

Date \_\_\_ / \_\_\_ / \_\_\_



# Operating System

## Notes

— by Riti Kumari

Please Share And Help others ::

# Operating System

dec D

## 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

sequential

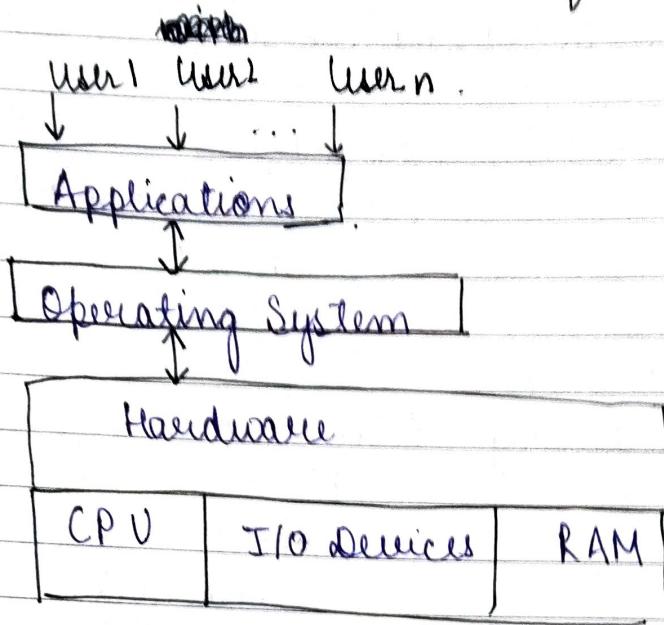
Random

linked

8. File management & Security

### 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

- ) Resource management ( kis user ko kitne particular time ke liye hardware ko provide krsana hai ).

Date: / /

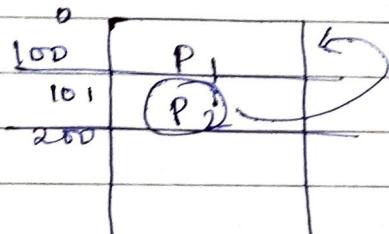
2. Process management - Managing & executing multiple process at a time. we use CPU scheduling

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

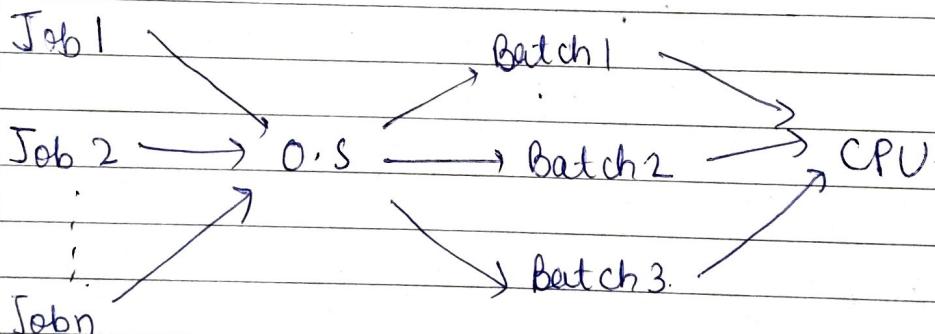
## Batch Operating System

### 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.

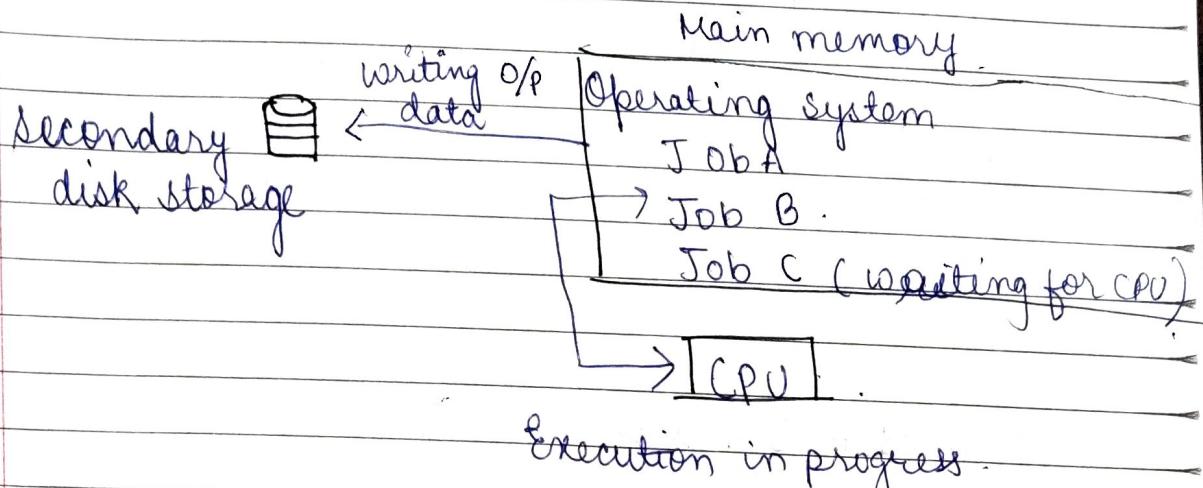


Same type of Jobs batch together and execute at a time.

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

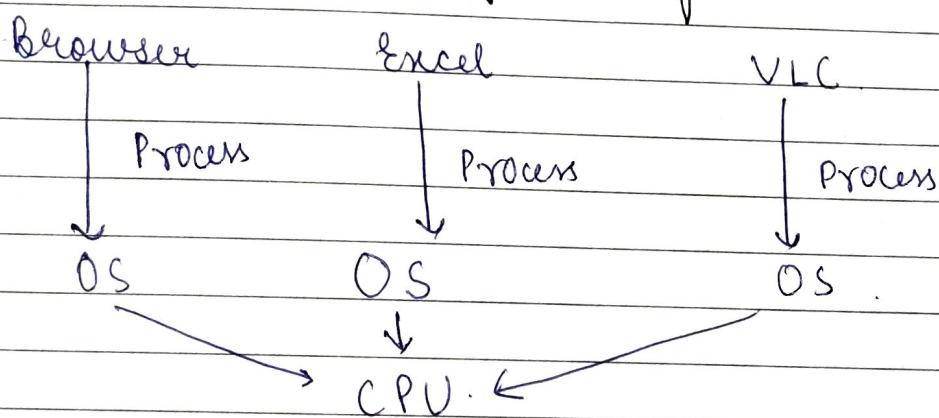


## L-1.3 (Multiprogramming & Multitasking system)



Execution in progress

## Multiprogramming OS / Time sharing



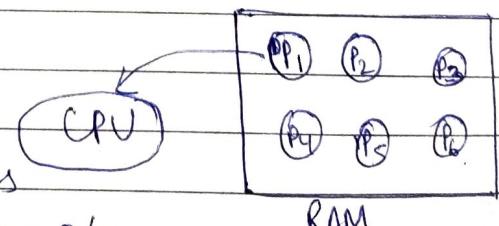
## Multitasking OS

### 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



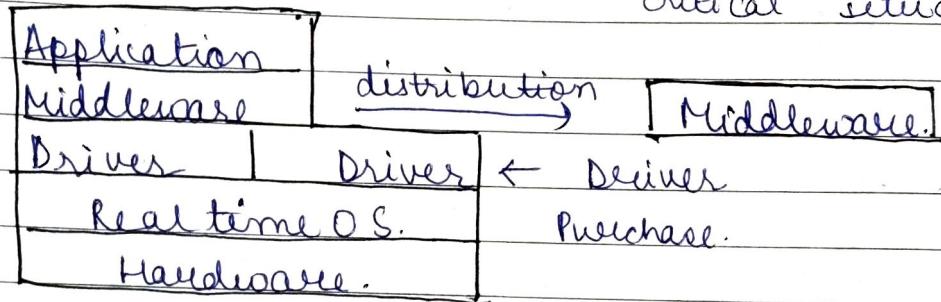


Multitasking / Time sharing - Provides particular time for every process. If its complete its great but if not it moves to other process P<sub>2</sub> & schedule P<sub>1</sub> for further.

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

#### Dec 1.4 Types of OS (Real time)

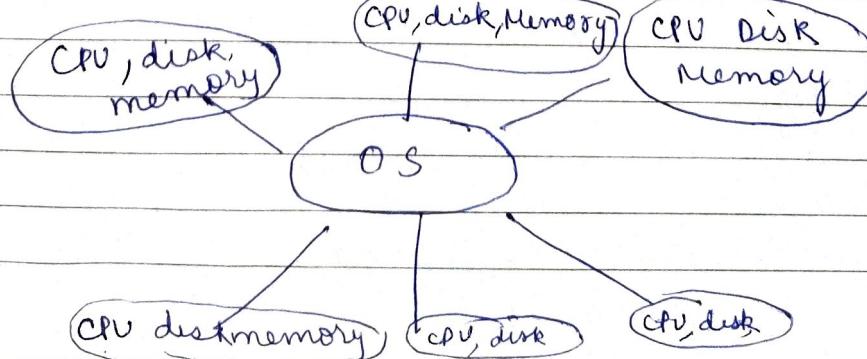
Real time OS → Hard (Restriction on time is more)  
soft Eg - In gaming no critical situation



Immediate output

- i) No delays

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

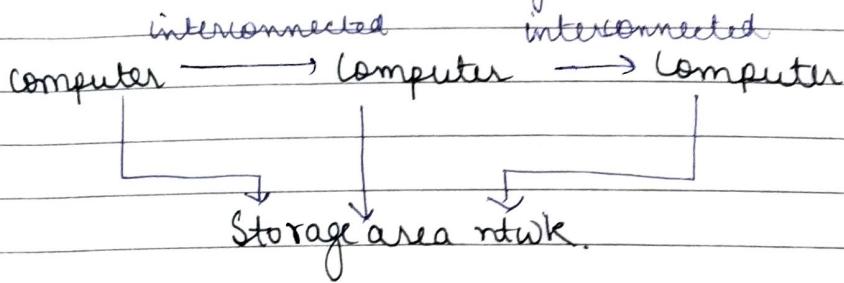


It uses multiple central processors to serve multiple real-time applications & multiple users using telephone wires.

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

Group of connected computer

\* fault tolerance, scalability

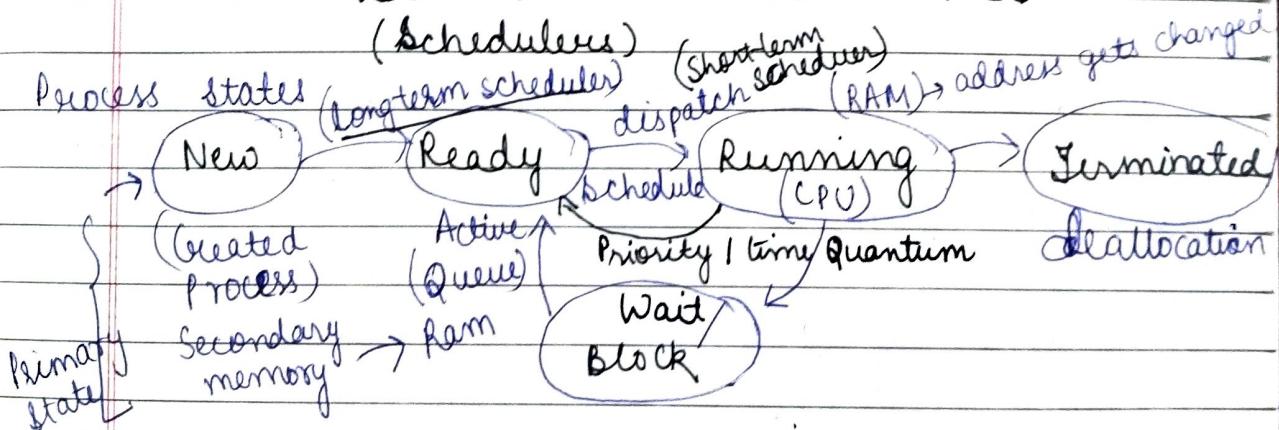


Embedded OS - It is a special purpose computer designed to perform one or few dedicated fun'n's. with real time computing. Works on a fixed functionality.

CPU

I/p → 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.

X — X

## Syllabus of OS.

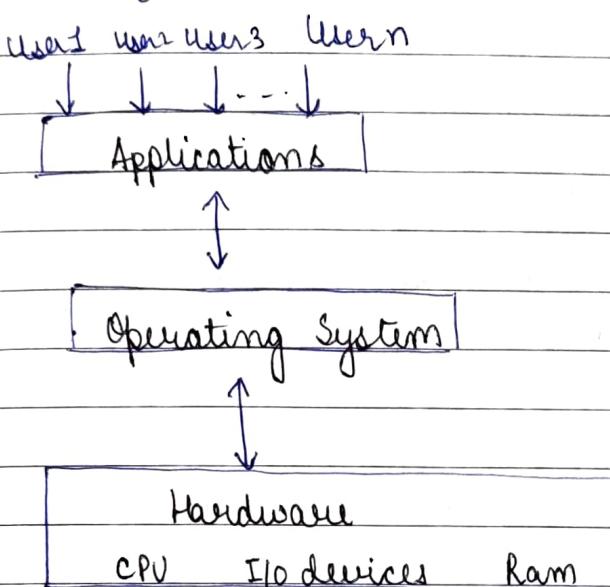
1. Basic Introduction : what is O/S, goals, need, system calls, diff' n OS.
2. Process Mdg : what is process, Algo - FIFO, SJF, Preemptive, Non preemptive, Round Robin
3. Deadlock - Banker Algo
4. Concurrency control - Semaphore, msg passing, race cond'n.
5. Memory - Paging, Segmentation, Fragmentation, virtual memory, Page Replacement.
6. Disk Mdg - C-look, Algo, PCFS, SSCF, look.
7. File mg : NIFS, FAT 32, Seg., Random
8. Unix System : commands.
9. Case study : MS DOS
10. Security - Attacks, Firewall, exception



### 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.



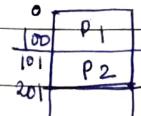
Throughput - No of tasks executed per unit time.  
 Eg - Linux

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.

