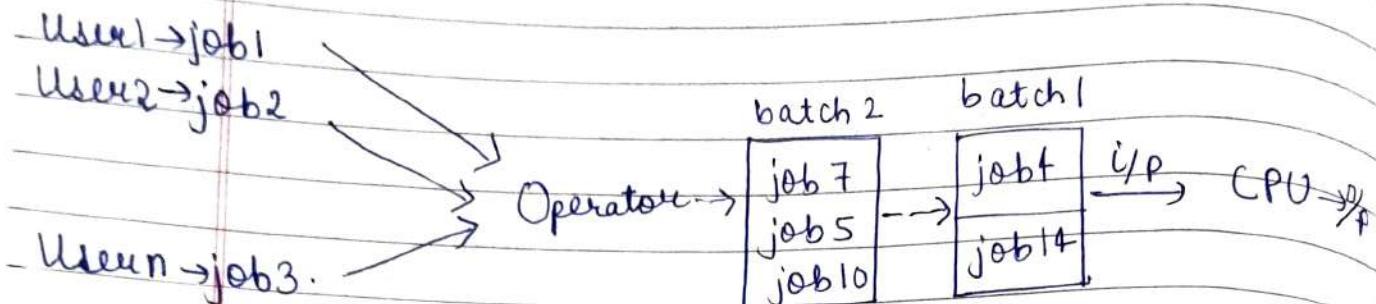


- 3) I/p ~~Punch~~ O/p (punch card)

Major problem was.

- 1) Speed mismatch (user & CPU)
- 2) Less memory
- 3) Every job has diffⁿ req^m

1. Batch processing (e.g.: payroll systems)



- 1) Jobs with similar needs are batched together and executed through the processor as a group.
- 2) Operator sorts job as a deck of punch cards in a batch with similar needs.
e.g. - FORTAN batch, COBOL batch etc.
- 3) first jobs → batch (with same req) → CPU

Advantages

- 1) In batch job execute one after another saving time for activities like loading compiler.
- 2) During a batch execution no manual intervention is needed.

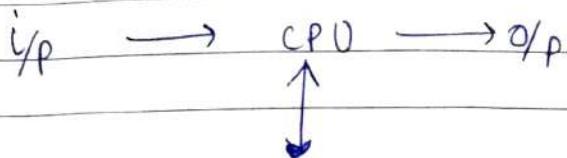
Disadvantage

- 1) Memory limitation
- 2) Interrelation of I/P & O/P devices directly with CPU.
- 3) CPU remains idle during loading & unloading



Spooling (Simultaneous peripheral device)

↑
I/p or O/p



temporarily process is stored in secondary storage.

Advantages

- 1) Increase the system performance.
- 2) Spooling resolve the problem of speed mismatch of diffⁿ. device
- 3) % of one job is overlapped with the computation of other jobs.
- 4) Spooling use the disk as a huge buffer.

Spooling

uses the hard disk as a large spool
(spool is a temporary storage area in hard disk)

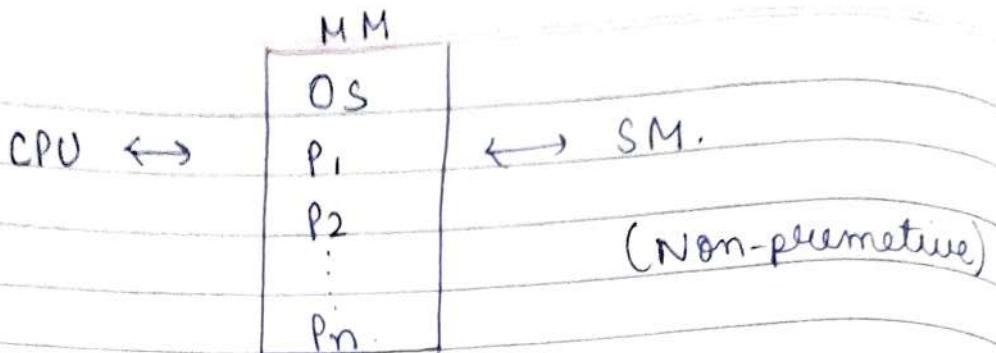
Buffering

uses limited memory space in RAM usually called buffer
(Buffer is a temporary storage area in RAM)

1.3

Multiprogramming OS

2.

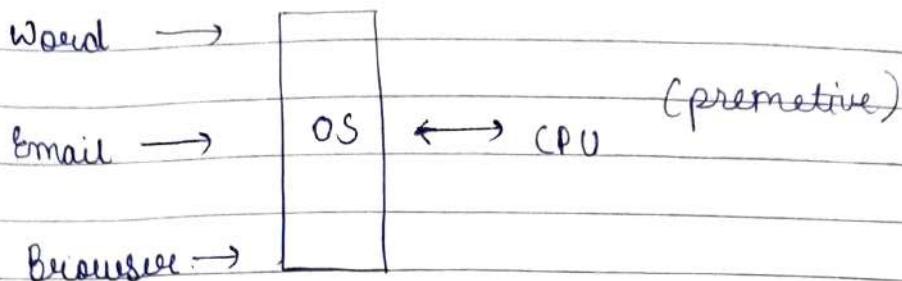


- 1) Maximize CPU utilization
- 2) Multiprogramming means more than one process in main memory which are ready to execute.
- 3) Process generally requires CPU time & I/O time. So if running process perform I/O or some other event which don't require CPU then instead of sitting idle, CPU makes context switch & pick some other process & this idea will continue.
- 4) CPU never ^{remains} idle unless there is no process ready to execute or at the time of context switch

Advantage	Disadvantage
1) High CPU utilization.	1) Difficult scheduling
2) less waiting time, response time etc	2) Main memory management is required
3) May be executed to multiple users	3) Memory fragmentation
4) Now a days useful when load is more	4) Paging (non contiguous memory allocation)

3. Multitasking OS. (Eg Unix)

Time sharing / Fair share / RR



1. Multitasking is multiprogramming with time sharing
 2. Only one CPU but switches bet'n process so quickly that it gives illusion that all executing at same time.
 3. the task in multiprogramming may refer to multiple threads of the same program.
 4. Main idea is better response time & executing multiple process together.
4. Real-time OS - It is used in environment where a large no of events mostly external to system must be accepted & processed in a short time within certain deadlines. The time interval required to process & response to input is very small.
Hard RTOS - It guarantees critical task to be completed within a range of time.
Eg - a robot hand to weld a car body
- Soft RTOS - It provides some relaxation in time.

Advantages

- 1) Max^{m.} utilization of devices & system
- 2) Better task shifting
- 3) Systems are error free

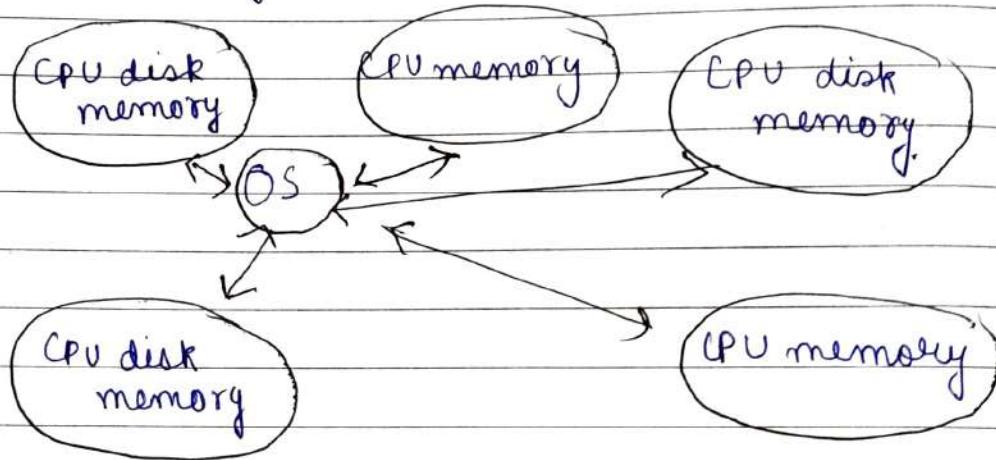
Disadvantages

- 1) limited tasks
- 2) complex algorithms
- 3) use heavy system resources

5) distributed OS. (Eg - LOCUS)

Distributed system uses multiple central processors to serve multiple real time application & multiple users. Data processing jobs are distributed among the processors accordingly.

Processors communicate with each other through various communication lines (such as high speed buses or telephone lines). These are referred as loosely coupled systems or distributed system.

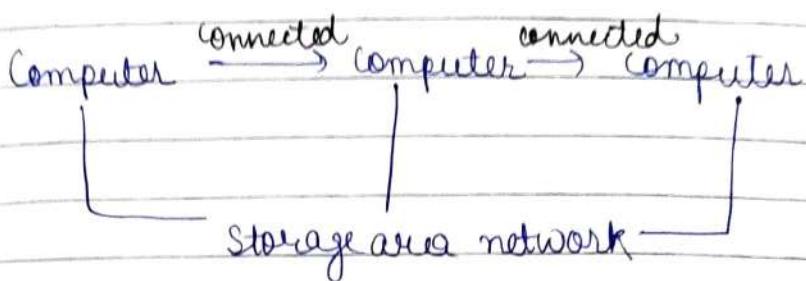


Advantages

- 1) Failure of one network connection doesn't affect other
- 2) Delay in data processing reduces

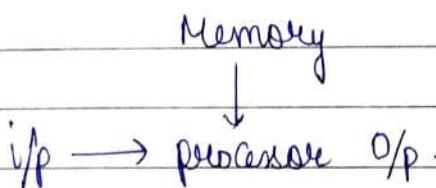
Disadvantages

- 1) Failure of main network will stop entire communication.
- 2) Clustered OS
- 3) Clustered computers share storage & are closely linked via LAN or faster connection.
- 4) Comb'n of multiprocessor + distributed OS.



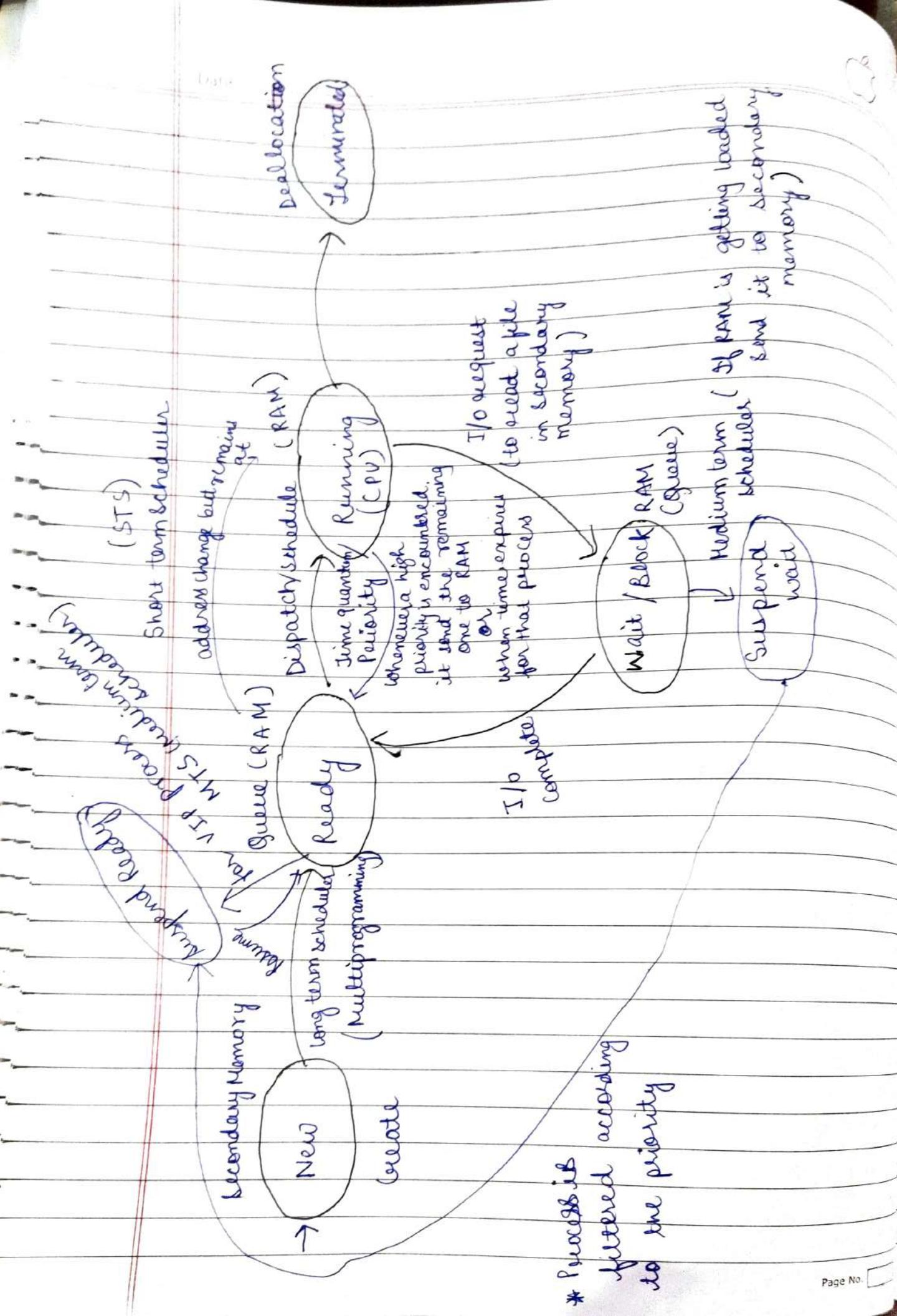
5) Embedded OS. - An OS in which we work on embedded systems. It is designed to perform a specific task for a device that is not in computer.

Eg - ATM



1.5

Process states in OS





- * Non preemptive - NO time quantum, no process can jump in between.
- * preemptive - We stop the running process in midway due to high priority process.

L-1.6

Linux commands.

