Date ___/___/_____

Operating System

Notes

— by Riti Kumari

Please Share And Help others :)

Operating System
Lec-0

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 → Bankers algorithm

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

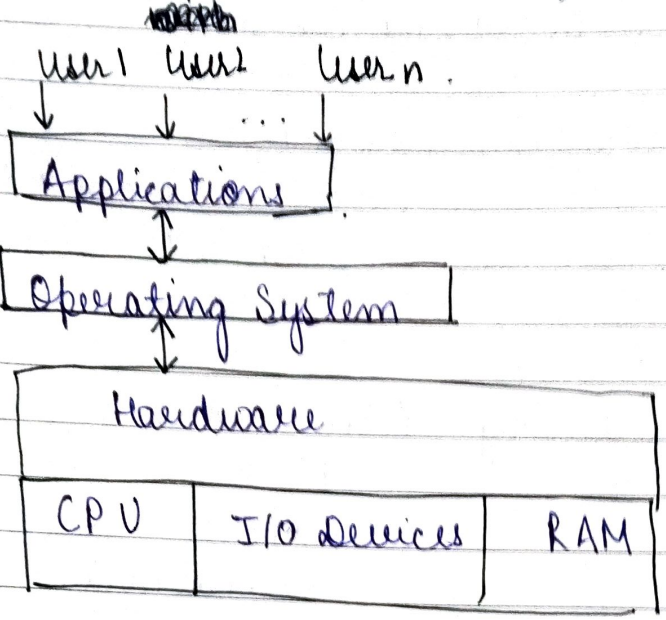
6. Disk scheduling ← SCAN
   CSCAN
   FCFS

7. Unix commands ← ls
   → mkdir
   → cd
   → chmod
   → Open system call

8. File mangament & 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 convinience 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 krwana hai).

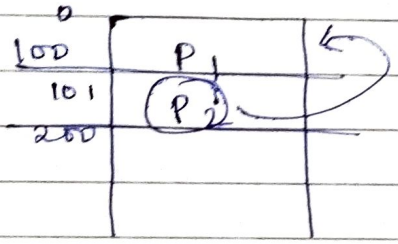
2. Process management - Managing & Executing multiple process at a time. We use CPU scheduling

program execute process

3. Storage management → How to store the data permanently using File system.
Eg - Harddisk.

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

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



It provides security bet^n. the process also.

Windows (Cmd)
Linux (Terminal)

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

Date __/__/____

L-1.2

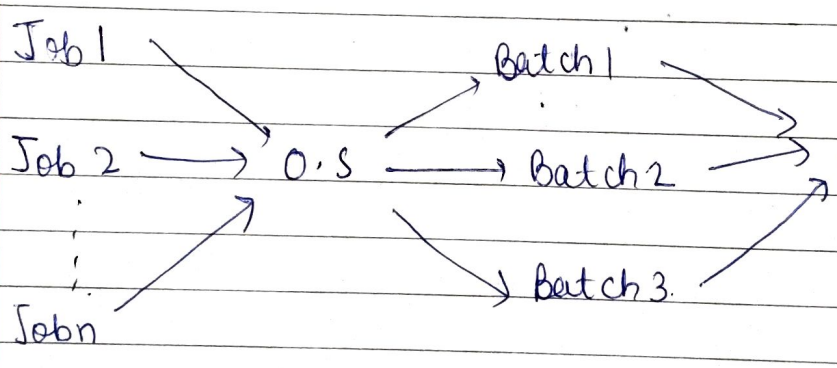
Batch Operating System

Types of Operating System

1) Batch OS.
2) Multiprogramming OS.] Imp.
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.

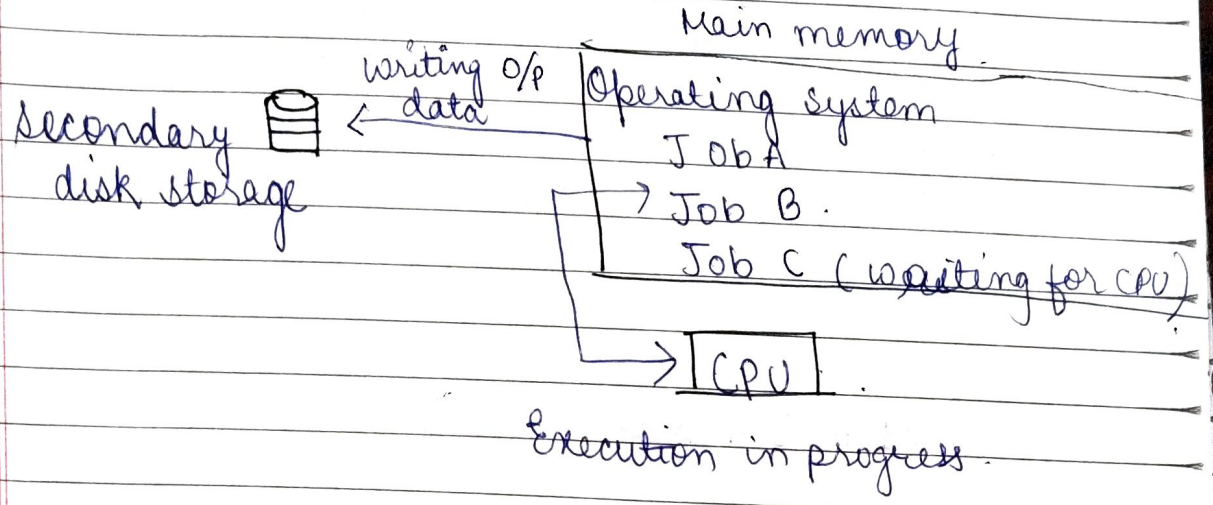


Same type of Jobs batch together and execute at a time.

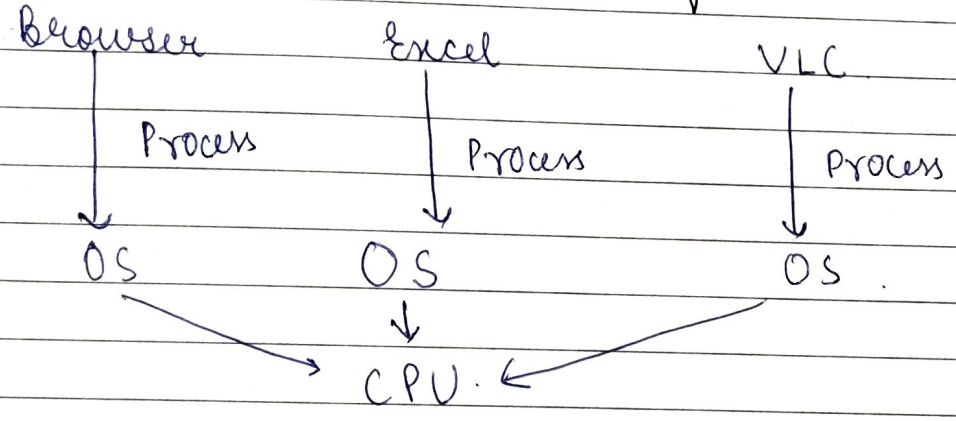
Demerit: It processes one job at a time till the time CPU remains idle.

Date ___/___/_____

L-1.3 (Multiprogramming & Multitasking System)

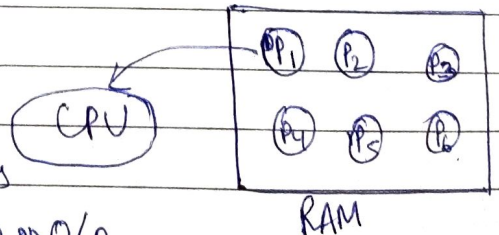


Multiprogramming OS / Time sharing



Multitasking OS

Multiprogramming OS.
Non preemptive
CPU executes a process completely until or unless process demands any i/p or O/p operation.
Idleness should be removed

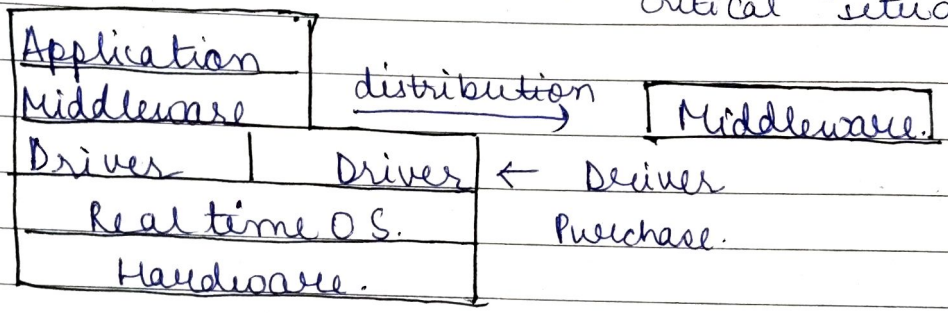


Multitasking / time sharing - Provides particular time for every process. If its complete its great but if not it moves to other process P2 & schedule P1 for further.

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

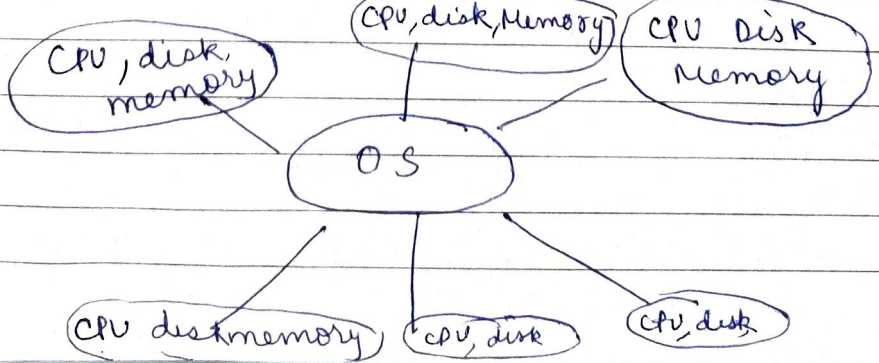
Lec1.4 Types of OS (Real time)

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



Immediate output
1) No delays

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

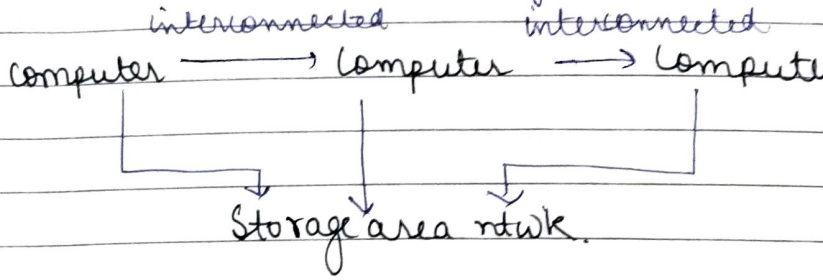


It uses multiple central processors to serve multiple real-time applications & multiple users using telephone wires.

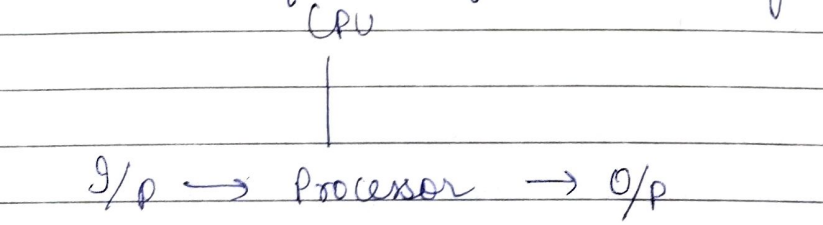
Clustered OS. → ~~Distributed~~ Clustered Os share storage and are closely linked via Lan or a faster inter connection.

Group of connected computers

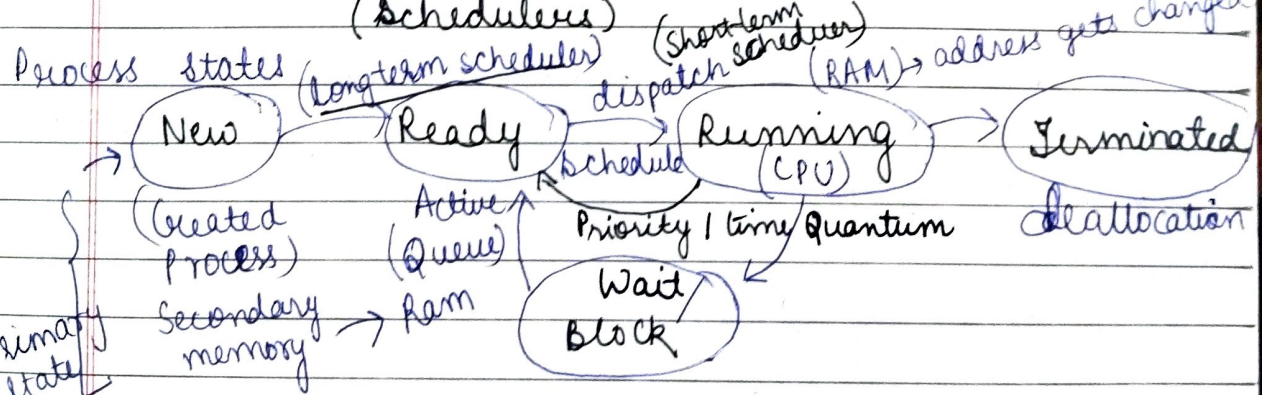
* fault tolerance, scalability



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.



dec - 1.5 Process states in OS (Schedulers)



Long term scheduler - Bring more to 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.