- Date \_\_\_\_\_
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 hard disk 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



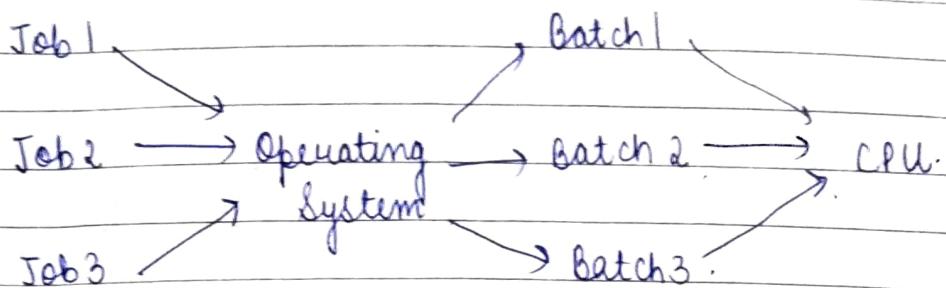
Windows (cmd) } system calls  
Linux (terminal) } (Read, open, write)

## 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

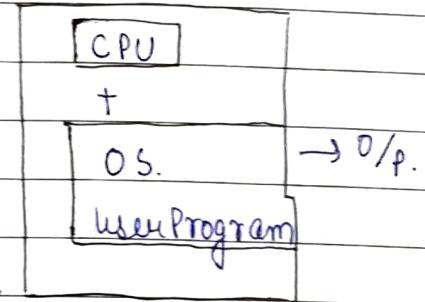


~~before~~

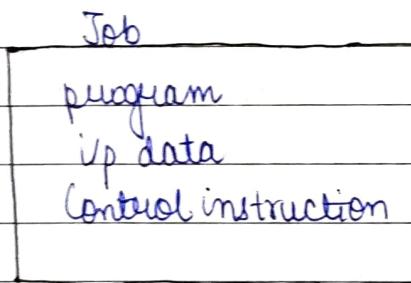
before batch OS., we used mainframe computers

- 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.

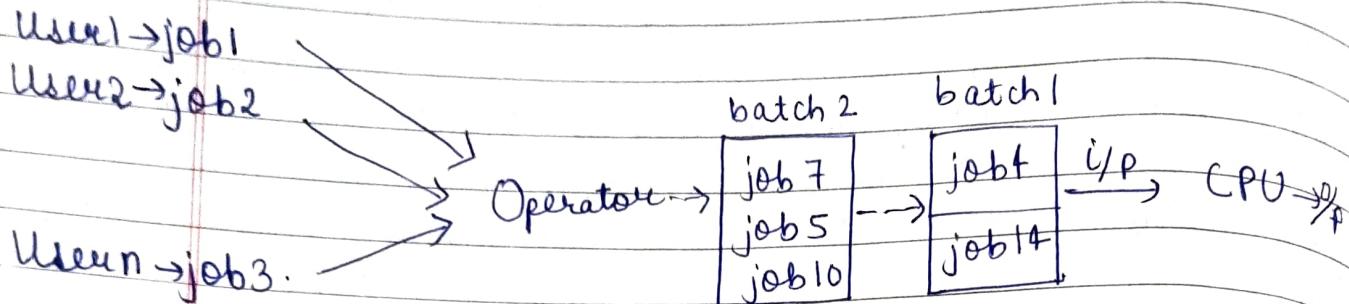


- 3) I/p ~~Punch~~ card O/p (punch card)

Major problem was.

- 1) Speed mismatch (user & CPU)
- 2) less memory
- 3) every job has diff<sup>n</sup> req<sup>m</sup>

## 1. Batch processing (e.g.: payroll system)



- 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.

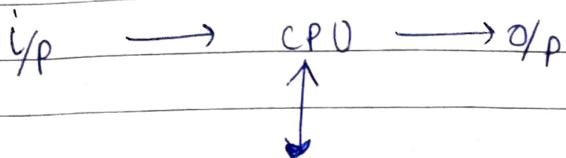
### Disadvantage

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



## Spooling (Simultaneous peripheral device)

↑  
I/p or O/p



Secondary storage.

temporarily process is stored in secondary storage.

### Advantages

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

### Spooling

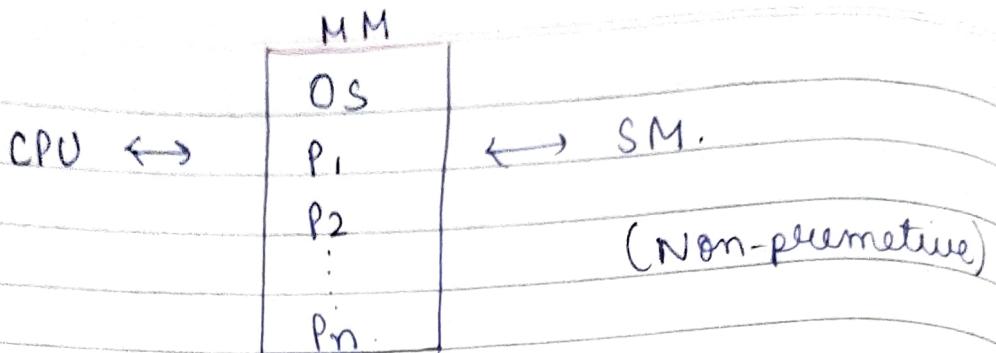
uses the hard disk  
as a large spool  
(spool is a temporary  
storage area in  
hard disk)

### Buffering

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

1.3

2. Multiprogramming O.S



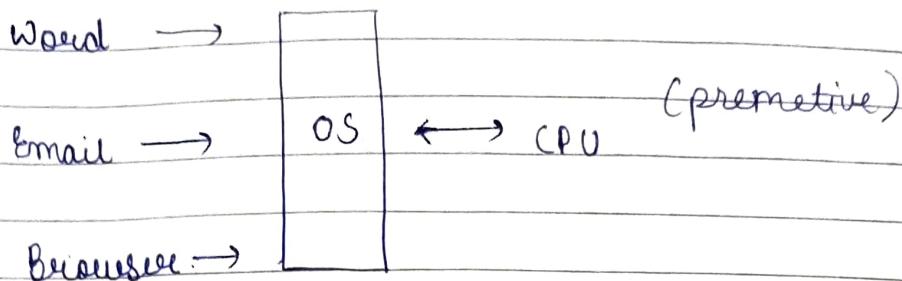
- 1) Maximize CPU utilization
- 2) Multiprogramming means more than one process in main memory which are ready to execute.
- 3) Process generally requires CPU time & I/O time. So if running process perform I/O or some other event which don't require CPU then instead of sitting idle, CPU makes context switch & pick some other process & this idea will continue.
- 4) CPU never <sup>remains</sup> 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. Multi-tasking OS. (e.g Unix)

# Time sharing / Fair share / R.R

Q3



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<sup>m.</sup> utilization of devices & system
- 2) Better task shifting
- 3) Systems are error free

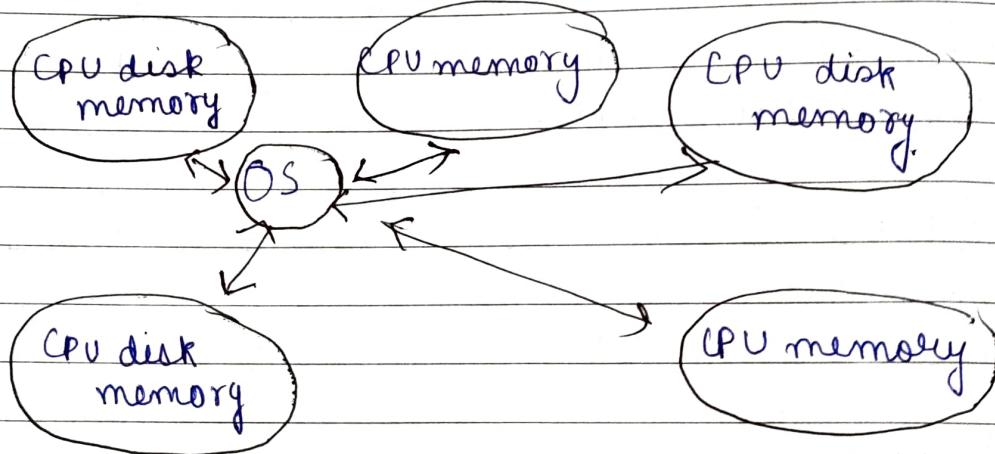
### Disadvantages

- 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.

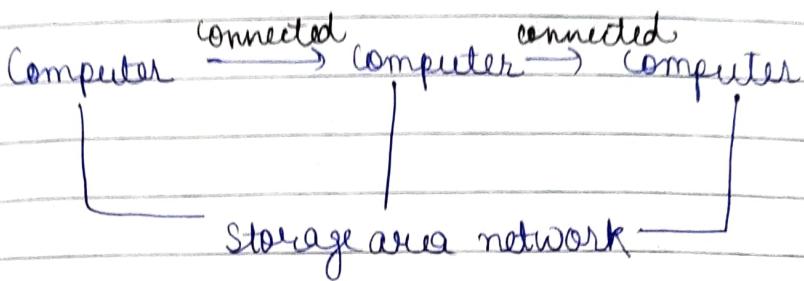


### Advantages

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

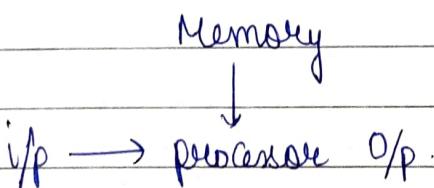
## Disadvantages

- 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.



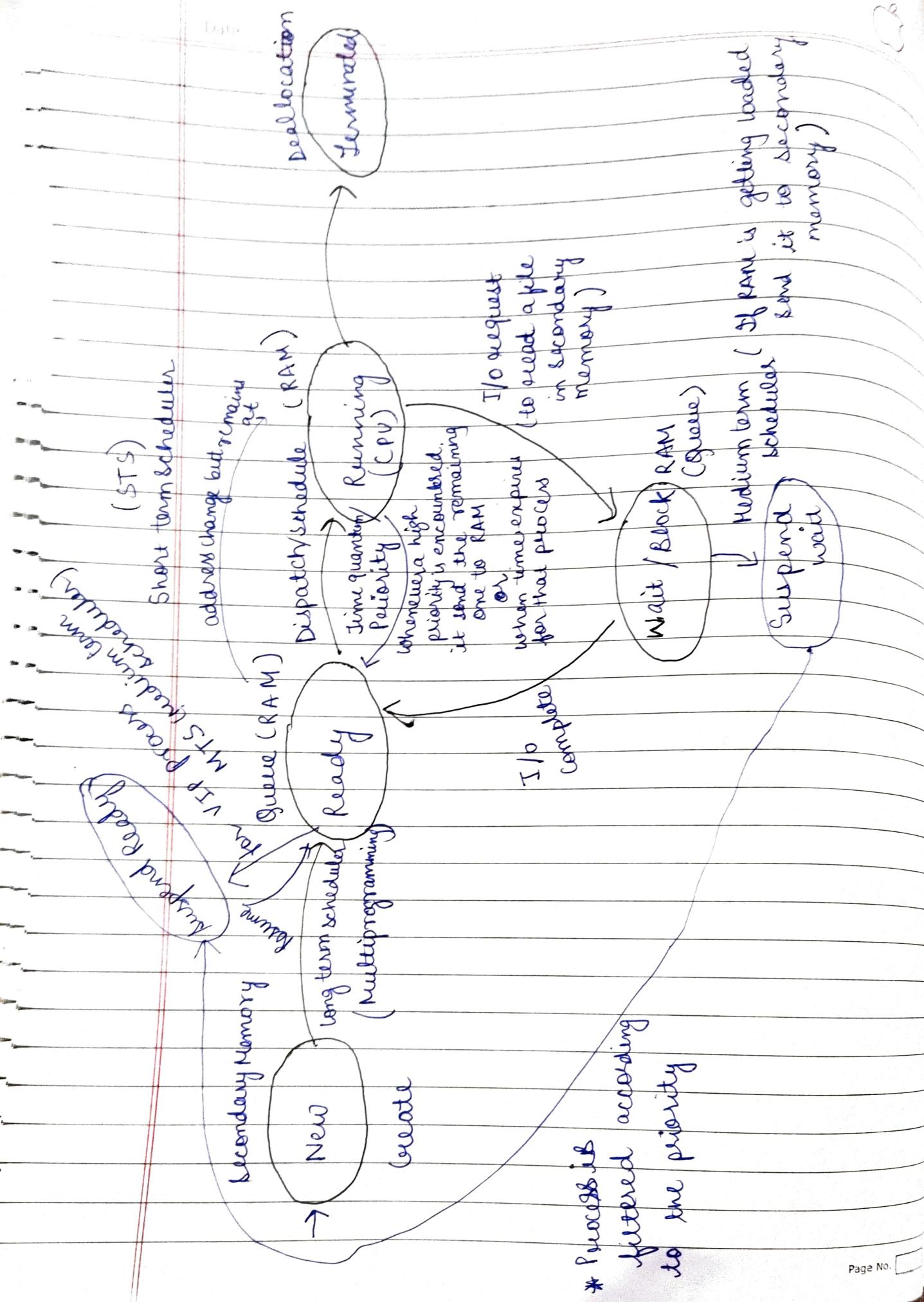
5) Embedded OS. - An OS in which we work on embedded systems. It is designed to perform a specific task for a device that is not in computer.

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~~ rwx
- chmod go+r note
- chmod ugo=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	rwx	r - 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      3) chmod 333 note  
 2) chmod 666 note      4) chmod 444 note

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

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

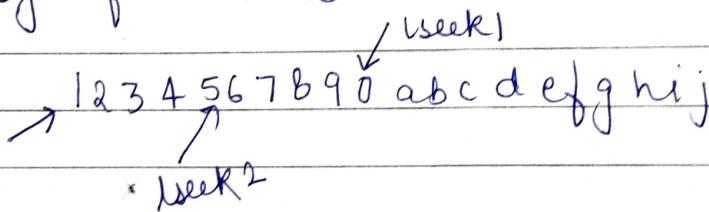
1 2 3 4 5 6 7 8 9 0 abcdefghij  
 \*seek is used 2 times sequentially.