Q. which command is used to assign only Read permission to all three categories of file 'note'.

- chmod a-~~rwx~~ rw
- chmod go+r note
- chmodugo=r note
- chmod. u+r, g+r, o-x note

chmod - change mode

<u>u</u>	<u>w</u>	<u>x</u>	<u>u</u>	<u>w</u>	<u>-</u>	<u>u</u>	<u>w</u>	<u>x</u>
user	user	user	group	group	group	other	other	other

u - user
g - group
o - others

rx	rw		u - read	4
6	7		w - write	2
			x - execute	1

Q. 'chmod ugo+rwx note' command can be represented in Octal notation as

- 1) chmod 555 note
 2) chmod 666 note
 3) chmod 333 note
 4) chmod 444 note

$$r+w = 4+2 = 6.$$

Q. Suppose you have a file "f1" whose contents are

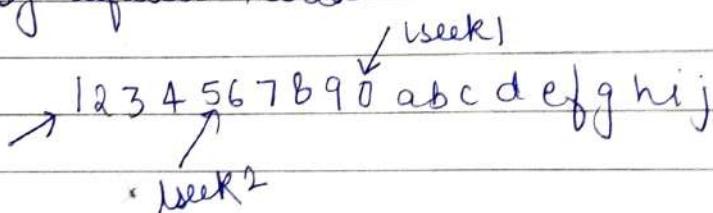
1 2 3 4 5 6 7 8 9 0 abcdefghij
 Here seek is used 2 times sequentially.

lseek (n, 10, SEEK_CUR);
 lseek (n, 5, SEEK_SET);

n is file descriptor. After applying seek Q times, what will be the current position of R/W head?
 (Index starts from 0)

- a) 0.
 b) 5
 c) 10
 d) 15.

lseek - system call (To move read write head we use lseek command)
 by default read write head is at 0.



1) seek_cue → move to the given no from the current

2) seek_set → set the current pos at given

3) seek_end → Count from the end towards left.

System calls in OS.

User mode $\xrightarrow{\text{system call}}$ Kernel mode

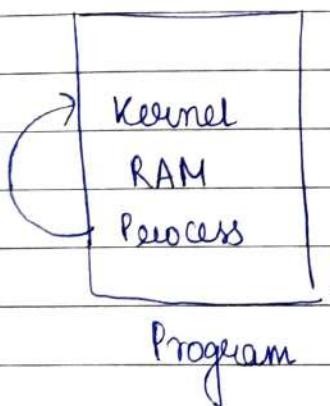
Q+4 monitor, print(6)

In Linux we can use system call directly.

printf() \longrightarrow accessing system call

System calls.

- 1) File related SC \rightarrow open(), Read(), write(), close(), create file etc.



- 2) Device related \rightarrow Read, write, reposition
ioctl(i/p o/p control), fcntl(file related control)

for hardware device

- 3) Information \rightarrow Process or regarding device attributes

For e.g.: getpid, attributes, get system time & data.

4) Process control \rightarrow Program being loaded
 \rightarrow load, execute, abort, fork, wait,
 signals, allocate etc.

5) Communication Control- Interprocess communication
 \rightarrow Pipe(), create/delete connection, Shmget()

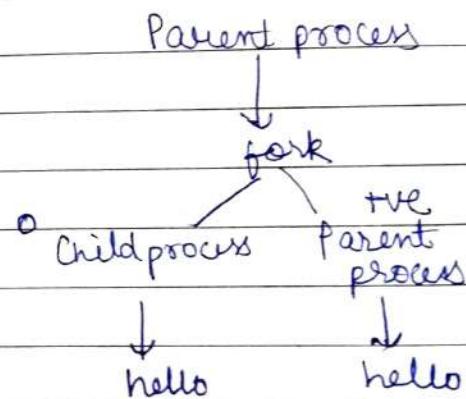
L-1.8 Fork system call

To create a child process from parent process
 We can use fork() or thread.

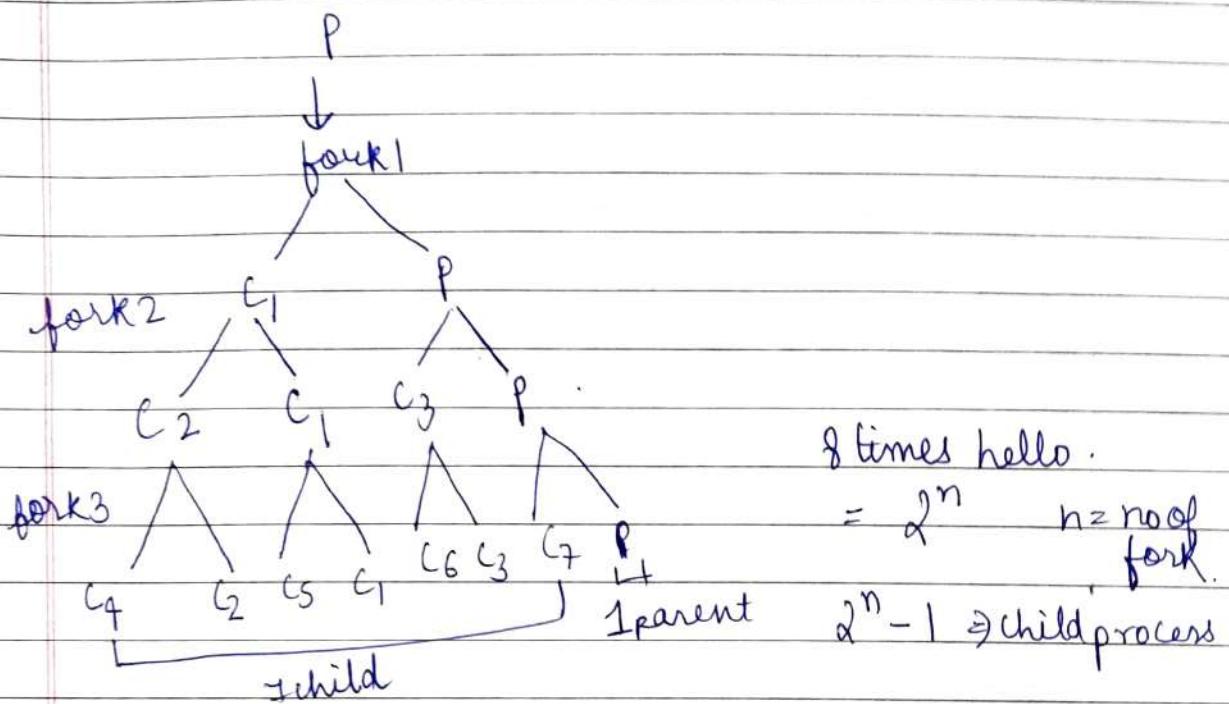
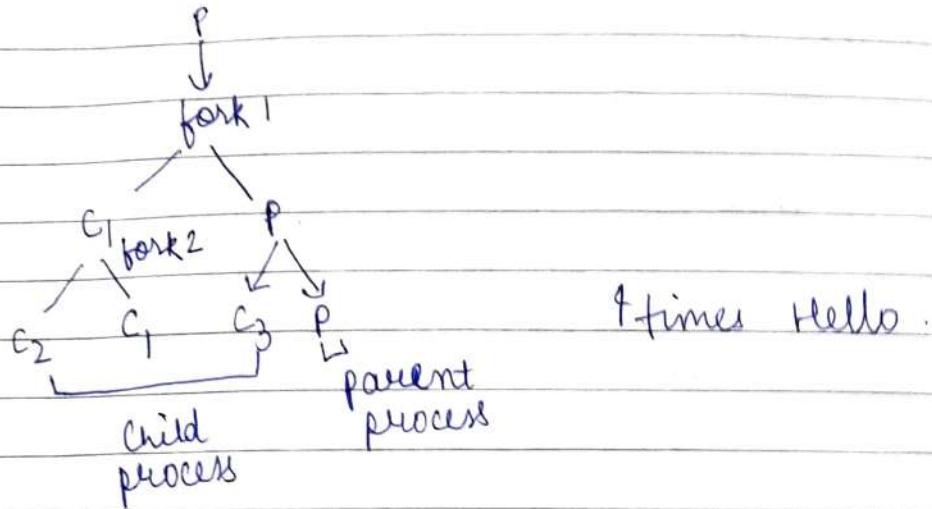
`Fork()`

- 0 child process
- +1 parent process
- 1 child process not created

```
main() {
    fork();
    printf("hello");
}
```



```
main() {
    fork();
    fork();
    printf("hello");
}
```

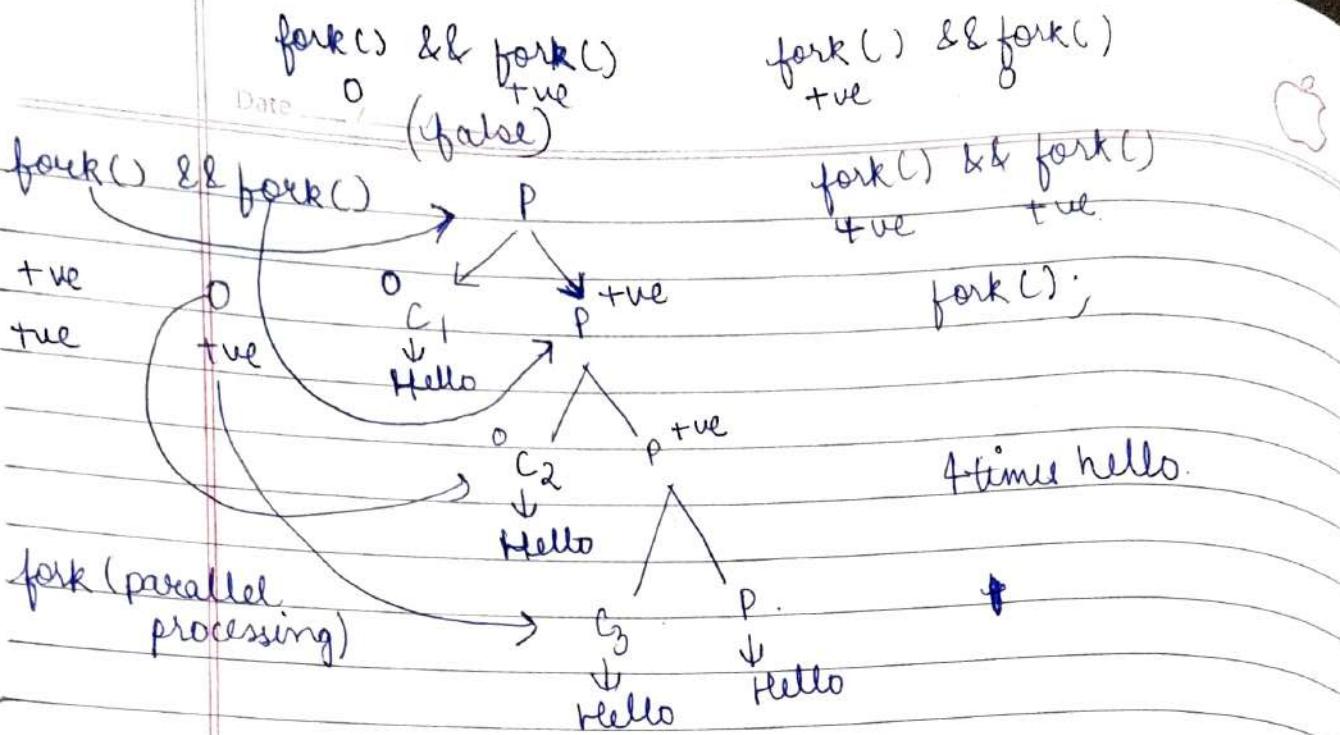


L-1.9

Questions on fork System Call

```
#include <stdio.h>
#include <unistd.h>
```

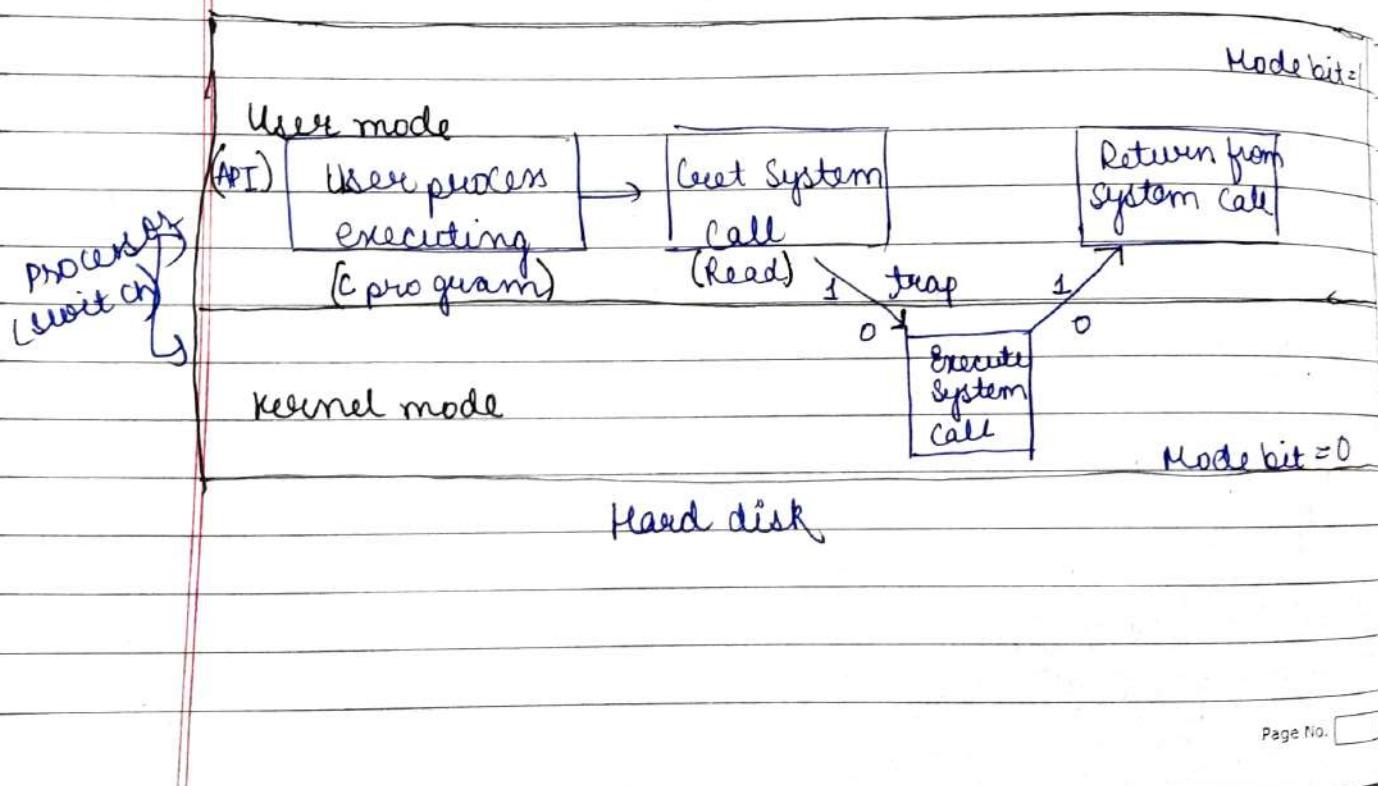
```
int main ()
{
    if (fork() & fork())
        fork();
    printf ("Hello");
    return 0;
}
```

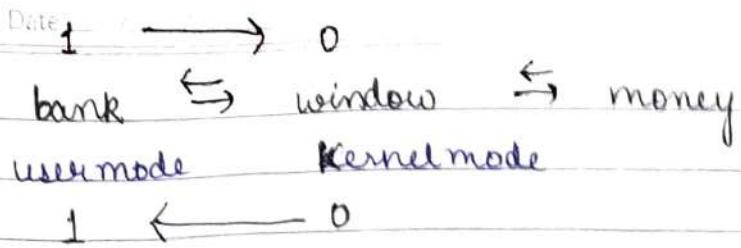


L-1.10

User mode & Kernel mode.