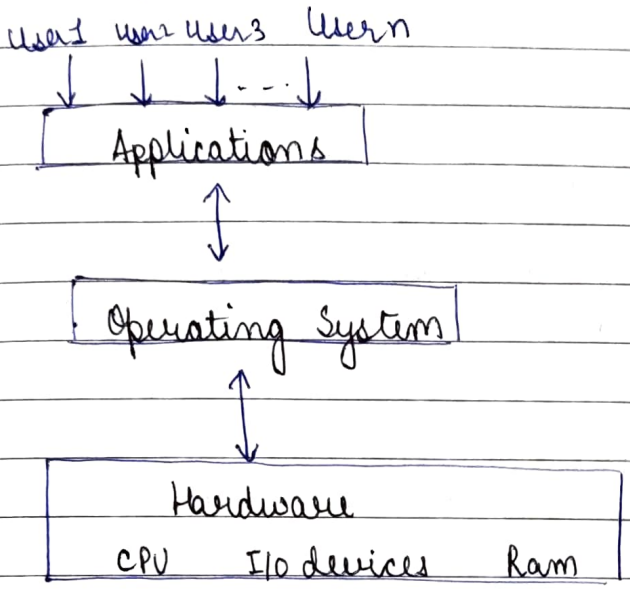
1. Basic Introduction. : What is O/S, Goals, need, System calls, diff^n OS.
2. Process M/g : what is process, Algo - FIFO, SJF, Preemetive, Non premtive, Round Robin
3. Deadlock - Banker Algo
4. Concurrency control - Semaphore, msg passing, race cond^n.
5. Memory - Paging, Segmentation, Fragmentation virtual memory, Page Replacement.
6. Disk M/g - C-look, Algo, FCFS, SSCF, look
7. File m/g : NIFS, FAT32, Seq:, Random
8. Unix System : commands.
9. Case study : MSDOS
10. Security - Attacks, Firewalls exceptions

Date

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

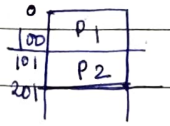
Functionalities of OS.

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.

2. ~~Storage~~ Process management → Performing multiple process at an time, OS uses CPU scheduling algorithms for processing & executing in a proper & efficient way

3. Storage management (Hard disk) - How to store data in harddisk 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.



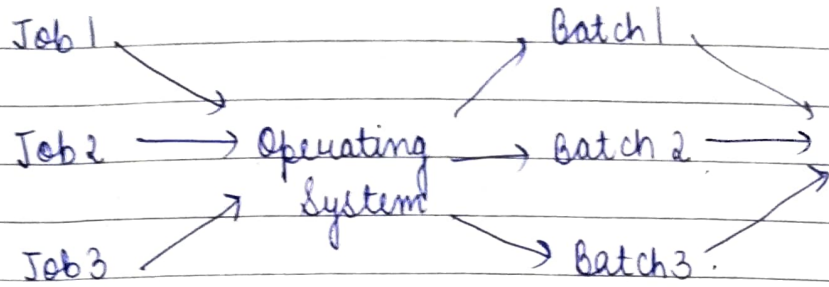
5. Security & privacy
It provides security & privacy between the process such that it doesn't gets blocked. Password protection

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

1.2. Types of OS

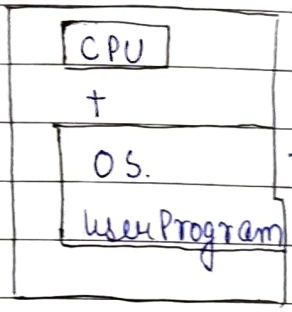
1) Batch OS
2) Multiprogramming
3) Multitasking
4) Real time OS
5) Distributed
6) Clustered
7) Embedded

1) Batch OS

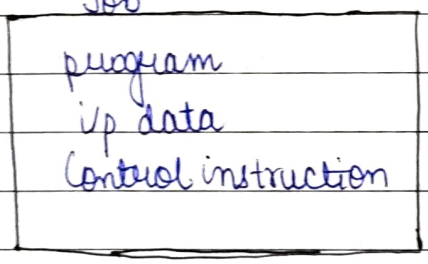


Before batch OS., we used mainframe computers

1) Common i/p & o/p devices were card readers & tape drivers.



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

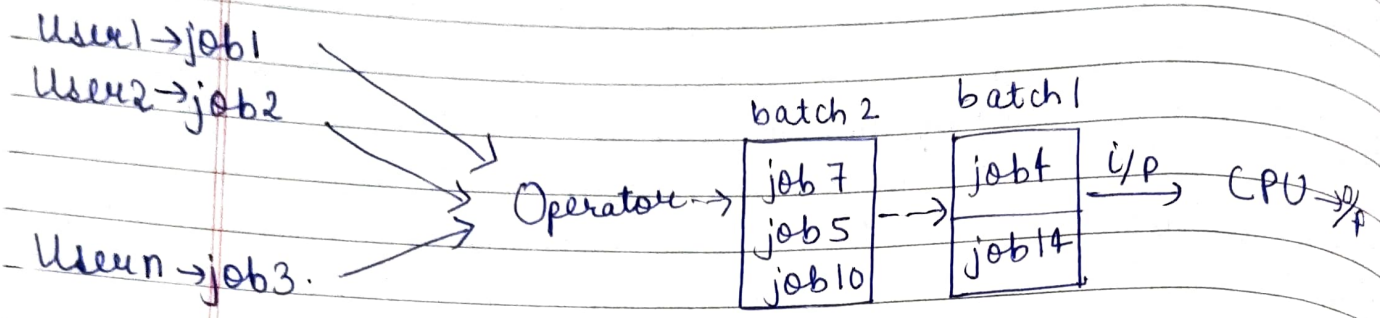


3) I/p Punch cards O/p (punch card)

Major problem was.

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

1. Batch processing os (Eg: 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.
Eg - 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.

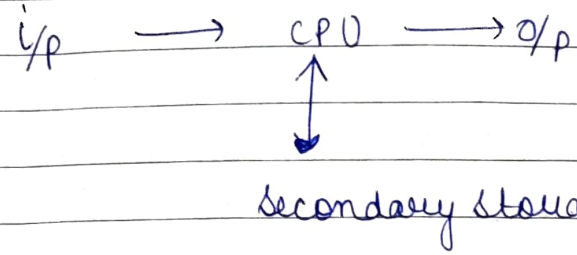
Disadvantage

1) Memory limitation
2) Interaction of i/p & o/p devices directly with CPU.
3) CPU remains idle during loading & unloading

Date

Spooling (Simultaneous peripheral device)

i/p or o/p



temporarily process is storage in secondary storage.

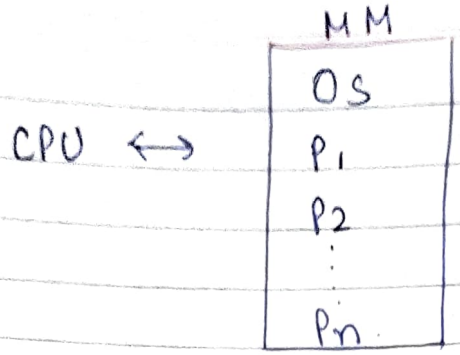
Advantages

1) Increase the system performance.
2) Spooling resolve the problem of speed mismatch of diff$^{n.}$ device
3) I/O of one job is overlapped with the computation of other jobs.
4) Spooling use the disk as a huge buffer.

Spooling	Buffering
Uses the hard disk as a large spool (spool is a temporary storage area in hard disk)	Uses limited memory space in RAM usually called buffer (Buffer is a temporary storage area in RAM)

1.3

2. Multiprogramming OS



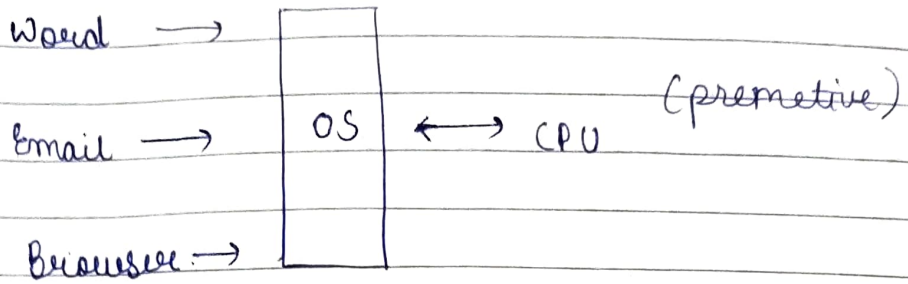
(Non-premetive)

1) Maximize CPU utilization
2) Multiprogramming means more than one proces 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 requir CPU then instead of sitting idle, CPU makes a 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 betn process so quickly that it gives illusion that all executing at same time.
3. the task in multitasking 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 guarentees critical task to be completed within a range of time.
Eg - a robot hierd to weld a car body

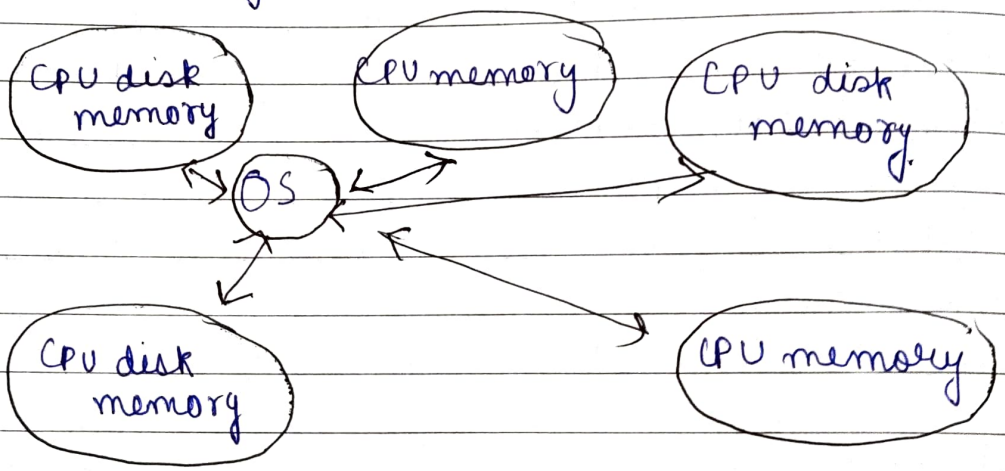
Soft RTOS - It provides some relaxation in time.

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

Disadvantages
1) Limited tasks
2) Compler 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 ntwk connection doesn't affect other
2) Delay in data processing reduces.

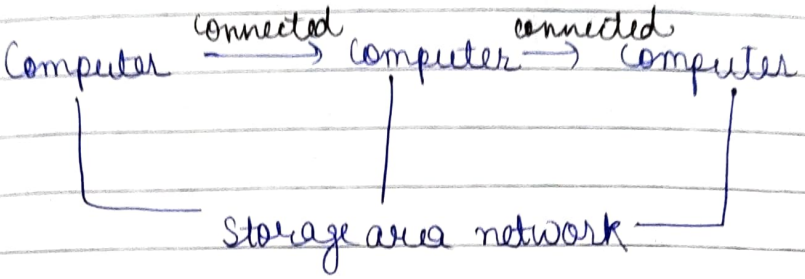
Disadvantages

1) Failure of main ntwk will stop entire communication.

6) Clustered OS

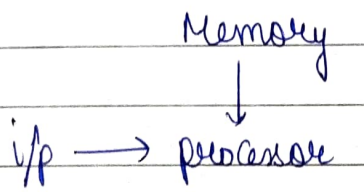
1) Clustered computers share storage & are closely linked via LAN or faster connection.

