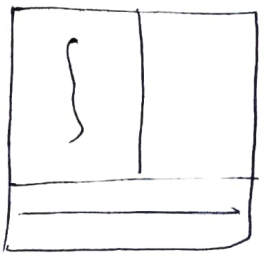
Problems with User level threads

① Blocking system calls. (OS block the threads if resources are not available).

② Block all the threads in a task if one threads get blocked). Mistake done by one thread suffers all the thread.

③ Unfair scheduling.
If two task have n and m process (threads) then despite of what values m and n have both task get same time quantum.

Solution of above problem is Kernel level thread :-



→ Each thread generate a system call to create new thread.
→ If a thread get blocked it will not blocked whole task.
→ If kernel knows of all the threads then it will schedule task according to no. of threads in task.

Problems : ① Expensive compare to user level thread bcoz of system calls, but less expensive compare to create new process.

② Switching require system call, in Kernel level thread: It also an overhead.

→ But overall threads are better than process creations.
→ Combination of both is hybrid thread which is very useful.

Types of OS

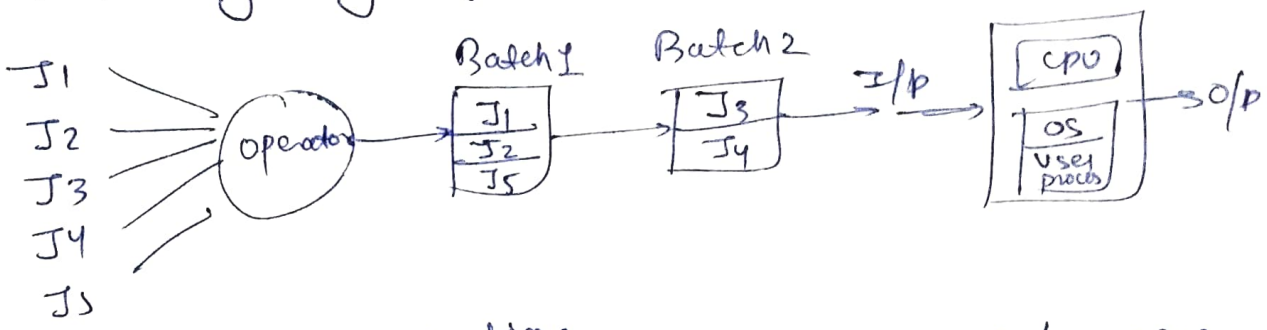
Batch OS : Input & Output devices were punch cards readers and tape readers.

→ User prepares a Job on punch cards that contains program, I/p data and control instructions.

→ Memory was very limited and Context switching was very tough. and it is very difficult to match speed (very poor CPU utilization).

So we needed an updated version of OS and we go for Batch OS.

→ In Batch OS we combined similar jobs together and input in system was coming in batches instead of single jobs

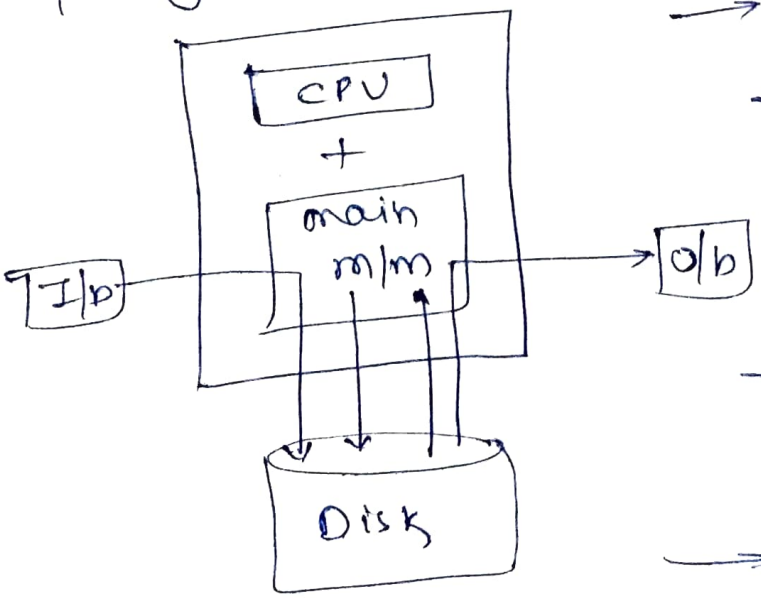


→ It reduces some time.

→ Disadv → Memory limitation, I/p and o/p were directly interacted with CPU, Job was also in form of punch cards Not Interactive System

→ we were saving the loading/unloading time of compiler for different jobs.