Kernel is the central component of an OS that manages operations of computer and hardware. It basically manages operat'n's of memory & CPU time





L-1.11

Process Vs Threads in OS

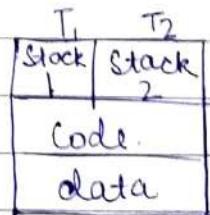
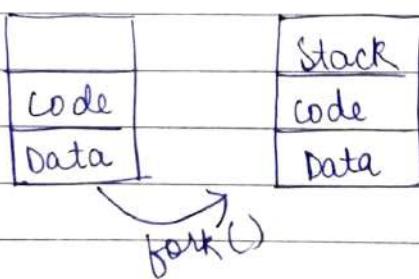
Process - It is a heavy weight task] multitasking
 Thread - It is a light weight task] environment

Process

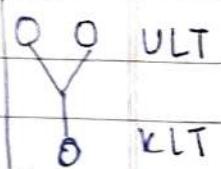
- 1) System calls involved
- 2) OS treats diffⁿ process differently.
- 3) diffⁿ process have diffⁿ copies of data, files, cache
- 4) Context switching is slow
- 5) Blocking a process-not other
- 6) Independent

Thread (User level)

- 1) No system call involved
- 2) All user level threads treated as single task for OS.
- 3) Threads share same copy of code
- 4) Context switch is fast
- 5) Blocking a thread - block entire process
- 6) Interdependent



Hybrid environment



L - 1.12

User level & Kernel level thread
code & data is shared
have own stack but

User level thread

- 1) User level threads are managed by user level library.

- 2) User level threads are typically fast

- 3) Context switching is faster.

- 4) If one user level thread performs blocking operation then entire process get blocked

Kernel level thread

- 1) Kernel level threads are managed by OS (system calls)

- 2) Kernel level threads are slower than user level

- 3) Context switching is slower.

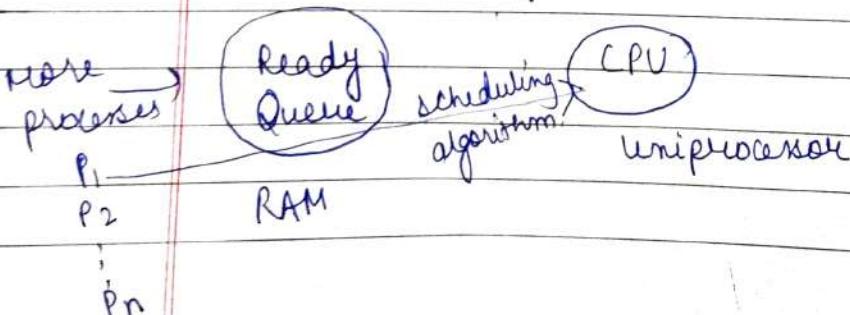
- 4) If one kernel level thread blocked no affect on others.

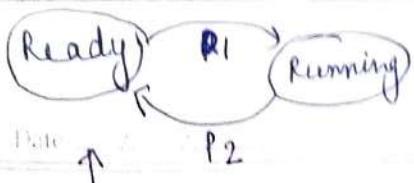
(context switch) Process \rightarrow KLT \rightarrow ULT

L - 2.1

Process scheduling Algorithms

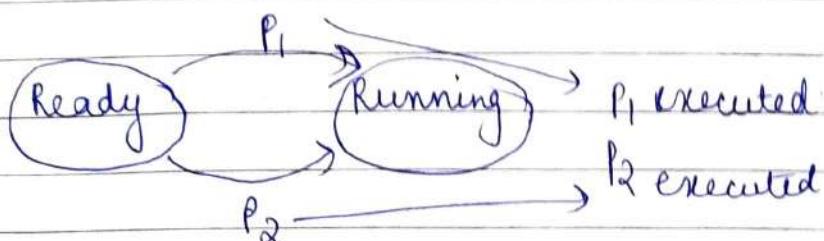
A way of selecting an algorithm from ready queue & putting it on the CPU.





- Reasons
- 1) Time quantum (pre-empted time for P₁)
 - 2) Priority process

- * Pre-emptive \rightarrow If a process is taken out from ready queue or RAM and is put in running process (queue) (CPU). Then we can stop the process P₁ & give another process p₂ & send p₁ to ready queue. It provides responsiveness.
- * Non pre-emptive \rightarrow If a process is taken out from ready queue it would be processed till its burst time, after then only new process can be introduced.



Scheduling Algorithms

Preemptive

Non-preemptive

- | | |
|--|-------------------------------------|
| BT | AT |
| \leftarrow 1) SRTF (shortest remaining time first) | 1) FCFS (First come first serve) |
| BT | BT |
| 2) LRTF (longest remaining time first) | 2) SJF (Shortest job first) |
| TQ | BT |
| 3) Round Robin | 3) LJF (longest job first) |
| Priority given | in descending order of burst time |
| 4) Priority based | 4) HRRN (High response ratio next). |
| | 5) Multilevel Queue |
| | 6) Priority based |

L-2.2

Different times in CPU scheduling

Arrival time - The time at which processes enter the ready Queue or stack.

Burst time - Time required by a process to get executed on CPU.

Completion time - The time at which process completes its execution

Turnaround time = Completion - Arrival time

Waiting time - Turnaround - Burst time

Response time - (The time at which process get CPU) -
(Arrival time.)

Arrival time → process execution → Completion time
(point of time) duration (point of time)

~~Arrived Bank~~ → Create new account → ~~left Bank~~

L-2-3

first come first serve (FCFS)



FCFS is an OS scheduling algorithm that automatically executes queued requests and processes in order of their arrival.

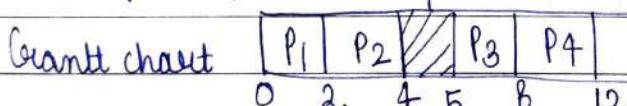
- The process which requests the CPU first get the CPU allocation first.
- Managed using FIFO queue.

Process No	Arrival time	Burst time	Completion time	TAT	WT	RT
				(CT - AT)	(TAT - BT)	
0 P ₁	0	2	2	2	0	0
2 P ₂	1	2	4	3	1	1
5 P ₃	5	3	8	3	0	0
8 P ₄	6	4	12	6	2	2

Order of arrival time

Mode : Non pre-emptive

provide



Time →

At t=12, all the processes got executed

$$\text{Avg TAT} = \frac{14}{4}$$

$$\text{Avg WAT} = \frac{3}{4}$$



SJF - (Shortest Job first) \rightarrow SJF is an algorithm in which the process having the smallest execution time is chosen for next execution.

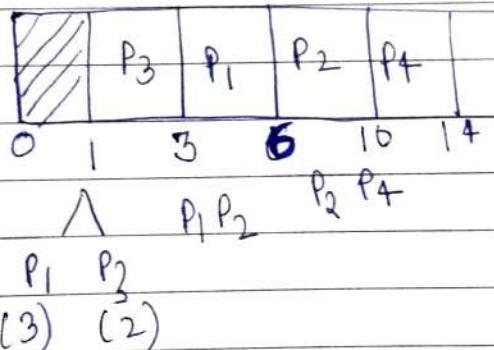
- i) It can be preemptive or non-preemptive
- ii) It reduces the waiting time for other process.
- iii) It is a greedy Algo.

Job Serial No.	Process No	AT	BT	CT	TAT	WT	RT
3	P ₁	1	3	6	5.	2.	0 2
6	P ₂	2	4	10	8.	4.	4
1	P ₃	1	2	3	2	0	0
10	P ₄	4	4	14	10	6	6

Criteria - Burst time

Mode - Non-preemptive

Gantt chart



when BT is same see the one having less arrival time else can use process id.

L-2.5

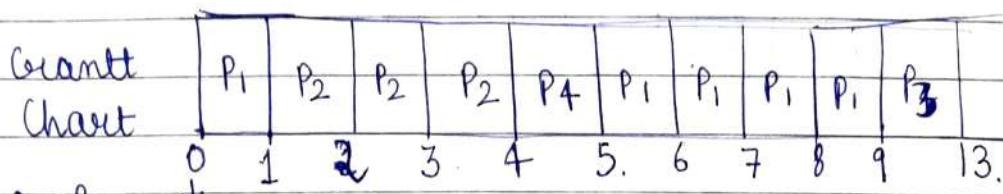
Shortest Remaining Time First
 ↓

(Shortest job first + pre-emptive)

Process no	AT	BT	Compt	TAT	WT	RT	CPU first time
P ₁	0	3 5 4	9	9	4	0	0
x P ₂	1	2 3 2	4	3.	0.	0	1
P ₃	2	4	13	11	7	.7	9
x P ₄	4	5	5	1	0	0	4

Criteria : "Burst time"

Mode : "Pre-emptive"



0 → P₁

↓

1 → P₁, P₂

2 → P₁, P₂, P₃

3 → P₁, P₂, P₃

4 → P₁, P₃, P₄

5 → P₁, P₃

6 → P₁, P₃

7 → P₁, P₃

8 → P₁, P₃

9 → P₁, P₃

$$\text{Avg TAT} = \frac{24}{4} = 6$$

$$\text{Avg WT} = \frac{11}{4} = 2.75$$

$$\text{Avg RT} = \frac{7}{4} = 1.75$$

P₁ → 8 +

P₂ → 3 × 2 × 10

P₃ → 4

P₄ → 1

P ₁	P ₂	P ₂	P ₄	P ₁	P ₃
0	1	2	4	5	9

L-2.6

SJF with preemption ex:

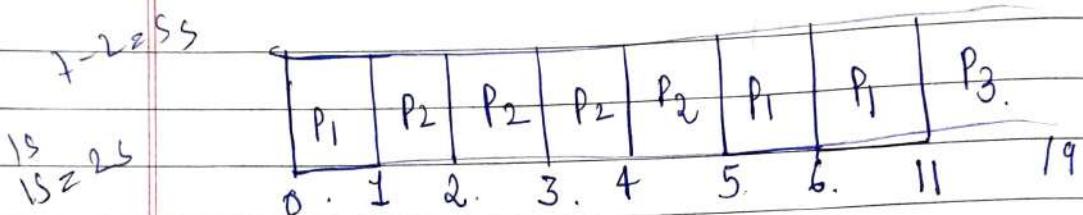
process	AT	BT	CT	TAT	WT	RT
P ₁	0	4.768				
P ₂	1	4.282				
P ₃	2	8.				

Criteria : burst time

Mode : preemptive .

Gantt

chart

0 → P₁1 → P₁, P₂2 → P₁, P₂, P₃3 → P₁, P₂, P₃4 → P₁, P₂, P₃5 → P₁, P₃

P ₁	P ₂	P ₁	P ₃
0	1	5	11



L-2.7

Round Robin Scheduling Algo.

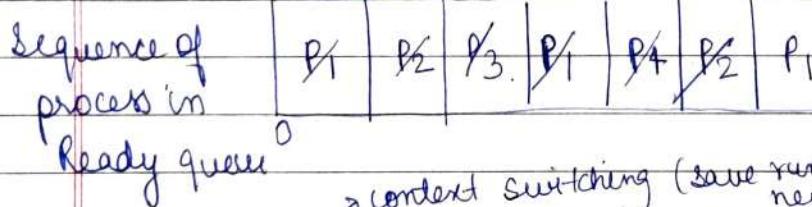
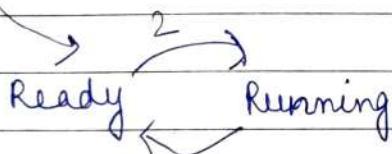
Round robin is preemptive process scheduling algorithm. Each process is provided a fix time to execute, it is called quantum. Once a process is executed for a given time period, it is preempted & other process executes for a given time period.

Process no	AT	BT	CT	TAT	WT	RT	clv first time
P ₁	0	8 31	12	12	7	0	0:
→ P ₂	1	0 42	11	10.	6	1	2
→ P ₃	2	2 0	6	4	2	2	4
P ₄	4	1	9	5.	4	4	8.

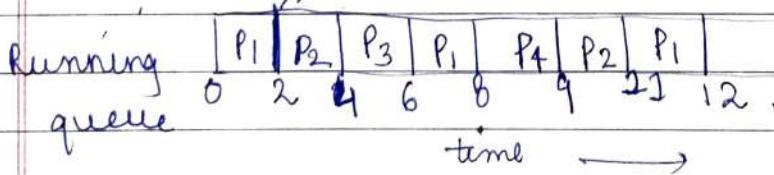
TQ=2

Criteria : "Time Quantum"
Mode : "Preemptive"

(we have to resume the process not restart it)



context switching (save running process & welcome new process)



1 - 2.8 Scheduling algo.
Preemptive Priority ~~Process~~.

