Operating System:-

→ acts as an interface between user and hardware.

Software :- Tested programs + documentation.

Appln software :- Designed for specific task
eg :- VLC, MS Office.

System Software :- Provides ~~prog~~ platform to run appln software. e.g OS.

OS → are system software.

Operating System :-

→ Resource Manager.

CPU Memory I/O devices.

→ Process Management (Scheduling)

→ Memory management. (mainly RAM)(Primary memory)

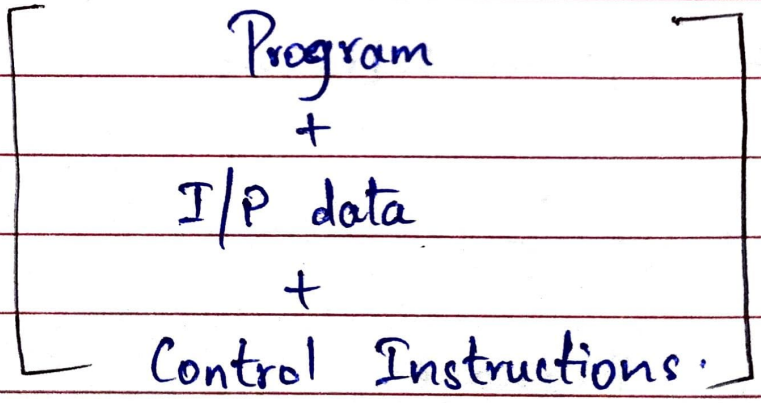
→ I/O devices management.

→ Storage management :- (Hard disk)(Secondary storage).

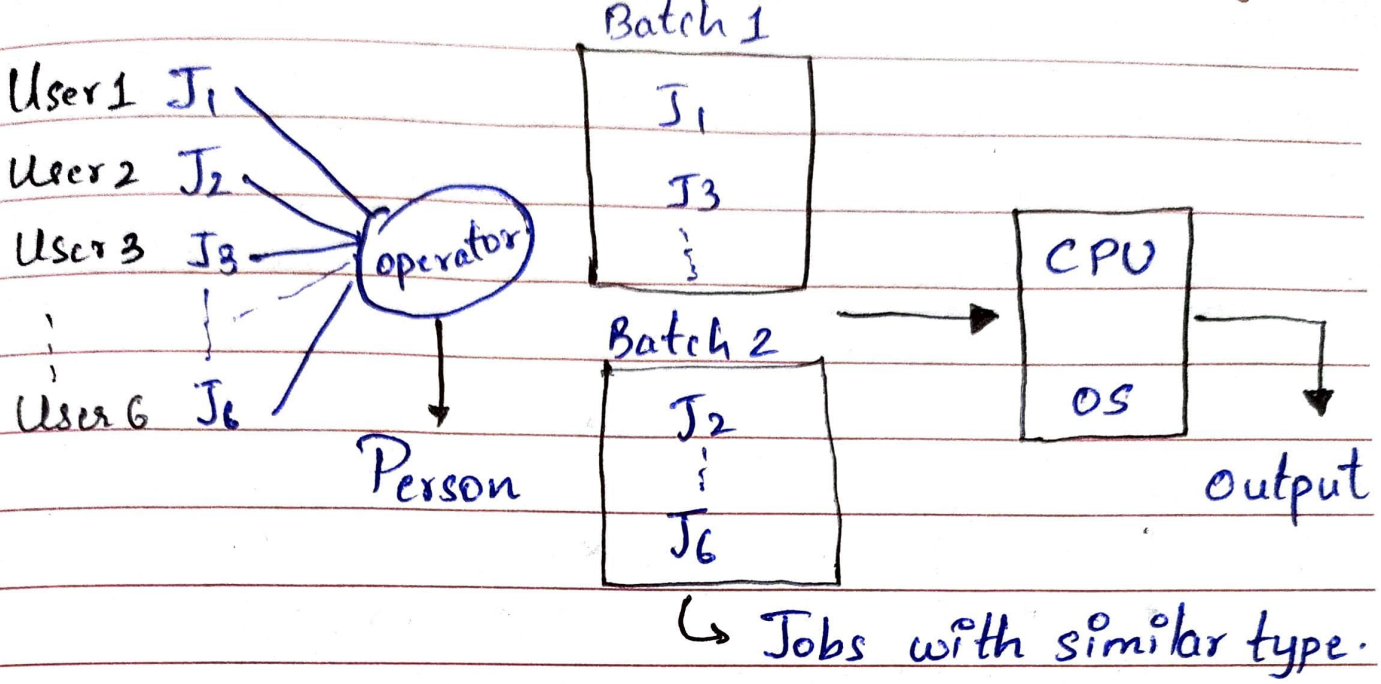
→ Security & ~~Protester~~ Protection:

Types of Operating System :-

1.) Batch OS :- Job

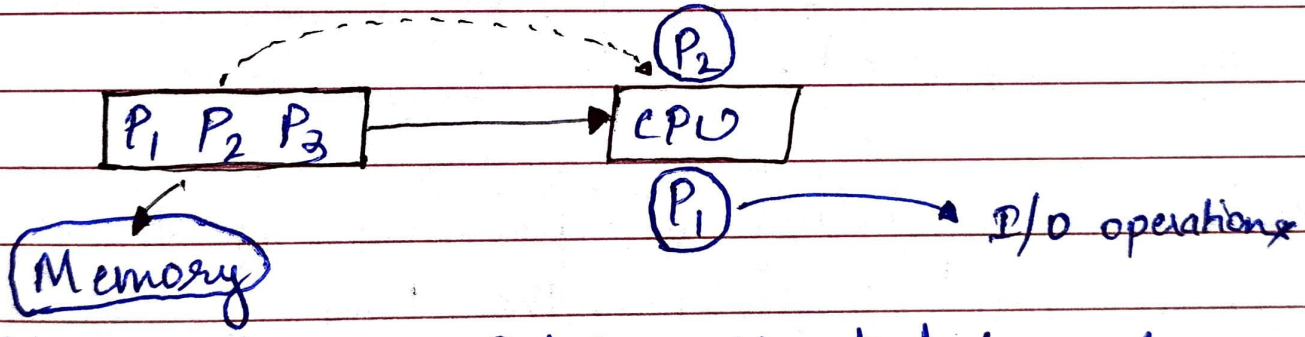


→ Non-Interactive system.
→ Punch-cards were used.
→ CPU Utilisation is very low. (CPU is idle oftenly)



↳ Jobs with similar type.