Spooling [Simultaneous Peripheral operation Online]



→ first time introduce sec. memory.

→ first load jobs in disk via m/m

→ Peripherals are very slow than CPU.

→ No Interaction of peripherals direct with CPU.

→ Increases CPU utilization

Disadv → It was uniprogramming not multiprogramming.

Multiprogramming: Maximize CPU utilization.

→ More than one process in main m/m at a time.
→ If one process doing I/O other can do CPU.
→ Context switching started.
→ Less waiting time of CPU, never idle.
→ **Disadv:** Scheduling Algo. required, main m/m management required, m/m fragmentation,
→ One processor can run only one process at a time.
→ Nowadays Octa Core → means 8 processor, in which we can run 8 process at a time. This method called multitasking / multiprocessing.
→ Less waiting & response time.

Multi-tasking / Time-sharing / fair Sharing / Multiprogramming with Round Robin

→ Multi-tasking means multiprogramming with preemption / time-sharing.
→ ~~We can~~ CPU runs only a single process at a time but due to very fast context switching it looks like lots of process running concurrently.

Multiprocessing: More than one processor to increase system performance.

→ Truly we achieve parallel execution of process

Operating System (Tenenbaumm) [Easy] (for Theory)

Process Management (Easy)

Introduction to OS

→ OS is interface between user and hardware.

OS provide the system calls for the instruction given by the user.

→ Resource Allocator :- Allocation of resource to process it might be shared or non-shared resources.

→ Manager → m/m, process, files, security management, etc.

Goals of OS

① Primary : Convinience

② Secondary : Efficiency [Generally considered in mainframe & Super computer]

Types-OS (Read and make Notes from GfG)

① Batch OS :- In very early computers were like,

→ All the process were given to computer collected in queue and Computer was doing it by taking time

→ Starvation & less interactive.

Process have two types of times :-

→ CPU Time

→ I/O Time

② Multiprogramming — Imp. for Gate

③ Multitasking

④ Multiprocessing

⑤ Real Time OS.

→ In Multiprogramming if any job doesn't need I/O picked up quickly in processing queue from waiting queue.

Theory points from textbook or GfG

→ Multiprogramming with preemption is called multitasking.

~~Process, PCB and Attributes~~ Cont..

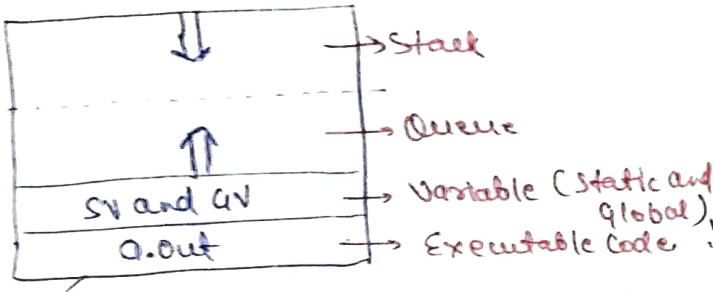
Process, PCB and Attributes

abc.c → (High level lang.)

Compiler ⇒ a.out (on ROM)

→ OS takes program from ROM and put it inside the RAM.

→ Process can be assumed as program under execution.



→ executable code tries to access the space alloted to other code it will give 'segmentation fault'

→ This alloted space called as process (activation record).

Attributes →

① **Process Id**: Unique number assigned to Process but it's not universal as port no.

② **Program Counter**: Keep the record that which instruction is going to executing next if you stop the process in between.

③ **Process State**: State of process Running, waiting etc.

④ **Priority**: Priority for execution of process. It is basically a number.

⑤ **General purpose registers**: Used by CPU,

R1 [1] R2 [2] R3 [3]

P_1: $R_1 = R_2 + R_3$ → Instruction

→ These registers are used for normal execution of instructions.

⑥ **List of open files**: files open by process to perform jobs

⑦ List of open devices

⑧ **Protection**: as stop the accessing data of another process.

PCB → (Process control block) alloted to each process contains all the attributes

All process combines using linked list.

Process states & Multiprogramming

PCB also called as 'context of process.'

States of a process

① **New**: Whenever process is created or program in ROM that is ready to get picked up by OS to kept in RAM.

→ Process is new if it is just created or about to create.

② **Ready**: Process in main m/m that is ready for execution.

Multiprogramming

- with preemption (Multitasking) → (Remove the resource from process and allot them to another process) → (You can stop the process in between and can start it later)
- without preemption [Can't force process to leave]

⑤ Run: Process under execution.

④ Block or wait: Process in main memory, if that process needs I/O to perform then it will kept aside and once if complete I/O. It will kept in ready state

⑧ Termination or Completion
→ Delete the process and faces / context of process.