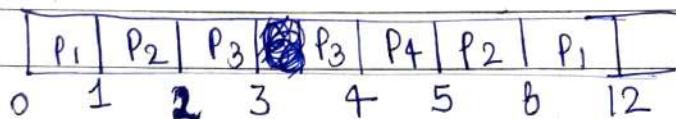
In pre-emptive scheduling, the tasks are mostly assigned with their priorities.

Priority scheduling is a method of scheduling process that is based on priority. In this algorithm, the scheduler selects the task to work as per the priority.

Equal priority - Round Robin or FCFS.

Priority	Process No	AT	BT	CT	TAT	WT	RT	CPU first
10	P ₁	0	48	12	12	7	0	0
20	P ₂	1	34	8	7	3	0	1
30	P ₃	2	120	4	2	0.	0	2
40	P ₄	4	08	5	1	0	0	4

Higher the no. higher the priority.



0 → P₁ time →

1 → P₁, P₂

2 → P₁, P₂, P₃

3 → P₁, P₂, P₃

4 → P₁, P₂, P₃, P₄

5 → P₁, P₂



L-2.9

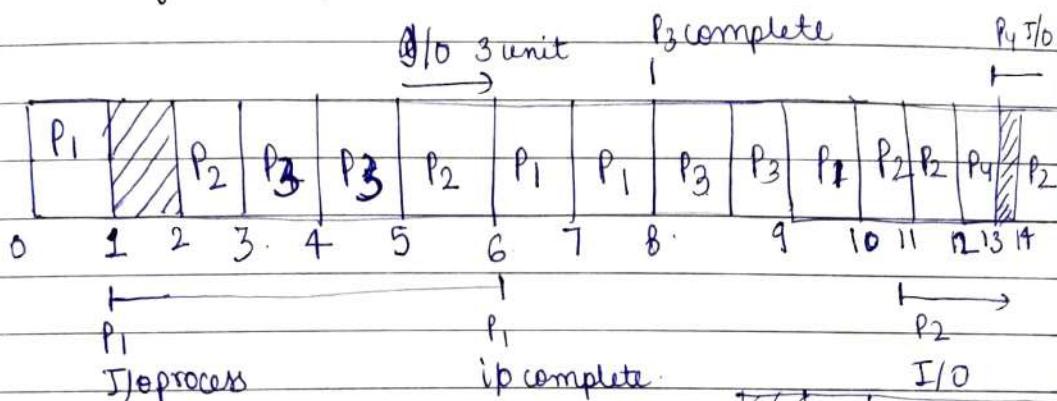
Example of Mix Burst Time (CPU & I/O both) in CPU

Process	AT	Priority	CPU	I/O	CPU
P ₁	0	2	X 0	8 0	8 2 10
P ₂	2	3	3 2 1	3	1
P ₃	3	1	2 X 0	3 0	2 0
P ₄	3	4	2	4	1

lowest the no. highest the priority.

Mode: Preemptive

Criteria: Priority based

Find CT of P₁, P₂, P₃, P₄0 → P₁

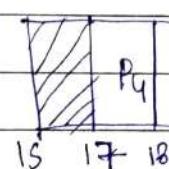
1 → no process

2 → P₂3 → P₃, P₄

high
priority

4 → P₁, P₂, P₃, P₄

5 →

6 → P₁, P₂, P₄8 → P₃, P₁9 → P₁, P₂, P₄

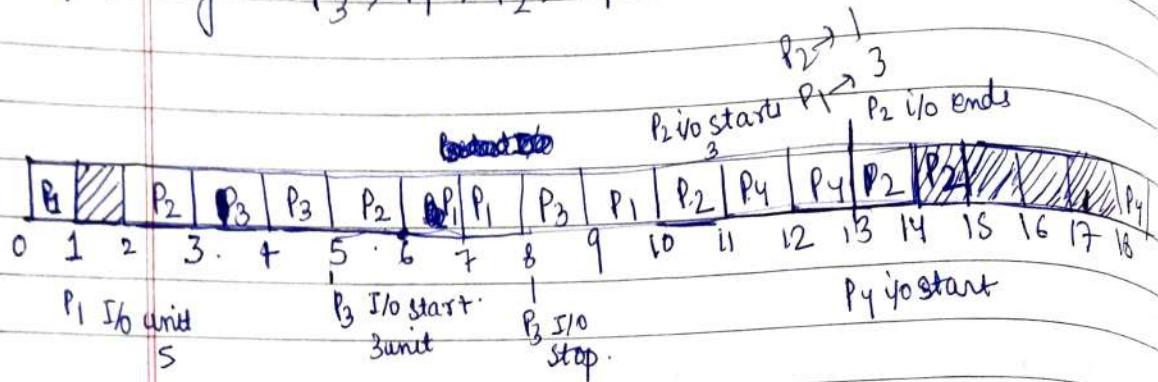
	CT
P ₁	10
P ₂	15
P ₃	9
P ₄	18

$$\text{CPU idleness} = \frac{4}{18}$$

$$\text{Usage} = \frac{14}{18}$$

Priority	AT	Priority	CPU	I/O	CPU
P ₁	0	2	20	5	3'20 X Mode: Preemptive
P ₂	2	3	320	3	1 Criteria Priority
P ₃	3	1	2X0	3	20 X based.
P ₄	3	4	210	4	1

Priority $P_3 > P_1 > P_2 > P_4$



In Ready queue.

I/O Process

- P₁ → 6
- P₃ → 5

6 → P₁, P₂, P₄

7 → P₁, P₂, P₄

8 → P₁, P₂, P₃, P₄

9 → P₁, P₂, P₄

10 → P₂, P₄

13 → P₂

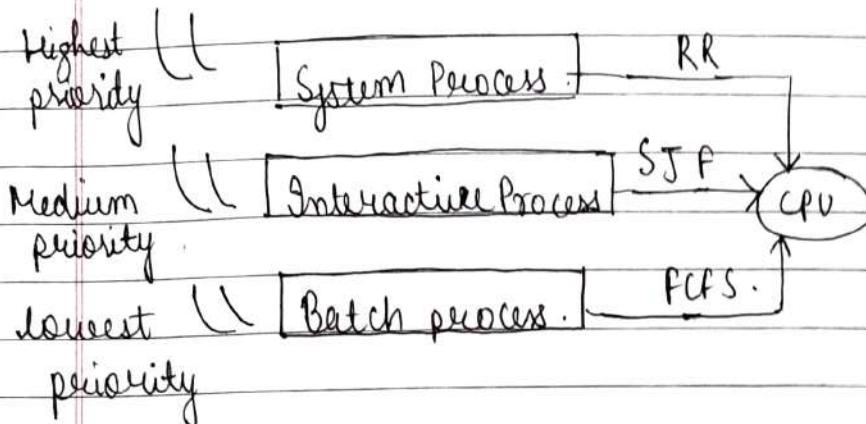
L - 2.10

Multilevel queue scheduling

In every scheduling algorithm till now we have taken one queue.

- i) There must be different queue for diffⁿ process
- ii) Every process can have their own algorithms.

- i) System calls - Round robin
- ii) Interactive calls - SJFS.



Problem :

Lower level or lower priority may experience starvation due to more no of calls on higher priority

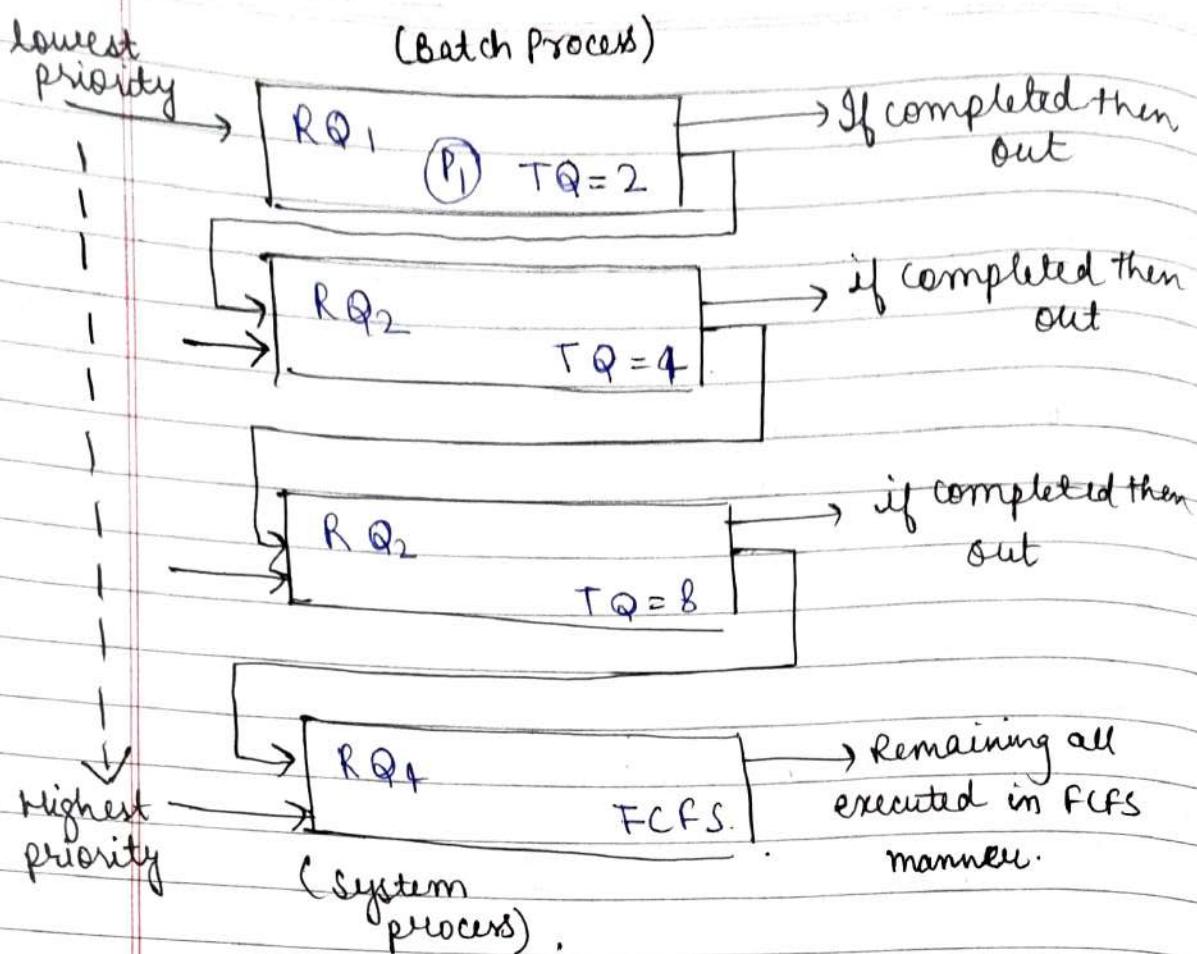
It can be resolved using multilevel feedback queue.

L-2.11

Multilevel Feedback Queue

More system process - lowest priority waits

lowest priority - feedback.



P ₁	19	17	13	5
	TQ = 2	TQ = 4	TQ = 8	

Process Synchronization

Cooperative processes

Processes

Independent processes

One's execution affects the other as they share same or common variable, memory/buffer, code, resources

One's execution doesn't affect other. They have nothing in common.

Ex - IRCTC .

PCB \rightarrow Process control block

Date _____



Process synchronization is used to solve the problems of co-operative process.

int shared = 5; // +

P₁

1. int X = Shared; 5
2. X++; (X=6)
3. Sleep(1); pause
4. shared = X; \leftarrow

context switching P₂

int y = shared;

y--; (y=5)

Sleep(1)

shared = y;

~ 6

(Terminated)

when they are parallel.

Jobs saved
in PCB

Uni processor

Processes aren't synchronized as the process at last gives either 4 & 6. but it should give 5.

This is called race cond'n.

L-3.2

Producer-consumer Problem

Co-operative process.

There is one producer that is producing something and there is one consumer that is consuming the products produced by producer. The producer & consumer share the same memory buffer.

Consumer

```
void consumer(void)
{
    int item c;
```

```
while (true)
{
    Buffer empty while (count == 0);
    item c = Buffer(out);
    out = (out + 1) mod n;
    count = count - 1;
}
```

Process-item (itemc)

1. load R_c, m[count] n = 8
2. DECR R_c
3. Store m[count], R_c

OUT



0
1
2
3
4
5
6
7

x₁

COUNT



IN

COUNT



Buffer [0 n-1]

index

1. Load R_p, m[count];

2. INCR R_p;

3. Store m[count], R_p

next empty slot address



Producer

int count = 0;

```
void producer(void)
{
    int item p;
```

```
while (true)
{
    produce item(item, p);
    while (count == n); → buff
    Buffer[in] = item p; if
    in = (in + 1) mod n full
    count = count + 1; its
    ↓ shelf
}
```

get from the buffer

give to buffers

n = size of buffer

Case I: x₁ (producer is processing x₁)

Out

∅ 1

(0+1) mod 8

1 mod 8 = 1

in

∅ 1

count

∅ 2 0

$$(0+1) \text{ mod } 8 = 1$$

$$\text{count} = 1 - 1 = 0$$

It forms a circular queue so we used mod n.

Consumer.

0	x_1
1	x_2
2	x_3
3	x_4
4	
5	
6	
7	

Producer

Count	In
3	4

Let x_4 be inserted in the in value.

Producer:

$R_p = 3$. Load $R_p, m[\text{count}]$ I₁

$R_p = 4$. INCR R_p . I₂

3

If after INCR R_p the process gets preempted
Resuming $R_p = 4$ I₃

Let x_1 be consumed.

Load $R_c, m[\text{count}]$ I₁
 $R_c = 3$

DEC R_c I₂
 $R_c = 2$

Resuming $R_p = 2$ I₃

I₁ I₂ consumer I₁ I₂ Producer I₃ Consumer I₃

There are 3 items in buffer but count says only 2 values is there, there is race condn. so we can't achieve process synchronization.