2.) Multiprogramming OS :-



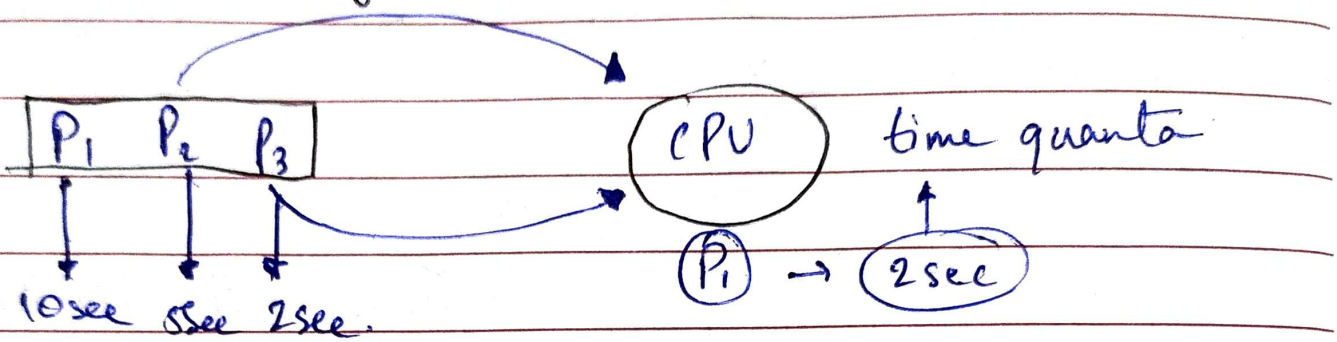
→ At a time CPU is allocated to only one process.

→ If P_1 is out for I/O operation then CPU is allocated to P_2. To reduce CPU idle time.

→ We cannot forcefully remove any process. the process itself has to be terminated or

has to perform I/O operations.

3. Multi-tasking OS:- / Time sharing OS :-



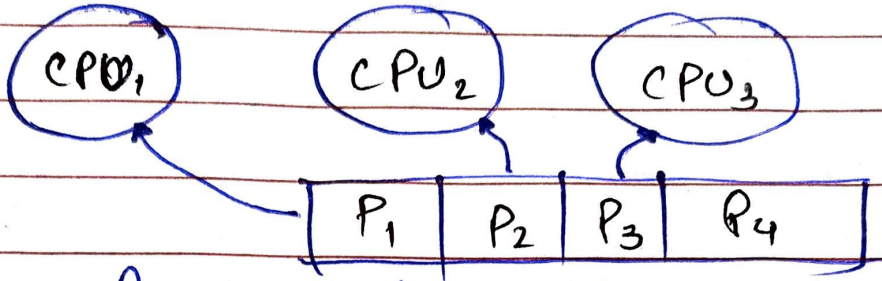
Time quanta :- fixed time for which the CPU is allocated to the process (each process in the memory).

After every shifting of processes if the process still requires the time to complete Then the process is sent back to main memory to wait for its next turn.

This OS is also known as multiprogramming with round robin.

NOTE:- CPU can be allocated to one process at one time.

4) Multi-processing OS :- (More than One CPU)



Because of multiple CPU's we can now say that more than one process is running at a time. But individually CPU will be allocated to one process at a time.

→ One computer Multiple CPU's are there

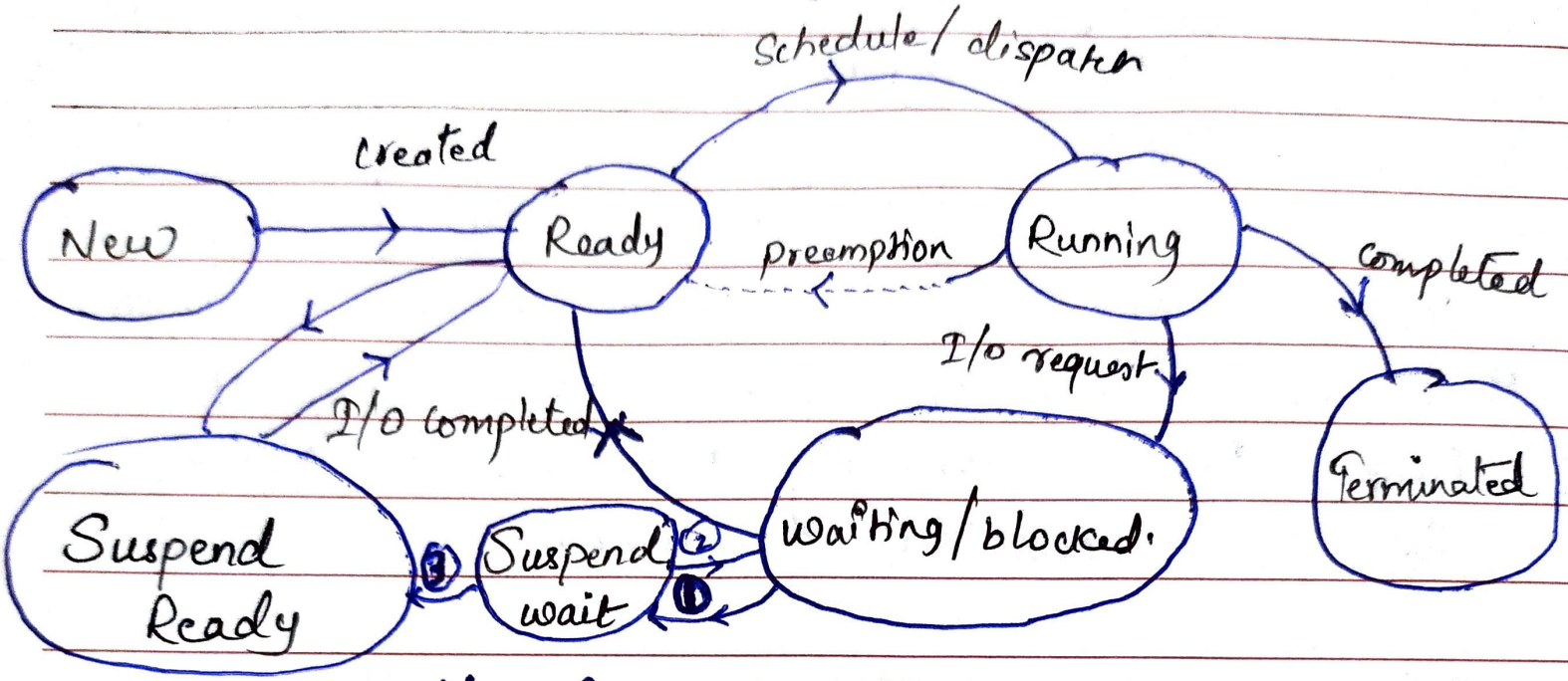
5) Real time Operating System :-

Soft RT , Hard ~~Time~~ RT ;

process can be completed anywhere within the time bound

The process must be completed at given fixed time.

→ Process State diagram:-



Non-Preemption diagram.

- Preemption :- In case of multitasking OS the process is returned to the ready state after spending a fixed time quanta in the CPU.

→ ~~Processing~~ Processees in the state — Ready, Running, waiting states are kept in the main memory, and there could be n-number of processes so, to avoid overcluttering of the main memory the OS swaps out the low priority processes to the secondary memory

and i.e., Suspend ready. and when the process has to resume it is again called back to ready state.

(1) If many processes are in waiting states performing their I/O operations. then also main memory can be exhausted so, to avoid this the OS moves low priority processess to the secondary storage i.e., Suspend wait.
And in this state also the process continues to perform its I/O operations.

(3) ~~If~~ Every process after Suspend wait is moved to suspend ready to be executed ~~resumed~~ again by the CPU.