③ Suspend ready: If space in main memory is not sufficient then less imp. process pulled out and send them to sec. memory.

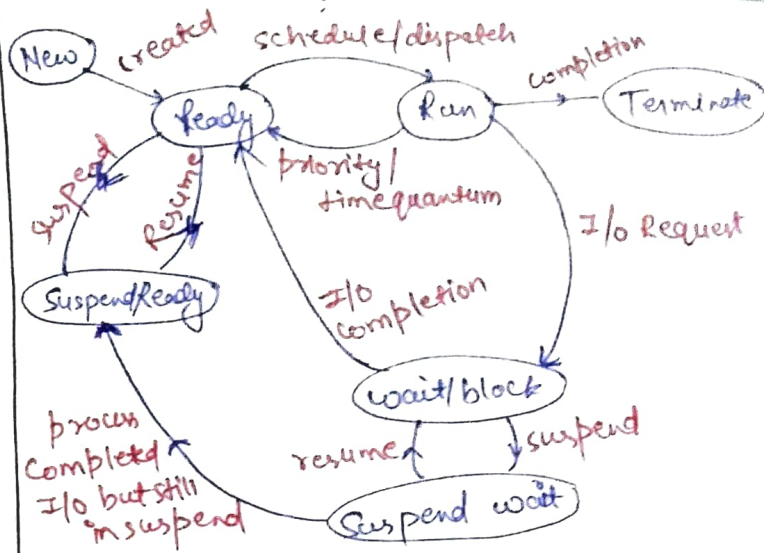
⑦ Suspend wait or Suspend block: It is similar to suspend ready but we suspend those which either get blocked or in wait state.

Operation on process

① Creation
② Scheduling
③ Execution
④ Killing or Delete

Process state Transition Diagram and various Schedulers.

→ we have to start from (new) and end at (terminate) and in between we can perform various operations



Ready → Run: Allotment of process to CPU

It means min. '4 states' require to complete the process without any disrupt.

CPU bound or I/O bound process. It can be said as if process is taking more time for CPU operations means CPU bound otherwise I/O bound.

Process at Run state should run only at CPU without asking for I/o so from 'wait/block' kept process back to Ready if its I/O get completed.

Run → Ready called as premption.

Suspend states are not generally used but it is necessary to be ready for worst case.

You can suspend or resume any time as required.

During suspension process stay in secondary memory.

① Long Term scheduler takes decision about How many process to make.

Short Term scheduler picks up the process from ready state where pool of process are available. Also called as 'dispatcher' (Ready → Run)

Medium Term Scheduler deide about suspension / resume decisions.

→ **Degree of multiprogramming** means max. no. of process to kept in ready state.

Main decision should be taken by long term scheduler that either the process is I/O bound or CPU bound. It manages mixture of both kind of process to maintain performance.

Context switching: Save the state of previous process and update the state of new process. It is done by short term scheduler to make min. context switching.

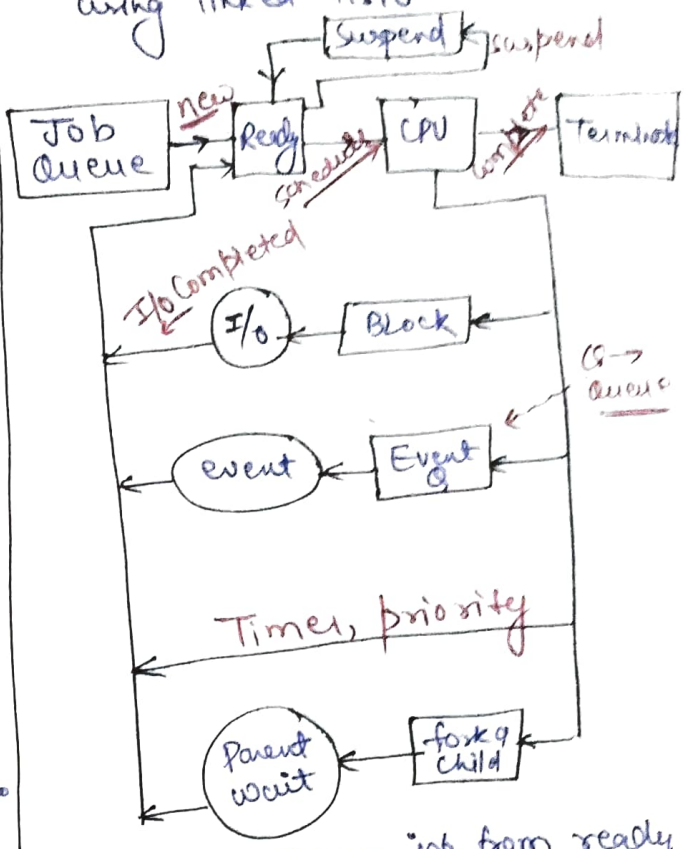
→ Medium terms is not much important. It maintains swapping (Loading and removing process from RAM to ROM or vice versa). Mainly in suspend / resume process.