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.      c) 10  
 (x) 5      d) 15.

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



1) seek\_cur → 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.

L - 1.7

## System calls in OS.

User mode      system call      → Kernel mode

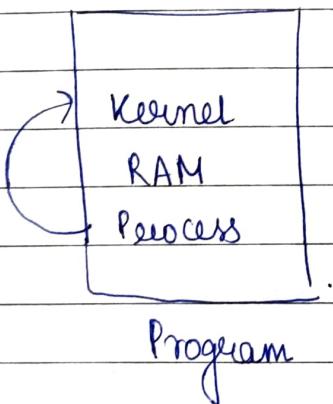
Q + 4      monitor, print(6)

In Linux we can use system call directly.

printf() → accessing system call

System calls.

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



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

for hardware device

- 3) Information → Process or regarding device attributes

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

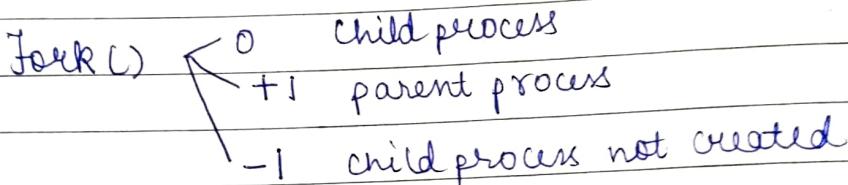


4) Process control → Program being loaded  
→ load, execute, abort, fork, wait,  
signals, allocate etc.

5) Communication Control- Interprocess communication  
→ pipe(), create/delete connection, shmget()

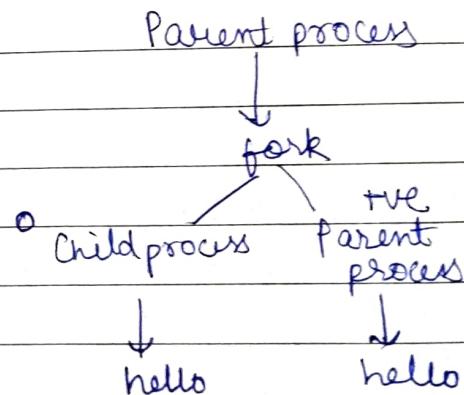
### L-1.8 Fork system call

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



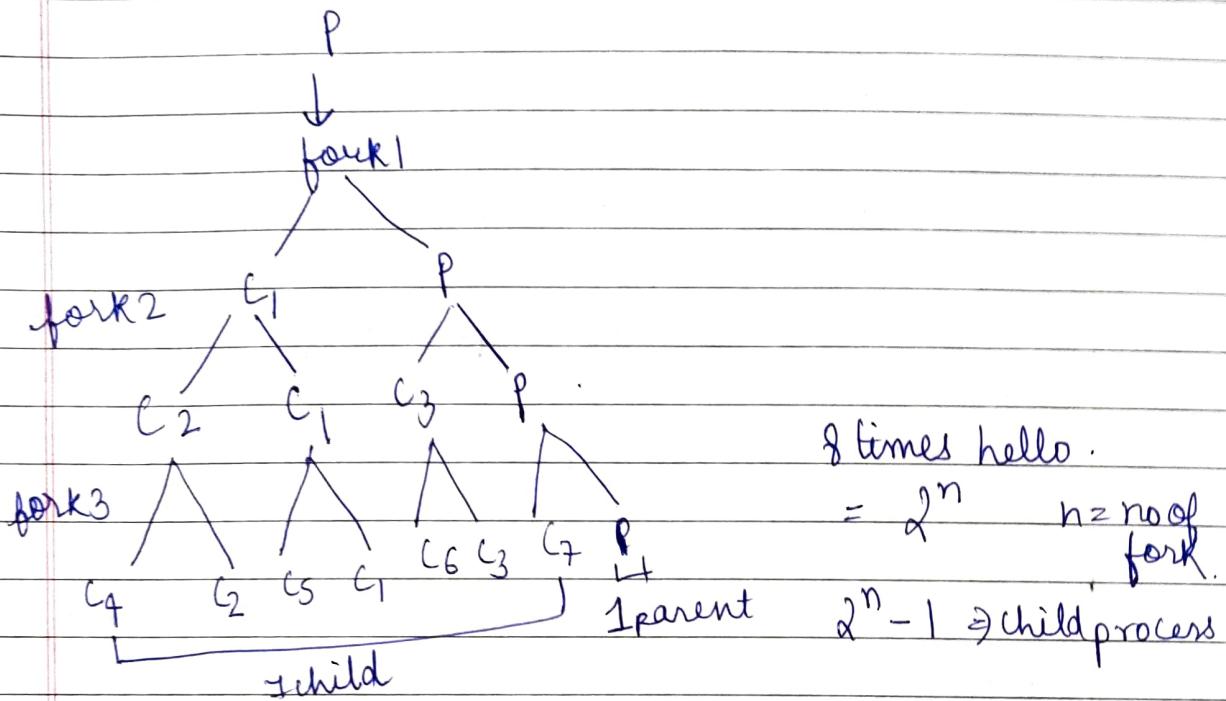
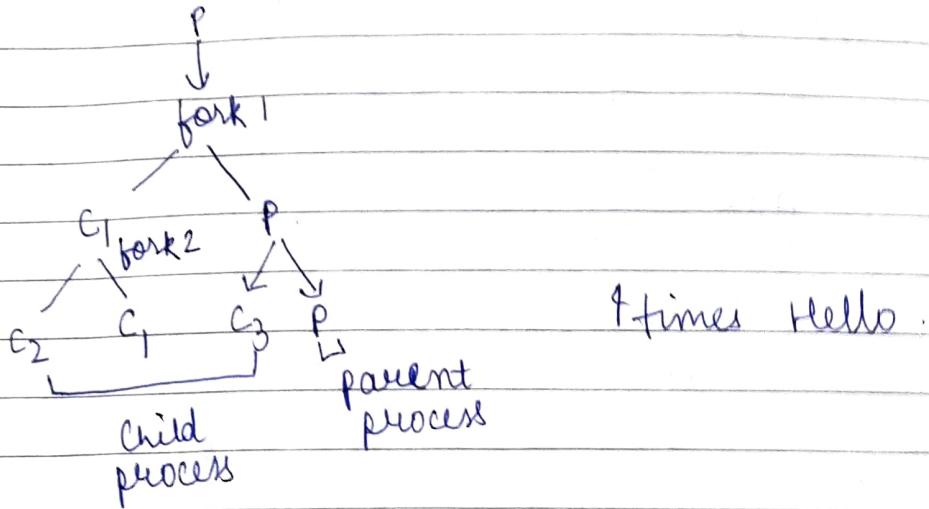
```

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;
}
```

~~fork() & fork()~~  
Date 0 / 7  
(false)

~~fork() & fork()~~  
+ve

~~fork() & fork()~~  
0 / 7  
+ve

~~fork() & fork()~~  
+ve +ve

+ve  
tue  
0 / 7  
+ve

~~fork()~~

tue  
0 / 7  
Hello

4-time hello.

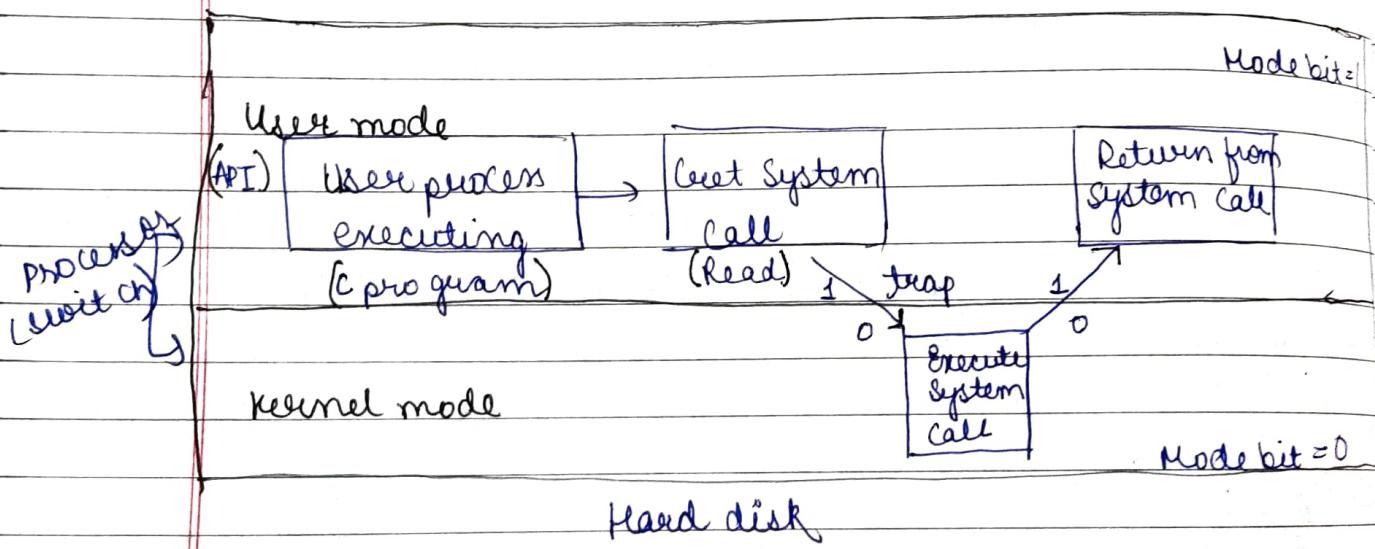
~~fork (parallel processing)~~

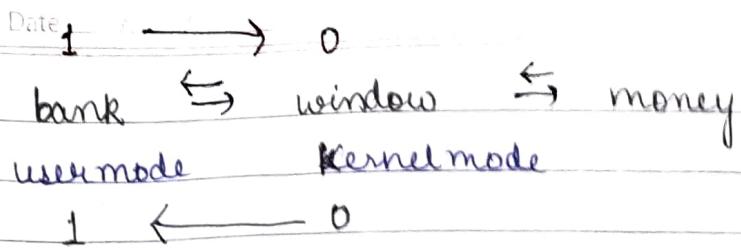
C<sub>1</sub>  
Hello  
C<sub>2</sub>  
Hello  
C<sub>3</sub>  
Hello

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

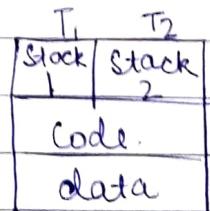
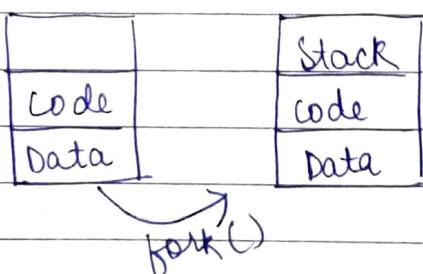
Process - It is a heavy weight task ] multitasking  
 Thread - It is a light weight task ] environment

### Process

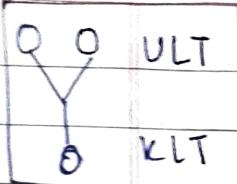
- 1) System calls involved
- 2) OS treats diff<sup>n</sup> process differently.
- 3) diff<sup>n</sup> process have diff<sup>n</sup> copies of data, files, cache
- 4) Context switching is slow
- 5) Blocking a process-not block other
- 6) Independent

### 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

(context switch) Process  $\rightarrow$  KLT  $\rightarrow$  ULT

### Kernel level thread

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

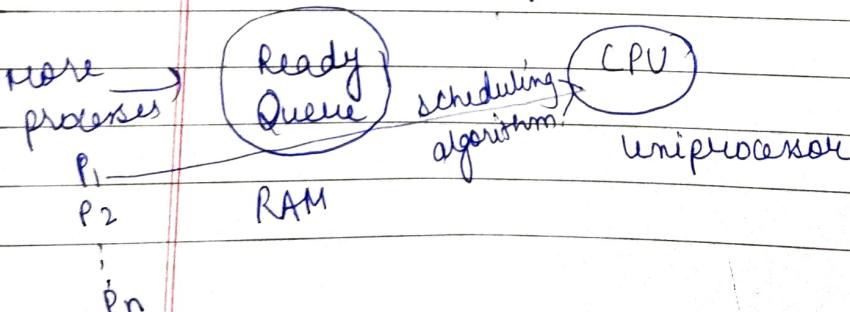
- 2) Kernel level threads are slower than user level

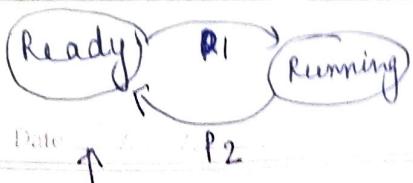
- 3) Context switching is slower.

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

## L - 2.1 Process scheduling Algorithms

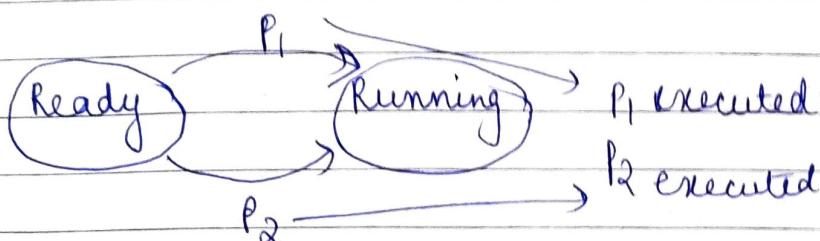
A way of selecting an algorithm from ready queue & putting it on the CPU.