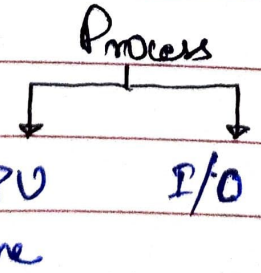
Schedulers:-

Processes are picked from 'New' state to 'ready' state with the help of 'Long term scheduler'

Degree of Multiprogramming:- The max. no. of processes that can reside in the ready state at one time.

Processes are picked from 'Ready' state to 'Running' with the help of short term scheduler. (Dispatcher)
This short term scheduler decides which processes to be picked.

Processes picked from 'Ready' to 'Suspend ready' or 'waiting' to 'Suspend wait' state with the help of 'Medium term scheduler.



→ If a process is spending more time in CPU then that process is known as "CPU bound".

→ Similarly if a process is spending more time in I/O operation then that process is known as "I/O bound".

FCFS — First come first serve - Simplest CPU scheduling algorithm.

→ assigns CPU to the process which comes first. (non-preemptive)

The process which comes in ready state first, is allocated to CPU.

Non-preemptive: Once the process is allocated to the CPU then that process cannot be removed from the CPU forcefully, until its termination.

Process	Arrival time (AT)	Burst time (BT) → CPU Time
P_1	2	2
P_2	0	1
P_3	2	3
P_4	3	5
P_5	4	4

AT → time at which the process arrives in ready-queue.

CT → Completion time → The time at which the process is terminated.

TAT → Turnaround time → TAT = WT + BT
TAT = CT − AT

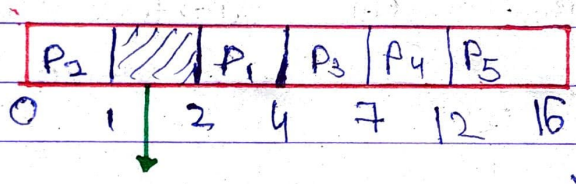
WT → Time for which the process has to wait in ready queue.

WT = TAT − BT

Response time → time at which the process is called for the first time.

process	AT	BT	CT	TAT	WT	Response time
P_1	2	2	4	2	0	0
P_2	0	1	1	1	0	0
P_3	2	3	7	5	2	2
P_4	3	5	12	9	4	4
P_5	4	4	16	12	8	8

Gantt chart :



If AT of two processes is same then the process which is registered first ~~has to be~~ in the process column has to be allocated to the CPU first.

In non-premptive scheduling Response time = WT.

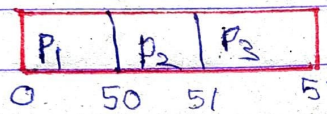
RT ⇒ (Time at which the process is executing) − (Arrival time)

FCFS - Convoy Effect :-

case - I

process	AT	BT	TAT	WT
P_1	0	50	50	0
P_2	1	1	50	49
P_3	1	2	52	50

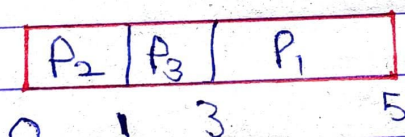
$99/3 = 33$



case - II

process	AT	BT	TAT	WT
P_1	1	50	51	2
P_2	0	1	1	0
P_3	0	2	3	1

$3/3 = (1)$ → Avg. waiting time



Convoy Effect :- processes with larger burst time should be called at the last to get maximum efficiency in waiting time.

SJF / SJN – Shortest Job First / Shortest Job Next.

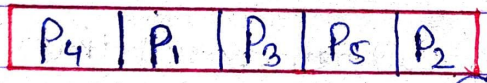
→ Out of all available (waiting) processes, it selects the process with the smallest burst time to execute next.

→ Pre-emptive → (SRTF) → Shortest remaining time first.
→ Non-pre-emptive → default, when nothing is specified.

Non-pre-emptive :- (SJF / SJN) :-

Process	AT	BT	CT	TAT	WT	Response time
P_1	2	1	7	5	4	4
P_2	1	5	16	15	10	10
P_3	4	1	8	4	3	3
P_4	0	6	6	6	0	0
P_5	2	3	11	9	6	6

Grantt Chart :-



= BT (total)

SJF / SJN – with Pre-emption :- (SRTF) :-

Process	AT	BT	CT	TAT	WT	Response Time
P_1	2	1	3	1	0	0
P_2	1	5	16	15	10	10
P_3	4	1	5	1	0	0
P_4	0	6	11	11	5	0
P_5	2	3	7	5	2	1
				Avg	Avg	
				6.6	3.6	

SRTF gives most minimal WT.

Gantt :-

Chart :-

P_4	P_4	P_1	P_5	P_3	P_5	P_4	P_2

0 1 2 3 4 5 7 11 (16) = Total Burst time.

Whenever a new process arrives, there may be pre-emption of the running processes.

Only if the newly arrived process has shorter burst time than the remaining burst time of currently running process then only pre-emption is allowed / done.

→ Firstly processes with shortest AT has to be allocated first to the CPU for 1 unit time until another process arrives.
After every allocation again the burst time of all the available processes has to be checked.

→ Once all the processes has been executed once, then the process with shortest burst time is executed till its termination.

* Advantages and Disadvantages of Shortest Job first (SJF):-

→ **Advantages :-**

1. SRTF gives the minimum average WT and min avg. TAT.
2. SRTF gives the better response time than FCFS.
3. SRTF provides a standard for other algorithms in case of average WT.
4. SRTF gives maximum throughput.

$$\text{Throughput} = \frac{\text{Number of processes}}{\text{Time taken to complete.}}$$

→ **Disadvantages :-**

1. SRTF cannot be implemented.

Reason:- Because before execution we cannot determine the burst time of processes that is why this is not implementable.

2. SRTF provides starvation problem with processes having larger BT.

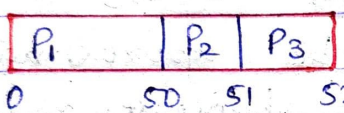
→ processes having larger BT has to wait alot for its termination or its turn to execute.

→ SJF can also suffer with convoy effect (Non-preemptive)

Case-1 :-

Process	AT	BT	CT	TAT	WT	Response time
P_1	0	50	50	50	0	0
P_2	1	1	51	50	49	50
P_3	2	1	52	50	49	51
					Avg	

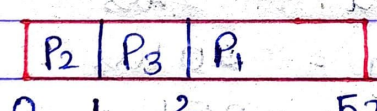
⇒ 32.6



Case-2 :-

Process	AT	BT	CT	TAT	WT	Response time
P_1	1	50	52	51	1	
P_2	0	1	1	1	0	
P_3	0	1	2	2	1	
					Avg	

= 0.6



Avg WT = 32.6 (case 1) } convoy effect

Avg WT = 0.6 (case 2) }

→ SRTF → do not have convoy effect.

SJF with predicted burst time (BT)

1) Prediction based on process size :-

Process	Process size	Burst time
Pold	100 KB	5 units
Pnew	102 KB	≃ 5 units

Sometimes this process doesn't work because the time taken by the processes might be approximately same but the nature of the processes may be different.