→ All 3 schedulers have some impact on performance of the system.

Process Queues

Scheduling Queues

→ These queues are implemented using linked lists



→ We can choose job from ready queue on the basis of certain criterias.

→ Event Q have various event in queue, it can be discuss later.

→ Timer means (time quantum) in case of round robbin.

Consider a system with 'N' CPU processors and 'M' processes then,

	min	max
ready	0	M
running	0	N
block	0	M

→ All might needed I/O.

Various times related to process...

① **Arrival Time** → Time at which process enteres in ready queue.

② **Burst Time** → Amount of CPU time required to get finished.

③ **Completion Time**: Time after which process get terminate.

→ In between Ready and Termination state.

(Completion Time - Arrival Time) is equals to (Burst Time + wait Time)

④ **Turn around time**: (TAT)

(Completion time - Arrival Time)

⑤ **Waiting time**: (WT) = (TAT - BT)

Turn around time — Burst Time

⑥ **Response Time**: First time at which process get scheduled.

CPU scheduling:

→ It is done by short term scheduler (STS)

→ STS pickup a process from ready state and kept it in running state.

→ CPU scheduling requires when a process moves from:

a) 1) Run → Termination
2) Run → wait
3) Run → Ready

b) New → Ready i.e when a process is just created.

c) wait → Ready

FCFS (Fist come first serve)

→ We are considering 'only one CPU.'

→ Selection criteria is 'arrival time.'

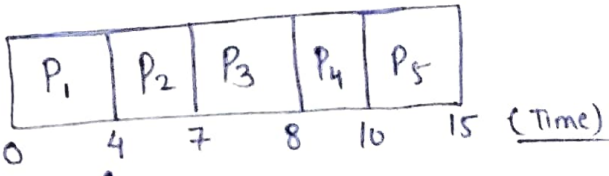
→ Mode is non-preemption means if a process selected we can't preempt it.

→ We are assuming that there is 'No' I/o Time.

Ex.

Process No.	Arrival Time	Burst Time
1	0	4
2	1	3
3	2	1
4	3	2
5	4	5

Ganttchart



All process comes upto this time in pool.

→ If two process having same arrival time then choose process with low of i in Pi.

Ex $P_2 < P_3$

Completion Time	TAT	WT	RT
4	4	0	0
7	6	3	3
8	6	5	5
10	7	5	5
15	11	6	6

↓ (Completion Time - AT)

In case of non-preemptive Algo. 'WT and response time' is same.

Avg. TAT = $\frac{4+6+6+7+11}{5} = \underline{6.8}$

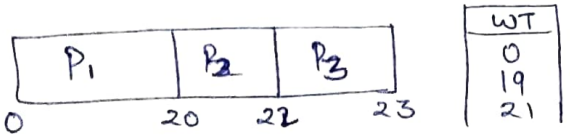
Avg. WT = $\frac{0+3+5+5+6}{5} = \underline{3.8}$

→ Property of FCFS is convoy effect.

Convoy Effect:- Disadvantage of FCFS.

Ex-

Process No.	AT	BT	CT	TAT
1	0	20	20	20
2	1	2	22	21
3	1	1	23	22

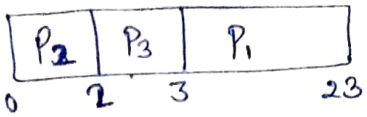


WT
0
19
21

Convoy effect independent of AT.

Avg WT = $\frac{40}{3} = \underline{13.33}$

P.No	AT	BT	CT	TAT	WT
1	1	20	23	22	2
2	0	2	2	2	0
3	0	1	3	3	2

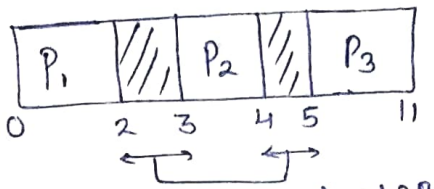


Avg. WT = $\frac{4}{3} = \underline{1.33}$

→ If process with large burst time comes first then all other process have to wait for long and they might get into starvation called as convoy effect.

Ex → Process with Gap.

Pno.	AT	BT	CT	TAT	WT
1	0	2	2	2	0
2	3	1	4	1	0
3	5	6	11	6	0



→ Idle state means CPU is not doing anything.