- Reasons
- 1) Time quantum (provided time for P<sub>1</sub>)
  - 2) Priority process

- \* Preemptive  $\rightarrow$  If a process is taken out from ready queue off RAM and is put in running process (queue) (CPU). Then we can stop the process P<sub>1</sub> & give another process p<sub>2</sub> & send p<sub>1</sub> to ready queue. It provides responsiveness.
- \* Non preemptive  $\rightarrow$  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
  - $\leftarrow$  1) SRTF (shortest remaining time first)
  - $\leftarrow$  2) LRTF (longest remaining time first)
  - $\leftarrow$  3) Round Robin
  - 4) Priority based
    - $\downarrow$
    - Priority given
- AT
  - 1) FCFS (first come first serve)
  - 2) SJF (Shortest job first)
  - 3) LJF (longest job first)
  - 4) HRRN (High response ratio next).
  - 5) Multilevel Queue
  - 6) Priority based
- TQ

## L-2.2

### Different times in CPU scheduling

**Arrival time** - The time at which pieces enter the ready Queue or stock

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 time - Arrival time

Waiting time - Turn around - 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.

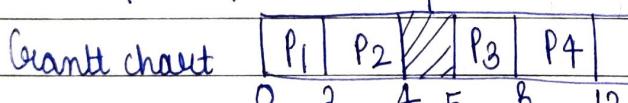
- The process which requests the CPU first get the CPU allocation first.
- Managed using FIFO queue.

CPU idle Kb/mile	Process No	AT	BT	CT	TAT (CT - AT)	WT (TAT - BT)	RT
		Arrival time	Burst time	completion time			
0	P <sub>1</sub>	0	2	2	2	0	0
2	P <sub>2</sub>	1	2	4	3	1	1
5	P <sub>3</sub>	5	3	8	3	0	0
8	P <sub>4</sub>	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.

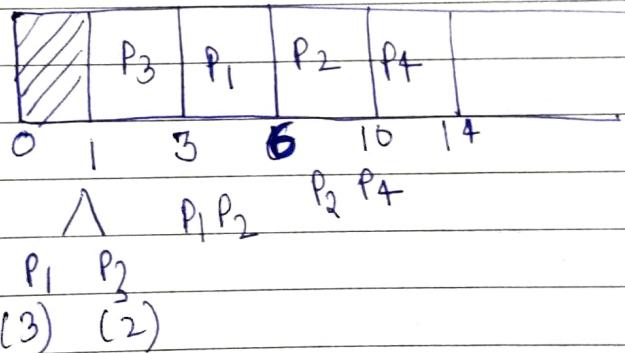
- ii) It can be preemptive or non-preemptive
- iii) It reduces the waiting time for other process.
- w) It is a greedy Algo.

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

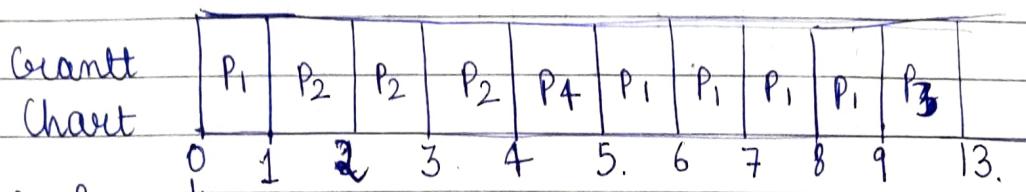
Shortest Remaining Time First  
 ↓

(Shortest job first + pre-emptive)

Process no	AT	BT	Compt	TAT	WT	RT	CPU first time
P <sub>1</sub>	0	3 5 4	9	9	4	0	0
x P <sub>2</sub>	1	2 3 2	4	3.	0.	0	1
P <sub>3</sub>	2	4	13	11	7	.7	9
x P <sub>4</sub>	4	8	5	1	0	0	4

Criteria : "Burst time"

Mode : "Pre-emptive"



0 → P<sub>1</sub>

↓

1 → P<sub>1</sub>, P<sub>2</sub>

2 → P<sub>1</sub>, P<sub>2</sub>, P<sub>3</sub>

3 → P<sub>1</sub>, P<sub>2</sub>, P<sub>3</sub>

4 → P<sub>1</sub>, P<sub>3</sub>, P<sub>4</sub>

5 → P<sub>1</sub>, P<sub>3</sub>

6 → P<sub>1</sub>, P<sub>3</sub>

7 → P<sub>1</sub>, P<sub>3</sub>

8 → P<sub>1</sub>, P<sub>3</sub>

9 → P<sub>1</sub>, P<sub>3</sub>

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

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

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

P<sub>1</sub> → 8 4

P<sub>2</sub> → 3 2 1 0

P<sub>3</sub> → 4

P<sub>4</sub> → 1

P <sub>1</sub>	P <sub>2</sub>	P <sub>2</sub>	P <sub>4</sub>	P <sub>1</sub>	P <sub>3</sub>
0	1	2	4	5	9

L-2 b

SJF with preemption ex:

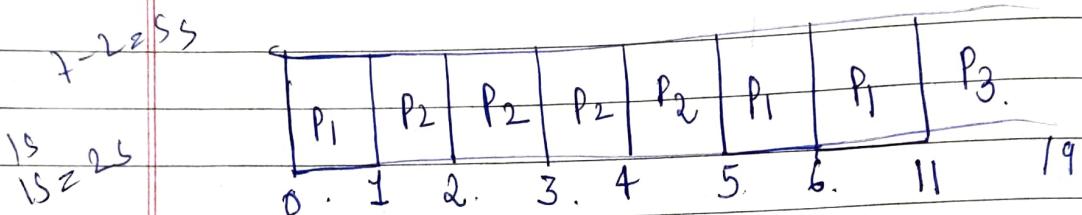
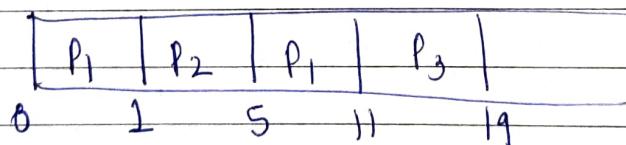
process	AT	BT	CT	TAT	WT	RT
P <sub>1</sub>	0	4.7	6.8			
P <sub>2</sub>	1	4.2	5.1			
P <sub>3</sub>	2	8.				

Criteria : burst time

Mode : preemptive .

Gantt

chart

0 → P<sub>1</sub>1 → P<sub>1</sub>, P<sub>2</sub>2 → P<sub>1</sub>, P<sub>2</sub>, P<sub>3</sub>3 → P<sub>1</sub>, P<sub>2</sub>, P<sub>3</sub>4 → P<sub>1</sub>, P<sub>2</sub>, P<sub>3</sub>5 → P<sub>1</sub>, P<sub>3</sub>



L-2.7

## Round Robin Scheduling Algo.

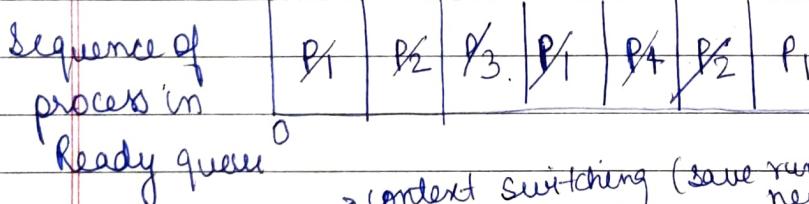
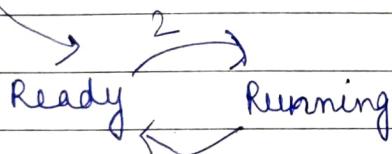
Round robin is preemptive process scheduling algorithm. Each process is provided a fix time to execute, it is called quantum. Once a process is executed for a given time period, it is preempted & other process executes for a given time period.

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

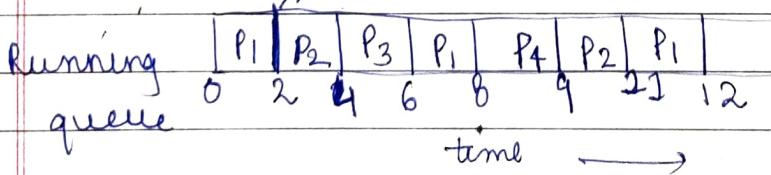
TQ=2

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~~.

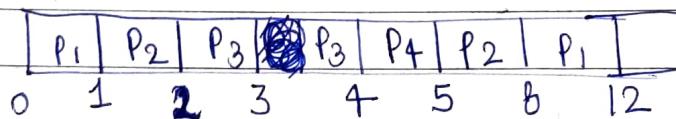
In pre-emptive scheduling, the tasks are mostly assigned with their priorities.

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 <sub>1</sub>	0	48	12	12	7	0	0
20	P <sub>2</sub>	1	34	8	7	3	0	1
30	P <sub>3</sub>	2	120	4	2	0.	0	2
40	P <sub>4</sub>	4	08	5	1	0	0	4

Higher the no. higher the priority.



0 → P<sub>1</sub>

time →

1 → P<sub>1</sub>, P<sub>2</sub>

2 → P<sub>1</sub>, P<sub>2</sub>, P<sub>3</sub>

3 → P<sub>1</sub>, P<sub>2</sub>, P<sub>3</sub>

4 → P<sub>1</sub>, P<sub>2</sub>, P<sub>3</sub>, P<sub>4</sub>

5 → P<sub>1</sub>, P<sub>2</sub>



L-2.9

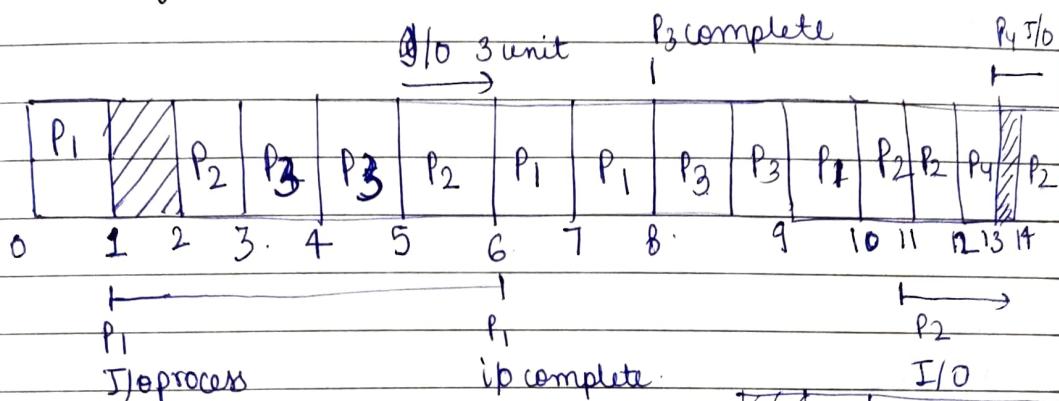
## Example of Mix Burst Time (CPU &amp; I/O both) in CPU

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

lowest the no. highest the priority.

Mode: Preemptive

Criteria: Priority based

find CT of P<sub>1</sub>, P<sub>2</sub>, P<sub>3</sub>, P<sub>4</sub>0 → P<sub>1</sub>

1 → no process

2 → P<sub>2</sub>3 → P<sub>3</sub>, P<sub>4</sub>

↓  
high  
priority

4 → P<sub>1</sub>, P<sub>2</sub>, P<sub>3</sub>, P<sub>4</sub>

5 →

6 → P<sub>1</sub>, P<sub>2</sub>, P<sub>4</sub>

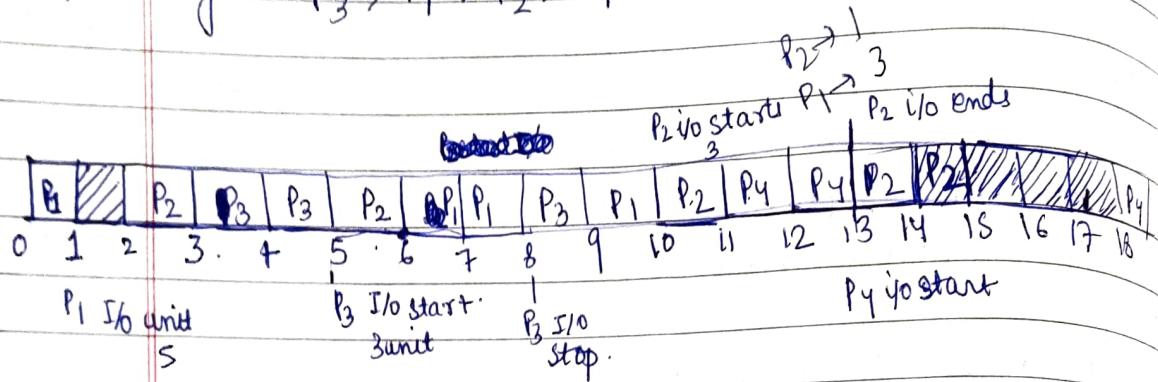
	CT
P <sub>1</sub>	10
P <sub>2</sub>	15
P <sub>3</sub>	9
P <sub>4</sub>	18

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

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

Priority	AT	Priority	CPU	I/O	CPU
P <sub>1</sub>	0	2	20	5	3'2x0 X Mode: Preemptive
P <sub>2</sub>	2	3	320	3	1 Criteria Priority
P <sub>3</sub>	3	1	2X0	3	based.
P <sub>4</sub>	3	4	210	4	1

Priority  $P_3 > P_1 > P_2 > P_4$



In Ready queue.

I/O events

$P_1 \rightarrow 6$

$P_3 \rightarrow 5$

6  $\rightarrow P_1, P_2, P_4$

7  $\rightarrow P_1, P_2, P_4$

8  $\rightarrow P_1, P_2, P_3, P_4$

9  $\rightarrow P_1, P_2, P_4$

10  $\rightarrow P_2, P_4$

13  $\rightarrow P_2$

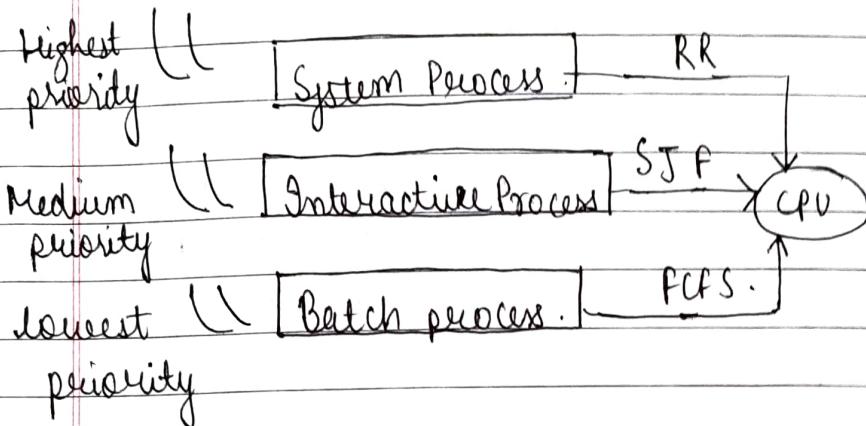
L - 2.10

Multilevel queue scheduling



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

- i) There must be different queue for diff<sup>n</sup> process
- ii) Every process can have their own algorithms.
- i) Systems calls - Round robin
- 2) Interactive calls - SJFS.