2) Prediction based on process type :-

Example :-

Delhi ——— 1500 km ——— Mumbai

flight - 2hrs
train = 20 hrs
car = 25 hrs

although the distance (size) is same but the nature of transportation (process) is different thus the time (burst time) taken is different.

3) Prediction based on simple Averaging :-

Suppose 5 processes have been completed P_1, P_2, P_3, P_4 & P_5, in their respective time t_1, t_2, t_3, t_4, t_5. then

P_1 P_2 P_3 P_4 P_5

$$\left[\frac{t_1 + t_2 + t_3 + t_4 + t_5}{5}\right] = \text{Avg.}$$

and this avg BT may be the BT of the P_6.

$$\left[\text{Simple averaging} = \frac{1}{n}\sum_{i=1}^{n} t_i\right]$$

4) Prediction based on exponential Averaging :-

$$\mathcal{E}_{i+1} = \alpha A_i + (1-\alpha)\mathcal{E}_i \quad \text{--- (1)}$$

$\mathcal{E}_{i+1}$ => expected time for processes $i+1$.

A_i => Actual burst time of process i.

α => Smoothening factor.

$$0 \le \alpha \le 1$$

eqn (1) => The expected time for process depends on
= the Actual & expected burst time of the previous process.

$$\varepsilon_i^\circ = \alpha A_{i-1} + (1-\alpha)\varepsilon_{i-1}$$

A_{i-1} → Actual BT of previous process

ε_{i-1} → Expected BT of previous process.

$$\varepsilon_{i-1} = \alpha A_{i-2} + (1-\alpha)\varepsilon_{i-2}$$

⋮

till the very first process / initial process.

In eqn (1) if $[\alpha = 0]$ then the equation becomes.

$$\varepsilon_{i+1}^\circ = (1-\alpha)\varepsilon_i^\circ$$

⇒ the expected time of the process totally depends on 'ε_i' i.e., expected time of the previous process, But this is not true

as $\varepsilon_i^\circ = \alpha A_{i-1} + (1-\alpha)\varepsilon_{i-1}$

↳ ε_i depends on this ↑

also the same if $[\alpha = 1]$ in eqn (1)

the $\varepsilon_{i+1} = A_i$

↳ Actual burst time. → this is not true due to the same reason as above.

α is neither '0' nor '1' it is in b/w. always.

Question:- $\alpha = 0.5$, $\varepsilon_1 = 5$, $\varepsilon_5 = ?$

Process	BT
P_1	4
P_2	8
P_3	5
P_4	6

$\Rightarrow \varepsilon_5 = (0.5)6 + (0.5) \times \varepsilon_4$

$\Rightarrow \varepsilon_4 = (0.5)5 + (0.5) \times \varepsilon_3$

$\Rightarrow \varepsilon_3 = (0.5)8 + (0.5) \times \varepsilon_2$

$\Rightarrow \varepsilon_2 = (0.5)(4) + (0.5) \times \varepsilon_1$

$\Rightarrow \varepsilon_2 = 2 + 0.5 \times 5$

$\varepsilon_2 = \underline{4.5}$

$\varepsilon_3 = (0.5)8 + (0.5) \times 4.5$

$4 + 2.25$

$\varepsilon_3 = \underline{6.25}$

$\varepsilon_4 = (0.5)5 + (0.5)6.25$

$\varepsilon_4 = 2.5 + 3.125$

$\varepsilon_4 = \underline{5.625}$

$\varepsilon_5 = (0.5)6 + (0.5)5.625$

$3.0 + 2.8125$

$= \underline{5.8125}$

$[\varepsilon_5 = 5.8125]$ ✓

Ans:- Expected time for the process P_5 is 5.8125

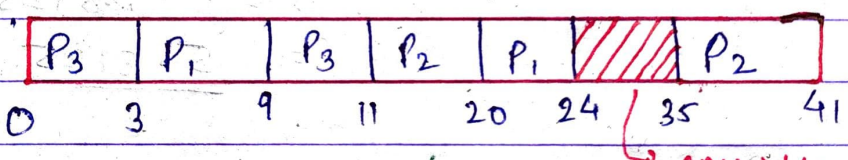
NOTE:- When the arrival time of all the processes is same then the SRTF acts as S.J.F with Non-preemption.

Question:- Three processes P_1, P_2, P_3 with process time 20, 30, 10 respectively. Each process uses the first 30% of its process time in CPU, then 50% in I/O & last 20% in CPU find avg. WT, TT, and Response Time RT if system follows SJF scheduling.

Ans:- Process time = CPU burst time + I/O burst time.

	Process time	AT	CPU BT	I/O BT	CPU BT	TAT	WT	Response time
P_1	20	0	6	10	4	24	14	3
P_2	30	0	9	15	6	41	26	11
P_3	10	0	3	5	2	11	6	0

Gantt's chart



Here WT will be considered only for CPU BT

i.e., TAT = WT + CPU BT.

WT = TAT − BT (CPU).

$$\text{Avg TAT} = \frac{24+41+11}{3} = \frac{76}{3} = \boxed{25.33}$$

$$\text{Avg WT} = \frac{14+26+06}{3} = \frac{46}{3} = \boxed{15.3}$$

$$\text{Avg. RT} = \frac{3+11+0}{3} = \frac{14}{3} = \boxed{4.66}$$

$= 0.7319.$

CPU utilization / efficiency :- $\frac{\text{Expected CPU time}}{\text{Actual CPU time}} = \frac{10+15+5}{41} = \frac{30}{41}$

Expected CPU time :- Only CPU burst time of the processes.

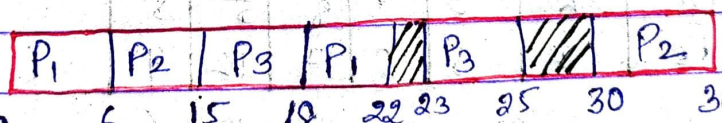
$= 73.17\%$

CPU idle time or % $= \frac{11}{41} = 0.2682$ or 26.82%

Question :- Same as previous → but with FCFS scheduling.

	Process time	AT	CPU BT	I/O BT	CPU BT	TAT	WT	RT
P_1	20	0	6	10	4	22	12	0
P_2	30	0	9	15	6	36	21	6
P_3	10	0	3	5	2	25	20	15
						83/3	53/3	21/3
						27.6	17.6	7
						Avg	Avg	Avg

Cantt Chart / graph :-



CPU utilization :- $\frac{\text{Expected}}{\text{Actual}} = \frac{30}{36} = \frac{5}{6} = 0.833 = 83\%$

CPU idle time :- $\frac{1+5}{36} = \frac{6}{36} = \frac{1}{6} = 16.66\%$

Round Robin Scheduling :-

Mode :- Pre-emptive.

-> This algorithm is used in time sharing systems.

-> Similar to FCFS with time Quantum.