→ Waiting time for all the processes are zero.

→ TAT = BT bcoz as the process arrive, it suddenly go for scheduling.

→ Time Complexity of Algo. : O(n)

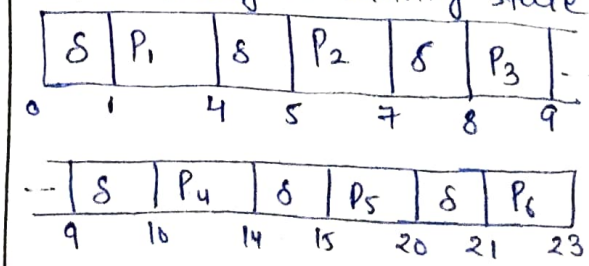
Data structure used → Queue

FCFS with overhead :

P.NO	AT	BT
1	0	3
2	1	2
3	2	1
4	3	4
5	4	5
6	5	2

δ = 1 unit

δ → Dispatching time (Time taken by dispatcher to take a process from ready to running state.



Efficiency (η) = $\left(1-\frac{6}{23}\right)\times 100$

6 → no. of times δ comes multiply with dispatch time (6×1)

23 → Total time taken

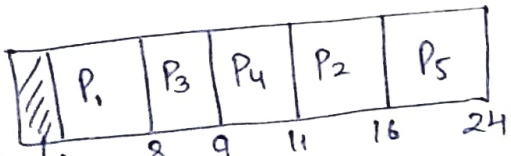
Introduction to SJF

SJF : Shortest Job First

→ we choose the process on the basis of burst time.

→ It works in mode of non-preemption.

P.No	AT	BT	CT	TAT	WT
1	1	7	8	7	0
2	2	5	16	14	9
3	3	1	9	6	5
4	4	2	11	7	5
5	5	8	24	19	11



Idle state

→ If process enters in queue then complete it first bcoz we follow non-preemptive version.

→ Upto 8 units all the jobs were inside Ready state so we choose shortest job already inside the ready queue.

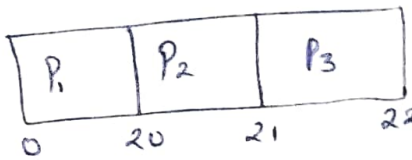
→ It can also be a victim of convoy effect. (Not given in any book)

→ Suitable Data Structure → Min heap.

Time Complexity : O(n log n)

Analysis of SJF

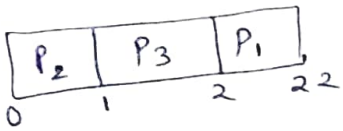
P.No	AT	BT	CT	TAT	WT
1	0	20	20	20	0
2	1	1	21	20	19
3	2	1	23	21	20



Avg WT = $\frac{39}{3}$ = 13

→ If two process having same BT we can choose the one with low AT first.

P.No	AT	BT	CT	TAT	WT
1	2	20	22	20	0
2	0	1	1	1	0
3	1	1	2	1	0



Avg WT = 0

→ SJF is not practical to implement bcoz we take decision according to BT but the problem is we can't measure the BT before it enters in running queue.

Throughput → No. of process finish per unit time.

for above ex,
Throughput = 3/22

SJF with prediction of BT

SJF is best known Algo. till now but we can't implement.

Adv. : ① Maximum throughput
② Min. Avg WT and TAT

Disadv. : ① Starvation to longer jobs.
② Can't implemented.

Solution: SJF with predicted BT :-

Prediction Technique
- Static
  1. Process Size
  2. Process Type
- Dynamic
  1. Simple averaging
  2. Exponential or Aging

① **Process Size**: Predict BT according to process size

Process size in Bytes

$P_{old} = 200KB \Rightarrow 20$ units

$P_{new} = 201KB \Rightarrow 20$ units

Match size of process with already completed process and allot the BT according to it.

② **Process Type**:
- Process
  - OS (3-5 units)
  - User
    - Interactive (5-10 Unit)
    - Foreground (10-15 Unit)
    - Background (15-20 Unit)

→ If prediction is right then definitely we get best result using SJF but prediction is such a tedious task.

③ **Simple Average**

→ Given n processes ($P_1 \dots P_n$)

→ Let t_i be the actual BT

→ Let τ_i denotes predicted BT

$$\Rightarrow \tau_{n+1} = \frac{1}{n} \sum_{i=1}^{n} t_i$$

BT of process equals to avg. BT of previous processes.