2) Combⁿ of multiprocessor + distributed OS



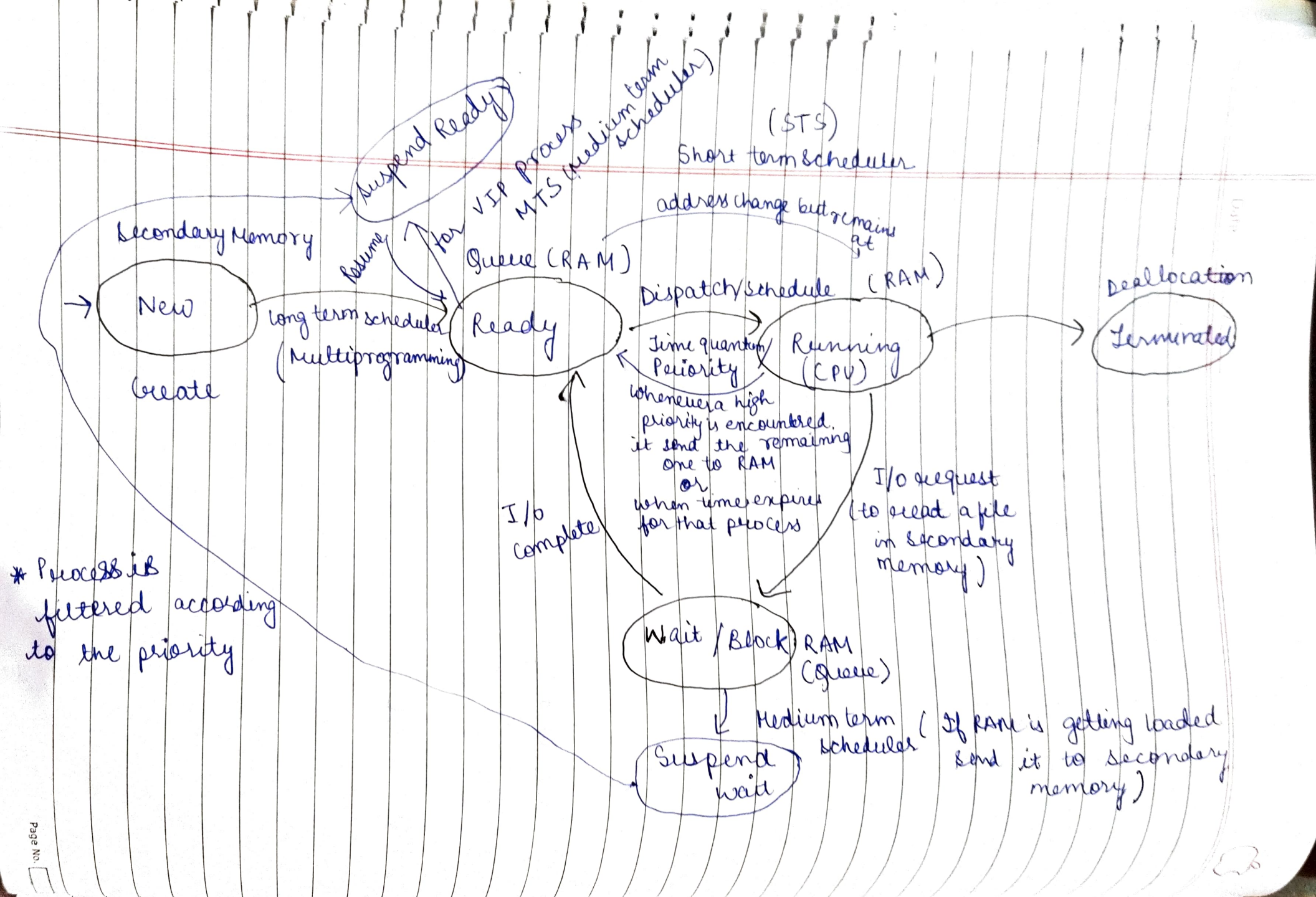
7) Embedded OS. - An OS in which we work on embedded systems. It is designed to perform a specific task for a device that is not in computer

Eg - ATM



1.5

Process states in OS


Suspend Ready
(MTS (Medium term scheduler))
for VIP process
(STS)
Short term scheduler
address change but remains at
Secondary Memory
Resume
Queue (RAM)
New
long term scheduler
(Multiprogramming)
Ready
Dispatch/schedule
(RAM)
Deallocation
Time quantum/ Priority
Running (CPU)
Terminated
create
whenever a high priority is encountered, it send the remaining one to RAM
or
when time expires for that process
I/O request (to read a file in secondary memory)
I/O complete
* Process is filtered according to the priority
Wait / Block
RAM (Queue)
Medium term scheduler
(If RAM is getting loaded send it to secondary memory)
Suspend wait

Date __/__/____

* 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'.

a) chmod a- rw
b) chmod go+r note
c) chmod ugo = r note
d) chmod u+r, g+r, o-x note

chmod — change mode

r w x r w - r w x
user group others

u - user
g - group
o - others

rx rwx
6 7

r - read 4
w - write 2
x - execute 1

Q. Chmod ugo+rw note command can represented in Octal notations 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 a b c d e f g h i j

there lseek is used 2 times sequentially.

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

n is file descriptor. After applying lseek 2 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.

lseek1

1 2 3 4 5 6 7 8 9 0 a b c d e f g h i j

lseek2

10, seek_cur → move to the given no from the current
5, seek_set → set the current pos at given
-2, seek_end → Count from the end towards left.

L-1.7

System calls in OS.

User mode —system call→ Kernel mode

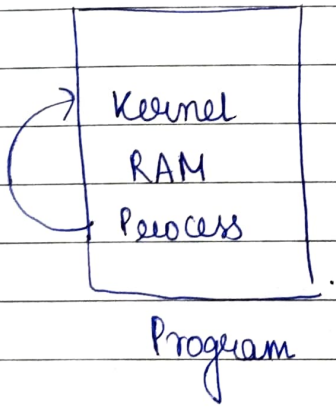
2+4 —monitor→ print (6)

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 devices

3) Information → Process or regarding devices attributes
Foreg:- getPid, attributes, get system time & data.

Date ___/___/_____

4) Process control → Program being loaded
→ load, Execute, abort, fork, wait, signal, Allocate etc.

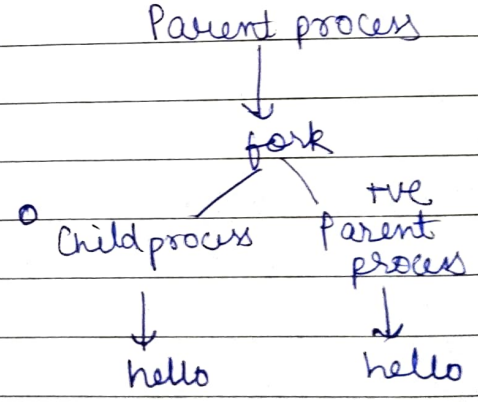
5) Communication Control- Interprocess communication
→ Pipe(), create/delete connections, shmget()

L-1.8
Fork system call

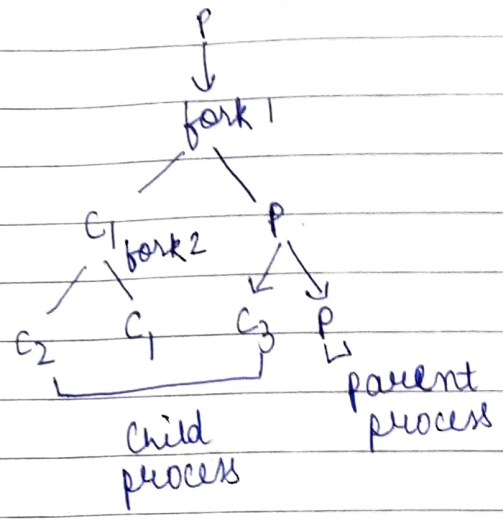
To create a child process from parent process we can use fork() or thread.

fork() ← 0 Child process
+1 parent process
-1 child process not created

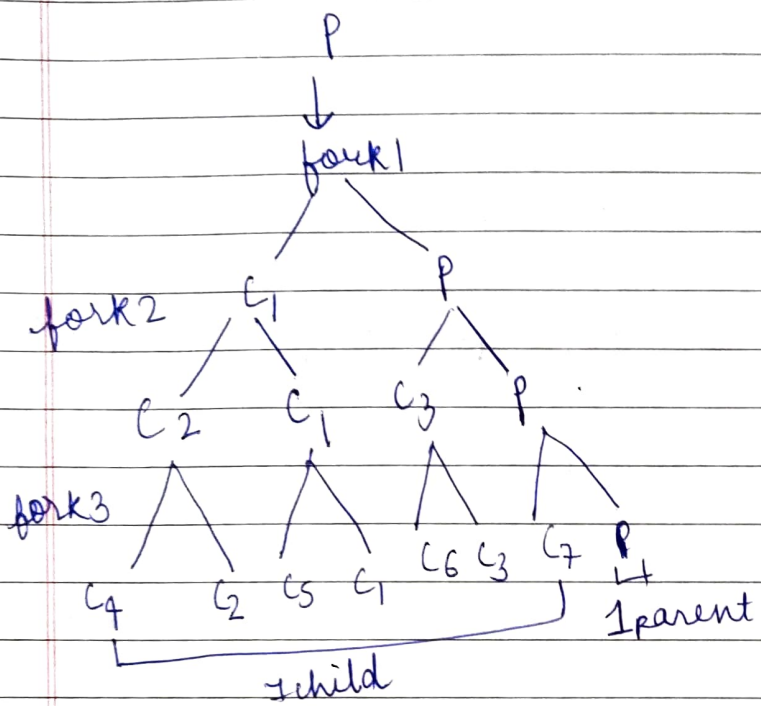
```
main() {
fork();
printf("hello");
}
```




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



4 times Hello.