Problem :

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

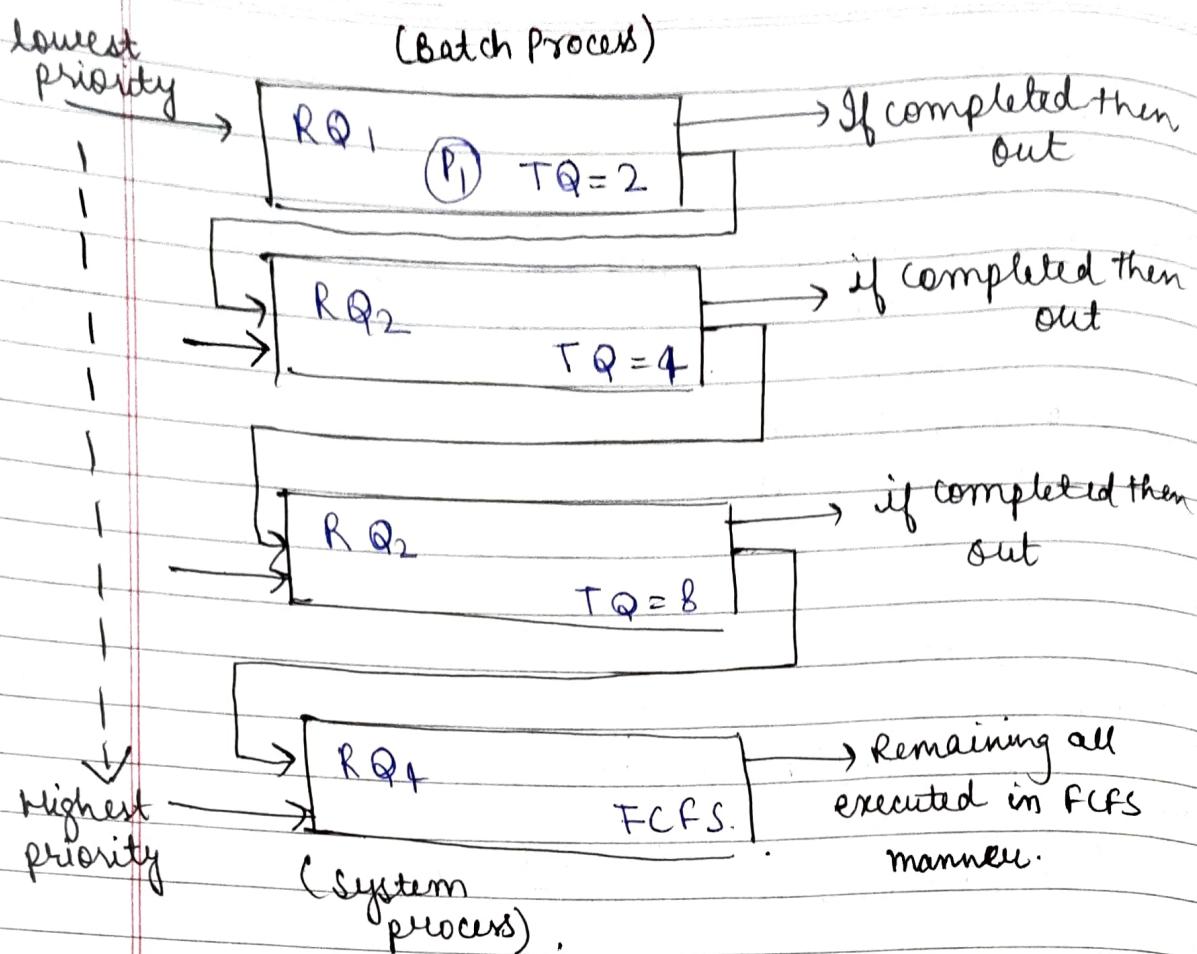
It can be resolved using multilevel feedback queue.

L-2.11

### Multilevel Feedback Queue

More system process - lowest priority waits

lowest priority - feedback.



P <sub>1</sub>	19	17	13	5
TQ = 2	TQ = 4		TQ = 8	

### Process Synchronization

Co-operative processes / <sup>Processes</sup>

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

Independent process

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

## PCB $\rightarrow$ Process control block

Date \_\_\_\_\_



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

int shared = 5; //

P<sub>1</sub>

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

context switching P<sub>2</sub>

int y = Shared;

y--; ( $y=5$ )

Sleep(1)

shared = y;

$\sim 6$   
(Terminated)

when they are parallel.

gets saved  
in PCB

Uniprocessor

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.

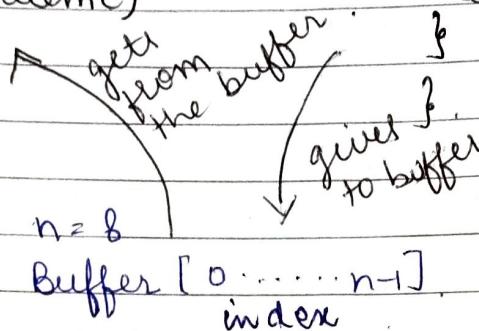
Consumer

```
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<sub>c</sub>, m[count] n = 8
2. DECR R<sub>c</sub>
3. Store m[count], R<sub>c</sub>



Producer