④ **Exponential Average / Aging**

$$\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n \quad -①$$

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

$\alpha \rightarrow$ Smoothing factor.

$$\tau_n = \alpha t_{n-1} + (1-\alpha)\tau_{n-1} \quad -②$$

put value of τ_n in eq^n ①

$$\tau_{n+1} = \alpha t_n + (1-\alpha)t_{n-1} + (1-\alpha)^2 \tau_{n-1}$$

So, it can be expand further and we can say that BT of process depends on all the previously occurs processes.

Ex $\alpha = 0.5$ $\tau_1 = 10$ (Initial Guess)

actual BT $(t_1, t_2, t_3, t_4) = (4, 8, 6, 7)$ then τ_5 ?

$\tau_2 = 0.5 \times 4 + 0.5 \times 10$
$= 2 + 5 = 7$

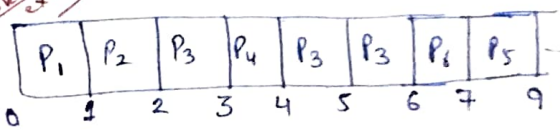
$\tau_3 = 0.5 \times 8 + 0.5 \times 7$
$= 4 + 3.5 = 7.5$

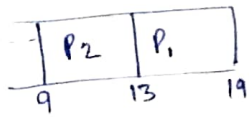
$\tau_4 = 0.5 \times 6 + 0.5 \times 7.5$
$= 3 + 3.75 = 6.75$

$\tau_5 = 0.5 \times 7 + 0.5 \times 6.75$
$= 6.875$ Ans//

SRTF: It is based on criteria of BT and mode is preemptive

Must look for example





PNO	AT	BT	CT	WT
1	0	~~6~~	19	12
2	1	~~8~~ 4	13	7
3	2	~~3~~ 2	6	1
4	3	~~1~~ 0	4	0
5	4	~~2~~ 0	9	3
6	5	~~1~~ 0	7	1

SRTF → Shortest remaining Time First

→ It have no convoy effect or starvation.

→ It can be said as SJF with preemption.

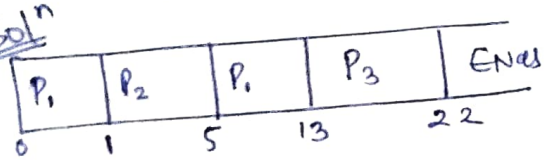
Response time of P_6 is 1 unit and all other Processes have response time as 0.

→ It is not implemented practically.

	PNo	AT	BT	CT	TAT	WT
Q	1	0	9	13	13	4
	2	1	4	5	4	0
	3	2	9	22	20	11

find avg. WT?

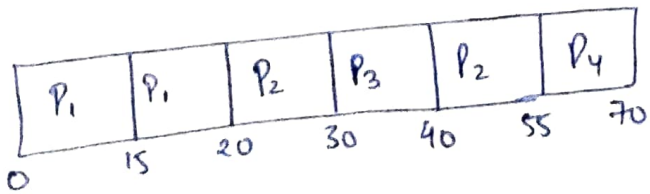
Soln



Avg WT = $\frac{15}{3}$ = 5 units

Q find WT of P_2 using SRTF.

PNo	AT	BT	CT	TAT	WT
1	0	20	20	20	0
2	15	25	55	40	15
3	30	10	40	10	0
4	45	15	70	25	10



Avg - 15 units

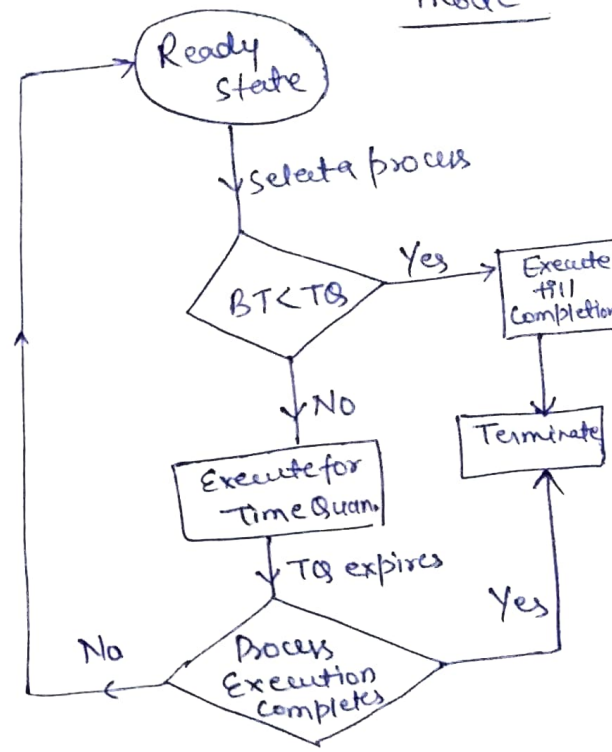
Round Robin Algo. (Preemptive)

→ Practically implemented bcoz not depend on BT.