L - 3. 3.
Printer-Spooler Problem.

multiple programs → spoolers → take items sequentially → gives the print

1. Load $R_i, m[i:n]$

2. Store $SD[R_i], "F-N"$
 $f1.doc$

3. JNCR R_i

4. Store $m[i:n], R_i$

Spooler Directory

0	f1.doc	
1		
2		
3		
4		
⋮		

IN [0:1]

↳ empty slot

Case 1: f1.doc P1
R1 [0:1]

b4.DOC ↑ b5.DOC ↑

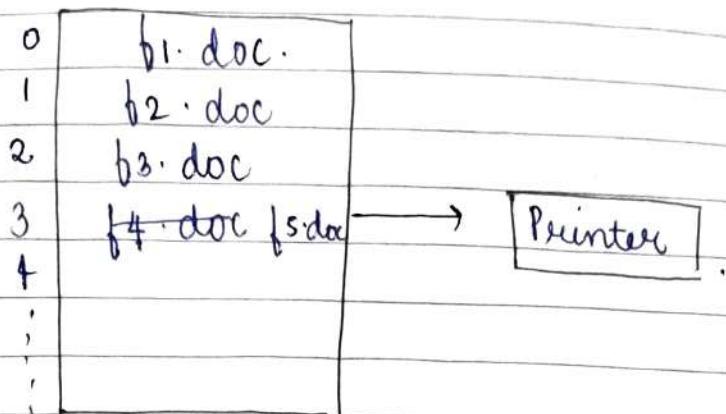
Case 2: Two process P1 & P2

↓

cooperative process



In



In [3] 4

R₁ [3] 4R₂ [3] 4
$$P_1 \cdot I_1 I_2 I_3 \quad | \quad P_2 \cdot I_1 I_2 I_3 I_4$$

P₁ & P₂ due to non synchronization put the file in the same index. So there is loss of data. When 2 processes share the same code, same data there is a problem created.

3.4

Critical Section Problem

"It is the part of program where shared resources are accessed by various processes"

↓
co-operative

P₁

include

main()

A,B → non critical section

P₂

include

main() {

X,Y

↳ common code

→ non critical section

Common code

→ critical section

count → critical section → count

main()

A, B

Non critical
section

Should
clear
entry
section

{Entry section}

count++

critical section.

Entry section

↓
shouldn't clear
critical section.

P₁

P₂

{Exit section}

{Exit section}

Different solⁿ for process synchronization

4 conditions

1) Mutual Exclusion }
2) Progress }

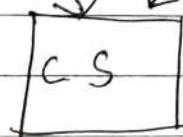
Primary

3) Bounded wait

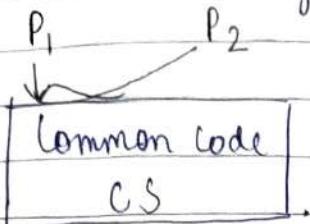
Secondary 4) No assumption related to H/W speed

Mutual Exclusion → If P₁ or P₂ is inside critical section the P₂ or P₁ is not allowed to enter in

P₁ P₂ critical section.

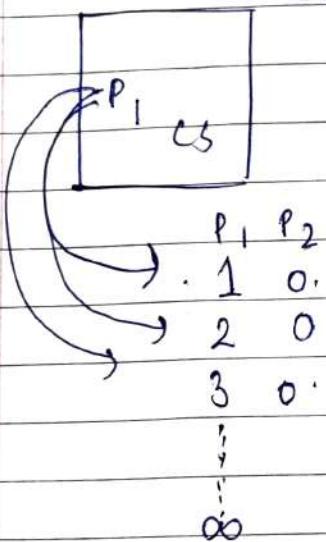


2. Progress \rightarrow when both can use common code there's progress.



P₁ can't enter due to some code written in P₂ so there is no progress.

3. Bound Wait - There shouldn't be a condition of starvation as one process uses the CPU every time. Let every process use the CPU.



P ₁	P ₂
1	0
2	0
3	0
⋮	
⋮	

If P₁ uses only the CS it will go in infinite loop

4) H/W speed - Every solution should be portable and universal. There shouldn't be anything dependent on H/W speed.

Now Solution to Critical Problem
L-3.5

Critical section solⁿ using lock.

CS - critical section

Date: / /

do {

acquire lock.

CS

release lock
}.

↑ entry code

{
1. while (LOCK == 1);
2. LOCK = 1
3. Critical Section
4. LOCK = 0.
}

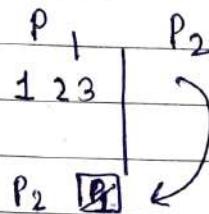
↓
exit code.

- * Execute in User Mode.
- * Multiple process solution.
- * No mutual exclusion
- * Guarantee

Case 1 : P₁ P₂

lock = 0 → CS is vacant

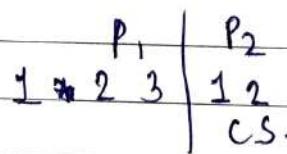
lock = 1 → CS is full



lock ≠ 0 & 1

Se

Case 2:



L = 0 & 1

[P₂ P₁] → No mutual exclusion
CS

3.6 Test and Set Instruction

1. while ($LOCK \geq 1$);	Entry code
2. $LOCK = 1$	
3. Critical section	
4. $LOCK = 0$	Exit code

It combines 1 & 2

while (test_and_set (&lock));

CS

lock = false;

boolean test_and_set (boolean *target)

{

boolean r = *target;

*target = TRUE;

return r;

}

lock	target	re	P ₁	P ₂
false	1000	false	1	2
1000				

True

Mutual exclusion as well as Progress achieved.

3.7 Turn variable

1) 2 process solution

2) Run in user mode

Process "P ₀ ".	Process "P ₁ ".
entry code \rightarrow while (turn != 0)	while (turn != 1)
CS	CS

entry code \rightarrow turn = 1 ;

exit code \rightarrow turn = 0 ;

1st turn = P. 1

P0

CS

P1

CS

P1 X,

P0 X,

} Mutual exclusion is satisfied

2) Progress (there is no progress)

CS

P1 X

when even = 0

3) Bounded wait \rightarrow so, there is no bound wait as turn value changes.

4) Not H/W dependent

3.8 Semaphore

Semaphore (It is a tool to prevent race cond)

Counting
(as to ∞)

Binary.
(0,1)

running cooperative
process side
by side

Semaphore is an integer variable which is used in exclusive manner by various concurrent cooperative process in order to achieve synchronization

Entry section

Date _____

Down (~~lock~~ semaphores)

{

s.value = s.value - 1;

if (s.value < 0)

{

put process (P(B)) in
suspended list sleep();

}

else.

return;
}.

Exit section



UP (semaphores)

{

s.value = s.value + 1

if (s.value > 0)

{

Select a process

from suspended list

wake up();

}

}.

P(), Down, Wait
v(), Up, (signal, post, Release)

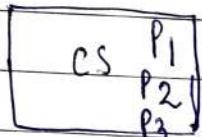
P₁, P₂, P₃

entry code

CS := ..

exit section

P₁, P₂, P₃, P₄ (process control block)
P₅



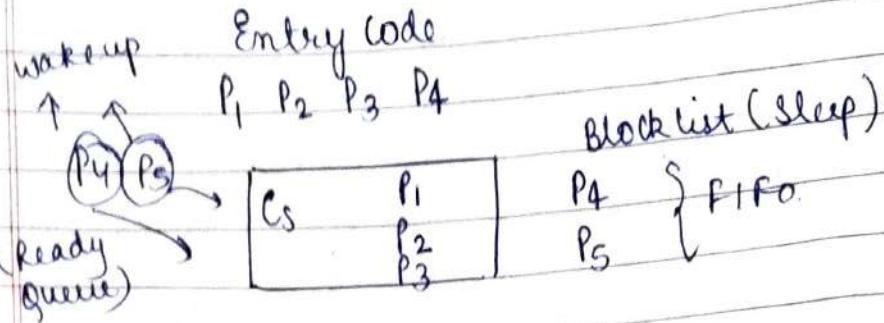
Block list (sleep())

P₄

P₅.

S

X 1 0 X (s.value < 0)
-2



down

S

-4
↓

4 processes are there which is in CS.

Exit code

S
$32 \times \emptyset - X - 2 \emptyset - X - 1$

S

0
↓

Further no process can come in CS.

up

S

0
↓

No process in suspend list

S

10
↓

How many process can successfully come in CS.

$$S = 10$$

Successful operation = 10

S

10
↓

6 P

4 V

final value of SP

$$10 - 6 = 4 + 4 = 8$$



S

SP 3V 1P

17

$$= 17 - 5 + 3 - 1$$

$$= 12 + 3 - 1$$

$$= 15 - 1$$

$$= 14$$

3.9 Binary Semaphore (integer variable shared by a lot of process in mutual exclusive manner).

Semaphore → 0 to ∞.

Binary Semaphore → 0, 1

Down (Semaphores)

{ if (value == 1)
 {

 s.value = 0;

}

else
 {

 Block this Process.

 And place in suspend
 list sleep();

}

}

Up (Semaphores)

{ if (suspend list is empty)

{

 s.value = 1;

}

else {

 Select a process from
 suspend list &

 wake up();

}

}

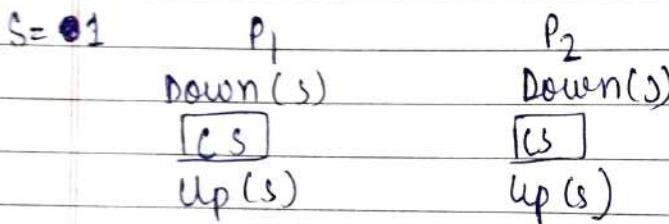
Down, p, wait
Up, v, signal

$S = 1$
Down } successful
 $S = 0$ openⁿ.

$S = 0$
sleep(); } unsuccessful
block openⁿ.

$S = 0$ $S = 1$
Up Up
 $S = 1$ $S = 1$

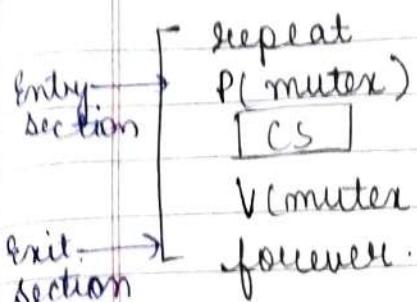
In up we check if the block queue is empty or not ($S = \emptyset \text{ or } 1$)
If empty ($S = \emptyset \text{ or } 1$)
If not empty \rightarrow It will bring process from block state to unblock state.



- Q. Each process P_i ($i = 1 \text{ to } 9$) execute
the following code.

Pi { $i=1$ to q }

P10



mutex
1

repeat
V(mutex)
CS
V(mutex)
forever.

What is the max^m no of process that may present in CS at any point of time?

(Case 1) mutex 1
P(mutex)
mutex 0.

CS.

V(mutex) mutex = 1.

(Case 2) mutex

P(mutex) 1
0

P2 P3
P4
(block) → CS.
P1

ifore P10

mutex: & V & X & & P1 P10 P2 BLOCK

CS
P1 P10 P2

P3

P3

P4

P5

P1 P2 P3 P10

P6

P7

P1 P2 P3 P4 P5 P6 P7 P8
P1 P2 P3 P4 P10 P1 P2 P3 P4 P10 P9

max^m no of process can be 10 but putting P₁₀ in and out of critical section.

Solution of Producer Consumer using Binary Semaphore

Counting Semaphore

- full 0 = No of filled slots
- Empty N → No of empty slots

Produce item(item p);

1) down(empty);

2) down(s);

3) Buffer [IN] = item p;

4) IN = (IN + 1) mod n;

5) up(s);

6) up(full);

Consumer.

1) down(full);

2) down(s);

3) item c = Buffer[OUT];

4) OUT = (OUT + 1) mod n;

5) up(s);

6) up(empty);

N = 8

	0	a	
	1	b	
	2	c	
IN	3	d	OUT
	4	-	
	5		
	6		
	7		

$$\text{Empty} = 5 \quad 4 \\ \text{full} = 7 \quad 4$$

$$S = X \neq 1 \quad I_n = (3+1) \bmod 8 \\ = 4 \bmod 8 \\ = 4$$



For consumer.

Empty = $\emptyset \neq 5$ In = 4

full = $\neq 3$ Out = $\emptyset \neq 1$ $((0+1) \bmod 8)$

S = $\emptyset \neq 1$.

0	
1	b
2	c
3	d
4	
5	
6	
7	
8	

With preemption (context switching)

Empty = $5 \neq 5$

full = $3 \neq 2$ (consumer in CS)

S = $\emptyset \neq 0$

Out = $\emptyset \neq 1$ \hookrightarrow consumer can't do down of S.

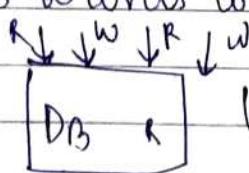
In = $3 \neq 4$

3.12 \rightarrow Solution of Reader writer problem.

We have a database used by Reader & writer

Reader - who only reads the data

Writer - who wants to update the data also.