Time Quantum :- The time or the period of time for which a process is allowed to run uninterruptedly in a pre-emptive multitasking OS. in one go.

criteria to pik any process for execution :-

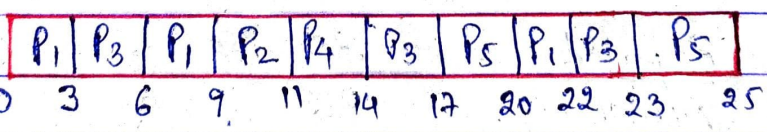
= Time quantum + Arrival time.

One of the best algorithm in case of Avg. response time.

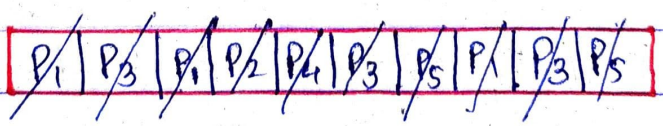
Time Quantum = 3 units

	AT	BT	CT	TAT	WT	RT
P_1	0	8	22	22	14	0
P_2	5	2	11	6	4	4
P_3	1	7	23	22	15	2
P_4	6	3	14	8	5	5
P_5	8	5	25	17	12	9
Avg :-				15	10	4

Gantt Chart.



Ready Queue :-



* If time quantum is very large then this Round Robin will function same as FCFS.

* If time quantum is very less then the Response time is the best we can get.

If time is very less then the number of process switching or context switching is much higher due to which the gantt chart becomes very large.
And also the context switching takes some time.

Time quantum range :- 10 - 100 ms.

Example :-

	BT	AT	CT	WT	TAT	RT
P_1	8	0	32	24	32	0
P_2	2	0	6	4	6	4
P_3	7	0	34	27	34	7
P_4	3	0	14	11	14	11
P_5	5	0	29	24	29	15

let context switching time is 1 unit.

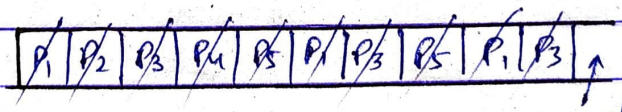
Q. Avg WT, TAT, RT, no. of context switches & CPU utilization ?

No. of context switches = 9

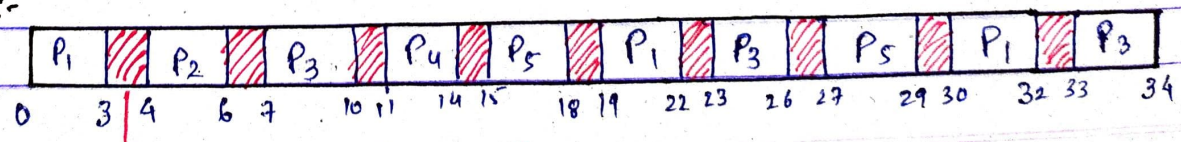
CPU utilisation = Expected / Actual = 25/34

= 73.5%

Ready Queue :-

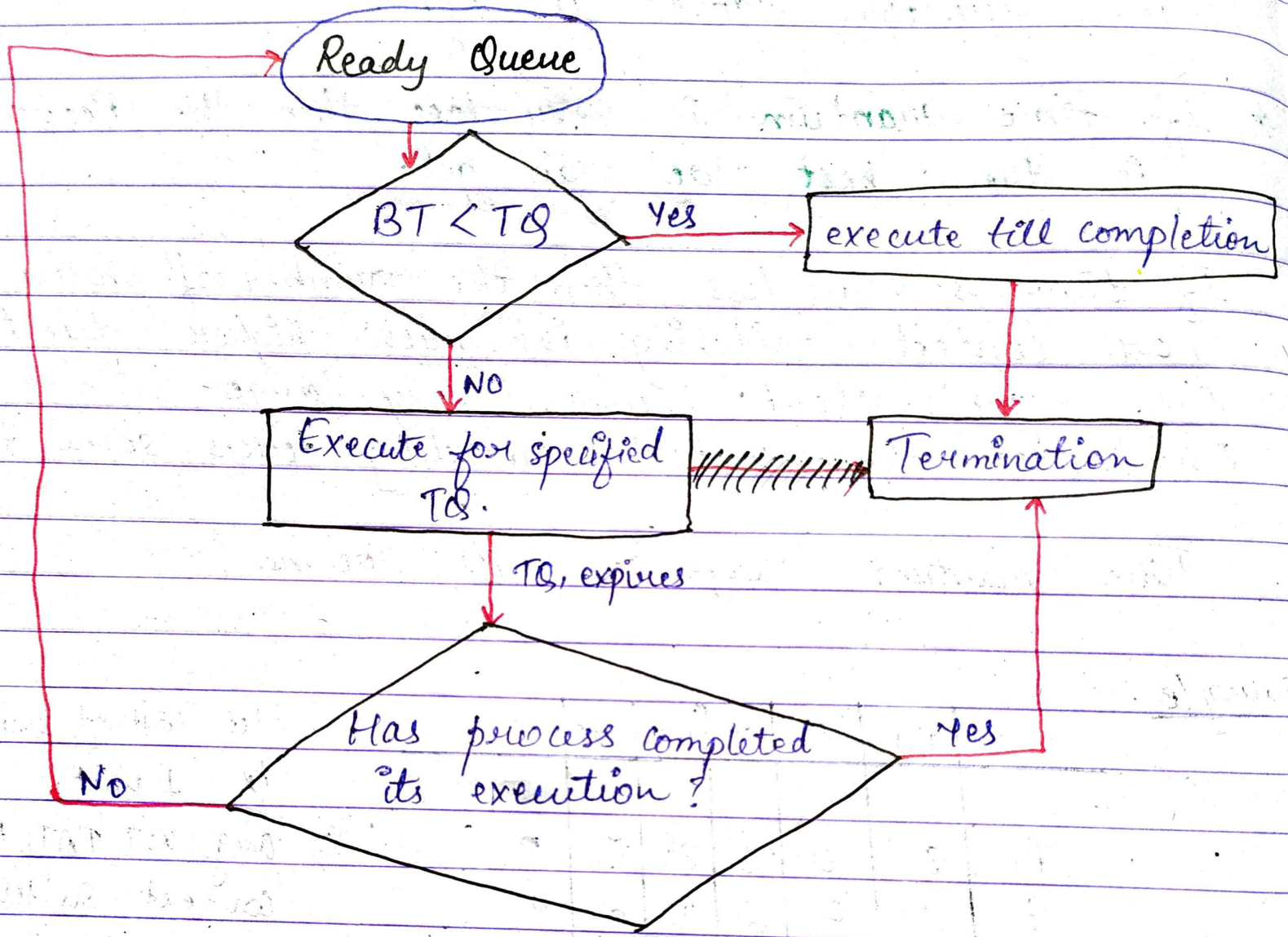


Gantt Chart :-



CPU idle time / context switching time.

Round Robin Algorithm Chart:-



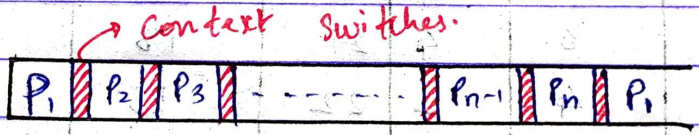
Advantages and disadvantages of RR algorithm.