→ Only a single Data structure require (Queue).

→ Concept of Time Quantum (TQ).

→ There is no chance of starvation or very less. → Premptive mode

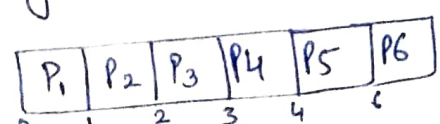


Ex →

P.NO	AT	BT	CT	TAT	WT
1	0	4	8	8	4
2	1	5	18	17	12
3	2	2	6	4	2
4	3	1	9	6	5
5	4	6	21	17	11
6	6	3	19	13	10

Time quantum = 2

Round Robin Based on the criteria of Arrival time and Time quantum



→ Order of Arrival.

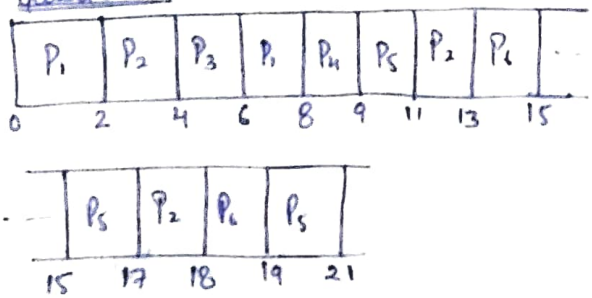
Conti--

Prepared two separate queues one for maintaining the sequence of process that is going to be executed next.

and second queue as Gantt chart

$P_1\ P_2\ P_3\ P_1\ P_4\ P_5\ P_2\ P_6\ P_5\ P_2\ P_6\ P_5$

Gantt chart



If one process complete its TQ, dequeue it and insert it at last to queue

→ Very or change the sequence after each time quantum.

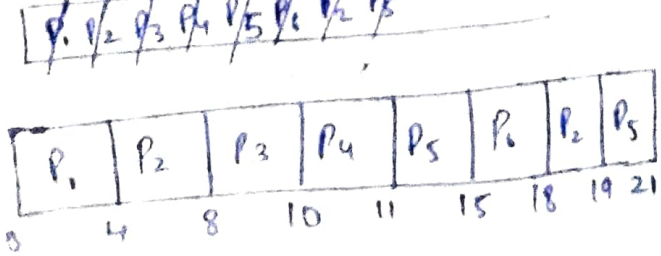
→ Take one process out and schedule next process called context switching.

→ More the TQ, lesser the context switching.

Solving same previous Ques. with TQ = 4

P.no	AT	BT	CT	TAT	WT
1	0	4	4	4	0
2	1	5	19	18	13
3	2	2	10	8	6
4	3	1	11	8	7
5	4	6	21	17	11
6	6	3	18	12	9

$P_1\ P_2\ P_3\ P_4\ P_5\ P_6\ P_2\ P_5$

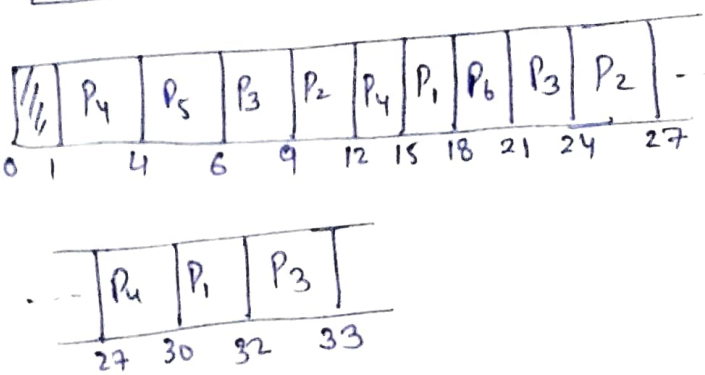


Ex3 TQ = 3

P.no	AT	BT	CT	TAT	WT	RT
1	5	5	32	27	22	10
2	4	6	27	23	17	5
3	3	7	35	30	23	3
4	1	9	30	29	20	0
5	2	2	6	4	2	2
6	6	3	21	15	12	12

Soln

$P_4\ P_5\ P_3\ P_2\ P_4\ P_1\ P_6\ P_3\ P_2\ P_4\ P_1\ P_3$



RT → Response Time in Table

= (Time at which arrive first) − (Arrival Time)

→ If, TQ ↑, context switching ↓, RT ↑
TQ ↓, ——— ↑, RT ↓

→ If TQ → ∞ then, RR slowly becomes FCFS.