8 times hello.
$= 2^n$ n = no of fork.
$2^n - 1 \Rightarrow$ child process

L-1.9

Questions on fork System Call

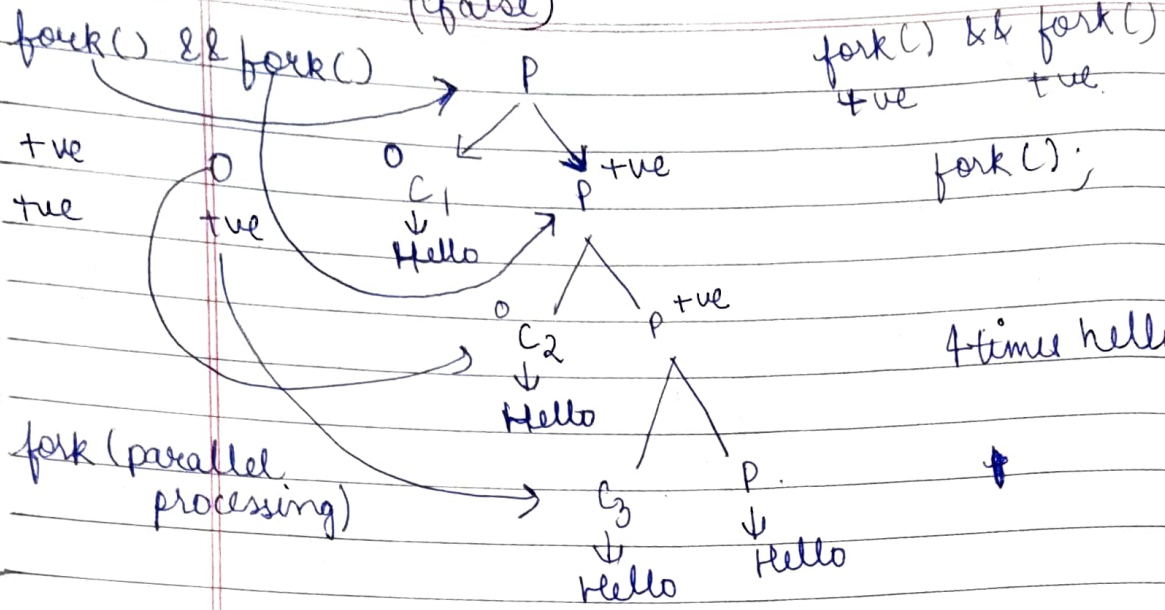
```
# include <stdio.h>
# include <unistd.h>

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

fork() && fork()
0 +ve
(false)

fork() && fork()
+ve

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

fork();

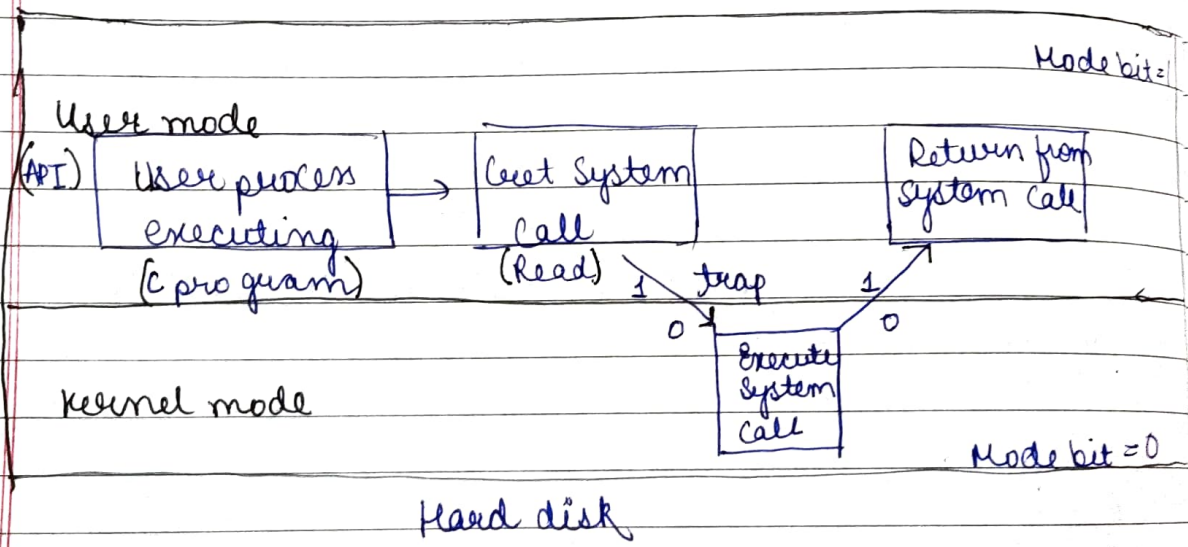


4 times hello.

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



Date

1 ⟶ 0

bank ⇆ window ⇆ money

usermode Kernelmode

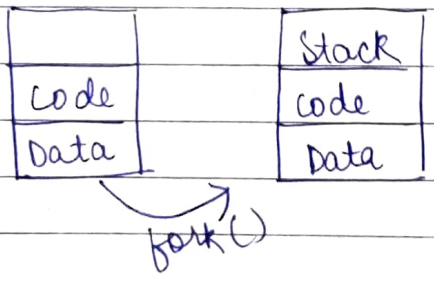
1 ⟵ 0

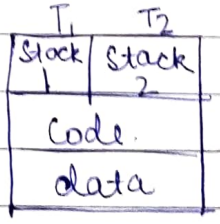
L-1.11

Process Vs Threads in OS

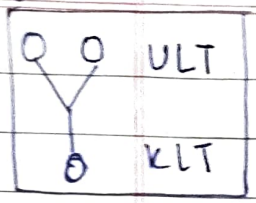
Process - It is a heavy weight task] multitasking
Thread - It is a light weight task] environment

Process	Thread (User level)
1) System calls involved	1) No system call involved
2) OS treats diff^n process differently.	2) All user level threads treated as single task for OS.
3) Diff^n process have diff^n copies of data, files, cache	3) Threads share same copy of code
4) Context switching is slow	4) Context switch is fast
5) Blocking a proces-not block other	5) Blocking a thread - block entire process
6) Independent	6) Interdependent





Hybrid environment



L-1.12

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

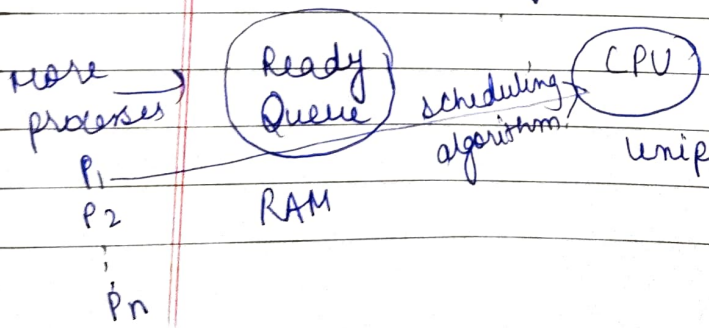
User level thread	Kernel level Thread
1) User level thread are managed by user level library.	1) Kernel level threads are managed by OS (system calls)
2) user level threads are typically fast	2) Kernel level threads are slower than user level
3) Context switching is faster.	3) Context switching is slower.
4) If one user level threads perform blocking operation then entire process get blocked	4) If one kernel level thread blocked no, affect on others.

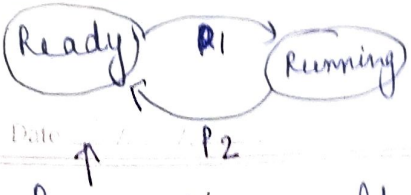
(Context Switch) Process > KLT > ULT

L - 2.1

Process scheduling Algorithms

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

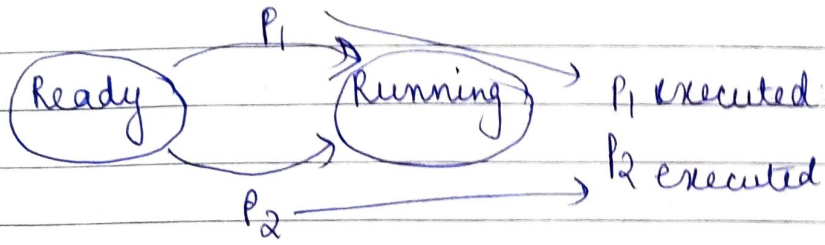




Reasons
1) Time quantum (provided time for P_1)
2) Priority process

* Pre emptive → 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 P_1 & give another process p2 & send p1 to ready queue. It provides responsiveness

* Non pre emptive ⇒ If a proces 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
← 1) SRTF (Shortest Remaining time first)

BT ← 2) LRTF (longest remaining time first)

TQ ← 3) Round Robin

4) Priority based
↓
Priority given

placed in ready queue in descending order of burst time

1) FCFS (First come first serve) → AT

2) SJF (Shortest Job first) → BT

3) LJF (Longest Job first) → BT

4) HRRN (High response ratio next).

5) Multilevel Queue

6) Priority based

L-2.2
Diff^{n} times in CPU scheduling

Arrival time - The time at which process enters 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 - Turn around - Burst time

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

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

Arrived Bank → Create new account → left Bank

L-2.3
First Come first Serve (FCFS)

Date ___/___/_____

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

i) The process which requests the CPU first get the CPU allocation first.

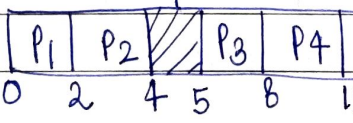
ii) Managed using FIFO queue.

CPU idle & busy	Process No	AT Arrival time	BT Burst time	CT Completion time	TAT (CT-AT)	WT (TAT-BT)	RT
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

Criteria: arrival time
Mode: Non preemptive

CPU idle

Gantt chart



Time ⟶

At t=12, all the processes got executed

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

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

L-24
Date ___/___/___ SJF

SJF - (Shortest Job first) → ~~Is~~ SJF is an algorithm in which the process having the smallest execution time is choosen for next execution.

ii) It can be preemptive or non-preemptive

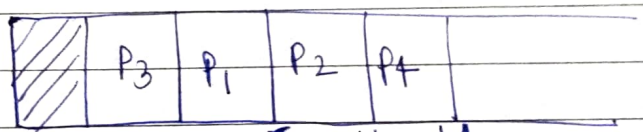
iii) It reduces the waiting time for other process.

iv) It is a greedy Algo.

kb burst hua	Process No	AT	BT	CT	TAT	WT	RT
3	P_1	1	3	5	5	2	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



P_1 P_2 P_2 P_4

P_1 P_3
(3) (2)

When bT is same see the one having less arrival time else can use process id.

Page No.

L-2.5

Shortest Remaining Time First

↓

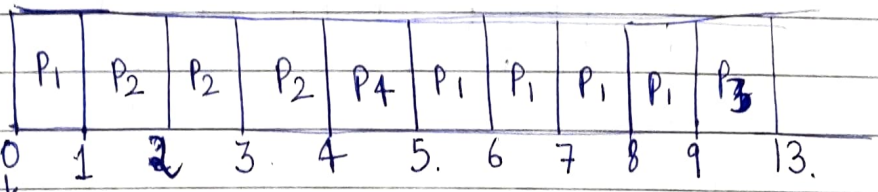
(Shortest job first + preemptive)

Process no	AT	BT	ComplT	TAT	WT	RT	CPU first time
P_1	0	35 ~~4~~	9	9	4	0	0
× P_2	1	~~13~~ ~~2~~	4	3	0	0	1
P_3	2	4	13	11	7	7	9
× P_4	4	~~1~~	5	1	0	0	4

Criteria: "Burst time"

Mode: "Preemptive"

Grantt Chart



0 → P_1

1 → P_1, P_2

2 → P_1, P_2, P_3

3 → P_1, P_2, P_3

4 → P_1, P_3, P_4

5 → P_1, P_3

6 → P_1, P_3

7 → P_1, P_3

8 → P_1, P_3

9 → P_1, P_3

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

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

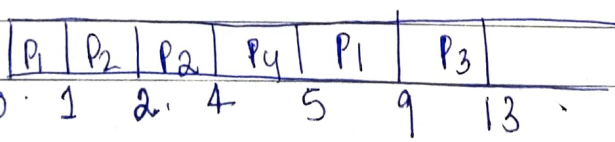
Avg RT = $\frac{7}{4}$ = 1.75

$P_1 \rightarrow$ ~~8~~ 4

$P_2 \rightarrow$ ~~3~~ ~~2~~ ~~1~~ ~~0~~

$P_3 \rightarrow$ 4

$P_4 \rightarrow$ ~~1~~



Date ___/___/___

L-2.6

SJF with preemption Ex:

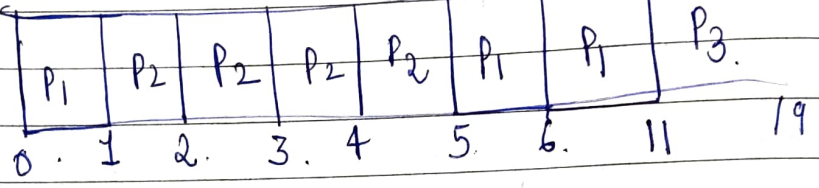
Process	AT	BT	CT	TAT	WT	RT
P_1	0	7 ~~6~~				
P_2	1	4 ~~3~~ 2				
P_3	2	8				

Criteria : Burst time
Mode : preemptive.

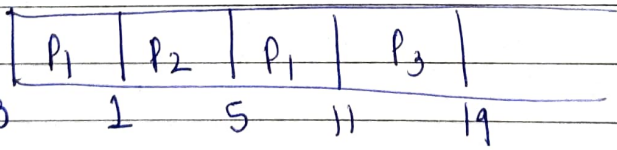
Gantt chart

7-2=5

15 15=2 5



0 → P_1
1 → P_1, P_2
2 → P_1, P_2, P_3
3 → P_1, P_2, P_3
4 → P_1, P_2, P_3
5 → P_1, [illegible], P_3



L-2.7

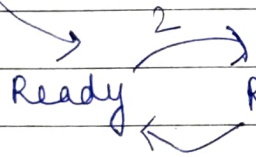
Round Robin Scheduling Algo.

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	CPU first time
P_1	0	5 3 1	12	12	7	0	0
→ P_2	1	4 2	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" (We have to resume the process not restart it)
Mode : "Preemptive"



3
4
2
1

Sequence of process in Ready queue

P_1	P_2	P_3	P_1	P_4	P_2	P_1

0

context switching (save running process & welcome new process)

Running queue

P_1	P_2	P_3	P_1	P_4	P_2	P_1

0 2 4 6 8 9 11 12

time →

Context Switches = 6.

L-2.8 Scheduling algo.

Pree emptive Priority ~~Process~~.

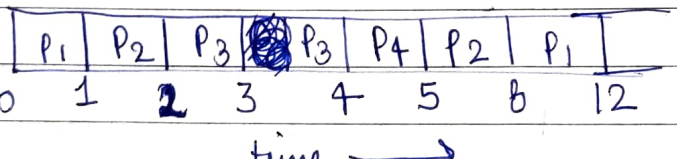
In pre emptive scheduling, the tasks are mostly assigned with their priorities.

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	4 5	12	12	7	0	0
20	P_2	1	3 4	8	7	3	0	1
30	P_3	2	1 2 0	4	2	0.	0	2
40	P_4	3 4	0 1	5	1	0	0	4

Higher the no. higher the priority.



0 → P_1
1 → P_1, P_2
2 → P_1, P_2, P_3
3 → P_1, P_2, P_3
4 → P_1, P_2, P_3, P_4
5 → P_1, P_2

L-2.9

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

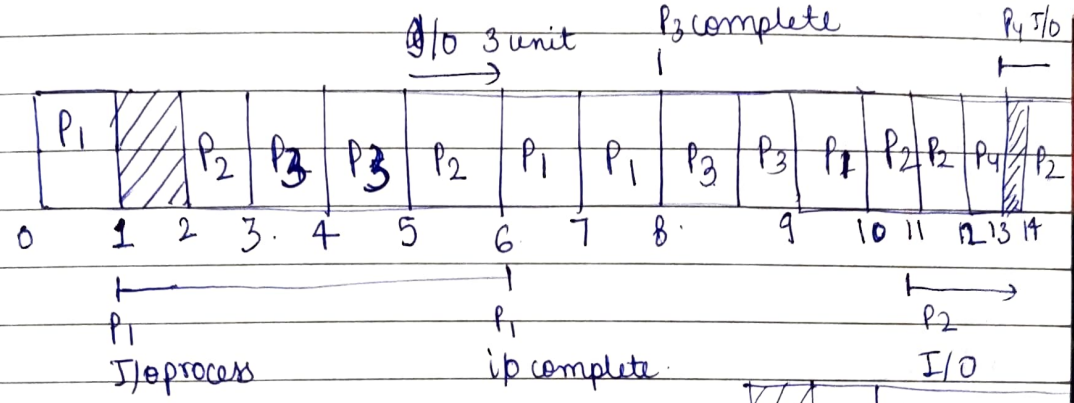
Process	AT	Priority	CPU	I/O	CPU
P_1	0	2	1 0	5 0	3 2 1 0
P_2	2	3	3 2 1	3	1
P_3	3	1	2 1 0	3 0	1 0
P_4	3	4	2	4	1

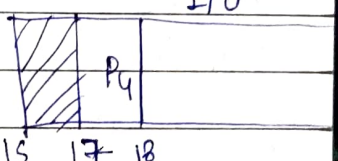
Lowest the no. highest the priority.

Mode: Preemptive

Criteria: Priority based

Find CT of P_1, P_2, P_3, P_4





0 → P_1

1 → no process

2 → P_2

3 → P_3, P_4 (high priority)

4 → P_1, P_2, P_3, P_4

5 →

6 → P_1, P_2, P_4

8 → P_3, P_1

9 → P_1, P_2, P_4

	CT
P_1	10
P_2	15
P_3	9
P_4	18

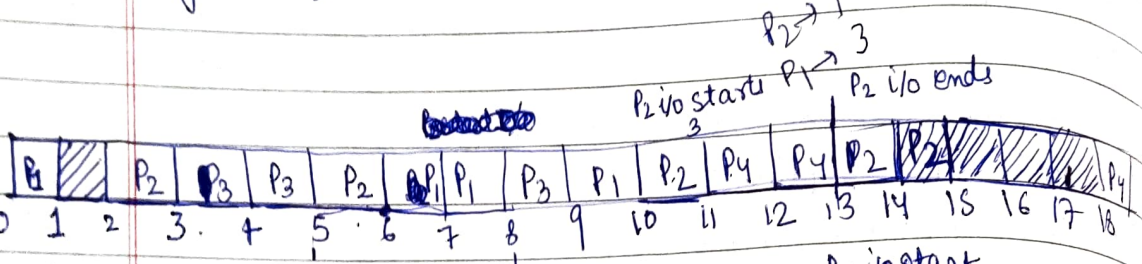
CPU idleness = 4/18

Usage = 14/18

Priority	AT	Priority	CPU	I/O	CPU
P_1	0	2	1 0	5	3 2 1 0
P_2	2	3	3 2 1	3	1
P_3	3	1	2 1 0	3	1 0
P_4	3	4	2 1 0	4	1

Mode: Preemptive
Criteria Priority based.

Priority $P_3 > P_1 > P_2 > P_4$



In Ready queue.

I/O Procss
$P_1 \rightarrow 6$
$P_3 \rightarrow 5$

6 → P_1, P_2 P_4
7 → P_1, P_2, P_4
8 → P_1, P_2, P_3, P_4
9 → P_1, P_2, P_4
10 → P_2, P_4

13 → P_2

L-2.10
Multilevel queue scheduling

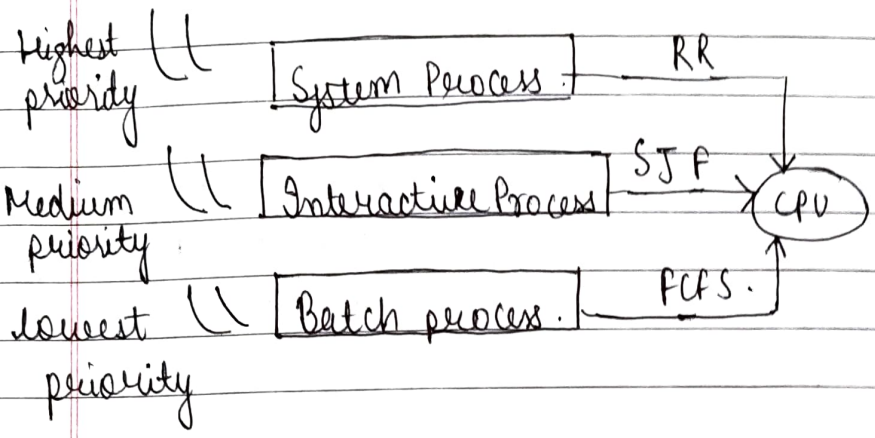
Page No.

Date __/__/____

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

i) There must be different queue for diff^n process
ii) Every process can have their own algorithms.

1) 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

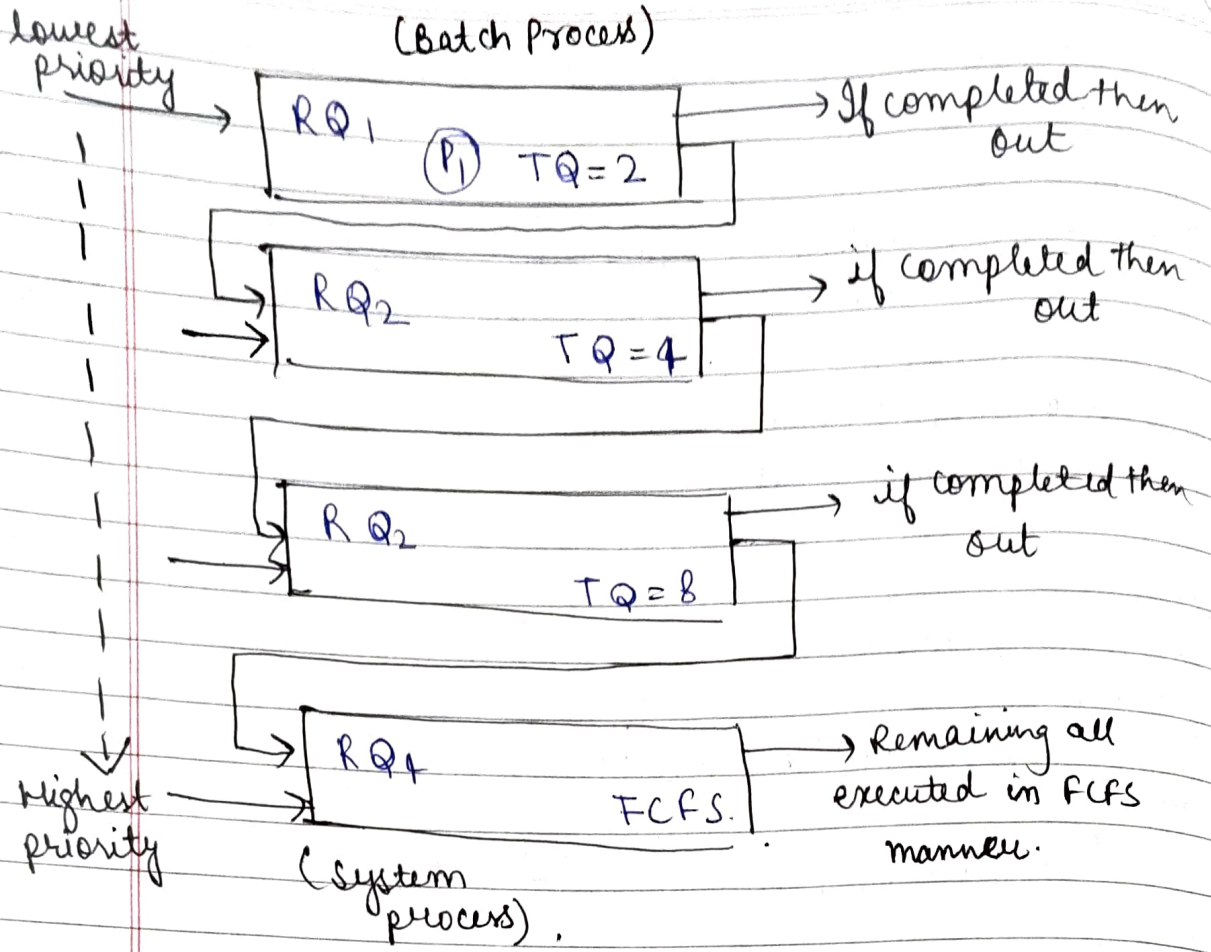
It can be resolved using multilevel feedback queue.

L-2.11

Multilevel Feedback Queue

More System process - lowest priority waits

lowest priority - feedback.



P_1	19	17	13.	5
		TQ=2	TQ=4	TQ=8

Process Synchronization

Processes

Cooperative process

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

Ex – IRCTC.

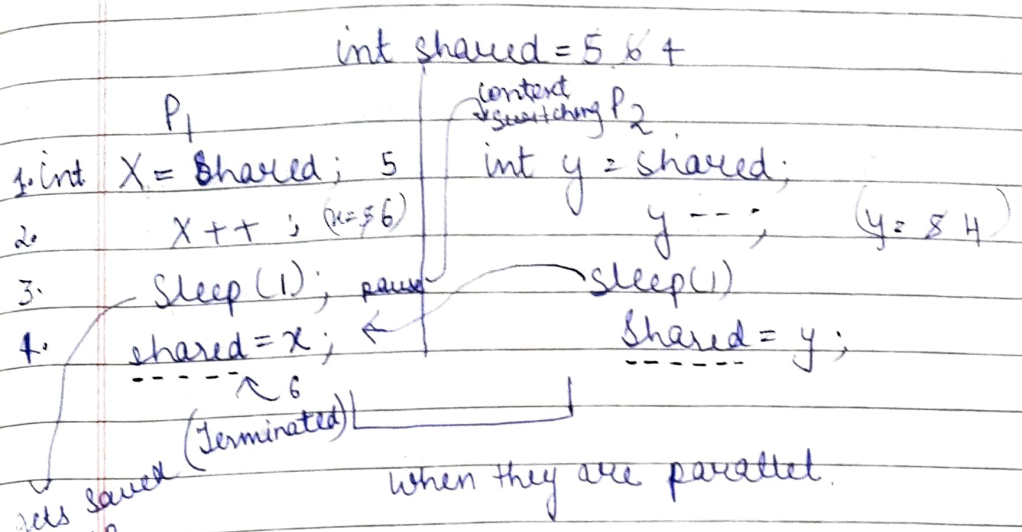
Independent process

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

PCB → Process control Block

Date

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



Processes aren't Synchronized as the process at last gives either 4 & 6. but it should give 5.
This is called race condn.

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

Buffer empty

local variable: out

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

Producer