Advantages:- ① Easy and simple to implement.
② Each process gets a fair share of CPU
③ No starvation effect.
④ No convoy effect.
⑤ deterministic response time.
⑥ Priority is same for each process.

Disadvantages:- ① Throughput depends on time quantum.
② If TQ is small - more overhead of context switching average wasting time increases.
③ If TQ is large - same as FCFS.
④ Deciding size of TQ is tough.

Question:- let there are 'n' processes in a system. Context switch time is 's' seconds what should be the max time quantum so that any process has to wait for atmost 'T' seconds enter in CPU again.

Gate exam



$$[tq * (n-1) + n * s] \leq T$$

$$\Rightarrow tq \leq \left(\frac{T - ns}{(n-1)}\right)$$

tq time quanta from P_2 to P_1.

no. of Context Switches from P_2 to P_1

Priority Scheduling:-

(1) Each process has its own priority.

(2) Out of all available processes, highest priority processes get the CPU.

(3) If tie then use FCFS.

(4) Priority → Static (doesn't chng throughout the execution of process)
→ Dynamic (changes after some interval of time)

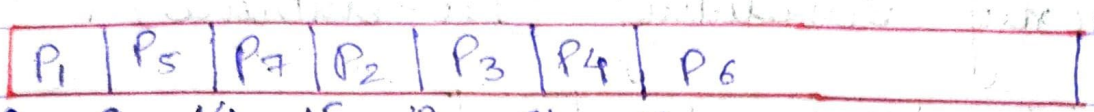
(5) Version → Non-preemptive
→ preemptive.

Example:- Priority Scheduling (Non-preemptive) :-

→ lesser the number higher the priority.

Process	Priority	AT	BT	CT	TAT	WT	RT
P_1	3	0	8	8	8	0	0
P_2	4	1	2	17	16	14	14
P_3	4	3	4	21	18	14	14
P_4	5	4	1	22	18	17	17
P_5	2	5	6	14	9	3	3
P_6	6	6	5	27	21	16	16
P_7	1	10	1	15	5	4	4

25 Sec



27 = BT (total)

Although the priority of P_7 is 1 therefore it must be assigned to the CPU first, but then at $t=0$, P_7 has not yet arrived. there we have to check the arrival time of the processes also. According to that we have to assign the CPU.

Now, since at $t=8$ we have P_2, P_3, P_4, P_5, P_6 in the ready queue therefore we now have to check the priorities of these processes and accordingly we have them CPU.

Once all the processes have arrived in the ready queue then there is no need to check the AT of each process for CPU allocation rather than that we can simply assign them CPU according to their priority.

↦ If there is a tie in the priority then use FCFS to break the tie.

Example:- Priority scheduling (Pre-emptive)

Process	Priority	AT	BT	CT	TAT	WT	RT
P_1	3	0	8	15	15	7	0
P_2	4	1	2	17	16	14	14
P_3	4	3	4	21	18	14	14
P_4	5	4	1	22	18	17	17
P_5	2	5	6	12	7	1	0
P_6	6	6	5	27	21	6	16
P_7	1	10	1	11	1	0	0
					13.7	9.8	8.7

lesser the number higher the priority.

P_1	P_1	P_1	P_1	P_5	P_5	P_7	P_5	P_1	P_2	P_3	P_4	P_6

0 1 3 4 5 6 10 11 12 15 17 21 22 27

Main drawback of this algorithm:-

Starvation problem:- If a process has to wait for indefinite amount of time for its CPU allocation. then this is known as Starvation problem.

Solution to the starvation problem is "aging".

Aging:- The processes which are in ready state, their priority after a regular interval of time is increased in by some amount.

Starvation & Aging :-

Starvation :- Indefinite blocking.

→ a process which is ready to run can wait indefinitely because of low priority.

→ High priority processes prevent a low priority process from ever getting the CPU.

Aging :- method to ensure that processes with lower priority will eventually complete their execution.

→ by gradually increasing the priority of processes that wait in the system for a long time.

Suppose :- After every 3 unit of time of CPU, priority of waiting processes will be decreased by 1.

→ lesser the number higher the priority.

	B.T	Priority
P_1	10	20
P_2	5	1
P_3	2	5
P_4	40	2

20 $\xrightarrow{3}$ 19 $\xrightarrow{3}$ 18 $\xrightarrow{3}$ 17 ------ (1) ... 0.

older priority

till this point this process may get a chance to be allocated to the CPU

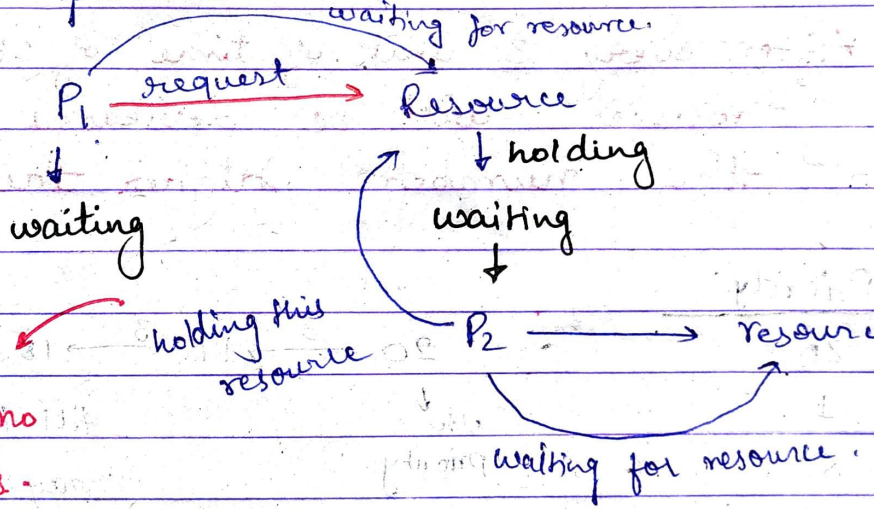
Deadlock in OS :-

↦ Waiting for something for infinite time, in which there is no progress for waiting processes.

↦ processes waits for one another's action indefinitely.

→ CPU is blocked (No process is running in the CPU). → Deadlock.

→ If a waiting process never changes its state because it is waiting for the resources which are held by some other waiting processes.



Hence there is no progress.

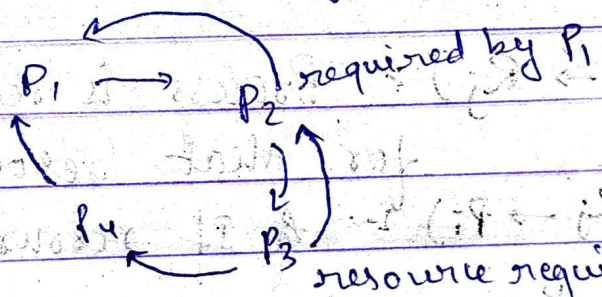
Coffman conditions :-

1) Mutual Exclusion :- atleast one resource must be held in non-sharable mode. → only one process can use the resource.

2) Hold and wait :- process holding one resource and waiting to acquire resources that are currently held by other processes.