```
int count = 0;
```

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

```
while (true)
{
    produce item(item, p);
    while (count == n); → buff
    Buffer[in] = item p; full
    in = (in + 1) mod n so
    count = count + 1; it's
}
```

1. Load R<sub>p</sub>, m[count];
2. INCR R<sub>p</sub>;
3. Store m[count], p

OUT



0  
1  
2  
3  
4  
5  
6  
7

x<sub>1</sub>

COUNT



IN



COUNT



n = size of buffer

Case I: x<sub>1</sub> (producer is processing x<sub>1</sub>)

Out

∅ 1

$$(0+1) \bmod b = 1$$

$$(0+1) \bmod 8$$

$$1 \bmod 8 = 1$$

in

∅ 1

count

∅ 20

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

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:

$$Rp = 3. \quad \text{load } Rp, m[\text{count}] \quad I_1$$

$$Rp = 4. \quad \text{INCR } Rp. \quad I_2$$

3

If after INCR Rp the process gets preempted  
Resuming  $Rp = 4 \quad I_3$

Let  $x_1$  be consumed.

3

$$\text{load } Rc, m[\text{count}] \quad Rc = 3 \quad I_1$$

$$\text{DEC } Rc \quad Rc = 2 \quad I_2$$

$$\text{Resuming } Rp = 2 \quad I_3$$

$I_1, I_2$  consumer  $I_1, I_2$  producer.  $I_3$  consumer  $I_3$

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

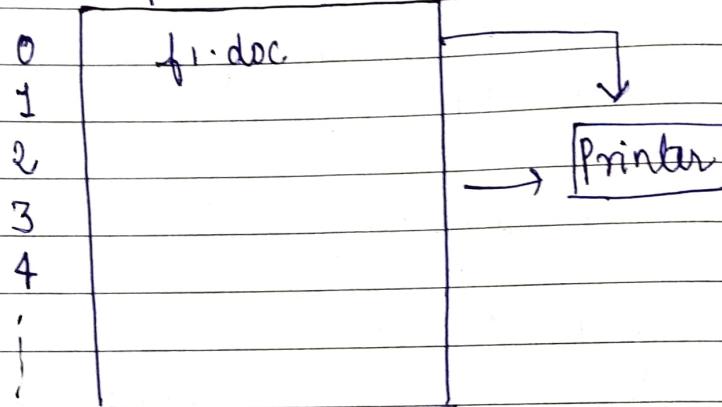
1. Load  $R_i, m[i:n]$

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

3. JNCR  $R_i$

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

Spooler Directory



IN [0:1]

↳ empty slot

Case 1:  $f1.doc$   $P_1$   
 $R_1$  [0:1]

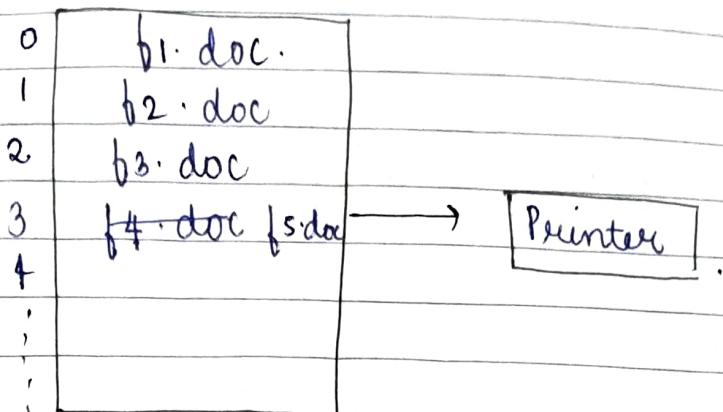
$b4.DOC$  ↑  $b5.DOC$  ↑

Case 2: Two processes  $P_1$  &  $P_2$

Cooperative process



In



In [3] 4

R<sub>1</sub> [3] 4R<sub>2</sub> [3] 4
$$P_1 \cdot I_1 I_2 I_3 | P_2 I_1 I_2 I_3 | I_4$$

P<sub>1</sub> & P<sub>2</sub> 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

P<sub>1</sub>

#include

main()

A,B → non critical section

P<sub>2</sub>

#include

main() {

X,Y

(common code)

→ non critical section

count → critical section → count

Common code

→ critical section

main()

A, B

Non critical  
section

Should  
clear  
entry  
section

Entry section

count++

critical section.

Entry section

shouldn't clear  
critical section

P<sub>1</sub>

P<sub>2</sub>

(Exit section)

(Exit section)

Different sol<sup>n</sup> for process synchronization

4 conditions

1) Mutual Exclusion }  
2) Progress }

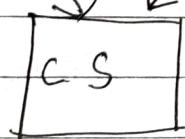
Primary

{ 3) Bounded wait

Secondary 4) No assumption related to H/W speed

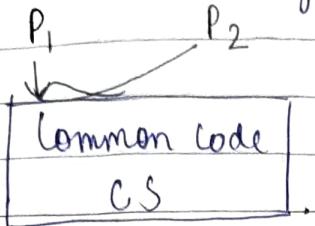
Mutual Exclusion → If P<sub>1</sub> or P<sub>2</sub> is inside critical section the P<sub>2</sub> or P<sub>1</sub> is not allowed to enter in

P<sub>1</sub>      P<sub>2</sub> critical section.



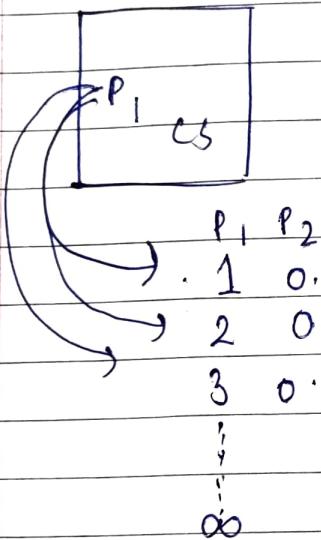
Date \_\_\_\_\_

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



P<sub>1</sub> can't enter due to some code written in P<sub>2</sub> 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.



If P<sub>1</sub> uses only the CS it will go in infinite loop

f) 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 sol<sup>n</sup> 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; ↓

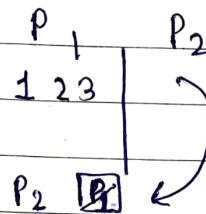
↓ exit code.

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

Case 1 : P<sub>1</sub>    P<sub>2</sub>

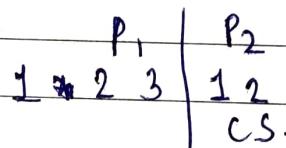
lock = 0 → CS is vacant

lock = 1 → CS is full



lock ≠ 1 ≠ 0

Case 2:



L = 0 X 1

P<sub>2</sub> P<sub>1</sub> → CS      No mutual exclusion

3.6 Test and Set Instruction

Q

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)`

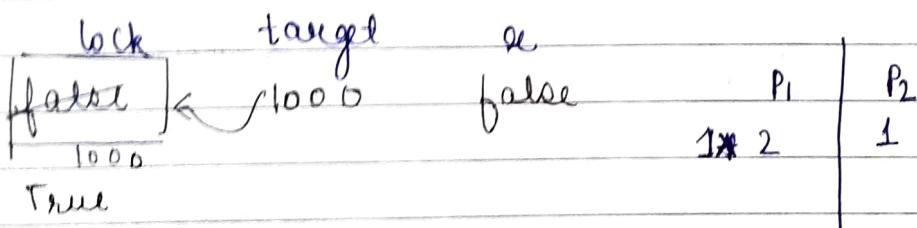
{

`boolean r = *target;`

`*target = TRUE;`

`return r;`

}



Mutual exclusion as well as Progress achieved.

### 3.7 Turn variable

1) 2 process solution

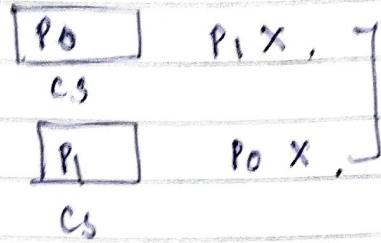
2) Run in user mode

Process "P <sub>0</sub> ".	Process "P <sub>1</sub> "
entry code $\rightarrow$ while (turn != 0)	while (turn == 1)
CS	CS

exit code  $\rightarrow$  turn = 1;

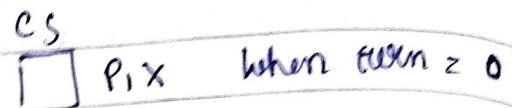
turn = 0;

int turn = 0; 1



↳ Mutual exclusion is satisfied

2) Progress (there is no progress)



3) Bounded wait  $\rightarrow$  So, there is bound wait as turn value changes.

4) Not H/W dependent

### 3.8 Semaphore

Semaphore (It is a tool to prevent race cond)

Counting      Binary.  
( $\infty$  to  $\infty$ )      ( $0, 1$ )

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)

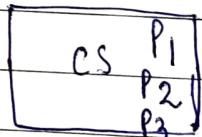
P<sub>1</sub>, P<sub>2</sub>, P<sub>3</sub>

Entry code

CS := :-

exit section

P<sub>1</sub>, P<sub>2</sub>, P<sub>3</sub>, P<sub>4</sub> (process control block)  
P<sub>5</sub>



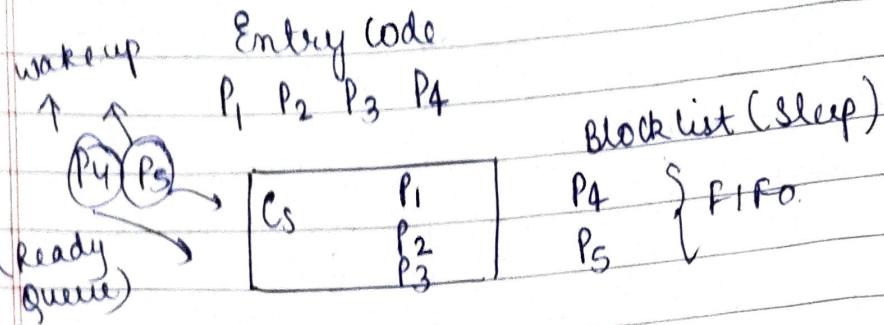
Block list (sleep())

P<sub>4</sub>

P<sub>5</sub>.

S

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



down

S

-4

↓

4 processes are  
there which  
is in CS.

Exit code

S  
 $\boxed{3} \times \infty - 1 = 2 \infty - 1$

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 3 V 1 P

17

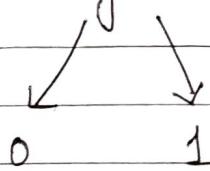
$$= 17 - 5 + 3 - 1$$

$$= 18 + 3 - 1$$

$$= 15 - 1$$

$$= 14$$

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();

}

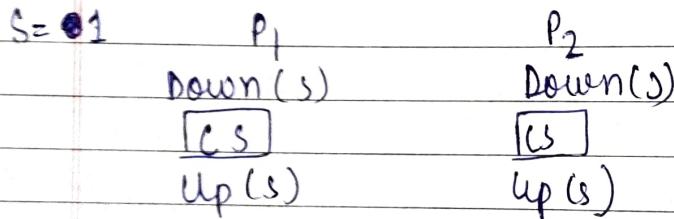
}

Down, p, wait  
Up, v, signal

$S = 1$        $S = 0$   
Down } successful sleep();  
 $S = 0$       open<sup>n</sup>. block } unsuccessful  
                  open<sup>n</sup>.

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

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

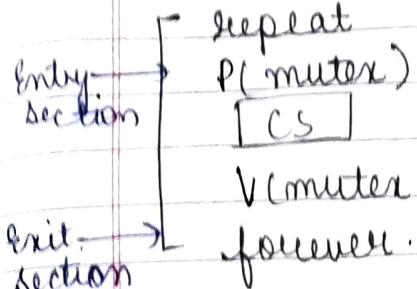


- Q. Each process  $P_i$  { $i = 1$  to 9} execute the following code.

Date

$P_i \{ i=1 \text{ to } 9 \}$

$P_{10}$



mutex  
1

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

What is the max<sup>m</sup> 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

P<sub>2</sub> P<sub>3</sub>  
P<sub>4</sub>  
(block) →  
P<sub>1</sub>

ifore P<sub>10</sub>

mutex: & V & X & & P<sub>1</sub> P<sub>10</sub> P<sub>2</sub>      BLOCK

CS  
P<sub>1</sub> P<sub>10</sub> P<sub>2</sub>

↓ P<sub>3</sub>

P<sub>1</sub> P<sub>2</sub> P<sub>3</sub> P<sub>10</sub>

P<sub>3</sub>

P<sub>4</sub>

P<sub>5</sub>

P<sub>6</sub>

P<sub>7</sub>

P<sub>8</sub>

P<sub>9</sub>

P<sub>1</sub> P<sub>2</sub> P<sub>3</sub> P<sub>4</sub> P<sub>5</sub> ← P<sub>1</sub> P<sub>2</sub> P<sub>3</sub> P<sub>4</sub> P<sub>5</sub> P<sub>6</sub> P<sub>7</sub> P<sub>8</sub> P<sub>9</sub> P<sub>10</sub> ← P<sub>10</sub> P<sub>1</sub> P<sub>2</sub> P<sub>3</sub> P<sub>4</sub> P<sub>5</sub> P<sub>6</sub> P<sub>7</sub> P<sub>8</sub> P<sub>9</sub>