```
int count = 0.
void producer (void)
{
    int item p;
    while (true){
        produce item (item p);
        while (count == n);
        Buffer [in] = item p;
        in = (in + 1) mod n
        count = count + 1;
    }
}
```

local variable: in

Buffer is full so its overflow

1. load R_p, m[count];
2. INCR R_p;
3. Store m[count], R_p

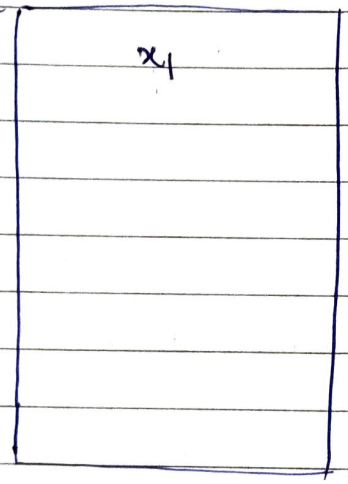
gets from the buffer

gives to buffers

$n = 8$

Buffer [0 ..... n-1]

index



OUT

COUNT

IN (next empty slot address)

0

COUNT

R_p → Registers

INCR → increment

n = size of buffer

Case I. x_1 (producer is processing x_1)

Out	(O+1) mod 8	in	count
0 1	1 mod 8 = 1	0 1	0 1 0

(O+1) mod 8 = 1

count = 1 - 1 = 0

Page No.

It forms a circular queue so we used mod n.

Consumer.

Out
0 1

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

Producer

Count In
3 4
2

Let x_4 be inserted in the in value.

Producer:

3
Rp = 3. load Rp, m[count] I_1
Rp = 4 INCR Rp. I_2

If after INCR Rp the process gets preempt
Resuming Rp = 4 I_3

Let x_1 be consumed.

3
load R_C, m[count] $R_C = 3$ I_1
DEC R_C $R_C = 2$ I_2
Resuming Rp = 2 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.

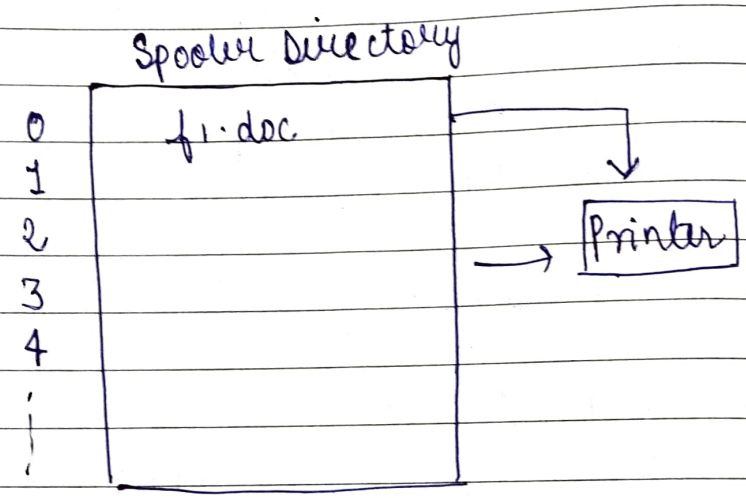
L-3.3.

Printer-Spooler Problem

multiple Programs → Spooler → take items sequentially → Gives to the printer

0

1. Load Ri, m[in]
2. Store SD[Ri], "F-N"
   f1.doc
3. INCR Ri
4. Store m[in], Ri



IN ∅1

↳ Empty slot

Case1: f1.doc P_1

R_1 ∅1

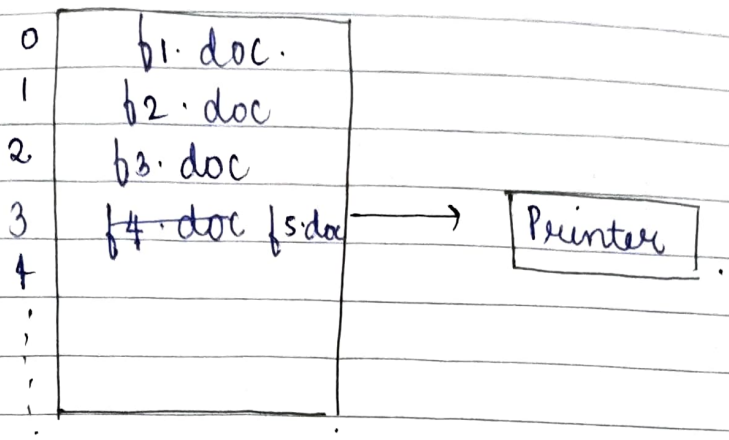
f4.DOC f5.DOC

Case2: Two process P_1 & P_2

Cooperative process

Date

In



In ~~3~~ 4

R_1 ~~3~~ 4 R_2 ~~3~~4

P_1 . I_1 I_2 I_3 | P_2 I_1 I_2 I_3 ~~4~~ | I_4

P_1 & P_2 due to non synchronisation 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_1