(Same data)

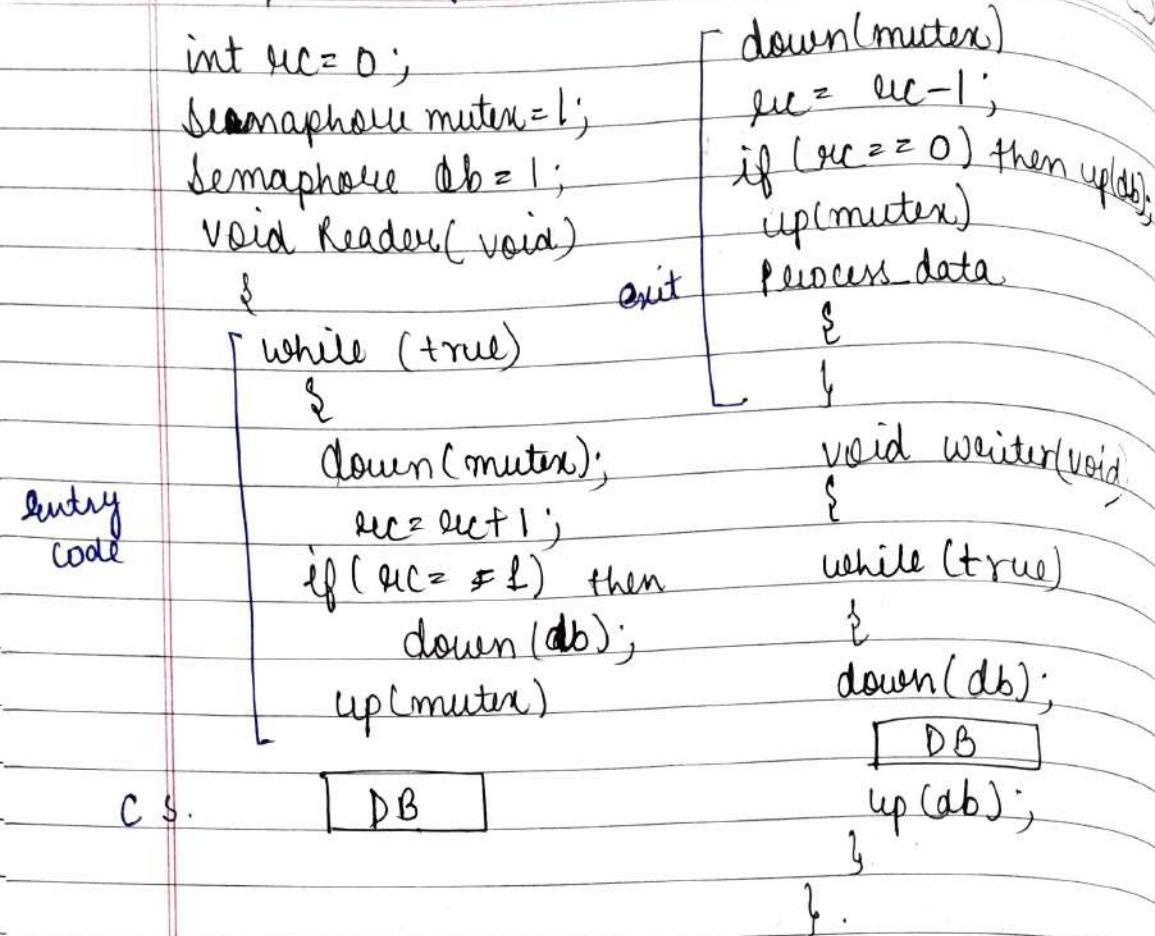
R-W (problem)

W-R (problem)

W-W (problem)

R-R (no problem)

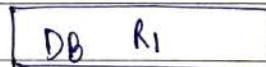
read count
no of readers in buffer.



To synchronise problem we used Semaphores.

Case 1: R₁ comes first

R_C = 0 1 mutex = 1 or 1 db = X₀
down(mutex)



For writer

R_C = 1 mutex = 1 db = 0
down(db)

→ db ≠ -1 so the process gets blocked.

Case 2: W₁ comes first



$$R_C = \emptyset \times 1 \quad \text{muter} = \times 0 \cdot \quad db = \times 0$$

down(db)



$db \neq -1$ can't be negative
so r_1 gets blocked.

Case 3: write-write problem

when writer w_1 comes down(db) $db = \times 0$.

when w_2 comes $db \neq -1$ so it gets blocked.

Case 4: Read - Read

$$R_C = \emptyset \times 2 \quad \text{muter} = \times \emptyset \times \emptyset \cdot 1 \quad db = \times 0$$

$\boxed{DB \ R_1, R_2}$

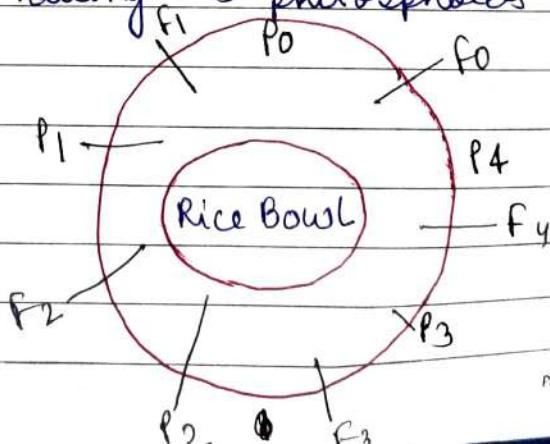
$z = 1$ no

down(~~muter~~
 db); \times

up(muter.)

3.13 Dining Philosophers Problem.

A dining table having 5 philosophers & 5 forks



void philosopher(void)

{

while (true)

{

Thinking();

table_fork(i); \leftarrow Left fork

table_fork((i+1) % N); \leftarrow Right fork

EAT();

put_fork(i);

Put_fork((i+1) % N);

}

}.

Philosopher \rightarrow think
 \downarrow eat

(Case 1): P₀ comes

i = 0

left fork = 0 (f₀)

right fork = (0+1)%N = 1 (f₁)

eating

f₀

f₁

end.

(Case 2): P₁

i = 1

left fork = 1 (f₁)

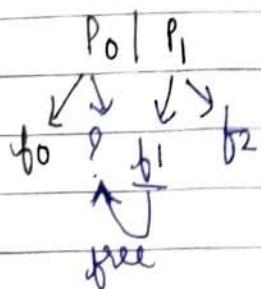
right fork = 2 (f₂)

eating

f₁

f₂

Case 3:



"problem of race cond" occurs.

$S[i]$ → use array of semaphores

So $S_1 S_2 S_3 S_4$
 Initially $\rightarrow \begin{matrix} x & x & 1 & 1 & 1 \end{matrix}$

Now $\begin{matrix} & & 1 & 1 & \end{matrix}$

void philosopher(void)

while (True)
 {

Thinking();

Wait (table-fork(si))

Wait (table-fork((i+1) mod N))

EAT();

Signal (Put fork(i));

Signal (Put fork((i+1) mod N))

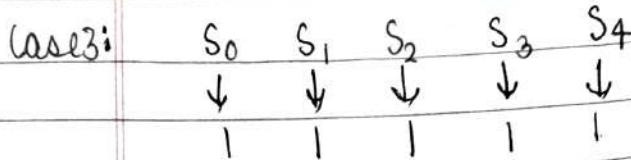
}

}.

(Initialize every semaphore
 with 1 when initialized
 with 0. it waits
 & get blocked)

$P_0 \rightarrow S_0 S_1$
$P_1 \rightarrow S_1 S_2$
$P_2 \rightarrow S_2 S_3$
$P_3 \rightarrow S_3 S_4$
$P_4 \rightarrow S_4 S_0$
Process 6

But in ab P_0 & P_2 can come. It is a special case of mutual exclusion as P_1 & P_2 are independent of each other.

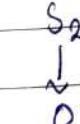


P_0 comes first



It gets preempt

P_1 comes P_2 comes P_3 comes

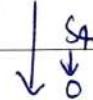


preempt

preempt

preempt

P_4 comes



preempt

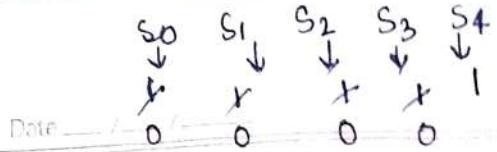
for eight fork (blocked)

The take left hand side fork but gets blocked for the right one. This situation is called deadlock (All processes get blocked)

Q. How to remove deadlock?

On changing the sequence of one process.

$P_4 \quad S_0 \quad S_4$



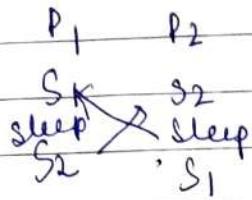
blocked $\rightarrow P_4$ S₀ S₄

We can change sequence of any philosopher.

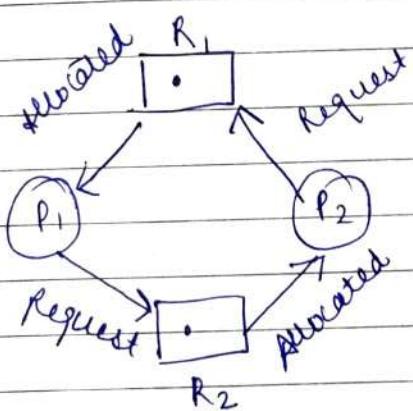
Nth philosopher:

Wait (take_fork (S_{(i+1) mod N}))
 wait (take_fork (S_i));

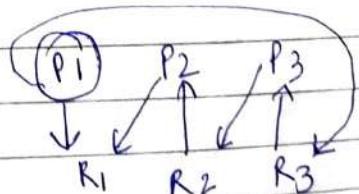
4.1 Deadlock concept



If two or more process are waiting on happening of some event which never happened we say these processes are involved in a deadlock then that state is called deadlock.



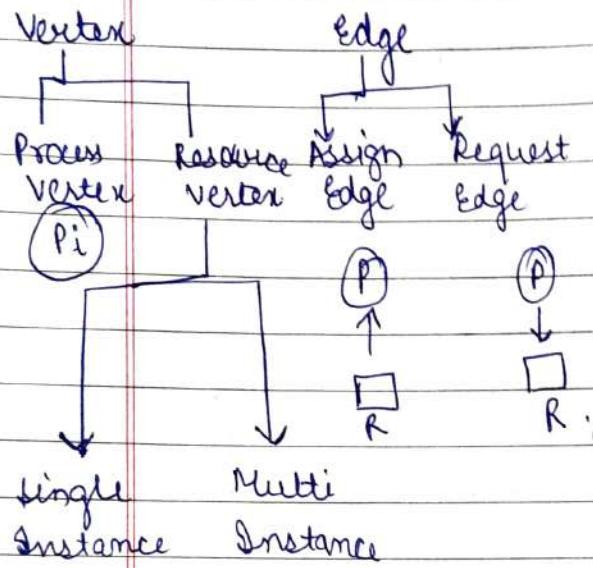
R₁ \rightarrow P₁
 R₁ is allocated to P₁
 P₂ request for R₁



Necessary conditions for deadlock.

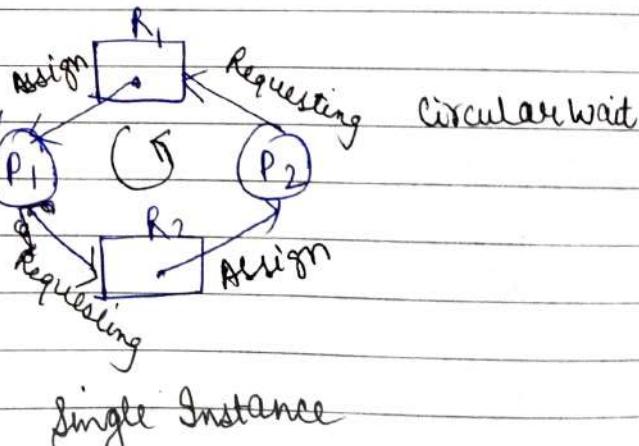
- 1) Mutual Exclusion
 - 2) No Preemption
 - 3) Hold & Wait
 - 4) circular wait

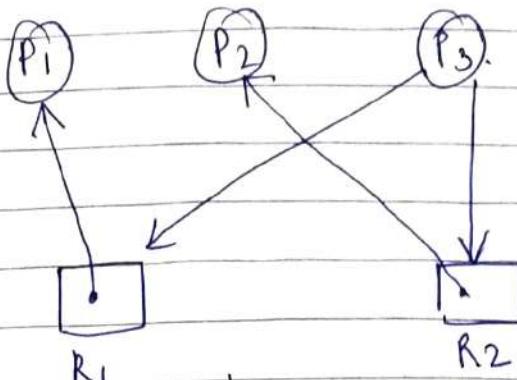
4.2 Resource Allocation Graph in Deadlock (RAG)



Eg: CPU eg : Register
Monitor

	Allocate		Request	
P _i	R ₁	R ₂	R ₁	R ₂
P ₁	1	0	0	1
P ₂	0	1	1	0





	Allocation		Request		Result
	R ₁	R ₂	R ₁	R ₂	
P ₁	1	0	0	0	✓ Terminate
P ₂	0	1	0	0	✓ Terminate
P ₃	0	0	1	1	✓

Availability (0, 0)

(1, 0) → after P₁ terminate

(1, 1) → after P₂ terminate

No deadlock occurs.

Waiting
 finite (Starvation) infinite (Deadlock)

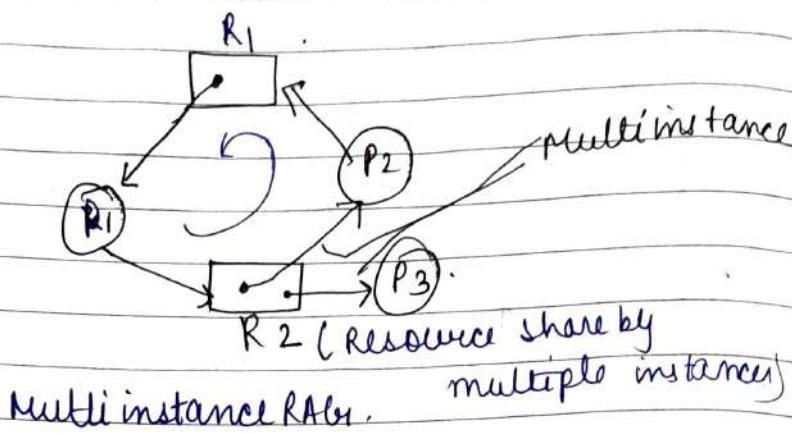
If RAG (Circular wait) cycle → Always in Deadlock

↓
 A single instance

Single instance + Circular wait = Deadlock.
 Multi instance + C.W. = Possible (may or may not)

Date / /

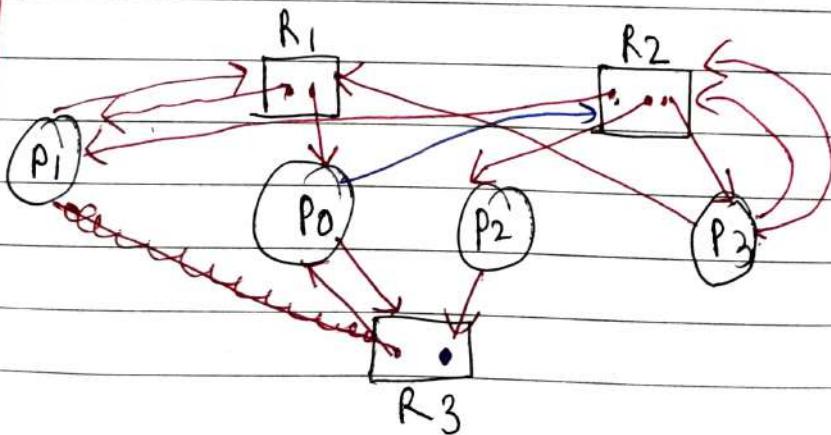
4.3 Multinode RA61.