max<sup>m</sup> no of process can be 10 but putting P<sub>10</sub> 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

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

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

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



For consumer.

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

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

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 C.S.)

S =  $\emptyset \neq \emptyset$

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

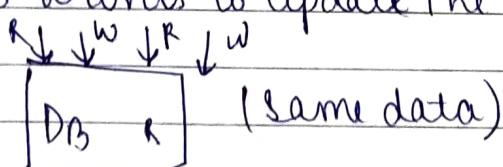
In =  $3 \neq 4$

3.12  $\rightarrow$  Solution of Readers writers problem.

We have a database used by Reader & writer

Reader - who only reads the data

Writer - who wants to update the data also.



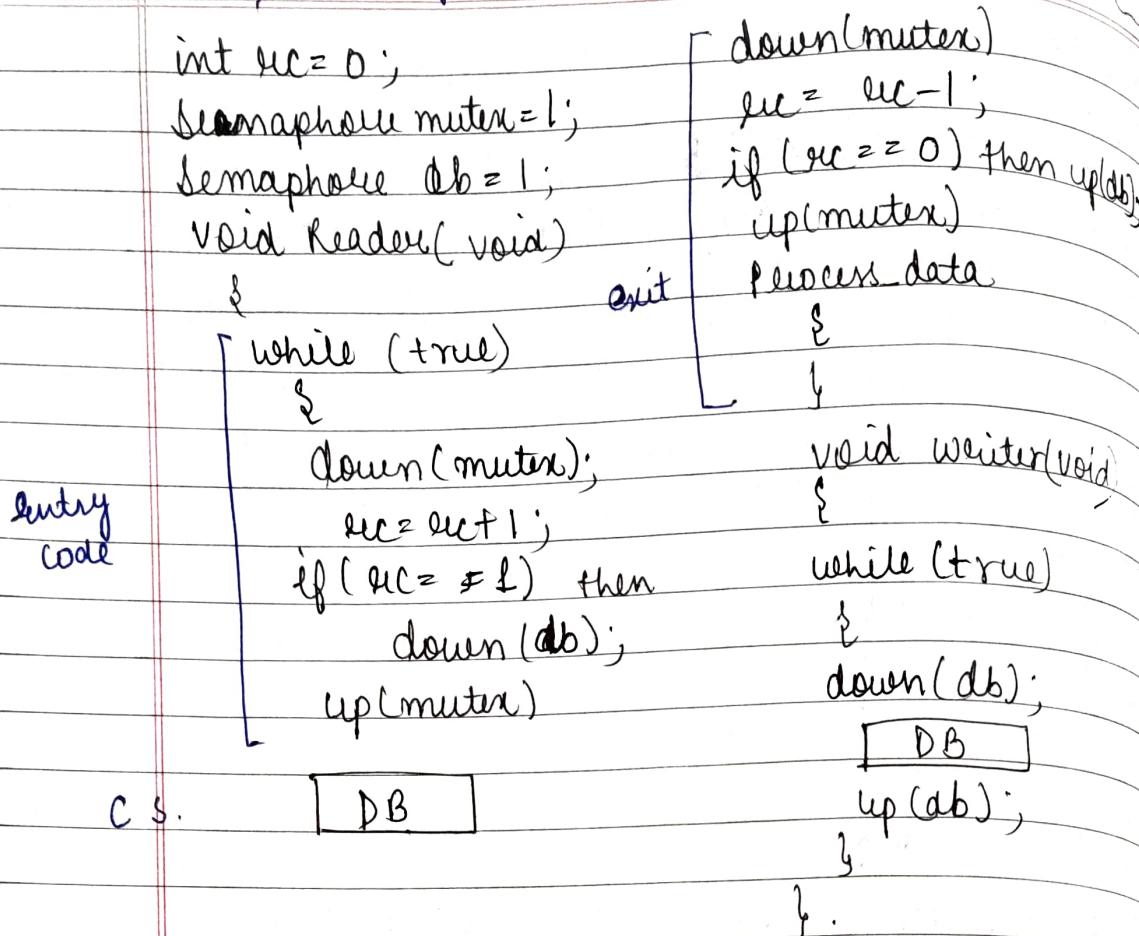
R-W (problem)

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: R<sub>1</sub> comes first

$RC = 0 \times 1$     mutex = 1 or 1    db = X0

**down(mutex)**

DB R<sub>1</sub>

For writer

$RC = 1$     mutex = 1    db = 0

**down(db)**

→ db ≠ -1 so the process gets blocked.

Case 2: W<sub>1</sub> comes first



Date: \_\_\_\_\_

$$R_C = \emptyset \times 1$$

$$\text{mutex} = X \emptyset$$

$$db = X \emptyset$$

down(db)



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

Case 3: writer-writer problem

when writer  $w_1$  comes down(db)

$$db = X \emptyset$$

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

Case 4: Read-Read

$$R_C = \emptyset \times 2$$

$$\text{mutex} = X \emptyset \times \emptyset \times 1$$

$$db = X \emptyset$$

$$\boxed{DB \quad R_1, R_2}$$

$$2 = 1 \text{ no}$$

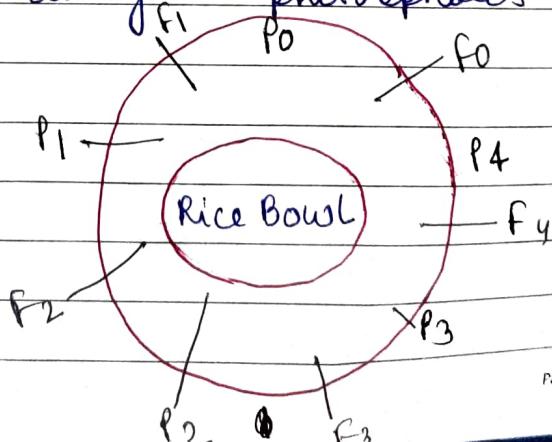
down(mutex)

$$db; X$$

up(mutex)

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:  $P_0$  comes

i = 0

left\_fork = 0 ( $f_0$ )

right\_fork =  $(0+1) \% N = 1$  ( $f_1$ )

Eating

$f_0$

$f_1$

end.

Case 2:  $P_1$

i = 1

left\_fork = 1 ( $f_1$ )

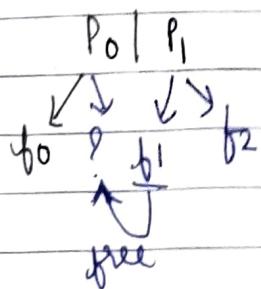
right\_fork = 2 ( $f_2$ )

Eating

$f_1$

$f_2$

Case 3:



"problem of race cond" occurs.

$S[i] \rightarrow$  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))

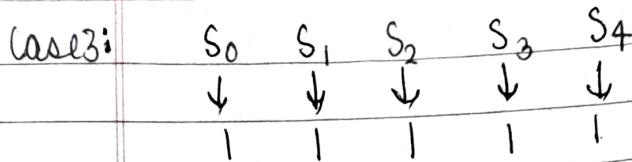
}

}.

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

$P_0 \rightarrow S_0$	$S_1$
$P_1 \rightarrow S_1$	$S_2$
$P_2 \rightarrow S_2$	$S_3$
$P_3 \rightarrow S_3$	$S_4$
$P_4 \rightarrow S_4$	$S_0$
<del>Process 6</del>	

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.

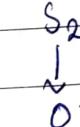
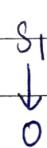


$P_0$  comes first



It gets preempt

$P_1$  comes       $P_2$  comes       $P_3$  comes

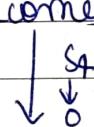


preempt

preempt

preempt

$P_4$  comes



preempt

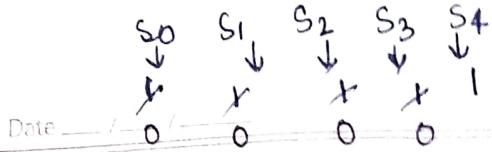
for eight fork (blocked)

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  $\rightarrow P_4$

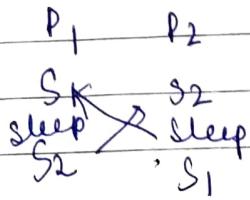
$S_0 \quad S_4$

We can change sequence of any philosopher.

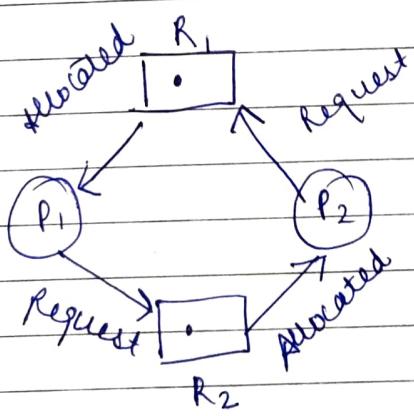
$N^{th}$  philosopher:

Wait (take\_fork ( $S_{(i+1) \bmod N}$ ))  
wait (take\_fork ( $S_i$ ));

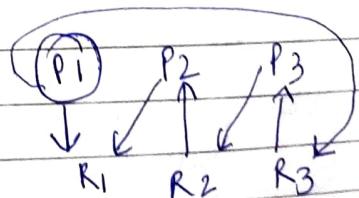
#### 4.1 Deadlock concept



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.



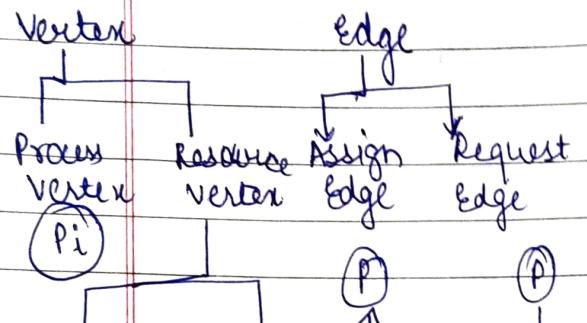
$R_1 \rightarrow P_1$   
 $R_1$  is allocated to  $P_1$   
 $P_2$  request for  $R_1$



## Necessary conditions for deadlock.

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

### 4.2 Resource Allocation Graph in Deadlock (RAG)



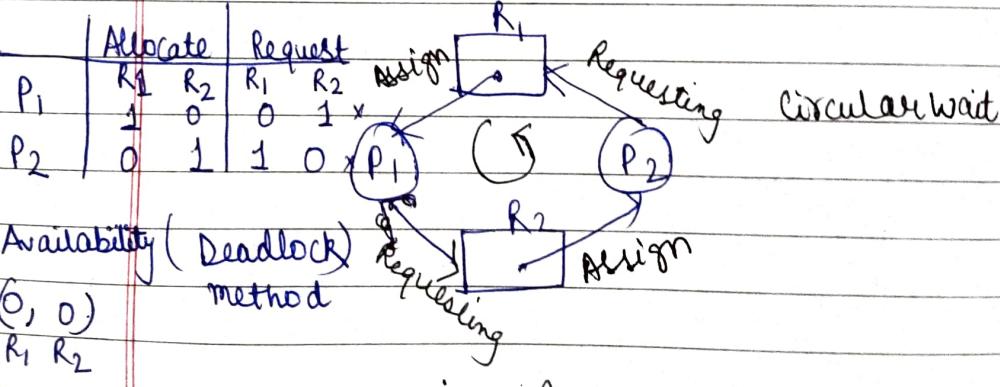
single instance

Multi instance

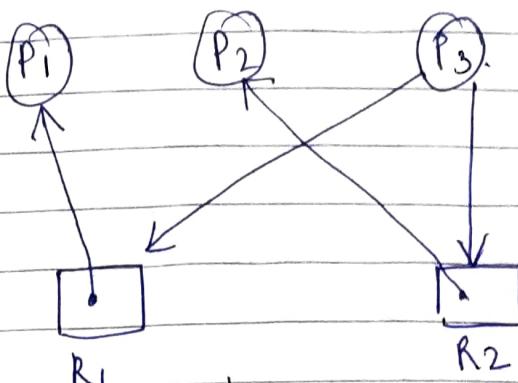
eg: CPU

Monitor

eg : Register



single Instance



	Allocation		Request		Result
	R1	R2	R1	R2	
P1	1	0	0	0	✓ Terminate
P2	0	1	0	0	✓ Terminate
P3	0	0	1	1	✓

Availability (0, 0)

(1, 0) → after P1 terminate

(1, 1) → after P2 terminate

No deadlock occurs.

Waiting  
↓  
finite      infinite  
(Starvation)      (Deadlock)

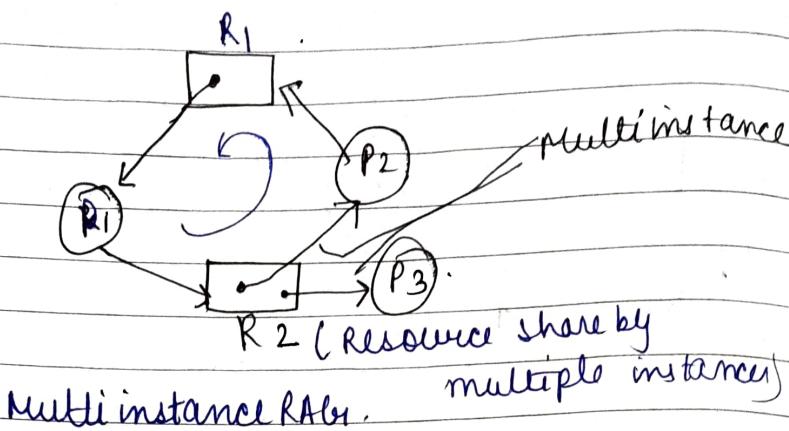
If RAG (Circular wait) cycle → Always in Deadlock

↓  
A single instance

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

Date \_\_\_\_\_ / \_\_\_\_\_ / \_\_\_\_\_

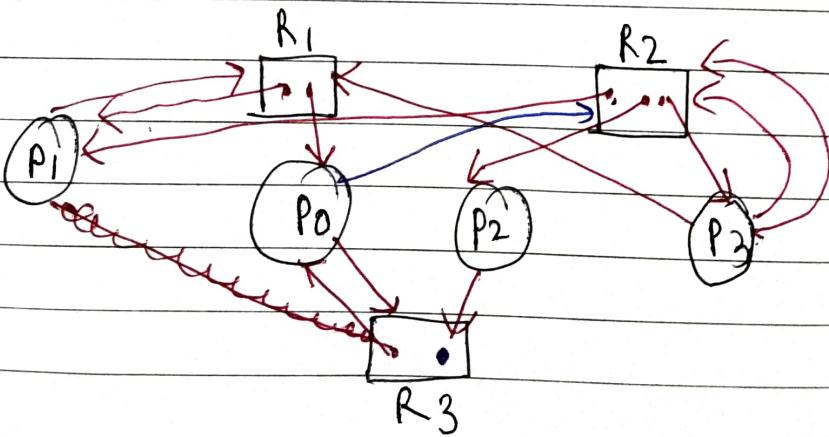
### 4.3 Multinode RAQs.



Process	Allocate	Request	
	$R_1$	$R_2$	
$P_1$	1	0	0 1 ✓
$P_2$	0	1	1 0
$P_3$	0	1	0 0 ✓

Current avail  $\rightarrow (0, 0)$   
 $\begin{matrix} 0 & 1 \\ 0 & 0 \end{matrix}$   
 $\begin{matrix} 1 & 0 \\ 0 & 1 \end{matrix}$

No deadlock





	Allocate		Request	
	R <sub>1</sub>	R <sub>2</sub>	R <sub>1</sub>	R <sub>2</sub>
P <sub>0</sub>	1	0	1	
P <sub>1</sub>				
P <sub>2</sub>				
P <sub>3</sub>				

	Allocate			Request		
	R <sub>1</sub>	R <sub>2</sub>	R <sub>3</sub>	R <sub>1</sub>	R <sub>2</sub>	R <sub>3</sub>
P <sub>0</sub>	1	0	1	0	1	1
P <sub>1</sub>	1	1	0	1	1	0
P <sub>2</sub>	0	1	0	0	0	1
P <sub>3</sub>	0	1	0	1	2	0

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

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

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

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

$$\underline{\quad 1 \quad 1 \quad 2 \quad}$$

	R <sub>1</sub>	R <sub>2</sub>	R <sub>3</sub>
P <sub>1</sub>	1	1	2
P <sub>2</sub>	1	1	0

→ No deadlock in the system.

	R <sub>1</sub>	R <sub>2</sub>	R <sub>3</sub>
P <sub>3</sub>	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 prevention 4 necessary conditions for deadlock

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

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

- 1) Mutual exclusion (there should be no sharing bet<sup>n</sup> diff<sup>n</sup> 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 bet<sup>n</sup> the resources)

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

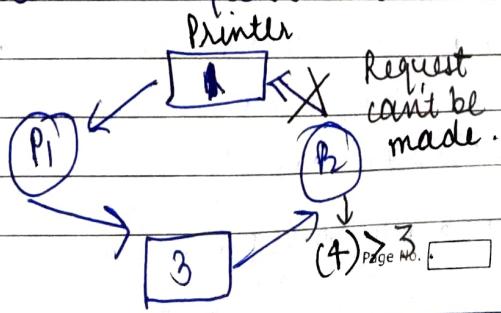
- 3) Hold wait (holding some resource & waiting for some resources).

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



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		
	(10-1)	(5-2)	(5-3)	A	B	C	A	B	C	A	B	C
P <sub>1</sub>	0	1	0	7	5	3	3(1)	3(1)	2(1)	7	4	3
P <sub>2</sub>	2	0	0	3	2	2	5(1)	3(1)	2(1)	1	2	2
P <sub>3</sub>	3	0	2	9	0	2	7(0)	4(0)	3(2)	6	0	0
P <sub>4</sub>	2	1	1	4	2	2	7(0)	4(1)	5(0)	2	1	1
P <sub>5</sub>	0	0	2	5	3	3	7(0)	5(0)	5(0)	5	3	1
+							10	5	7			
	7	2	5									

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

P<sub>2</sub> → P<sub>4</sub> → P<sub>5</sub> → P<sub>1</sub> → P<sub>3</sub> (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 <sub>0</sub>	1	0	1	4	3	1	3(1)	3(0)	0(1)	3	3	0