```
#include ___
main()
{
A, B → non critical section
___
count → critical section → count
}
```

P_2

```
#include ___
main() {
X, Y
___
}
```

common code → non critical section

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_1

Exit section

P_2

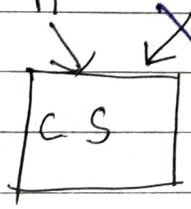
Exit section

Different sol^n for process synchronization

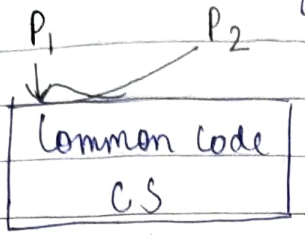
4 conditions

1) Mutual Exclusion } Primary
2) Progress
3) Bounded wait
4) No assumption related to H/W Speed } Secondary

Mutual Exclusion → If P_1 or P_2 is inside critical section the P_2 or P_1 is not allowed to enter in critical section.

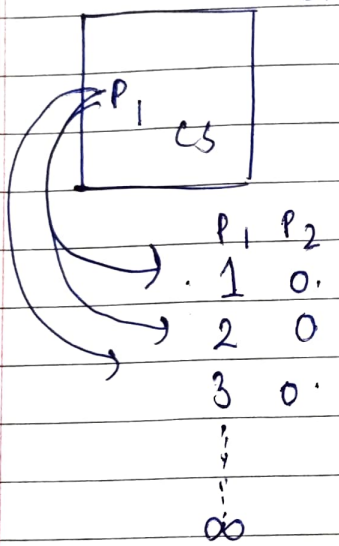


2. Progress → when both can use common code there is progress.



P_1 can't enter due to some code written in P_2 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_1 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 ___/___/___

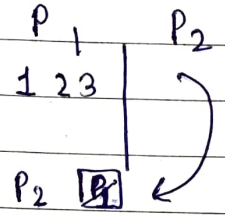
do {
acquire lock.
CS
release lock
}.

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 : P_1 P_2

lock = 0 → CS is vaccant
lock = 1 → CS is full



lock ∅ 1 ∅ 1

Case 2:

P_1	P_2
1 2 3	1 2
	CS.

L = ∅ 1 1

P_2P_1 → No mutual exclusion
CS

3.6 Test and Set Instruction

preemption

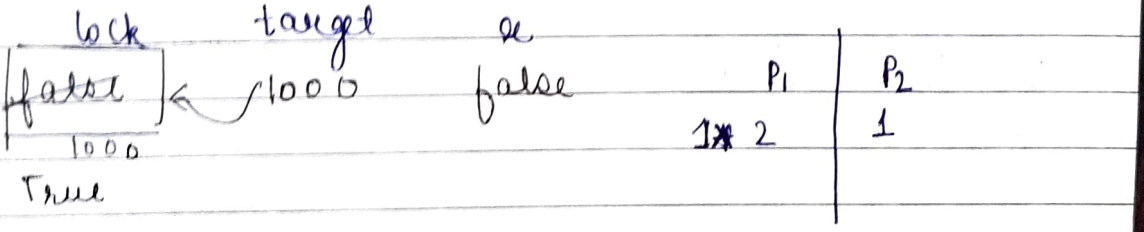
```
1. while (LOCK == 1);
2. LOCK = 1
```
Entry code

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 r1;
   }
```



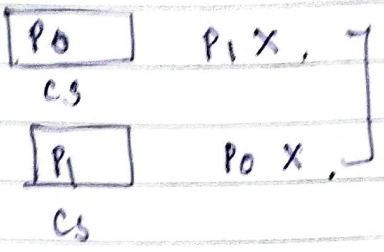
Mutal exclusion as well as Progress acheieved.

3.7 Turn variable

1) 2 process solution
2) Run in user mode

Process "P0"	Process "P1"
Entry code → while(turn != 0)	while (turn != 1)
CS	CS
Exit code → turn = 1 ;	turn = 0 ;

int turn = 0 / 1



} Mutual exclusion is satisfied

2) Progress (there is no progress)

when turn = 0

3) Bounded wait → 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) → Running cooperative process side by side

Semaphore
- Counting (-∞ to ∞)
- Binary (0, 1)

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

Entry section

Exit section

```
Down (Seamphore S)
{
S.value = S.value - 1;
if (S.value < 0)
  {
  put process (PCB) in
  suspended list sleep();
  }
else.
  {return;
  }.
```

```
UP (Seamphore S)
{
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_1 P_2 P_3

Entry code

CS :::

Exit section

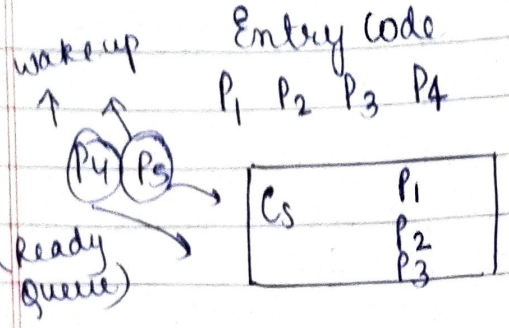
P_1 P_2 P_3 P_4 (Process control block)
P_5

CS P_1
P_2
P_3

Block list (sleep())
P_4
P_5.

S

3 2 1 0 -1 -2 (S.value < 0)



Down

S

-4

↓

4 processare there which is in CS.

Exit code.

S

3 2 1 0 -1 -2 -3 -4

S

0

↓

Further no process can come in CS.

Up

S

0

No process in suspend list

S

10

How many process can succesfully come in CS.

S = 10

Successful operation = 10

S	6 P	final value of S!
10	4 V	

10 - 6 = 4 + 4 = 8

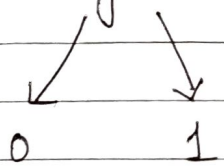
Date ___/___/_____

S 5P 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 - ∞ to ∞.
Binary Semaphore → 0, 1

```
Down (Semaphore S)
{
 if (value == 1)
    {
    S.value = 0;
    }
 else
    {
    Block this Process.
    And place in suspend
      list sleep();
    }
}
```

```
Up (Semaphore S)
{
 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
S = 0
} Successful opertn.

S = 0
sleep();
block
} Unsuccessful opertn.

S = 0
Up
S = 1

S = 1
Up
S = 1

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

S = 1

P_1	P_2
Down(s)	Down(s)
CS	CS
Up(s)	Up(s)

Q. Each process Pi [i = 1 to 9] execute the following code.

P_1 P_2

Pi {i=1 to 9}

```
repeat
P(mutex)
[CS]
V(mutex
forever.
```

Entry section → P(mutex)
Exit section → V(mutex

mutex
[1]

P10

```
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]

for P10

mutex : 0 1 0 1 0 1 0 P1 P10 P2

CS: [P1 P10 P2] → P3

P1 P2 P3 P10 → P4

P1 P2 P3 P4 P10 → P5

P1 P2 P3 P4 P5 ← P5 P1 P2 P3 P4. ← P10 P1 P2 P3 P4 P10

Block: P3 P4 P5 P6 P7 P8 P9

Page No.

Date

maxm no of process can be 10 but putting P10 in and out of critical Section.

Soultion of Producer Consumer using Binary Semaphore

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

```
Produce_item(itemp);
1) down(Empty);
2) down(s);
3) Buffer[IN] = itemp;
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
3	d
4	
5	
6	
7	

IN = 3

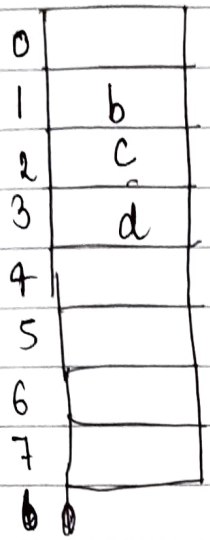
OUT = 0

Empty = ~~5~~ 4 full = ~~3~~ 4 S = ~~1~~ 1 In = (3+1) mod 8 = 4 mod 8 = 4

Page No.

For consumer.

Empty = 0 4 5 In = 4
full = 4 3 Out = 0 1 ((0+1) mod 8)
S = 1 0 1

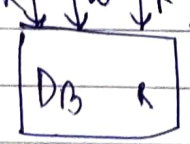


With preemption (context switching)

Empty = 5 4 5
full = 3 2 (Consumer in CS)
S = 1 0 1 0
out = 0 1 ↳ consumer can't do down of S.
In = 0 4

3.12 → 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) R-R (no problem)
W-R (problem)
W-W (problem)

Date ___ / ___

Read count → no of readers in buffer.

```
int rc = 0;
Semaphore mutex = 1;
Semaphore db = 1;
void Reader(void)
{
  while (true)
  {
    down(mutex);
    rc = rc + 1;
    if (rc == 1) then
        down(db);
    up(mutex)
```

Entry code

C.S. DB

```
    down(mutex)
    rc = rc - 1;
    if (rc == 0) then up(db);
    up(mutex)
    Process_data
  }
}
```

exit

```
void Writer(void)
{
while (true)
{
down(db);
```

DB

```
up(db);
}
}.
```

To synchronise problem we used Semaphores.

Case 1: R_1 comes first

$RC = 0 \to 1$ mutex $= 1 \to 0$ db $= 1 \to 0$

down(mutex)

DB R_1

For writer

$RC = 1$ mutex $= 1$ db $= 0$

down(db)

↳ db $\neq -1$ so the process gets blocked.

Case 2: W_1 comes first

Page No.

Date

$R_c = \not{0}\ 1$ mutex $= \not{1}\ 0$. db $= \not{1}\ 0$

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 $= \not{1}\ 0$.
When w_2 comes db $\neq -1$ so it gets blocked

Case 4: Read -Read

$R_c = \not{0}\ \not{1}\ 2$ mutex $= \not{1}\ \not{0}\ \not{1}\ 0\ 1$ db $= \not{1}\ 0$

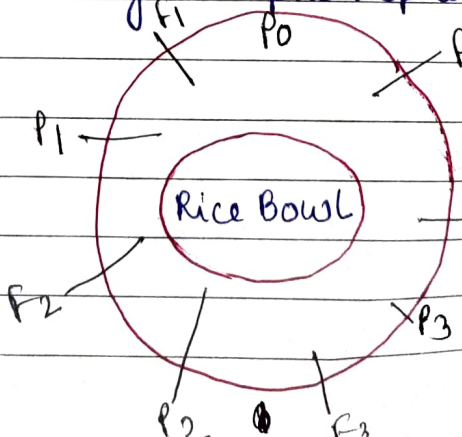
DB R_1 R_2

$2 == 1$ no
down(~~mutex~~ db); X

up(mutex)

3.13 Dinining Philosphers Problem.

A dining table having 5 philosphoers & 5 forks



```
void Philosopher (void)
{
  while (true)
  {
    Thinking ();
    take_fork (i);  ← left fork
    take_fork ((i+1) % N);  ← Right fork
    EAT ();
    put fork (i);
    put fork ((i+1) % N);
  }
}
```

Philosopher → think
→ 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)

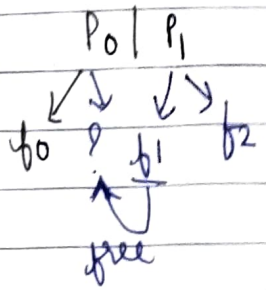
Right fork = 2 (f_2)

Eating

f_1

f_2

Case 3:



problem of race cond^n occurs.

S[i] → use array of semaphores

	S_0	S_1	S_2	S_3	S_4
Initially →	↓	↓	1	1	1
	0	0			
	1	1			

(Initialize every semaphore with as when initalized with 0 it waits & get blocked)

Now

```
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) % N)
   }
 }.
```

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

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

Date ___/___/_____

Case3: S_0 S_1 S_2 S_3 S_4

↓ ↓ ↓ ↓ ↓

1 1 1 1 1

P_0 comes first

S_0

↓

0

It gets preempt

P_1 comes

S_1

↓

0

preempt

P_2 comes

S_2

↓

0

preempt

P_3 comes

S_3

↓

0

preempt

P_4 comes

↓ S_4

↓ 0

prempt

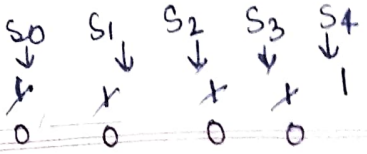
for right fork (blocked)

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

Q. How to remove deadlock?

On changing the sequence of one process.