3) No-preemption :- Resources cannot be pre-empted.

4) Circular wait :- Resources held by the process in circular form.



⇒ Every process in this circular wait is waiting for its next process to the ~~hold~~ leave the resources held by them.

Resource Allocation Graph:-

Set of vertices :-

V → Set of processes $\{P_1, P_2, P_3 ---- P_n\}$
→ Set of resources $\{R_1, R_2, R_3 ---- R_n\}$

Set of edges :-

E → $P_i \longrightarrow R_j$ (Request edge)
→ $R_j \longrightarrow P_i$ (Assignment edge)

Request edge $(P_i \rightarrow R_j)$:- Process is requesting or waiting for that resource.

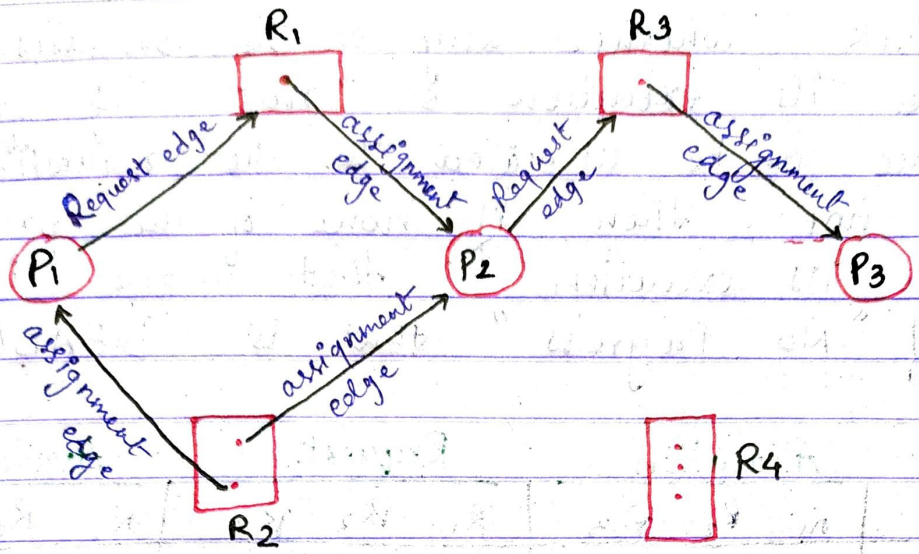
Assignment edge $(R_j \rightarrow P_i)$:- If resource is assigned to the process.

□ ⇒ Resource type

⋮□ ⇒ instance of a resource type
↳ no. of available resource type.

○ ⇒ Process

If there is no cycle in resource allocation graph then there will be no deadlock.
→ If cycle exists then deadlock may exist.



$P_i \rightarrow R_j$ = Request edge

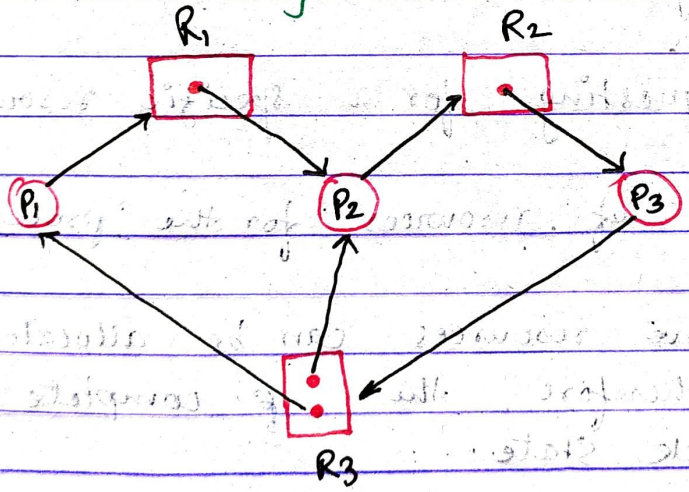
$R_j \rightarrow P_i$ = Assignment edge

$P = \{ P_1, P_2, P_3 \}$

$R = \{ R_1, R_2, R_3, R_4 \}$

$E = \{ P_1 \rightarrow R_1, P_2 \rightarrow R_3, R_1 \rightarrow P_2, R_2 \rightarrow P_2, R_3 \rightarrow P_1, R_3 \rightarrow P_3 \}$

Graph with cycle and ~~Deatock~~ Deadlock.



There are two possible methods to check whether it is a ~~deador~~ deadlock or not.

Method - 1:- Check, whether any of the processes can complete its execution or not if yes then that resource can be allocated to the requesting processes and if not then if none of the process can complete its execution that means it is in a State of "No Progress" that is DEADLOCK.

Method - 2 :-

	Allocation			Request			Availability.		
	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3
P_1	0	0	1	1	0	0	0	0	0
P_2	1	0	1	0	1	0			
P_3	0	1	0	0	0	1			

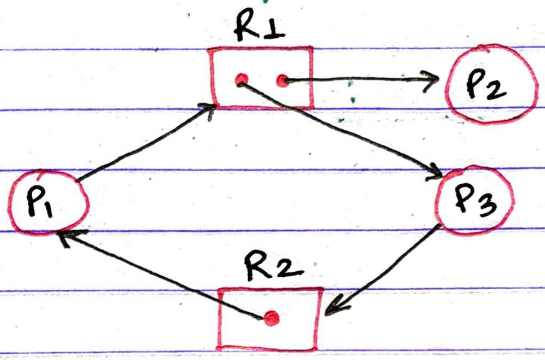
Allocation:- Allocation of ~~Resource~~ Process to the resource.

Request:- Processes requesting for a specific resource.

Availability:- Availability of resources for the processes.

Since none of the resources can be allocated to the required processes therefore, the complete system is in a deadlock state.

→ Graph with cycle and not Deadlock:-



∵ P_2 is not waiting for any process therefore it can complete its execution.

After sometime P_2 will complete its execution and then one instance of R_1 is then free and is allocated to the process P_1 and after sometime when P_1 completes its execution the Resources are again free and therefore other processes can also complete their ~~excad~~ execution and thus there is no condition for deadlock here.

	Allocation		Request		Availability.	
	R_1	R_2	R_1	R_2	R_1	R_2
P_1	0	1 (1.1)	1	0	0	0
P_2	1	0	0	0	+1	0 ← after P_2 is completed
P_3	1	0	0	1	1	0 ← allocated to P_1.
					0	0
					1	1

~~With~~ with the availability column we need to compare the request column matrix and then check whether we can fulfil any process's execution with the availability of the resources.

∴ Yes, we can fulfil P_2 with current availability of resources.

Complete explanation :-

Deadlock handling in OS :-

1) Deadlock prevention.
2) Deadlock Avoidance.
3) Deadlock Detection & Recovery.
4) Deadlock Ignorance (ostrich Method).

→ Deadlock prevention :- Violet any of the four necessary conditions at any time and deadlock can never occur in the system.

→ Removal of Mutual exclusion :- Make all the resources sharable and i.e., impossible to implement, therefore not feasible to implement.

→ Removal of Hold and wait :- 3 ways are there.