P <sub>1</sub>	1	1	2	2	1	4	4(1)	3(0)	1(2)	1	0	2
P <sub>2</sub>	1	0	3	1	3	3	5(1)	4(0)	3(0)	0	3	0
P <sub>3</sub>	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.

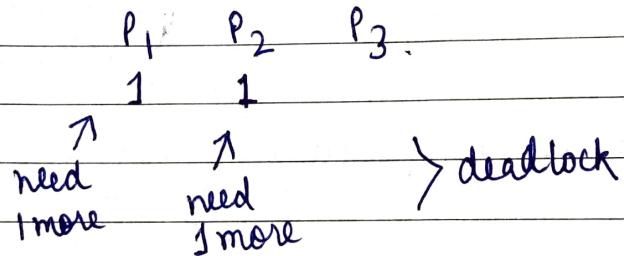
- a) 3
- b) 5
- ~~c)~~ 6
- d) 4

$P_1 \quad P_2 \quad P_3$

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

but we need minimum no of resources.

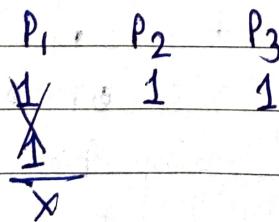
$$R = 2$$



$P_1 \quad P_2 \quad P_3$

$$P_1 \rightarrow P_2 \rightarrow P_3$$

but not always true



R = 3

P <sub>1</sub>	P <sub>2</sub>	P <sub>3</sub>
1 1 1	1 1 1	1 1 1
{x}	{x}	{x}

P <sub>1</sub>	P <sub>2</sub>	P <sub>3</sub>
1	1	1

→ deadlock can occur

R = 4

P <sub>1</sub>	P <sub>2</sub>	P <sub>3</sub>
1 1 1	1 1 1	1 1 1
x	{x}	{x}

→ Deadlock never occurs.

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

Max<sup>m</sup> resources allocated still deadlock - 1

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

Max<sup>m</sup> resources allocated still deadlock = 9

## 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 x	1 x	1 x
	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<sup>m</sup> resources but still deadlock

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

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

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

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

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

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

total resource      total processes       $\rightarrow$  total demand

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

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

4 processes  
4 resources

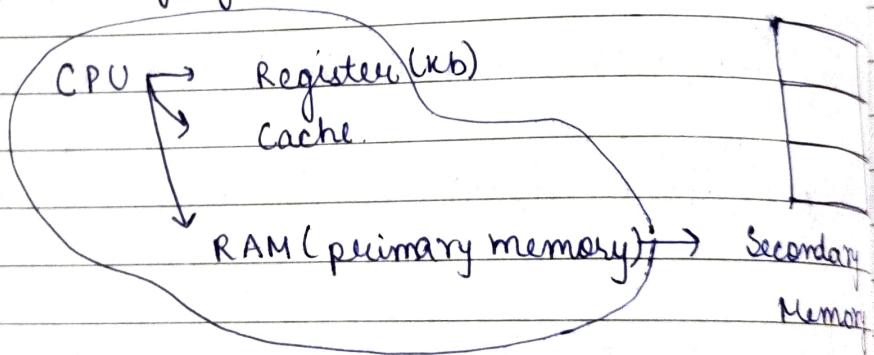
$$4+4 > 4 \times 1$$
$$8 > 4 \quad (\text{Valid})$$

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

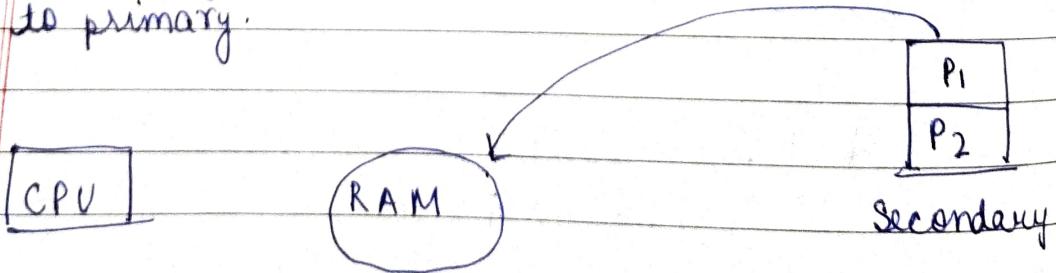
## 5.1 Memory Management and Degree of Multi programming

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

Method of managing primary memory:

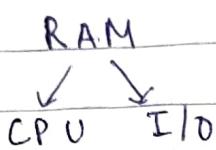


Secondary memory is not directly connected with CPU because secondary memory is a bit slower in comparison to primary.

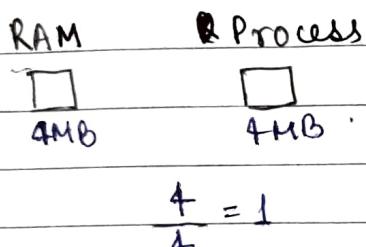


 multiprogramming - whenever we are giving the programs in RAM being more process in main memory.

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

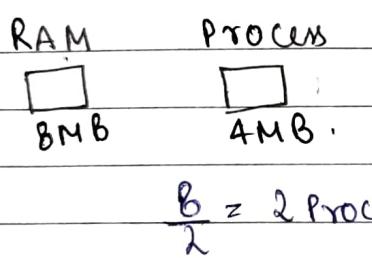


when we bring a processes.  $P_1$  & if it wants to have some I/O then CPU shouldn't be idle.



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

$$\frac{4}{4} = 1$$

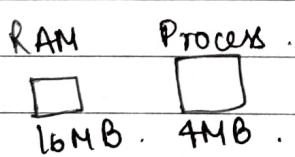


$$k^2 \rightarrow \text{I/O operat'n.} \\ \text{CPU utilization} = (1-k^2)$$

$$\frac{8}{2} = 2 \text{ Process.}$$

$$= 1 - (0.7)^2 \\ = 76\%$$

$$k = 70\%$$



$$\frac{16}{4} = 4$$

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

- \* 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)

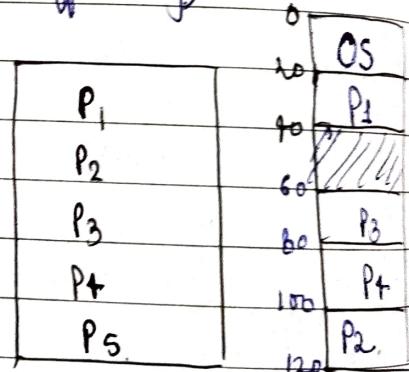
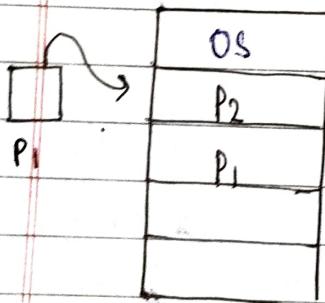
variable partition.  
(Dynamic)  
(provide at run time)

(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		8MB.
1			8MB
2	6MB.		8MB
3	8MB		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 diff<sup>n</sup>. partition  
we couldn't accomodate a process in pieces for that  
fragments.

### Limitations

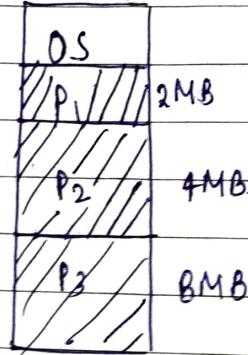
- 1) Internal fragmentation
- 2) limit in process size.
- 3) limitation on degree of multiprogramming.  
We can't bring more and more processes.
- 4) External fragmentation

## 5.4 Variable Size Partitioning

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

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

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

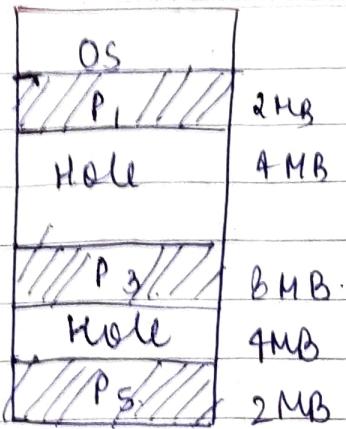


### Notes

- 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.

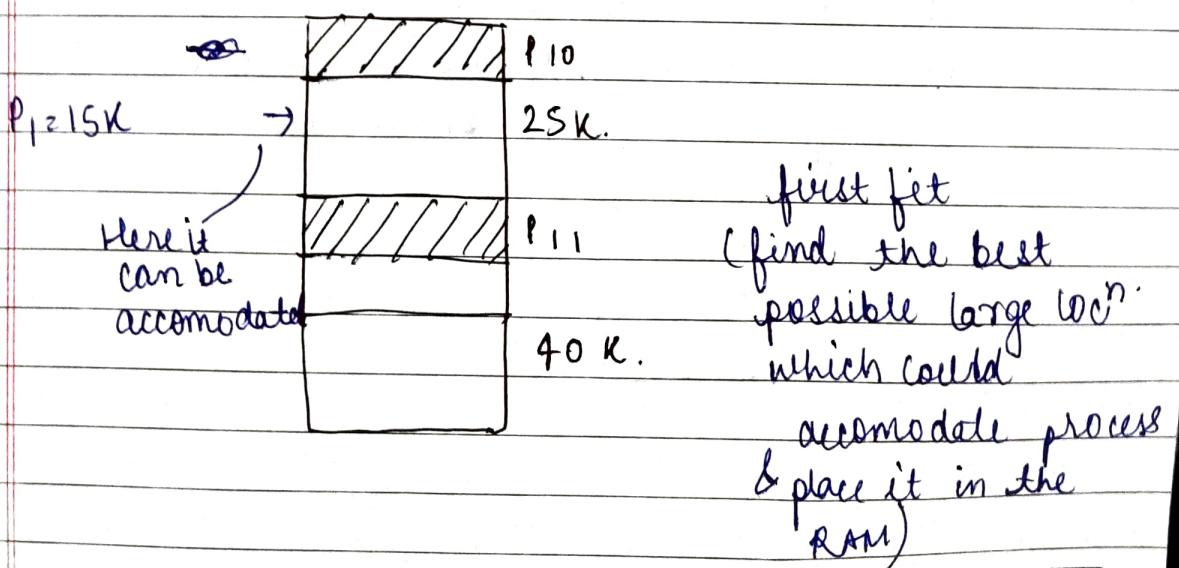
### 5.5 First fit, Next fit, Best fit Worst fit

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

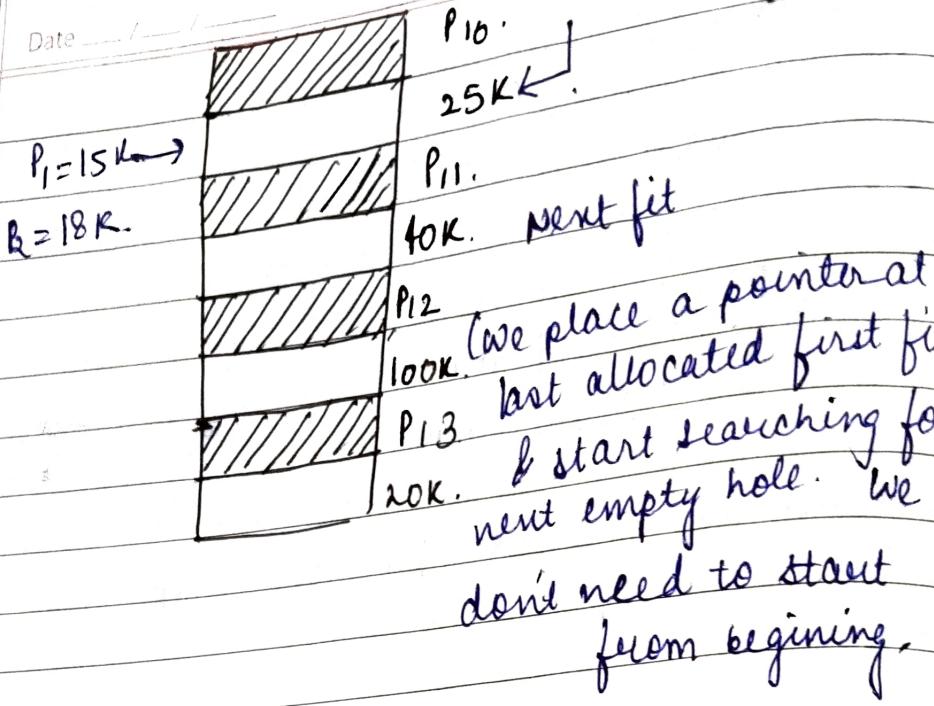
Next fit  $\rightarrow$  Same as first fit but start search always from last allocated hole.

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

Worst fit  $\rightarrow$  Allocate the largest hole.



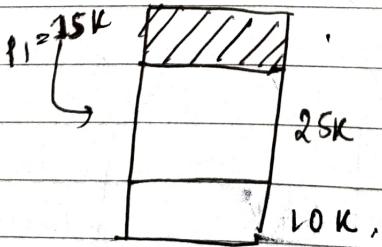
Date \_\_\_\_\_ / \_\_\_\_\_ / \_\_\_\_\_



It would search the entire list & then return the hole where there is min<sup>m</sup> fragmentation  
It is slow

~~Interval fragmentation is least~~

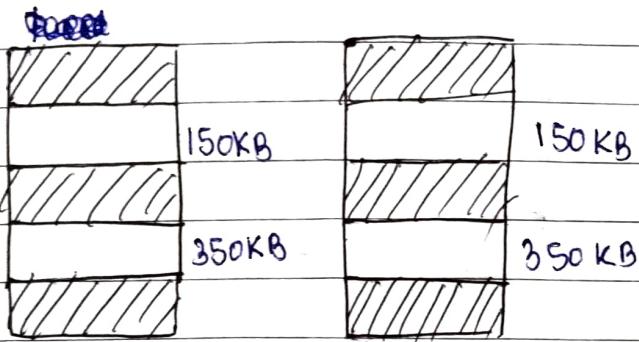
worst fit - It would search the entire list but would search the biggest hole among every partitions. (Slow due to searching of entire list).



5.6. Create Question On ~~fit~~ fits

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

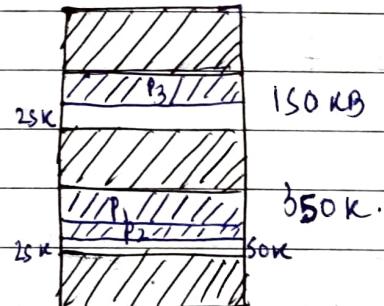
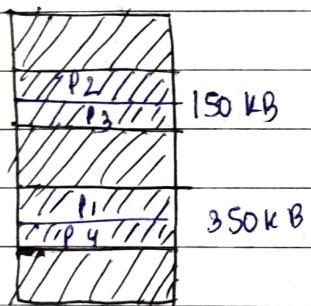
- a) best fit but not first fit
- b) first fit but not best fit
- c) Both
- d) None



For first fit

For best fit

P <sub>1</sub>	300 K
P <sub>2</sub>	25 K
P <sub>3</sub>	125K
P <sub>4</sub>	50K



P<sub>1</sub> P<sub>2</sub> P<sub>3</sub> P<sub>4</sub>  
300 25 125 50

350K.  
50K.P<sub>4</sub> can't be accomodated

## 5.7 Create 2007 Question on fits

## Best fit

Req. No	J <sub>1</sub>	J <sub>2</sub>	J <sub>3</sub>	J <sub>4</sub>	J <sub>5</sub>	J <sub>6</sub>	J <sub>7</sub>	J <sub>8</sub>
Request size	2K	14K	3K	6K	6K	10K	7K	20K.
Usage time	4	10	2	8	4	1	8	6.

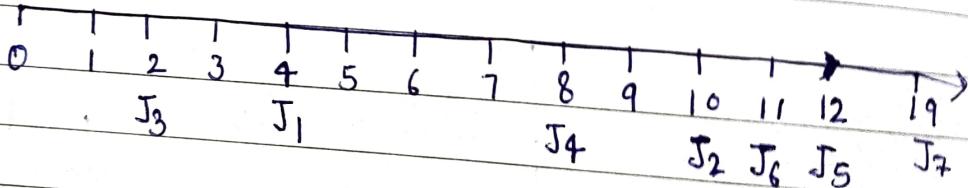
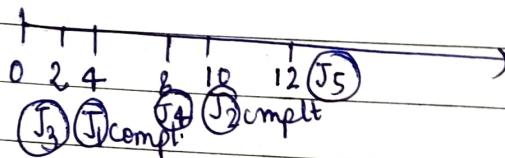
Calculate the time at which J<sub>7</sub> will be completed

- a) 17    b) 21    c) 19    d) 20

M/M

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

0	J <sub>3</sub>	4K
1	<del>J<sub>5</sub></del>	8K
2	<del>J<sub>2</sub></del> <del>J<sub>6</sub></del> J <sub>7</sub>	20K
3	J <sub>1</sub>	2K

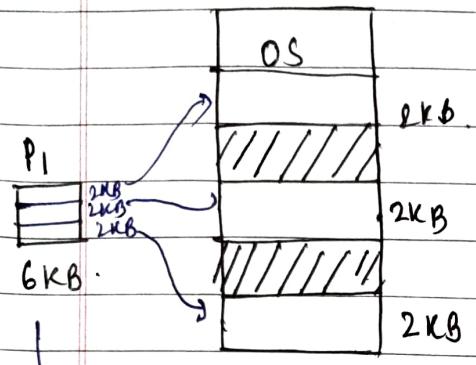


In case of J<sub>5</sub> & J<sub>6</sub> we would take out the one which is completed first. And then put it & calc. the final time by initial + usage-time.



## 5.8 Need of Paging (Non contiguous Memory Allocation)

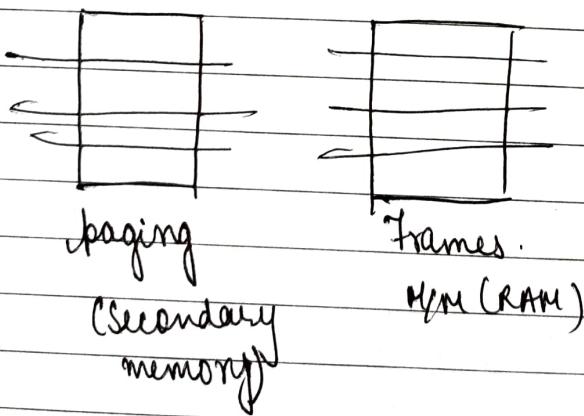
A process can be spanned & put up in fragments in different memory location.



process is divided acc<sup>n</sup>. to the hole size but the hole size changes on runtime so its a bit time consuming (division of process)

process is divided & put M/M  
divided & put in diff. memory locations. So a process is divided before its 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