Process	Allocate		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 ✓

$$\begin{matrix} \text{Current avai} & \rightarrow (0, 0) \\ & | \\ & 0 & 1 \\ & | & 0 \\ & 0 & . & \checkmark \end{matrix}$$

No deadlock





	Allocate		Request	
	R ₁	R ₂	R ₁	R ₂
P ₀	1	0	1	
P ₁				
P ₂				
P ₃				

	Allocate			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	1	0
P ₂	0	1	0	0	0	1
P ₃	0	1	0	1	2	0

curr aw = $\begin{matrix} R_1 & R_2 & R_3 \\ 0 & 0 & 1 \end{matrix}$

$P_2 \rightarrow \underline{\quad 0 \quad 0 \quad 1 \quad 0}$

$P_0 \rightarrow \underline{\quad 1 \quad 0 \quad 1 \quad}$

	R ₁	R ₂	R ₃
P ₁	1	1	0

→ No deadlock in the system.

	R ₁	R ₂	R ₃
P ₃	0	1	0

- curr availability

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

4.4 Deadlock Handling methods

1. Deadlock ignorance (Ostrich method)
2. Deadlock prevention
3. Deadlock avoidance (Banker's Algo)
4. Deadlock detection & Recovery

Deadlock ignorance
↓

(Just ignore the deadlock)

Deadlock occurs very rare: Windows has lots of code in this. We want more & more speed so we don't waste any code for deadlock, so it's easy to avoid as it's rare.

Why ostrich method called?

Whenever there is a sand storm Ostrich puts its head in the sand assuming no sand. In the same way we ignore deadlock so that the performance & speed doesn't get degraded.

Deadlock prevention

Before the deadlock occurs find the preventions
4 necessary conditions for deadlock

- 1) Mutual exclusion
- 2) No preemption
- 3) Hold & wait
- 4) Circular wait

either remove all the conditions or try to false any one of the situation.

- 1) Mutual exclusion (there should be no sharing betⁿ diffⁿ process at same time).

So, if we make all resources shareable we can remove mutual exclusion but some resources like printer can't be made shareable.

- 2) No preemption (there should be any preemption in betⁿ the resources)

So, if we make the process preempted using timestamp or TQ we can prevent deadlock.

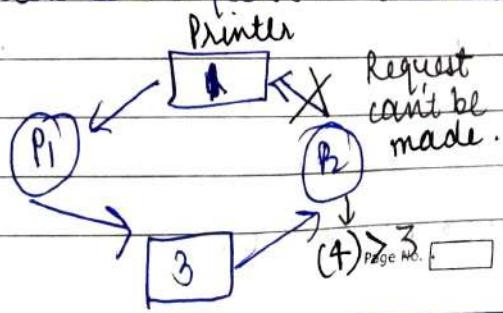
- 3) Hold & wait (holding some resource & waiting for some resources).

Before the process starts, give all the resources to the process.

- 4) Circular wait

Order all the resources (like numbering). Whenever a process request for a resource it would be ordered in increasing order. A process can request in increasing order.

1. Printer
2. Scanner
3. CPU
4. Register



Request
can't be
made.

Page No. _____

Deadlock Avoidance

while giving the resource to the process check its safe or not.

Deadlock detection & Recovery.

Step 1: Firstly check whether there is a deadlock or not using RAG (Resource Allocation graph).

Step 2: After detection do some recovery

- 1) kill the processes or a process (lower priority resources)
- 2) Resource preemption (preempt the resource it is holding)

4.5 Deadlock Avoidance (Banker's Algorithm)

Deadlock Avoidance Algorithm

While providing the resources to the process we check if deadlock occurs or not. It is also called deadlock prevention.

$$\text{CPU (A)} = 10 \text{ (7)}$$

$$\text{Memory (B)} = 5 \text{ (2)}$$

$$\text{Printer (C)} = 7 \text{ (5)}$$

(Total AT) Allocation \rightarrow how much is allocated

Max Need \rightarrow how much is needed

(Total AT - A) Availability \rightarrow how much resource is available

(Max Need - Allocation) Remaining Need

Process	Allocation			Max Need			Available			Remaining need Max-Allocation		
	(10-3)	(5-2)	(5-3)	A	B	C	A	B	C	A	B	C
P ₁	0	1	0	7	5	3	3 (2)	3 (1)	2 (0)	7	4	3
P ₂	2	0	0	3	2	2	5 (2)	3 (1)	2 (1)	1	2	2
P ₃	3	0	2	9	0	2	7 (0)	4 (0)	3 (2)	6	0	0
P ₄	2	1	1	4	2	2	7 (0)	4 (1)	5 (0)	2	1	1
P ₅	0	0	2	5	3	3	7 (0)	5 (0)	5 (2)	5	3	1
+							10	5	7			
	7	2	5									

If we can't fulfil need of every process it is called deadlock.

P₂ → P₄ → P₅ → P₁ → P₃ (safe sequence)

(sequence having no deadlock)

In real life it is not possible.

4.6. Gate Question.

Process	Allocation			Max			Available			Remaining need		
	A	B	C	A	B	C	A	B	C	A	B	C
P ₀	1	0	1	4	3	1	3 (1)	3 (0)	0 (1)	3	3	0
P ₁	1	1	2	2	1	4	4 (1)	3 (0)	1 (2)	1	0	2
P ₂	1	0	3	1	3	3	5 (1)	4 (0)	3 (0)	0	3	0
P ₃	2	0	0	5	4	1	6 (2)	4 (0)	6 (0)	3	4	1
+							8	4	6			
	5	1	6									

4.7 Question Explanation on Deadlock

Q A system requires 2 units of resources having 3 process. The min. no of units of 'R' such that no deadlock will occur.

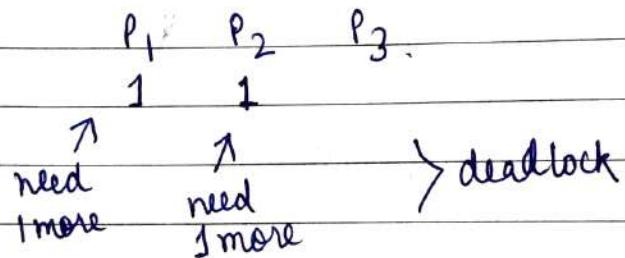
- a) 3
- b) 5
- c) 6
- d) 4

$P_1 \quad P_2 \quad P_3$

$$3 \times 2 = 6 \text{ resources}$$

but we need minimum no of resources.

$$R = 2$$



$P_1 \quad P_2 \quad P_3$

$P_1 \rightarrow P_2 \rightarrow P_3$
1
1

but not always true

$P_1 \quad P_2 \quad P_3$

1	1	1
1	1	1
X	X	X

$$R = 3$$

P_1	P_2	P_3
1 1 1	1 1 1	1 1 1
$\{x\}$	$\{x\}$	$\{x\}$

P_1	P_2	P_3
1	1	1

\rightarrow deadlock can occur

$$R = 4$$

P_1	P_2	P_3
1 1 x	1 1 $\{x\}$	1 1 $\{x\}x$

\rightarrow Deadlock never occurs.

$$\begin{matrix} (\min \text{ req}) + 1 \\ 3 + 1 = 4 \end{matrix}$$

Max^m resources allocated still deadlock - 1

$$\begin{array}{ccc} P_1 & P_2 & P_3 \\ 3 & 4 & 5 \\ 2 & 3 & 4 = (4+3+2) + 1 \\ & & \simeq 10 \end{array} \quad (\min \text{ req})$$

Max^m resources allocated still deadlock = 9

Date: _____ / _____ / _____

4.8 Create Question on Deadlock.

- Q. Consider a system with 3 processes that share 9 instances of same resource type. Each process can request a max. of 'K' instances. The largest value of 'K' that will always avoid deadlock is $\rightarrow 2$.

$$R = 4 \text{ units of } R$$

	P_1	P_2	P_3
$K=1$	1	1	1

	P_1	P_2	P_3
$K=2$	1 1	1 1	1 1

	P_1	P_2	P_3
	1	1	1
	1 1	1 1	1 1
	1	1	1

	P_1	P_2	P_3
$K=3$	1	1	1
	1	-	-
	-	-	-

Deadlock

Ans: 2.

Another method to find

'R' resources

'n' processes ($P_1, P_2, P_3 \dots P_n$)

'd' demand ($d_1, d_2, d_3 \dots d_n$)

max^m resources but still deadlock

$$\begin{array}{ccc} P_1 & P_2 & P_3 \\ 2 & 2 & 2 \\ 1 & 1 & 1 \end{array} = 6.$$

$$= 3 + 1 = 4 \text{ (free from deadlock)}$$

$$(d_1 - 1) (d_2 - 1) (d_3 - 1)$$

$$R \leq \sum_{i=1}^n d_i - n. \text{ (there will be a deadlock)}$$

$$R > \sum_{i=1}^n d_i - n \text{ (there will be no deadlock)}$$

$$R + n > \sum_{i=1}^n d_i$$

↓ $\sum_{i=1}^n d_i$ → total demand
total resource total processes

$$4 + 3 > 3 \times 2$$

$$7 > 6 \rightarrow \text{True}$$

4 processes
4 resources

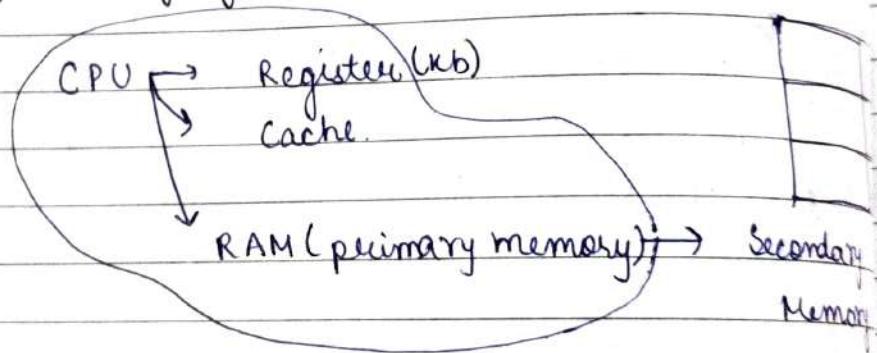
$$4+4 > 4 \times 1 \\ 8 > 4 \quad (\text{Valid})$$

$$4+4 > 4 \times 2 \\ 8 > 8 \quad (\text{false})$$

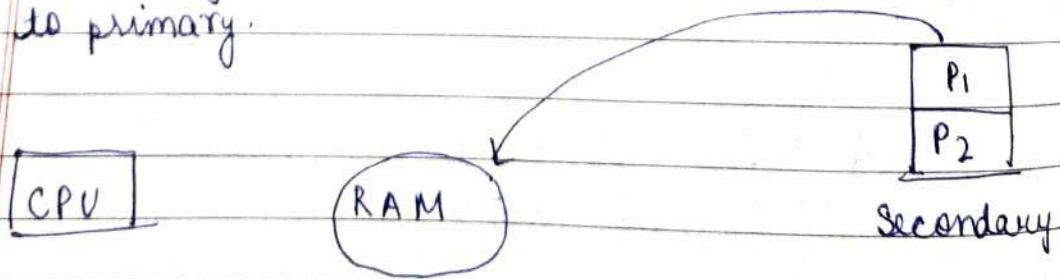
5.1 Memory Management and Degree of Multi programming

It is a kind of functionality to manage all the memory resources in a more & more efficient manner.

Method of managing primary memory:

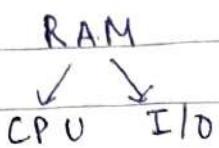


Secondary memory is not directly connected with CPU because secondary memory is a bit slower in comparison to primary.



Multiprogramming - whenever we are giving the programs in RAM being more process in main memory.

More process in RAM, degree of multiprogramming increases.
 ↓
 (more & more processes in RAM to increase efficiency)



When we bring a processes. P_1 & if it wants to have some I/O then CPU shouldn't be idle.

RAM 4MB	Process 4MB	$\frac{4}{4} = 1$ $k \rightarrow \text{I/O operation (70\%)} \\ \text{CPU utilization} = (1-k) \quad (30\%)$
------------	----------------	---

RAM 8MB	Process 4MB	$\frac{8}{4} = 2 \text{ Process.}$ $k^2 \rightarrow \text{I/O operat'n.} \\ \text{CPU utilization} = (1-k^2) \\ = 1-(0.7)^2 \\ = 76\%$
------------	----------------	---

$$k = 70\%$$

RAM 16MB	Process 4MB	$\frac{16}{4} = 4$ $k^4 \rightarrow \text{I/O operat'n.} \\ \text{CPU utilization} = 1 - (0.7)^4 \\ = 93\% \text{ approx.} \\ \downarrow \\ \text{Efficiency increases}$
-------------	----------------	---

- * Increase the no of processes as well as RAM of the system
- * OS needs to pay attention to allocation & deallocation

5.2 Contiguous and non contiguous.

Memory management Techniques

(continuous memory address allocation)

Contiguous

fixed Partition
(Static)
(make fixed slots)

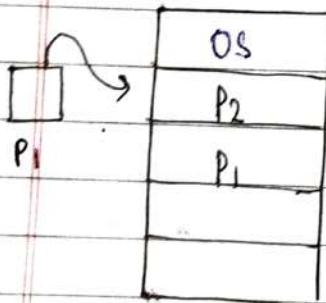
variable partition.
(Dynamic)
(provide at run time)

(Non contiguous memory allocation)

Non-contiguous

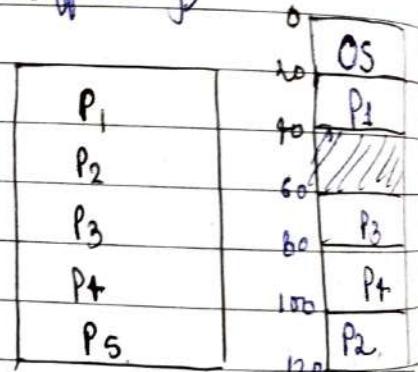
- Paging
- Multilevel Paging
- Segmented paging
- Segmentation
- Segmented Paging

Ready state process & CPU utilization (efficiency)



Memory Blocks
(RAM)

Contiguous allocation



Non-contiguous allocation



Date _____

5.3 Internal fragmentation | Fixed size Partitioning

Whenever the process are coming into RAM, how we are allocating it the space.