P_4 S_0 S_4



Date

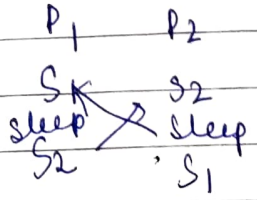
blocked → P4 S0 S4

We can change sequence of any philosopher.

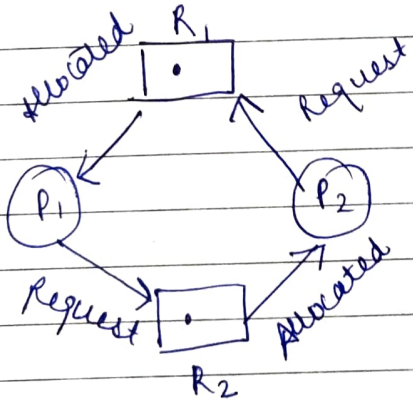
Nth philosopher

Wait (take fork (S(i+1) mod N))
wait (take_fork (si));

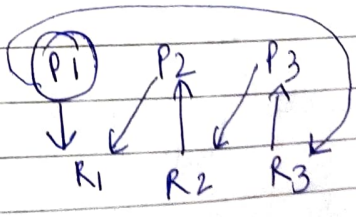
4.1 Deadlock concept



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



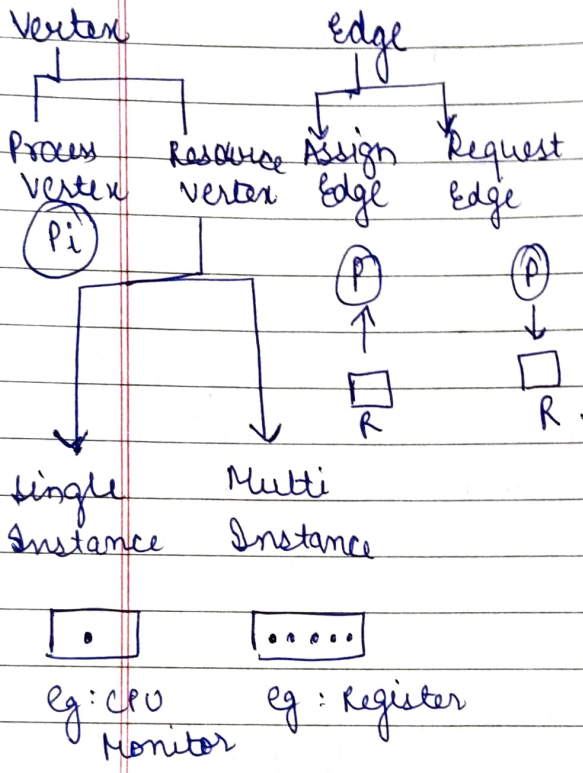
Page No.

Date

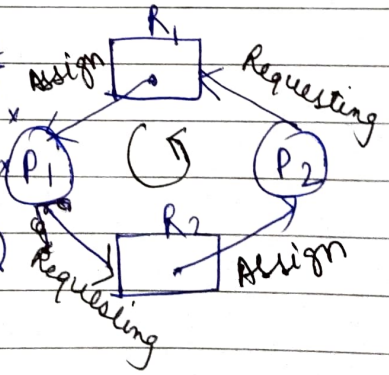
Necessary conditions for deadlock.

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

4.2 Resource Allocation Graph in Deadlock (RAG)



	Allocate		Request	
	R_1	R_2	R_1	R_2
P_1	1	0	0	1
P_2	0	1	1	0

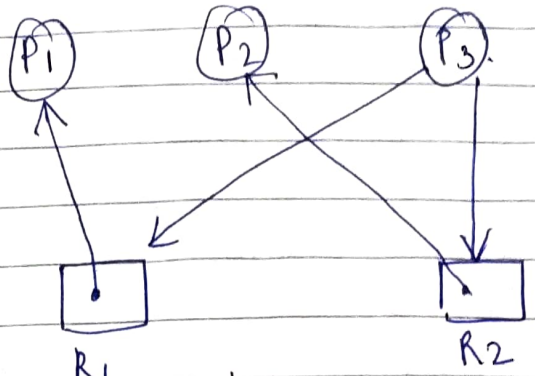


Circular wait

Availability (Deadlock) method

(0, 0)
R_1 R_2

Single Instance



	Allocation		Request		
	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	✓

Availaibility (0, 0)
(1, 0) → after P_1 terminate
(1 1) → after P_2 terminate

No deadlock occurs.

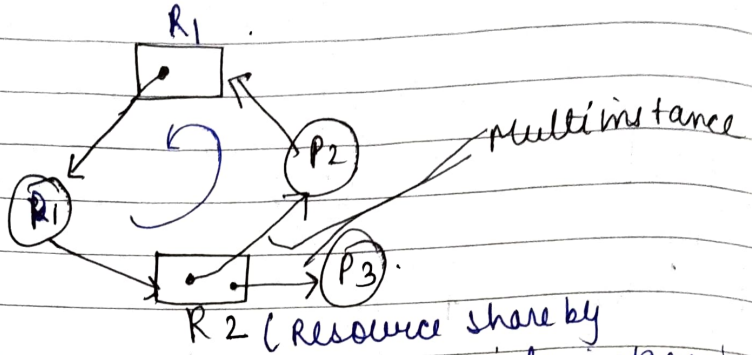
Waiting
- finite (Starvation)
- Infinite (Deadlock)

If RAG (Circular wait) (Cycle) → Always in Deadlock
↓
A single instance

Single instance + Circular weight = Deadlock.
Multi instance + CW = Possible (may or may not)

Date ___/___/___

4.3 Multinstance RAG.

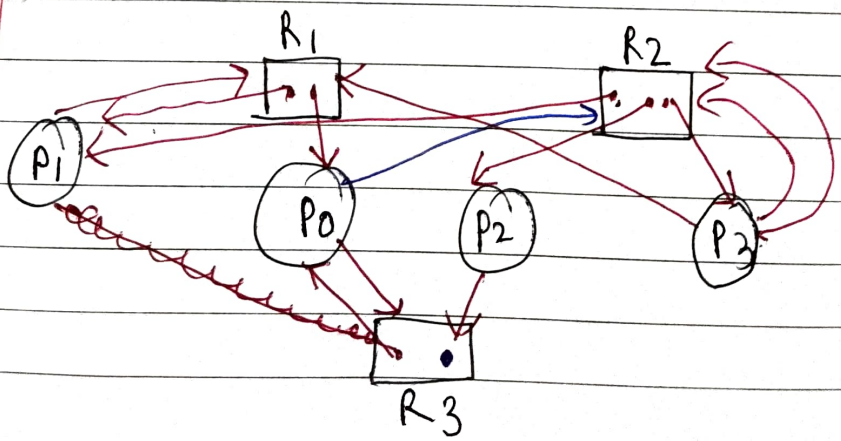


Multi instance RAG.

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 ✓

Current avai → (0, 0) –
0 1 –
1 0 ✓

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	0 1	1
P_1	1	1	0	1	1 0	0
P_2	0	1	0	0	0	1
P_3	0 0	1	0	1	2	0

curr av = R_1 R_2 R_3

(0 0 1)

$P_2 \rightarrow$ 0 0 1 0

0 1 1

$P_0 \rightarrow$ 1 0 1

1 1 2

	R_1	R_2	R_3
	1	1	2
P_1	1	1	0
	2	2	2
P_3	0	1	0
	2	3	2

2 3 2 - curr availability

→ No deadlock in the system.

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

Page No.

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 write any code for deadlock, so it's easy to avoid as its 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 gets 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

Date __/__/____

Either remove all the condition or try to false any one of the situation.

1) Mutual Exclusion (there should be no sharing bet^{n} diff^{n} 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^{n} the resources)

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

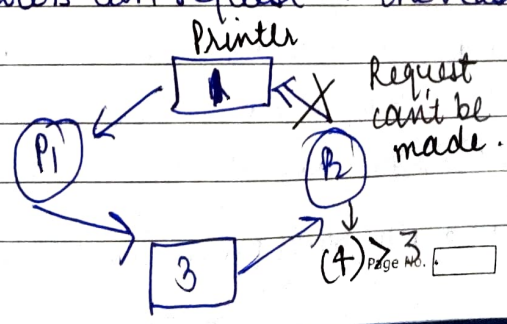
3) Hold wait (Holding some resource & waiting for some resources).

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. CD
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)
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

CPU (A) = 10 (7)
Memory (B) = 5 (2)
Printer (C) = 7 (5)

(Total AT) Allocation → how much is allocated
Max Need → how much is needed
(Total AT - A) Availability → how much resource is available
(Max Need - Allocation) Remaining Need

Process	Allocation			Max Need			Available (10-7)	(5-2)	(5-3)	Remaining need Max - Allocation		
	A	B	C	A	B	C	A	B	C	A	B	C
⇒ P_1	0	1	0	7	5	3	3 (2)	3 (0)	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.

$P_2 \rightarrow P_4 \rightarrow P_5 \rightarrow P_1 \rightarrow P_3$ (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 (1)	3 (3)	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									

8 4 6

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

No deadlock

Date ___/___/___

4.7 Question Explanation on Deadlock

Q A System requires 2 units of Resources having 3 process. The min^m. no of units of 'R' such that no deadlock will occur.

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

P_1 P_2 P_3

$3 \times 2 = 6$ resources

but we need minimum no of resources.

$R = 2$

P_1	P_2	P_3
1	1	
↑ need 1 more	↑ need 1 more	

→ deadlock.

P_1	P_2	P_3
1		
1		

$P_1 \rightarrow P_2 \rightarrow P_3$

but not always true

P_1	P_2	P_3
1	1	1
↑		
1		

Page No.

R = 3

P_1	P_2	P_3
1 } x	1 } x	1 } x
1 }	1 }	1 }

P_1	P_2	P_3	
1	1	1	→ deadlock can occur

R = 4

P_1	P_2	P_3	
1 }	1 }	1 }	→ Deadlock never occurs.
1 }	1 } x	1 } x	
x			

(min req) + 1

3 + 1 = 4

Max^m resources allocated still deadlock - 1

P_1	P_2	P_3	
3	4	5	(min req)
2	3	4	= (4+3+2) + 1
			= 10

Max^m resources allocated still deadlock = 9

Date / /

4.8 Crate Question on Deadlock.

Q. Consider a system with 3 processes that share 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 ? → 2.

R = 4 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

K=3.

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

K=3

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

Deadlock

Ans. 2.

Another method to find

'R' resources
'n' processes $(P_1, P_2, P_3 \cdots P_n)$
'd' demand $(d_1\ d_2\ d_3 \cdots d_n)$

max^m resources but still deadlock

P_1	P_2	P_3	
2	2	2	= 6
1	1	1	= 3 + 1 = 4 (free from deadlock)

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

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

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

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

↓ total resource

→ total processes.

$4 + 3 > 3 \times 2$

$7 > 6$ → True

4 processes
4 resources

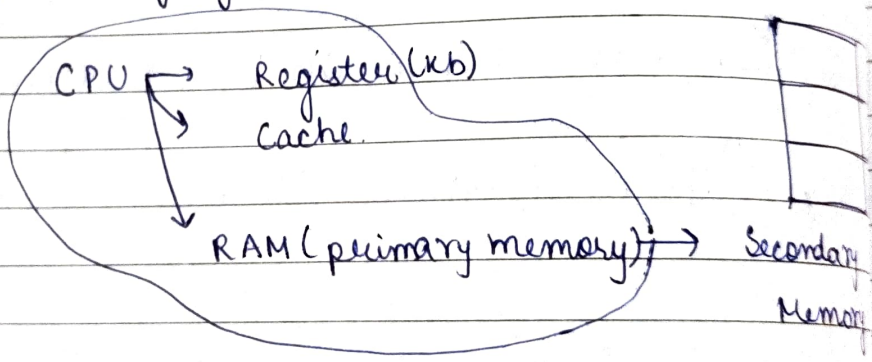
$4+4 > 4 \times 1$
$8 > 4$ (Valid)

$4+4 > 4 \times 2$
$8 > 8$ (false)

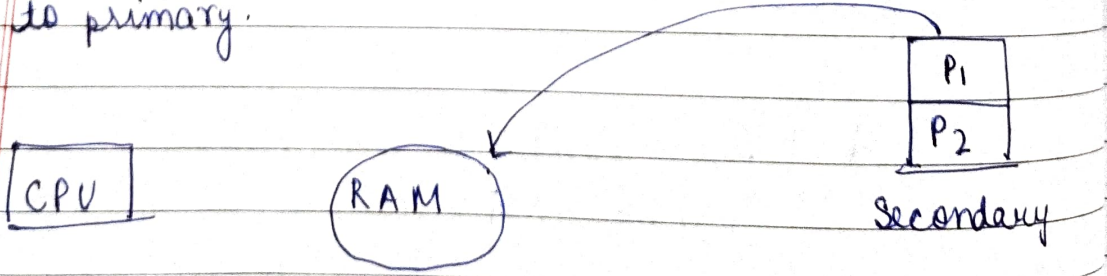
5.1 Memory Management and Degree of Multi programming

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 comparision 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)

RAM
↙ ↘
CPU I/O

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

RAM — Process

4MB — 4MB.