1) A process must acquire all the necessary resources before execution starts.
   → Impossible - Not implementable.

2) Process holding some resources and requesting for additional resource, then it must release the acquired resources first. (Starvation problem)

But in some cases the process may need the previous resources as well as additional resource to perform and complete execution.

3) Wait time out/bound.
↳ wait for the add^{n} resource for a time bound and if that time bound expires then the process has to release its acquired resources.

→ Removal of No-pre-emption:- Taking resources forcefully from the waiting processes not the executing or running processes. (only applied to higher priority processes)

→ Resources can be pre-empted from processes.

→ Processes holding some resources and requesting for another resource that can be immediately allocated then all the required resources will be pre-empted.

→ Process request a resource.

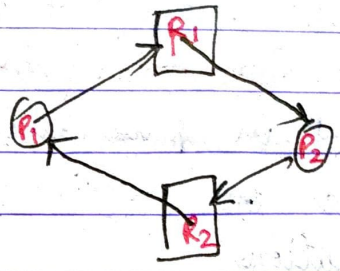
Available (Allocated)

Not-Available (Allocated to some other waiting processes)

→ Removal of Circular wait. Implementable

P_1	P_2
5	

Printer - 1
CPU - 5
Memory - 6
CD-Drive - 7



To request a resource R_j, a process must release all the acquired resources R_i such that $i >= j$.

only the ordering of resources is a bit difficult

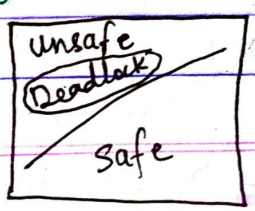
→ Deadlock Avoidance:-

→ System maintains some database using which it can take decision whether to entertain a request or not, just to be in safe state.

→ System (kernel) analyse the database (allocation state) to determine whether granting a request can lead to deadlock in future

→ if not lead to deadlock then granted.
→ Otherwise keep pending until they can be granted.
→ processes may face a long delay for obtaining a resource.

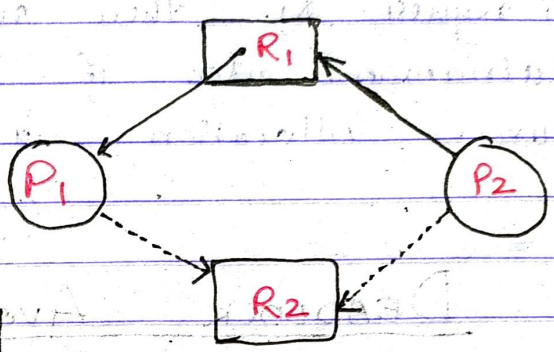
When the system is in unsafe state then deadlock may occur.



Deadlock Avoidance With Resource Allocation Graph.

V → Set of vertices.

E → Set of edges



Request edge ($P_i \rightarrow R_j$)

Assignment edge ($R_j \rightarrow P_i$)

claim edge ($P_i \dashrightarrow R_j$)

P_i may request R_j in future

a process may request a resource in future.

(1) $P_1 \dashrightarrow R_1$ (2) $P_i \rightarrow R_1$ (3) $P_1 \leftarrow R_1$ (4) $P_i \dashrightarrow R_1$

(1) At first a process may request a ~~request~~ resource and if it requests then (claim edge)

(2) The resource and process are converted into request edge from claim edge

(3) After that the resource may be allocated to the process.

(4) Again the ~~proc~~ edges are converted into claim edge.

Condition:- Resources must be claimed in advance.

if Pi request Rj then request edge can only be converted to assignment edge if it does not form a cycle in Resource Allocation Graph.

DEADLOCK AVOIDANCE WITH BANKER'S ALGORITHM :-

→ Handles multiple instances of same resources.

→ Prefer to use this when multiple instances of same resource is there.

These things must be known before applying banker's algorithm.

(1) How many instances of each resource each process can max request. [MAX] → 2D Array.

(2) How many instances of each resource each process currently holds. [ALLOCATION]

(3) How many instances of each resource is available in the system. [AVAILABLE] → 1D array

Example :-

P	Allocation				Max				Available			
	A	B	C	D	A	B	C	D	A	B	C	D
P_0	0	0	1	2	0	0	1	2	1	5	2	0
P_1	1	0	0	0	1	7	5	0				
P_2	1	3	5	4	2	3	5	6				
P_3	0	6	3	2	0	6	5	2				
P_4	0	0	1	4	0	6	5	6				
	2	9	10	12								

Find :-
1) Need Matrix.
2) Is system in safe mode/state if yes then find the safe sequence.

Allocation (ABCD) :- Provides information about the required instances of a process.

Max (ABCD) :- Maximum number of instances a process can have.

Available (ABCD) :- Instances of resources currently available after the process is allocated.

Total resources available in the system :- ABCD.

=> Allocation + Available =

A	B	C	D
2	9	10	12
1	5	2	0
3	14	12	12

Total resources available :-

A	B	C	D
3	14	12	12

Need Matrix :- Maximum resource available for allocation.

[Maximum - Allocation]

P	A	B	C	D
P_0	0	0	0	0
P_1	0	7	5	0
P_2	1	0	0	2
P_3	0	0	2	0
P_4	0	6	4	2

→ Need Matrix.

Now, we'll check whether the system is in safe state. Therefore, we apply Bankers ~~ap~~ algorithm.

→ Therefore, we need to find out whether the available number of resources can satisfy the need of resources.

⇒ Need of P_0 process is - 0 0 0 0 ✓
1 5 2 0 ✓

∴ We can satisfy this need for the process P_0

⇒ P_1 - 0750 We cannot satisfy
1520
✓ × × ✓

P_2 - 1002 We cannot satisfy
1520
✓ × ✓ ×

P_3 - 0020 We can satisfy.
1520
✓ ✓ ✓ ✓

P_4 - 0642 We cannot satisfy
1520
✓ × × ×

Safe Sequence :- | P_0 | P_2 | P_3 | P_4 | P_1 |

	Allocation				Available				Need			
P	A	B	C	D	A	B	C	D	A	B	C	D
P_0	0	0	1	2	1	5	2	0	0	0	0	0
P_1	1	0	0	0	1	5	3	2	0	7	5	0
P_2	1	3	5	4	2	8	8	6	1	0	0	2
P_3	0	6	3	2	2	14	11	8	0	0	2	0
P_4	0	0	1	4	2	14	12	12	0	6	4	2
					3	14	12	12				

n = no. of processes.
m = no. of resources.

Algorithm for Bankers Algorithm :-

Input :- Processes.
→ Any 2 out of 3 (Max, Need, Allocation)
→ Available or total no. of resources.

Step :- 1 flag[i] = 0 for i = 0 to (n-1) and
find Need[n][m] = Max[n][m] - allocation[n][m].

Step :- 2 Find a process P_i such that :- flag[i] = 0 and
Needi <= Available.

Step :- 3 If i exists then
flag[i] = 1, available = available + allocation.
goto step - 2 ; Otherwise goto step 4

Step :- 4 If flag[i] = 0 for all i then system is in safe state otherwise unsafe state.