Points to remember.

- 1) No of partitions are fixed
- 2) Size of each partition may or may not be same.
- 3) Contiguous allocation so spanning is not allowed.

- * No of size of partitions may differ or be same. No of partition is always same.

We can put process in any partition taking into consideration its size should be less than partition size.

	OS .		OS .
0		4MB.	
1			8MB.
2		6MB.	
3		BMB.	
		16MB .	
			8MB

When we allocate any process to the partition. But after allocating process we get some extra memory which is wasted. It is called ~~process~~ Internal fragmentation.

Although we having extra extra memory for diffⁿ. partition
we couldn't accomodate a process in pieces for that
fragments.

Limitations

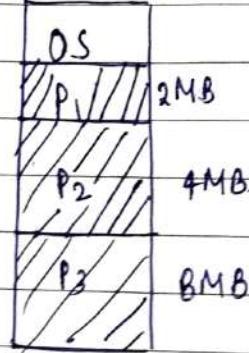
- 1) Internal fragmentation
- 2) limit in process size.
- 3) limitation on degree of multiprogramming.
We can't bring more and more processes.
- 4) External fragmentation

5.4 Variable Size Partitioning

Ram is empty: At run time processes are allocated to
the Ram at runtime

$$P_1 = 2 \text{ MB}$$

$$P_2 = 4 \text{ MB}$$

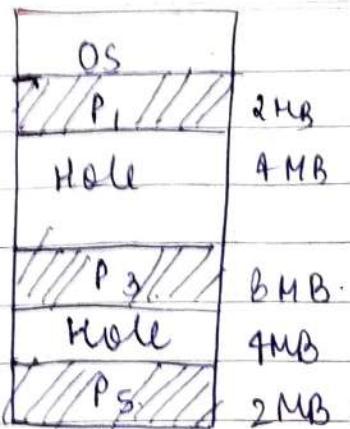


Notes

- 1) No internal fragmentation
- 2) No limitation on number of processes.
- 3) There is no limitation on the process size.

Limitations

- 1) External fragmentation occurs. We can remove it using compaction (one process is copied & pasted to other location) but it takes very much time.
- 2) Deallocation creates Hole. Allocation & deallocation is difficult.
- 3) list of holes is needed



In which we should put which process.

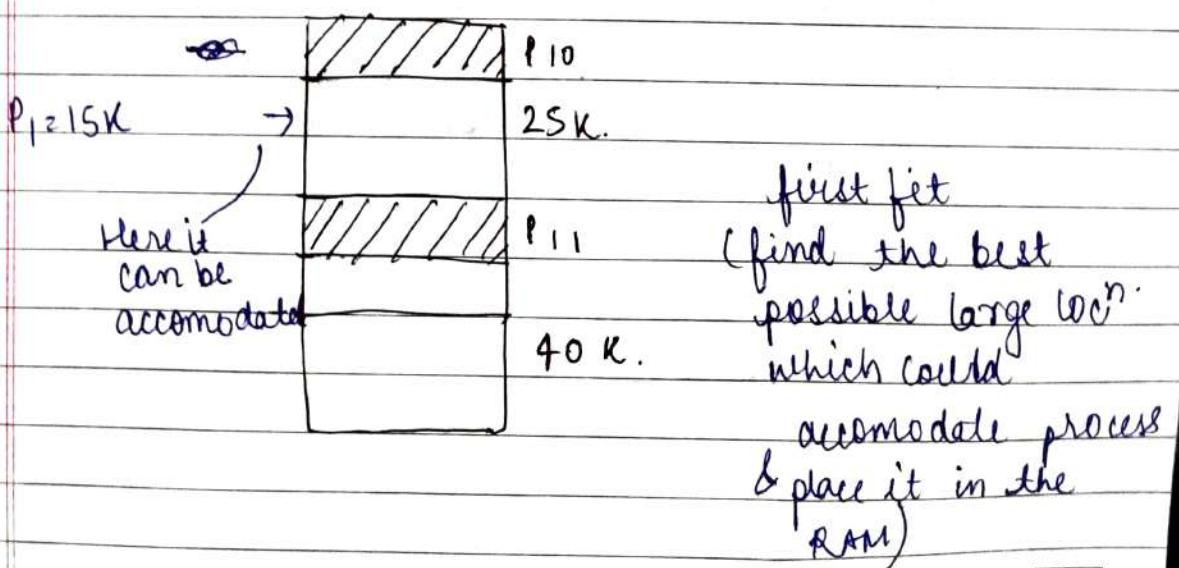
5.5 First fit, Next fit, Best fit Worst fit

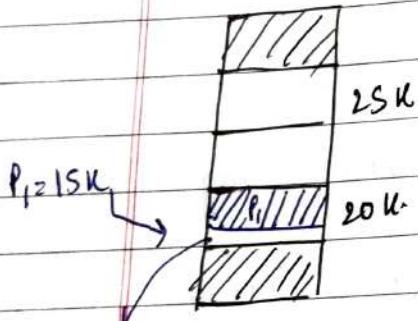
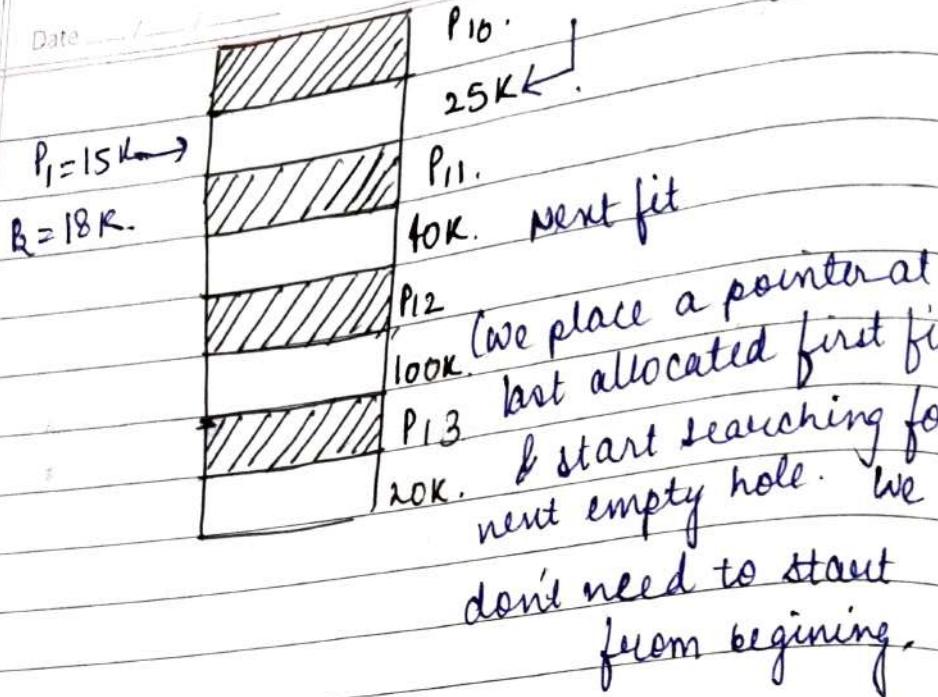
First fit \rightarrow Allocate the first hole that is big enough

Next fit \rightarrow Same as first fit but start search always from last allocated hole.

Best fit \rightarrow Allocate the smallest hole that is big enough

Worst fit \rightarrow Allocate the largest hole.



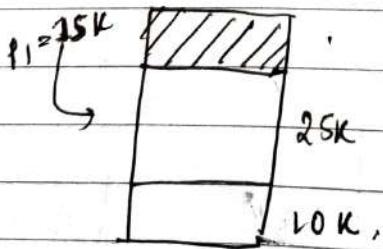


Interval fragmentation
is least

Best fit

It would search the entire list & then return the hole where there is min^m fragmentation.
It is slow

Worst fit - It would search the entire list but would search the biggest hole among every partitions. (Slow due to searching of entire list).

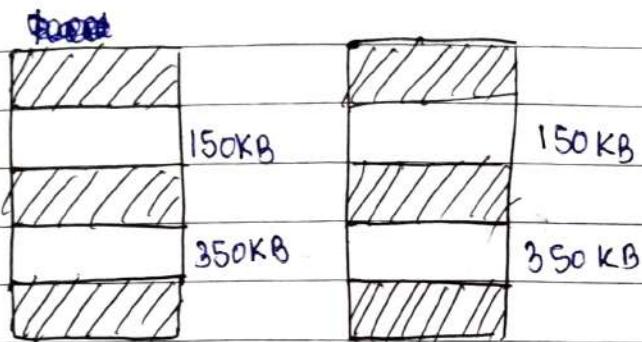




5.6 Create Question on fits

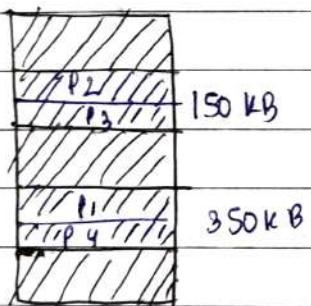
Q. Request from processes are 300K, 25K, 125K, 50K respectively. The above req. could be satisfy with.

- a) best fit but not first fit
- b) first fit but not best fit
- c) Both
- d) None

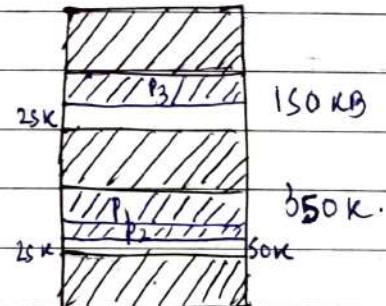


For first fit

P ₁	300 K
P ₂	25 K
P ₃	125 K
P ₄	50 K



For best fit



P₁, P₂, P₃, P₄
300 25 125 50
P₄ can't be accomodated

5.7 Create 2007 Question on fits

Date _____

Best fit

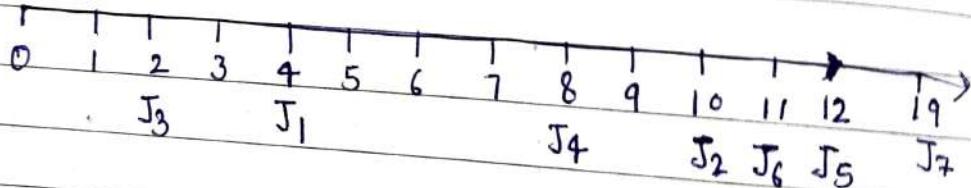
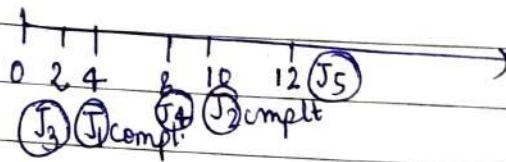
Req. No	J ₁	J ₂	J ₃	J ₄	J ₅	J ₆	J ₇	J ₈
Request size	2K	19K	3K	6K	6K	10K	7K	20K
Usage time	4	10	2	8	4	1	8	6

Calculate the time at which J₇ will be completed
 a) 17 b) 21 c) 19 d) 20

M/M

0	4K
1	8K
2	20K
3	2K

0	J ₃	4K
1	J₅	8K
2	J₄ J₆ J ₇	20K
3	J ₁	2K

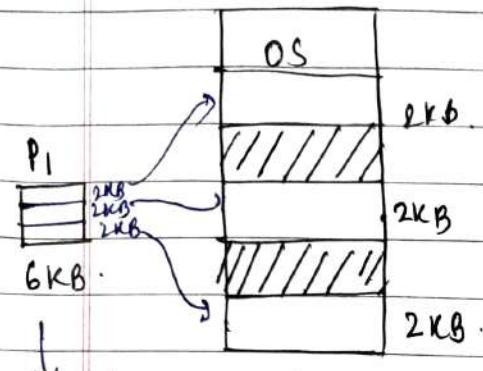


In case of J₅ & J₆ we would take out the one which is completed first. And then put it & calc. the final time by initial + usage-time.



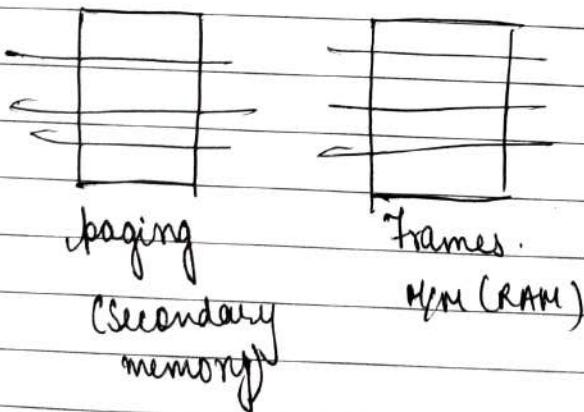
5.8 Need of Paging (Non contiguous memory Allocation)

A process can be spanned & put up in fragments in different memory location.



process is divided & put M/M
divided & put in diff. memory locations. So a process is divided before it gets memory in pages.

When we do partition of process in secondary memory it's called paging while when we do partition of process in RAM it is called framing.



JHANK You