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

$K \rightarrow$ i/o operation (70%)

CPU = $(1-K)$ (30%.)

utilization.

RAM — Process

8MB — 4MB.

$\frac{8}{4} = 2$ Process.

$K = 70\%$.

$K^2 \rightarrow$ i/o operⁿ.

CPU utilization = $(1-K^2)$

$= 1-(0.7)^2$

$= 76\%$.

RAM — Process.

16MB. — 4MB.

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

$K^4 \rightarrow$ i/o operⁿ.

CPU utilization = $1-(0.7)^4$

$= 93\%$ approx

↓

Efficiency increases

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

5.2 Contigious and non contigious.

Memory management Techniques

(Continuous memory address allocation)

Contiguous

Fixed Partition (Static) (Make fixed slots)

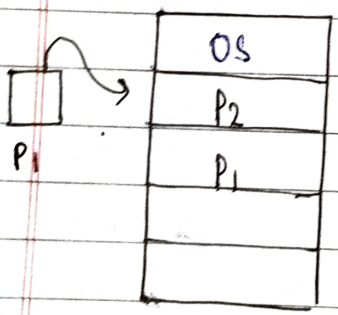
Variable partition. (Dynamic) (provide at run time)

(Non continuous memory allocation)

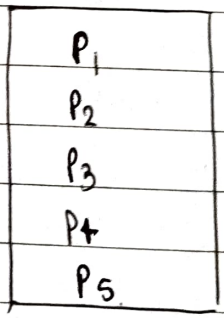
Non-contiguous

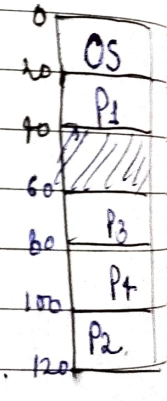
- Paging
- Multilevel Paging
- Inverted paging
- Segmentation
- Segmented Paging

Ready state process & CPU utilization (efficiency)



Contiguous allocation





Non-contiguous allocation

Date ___/___/_____

5.3 Internal fragmentation / Fixed size Partitioning

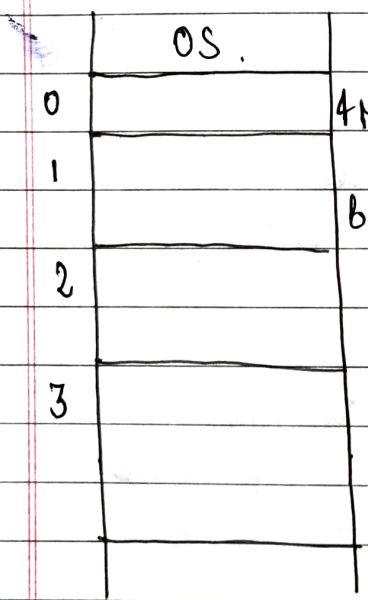
Whenever the process are coming into RAM, how we are allocating it the space.

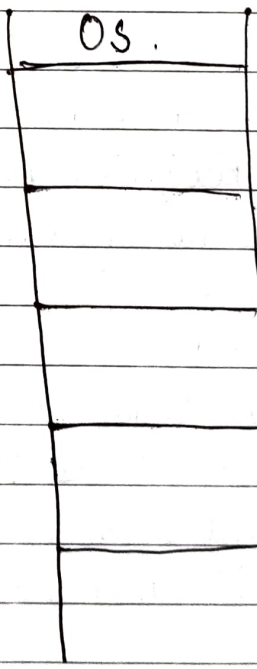
Points to remember.

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.





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

Page No.

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

Limitations

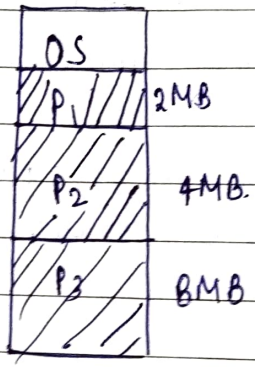
1) Internal fragmentation
2) Limit in process size.
3) Limitation on degree of multiprogramming. We can't bring more and more processes.
4) External fragmentation

5.4 Variable Size Partitioning

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

$P_1 = 2MB$

$P_2 = 4MB$

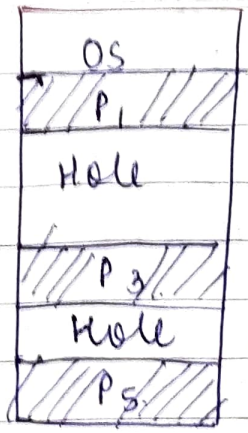


Notes

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) Lot of Holes is created



In which we should put which process.

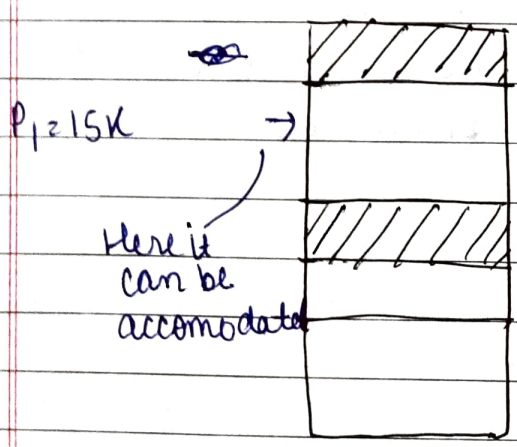
5.5 First fit, Next fit, Best fit Worst fit

First fit → Allocate the first hole that is big enough

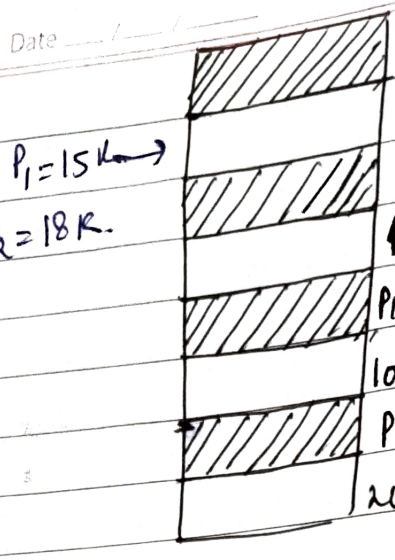
Next fit → Same as first fit but start search always from last allocated hole.

Best fit → Allocate the smallest hole that is big enough

Worst fit → Allocate the largest hole.



first fit
(find the best possible large loc^{n}. which could accomodate process & place it in the RAM)

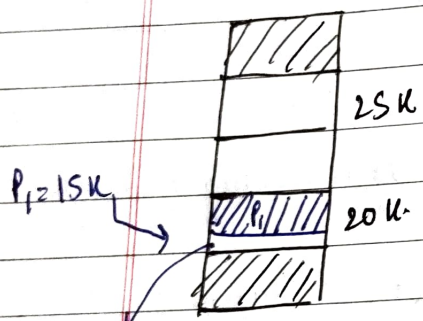


$P_1 = 15K$

$B = 18K$.

Next fit

(We place a pointer at last allocated first fit & start searching for next empty hole. We don't need to start from begining.



$P_1 = 15K$

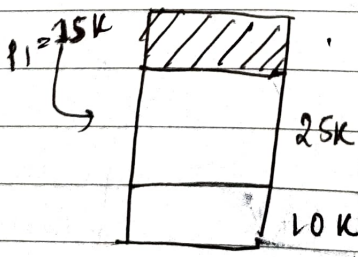
Internal fragmentation is least

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).

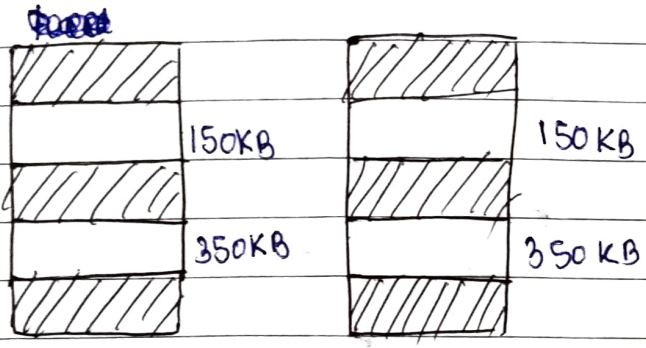


Date ___/___/_____

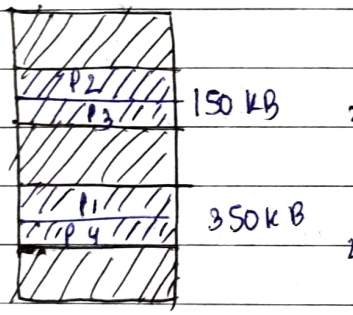
5.6. Gate Question On fits

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

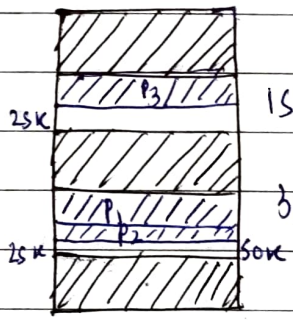
a) best fit but not first fit.
b) first fit but not best fit
c) Both
d) None.



For first fit



For best fit



P1 P2 P3 P4
300 25 125 50

P4 can't be accomodated

5.7 Gate 2007 Question on fits

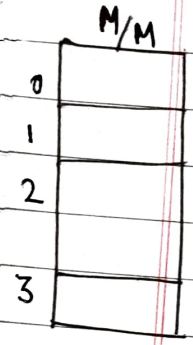
Date

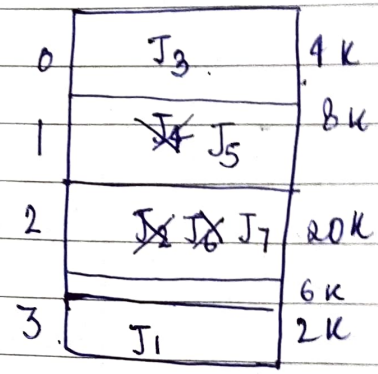
Best fit

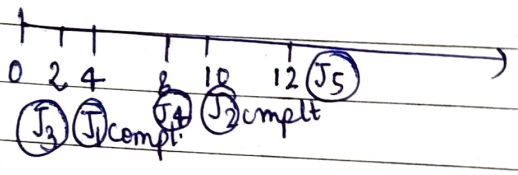
Req. No	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8
Request size	2K	14K	3K	6K	6K	10K	7K	20K
Usage time	4	10	2	8	4	1	8	6

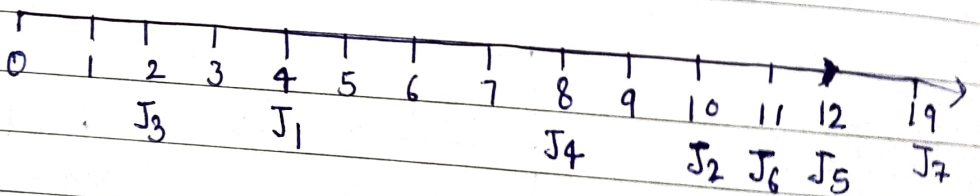
Calculate the time at which J_7 will be completed

a) 17 b) 21 c) 19 d) 20







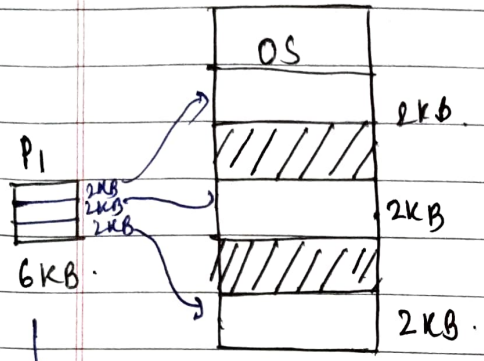


In case of J_5 & J_6 we would take out the one which is completed first. And then put it & calc. the final time by initial + Usage time.

Date ___/___/___

5.8 Need of Paging (Non contiguous Memory Allocation)

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

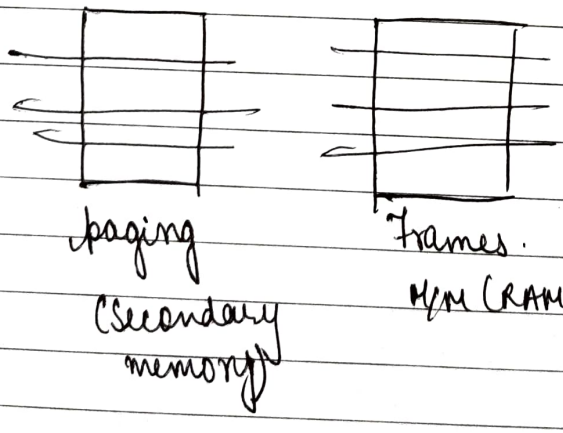


process is divided acc^n. to the hole size but the hole size changes on runtime so its a bit time consuming (division of process)

process is divided & put up in diff. memory location. M/M

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.



paging (Secondary memory)

Frames. M/M (RAM)

Thank You