→ Preparing Gantt chart is necessary for all the problems.

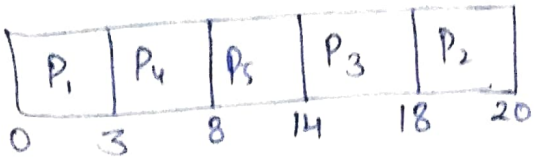
Longest Job First Algo.

Process having longest BT gets scheduled first.

mode → Non preemptive

Ex -

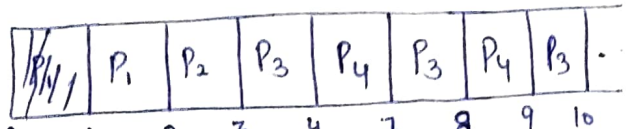
Pno	AT	BT	CT	TAT	WT	RT
1	0	3	3	3	0	0
2	1	2	20	19	17	19
3	2	4	18	16	12	12
4	3	5	8	5	0	0
5	4	6	14	10	4	4

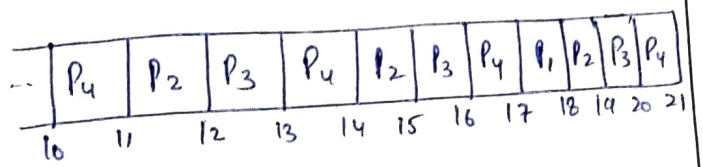


→ This Algo. is similar to SJF but we choose largest BT first.

Algo: Longest remaining time first

PNO	AT	BT	CT	TAT	WT	RT
1	1	2	18	17	15	0
2	2	4	19	17	13	0
3	3	6	20	17	11	0
4	4	8	21	17	9	0





→ LRTF can be assumed as similar to SRTF but choose longest Burst time first.

→ If two process are having same BT then go for low AT.

Gate: 2006 Question on LRTF todo.

HRRN: (Highest response ratio next)

Response ratio (RR) = $\frac{w+s}{s}$

w: waiting time for a process so far.

s: service time of a process or BT

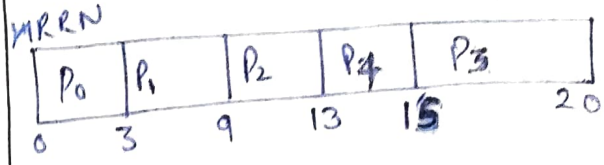
→ HRRN not only favours the shorter jobs but also limits the waiting time of longer jobs.

→ Mode: Non preemptive.

Ex

P NO	AT	BT	CT	TAT	WT	RT
0	0	3	3	3	0	0
1	2	6	9	7	1	1
2	4	4	13	9	5	5
3	6	5	20	14	9	9
4	8	2	15	7	5	5

HRRN



$RR(P_2) = \frac{(9-4)+4}{4} = \frac{9}{4} = 2.25$

$RR(P_3) = \frac{(9-6)+5}{5} = \frac{8}{5} = 1.6$

$RR(P_4) = \frac{(9-8)+2}{2} = \frac{3}{2} = 1.5$

at t = 9 units all 3 process (2, 3, 4) get enters so we calculate RR.

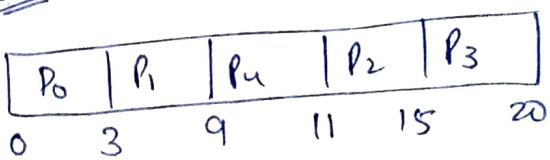
After completing P_2 at 13 again calculate RR.

$RR(P_3) = \frac{7+5}{5} = 2.4$

$RR(P_4) = \frac{5+2}{2} = \frac{7}{2} = 3.5$

So give preference to P_4 over P_3

SJF



✓ Order in both HRRN and SJF is changed.

✓ HRRN is better as compare to SJF.

Priority Scheduling

→ (1) Priority

- Static → Doesn't change throughout the execution of process (Gate)
- Dynamic → Change at regular interval of time.

(2) Priority

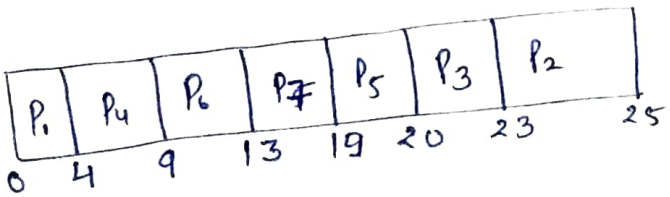
- Preemptive
- Non-preemptive

→ If two process are having same priority then we go for AT and if AT is also same then Pid.

Non-preemptive priority scheduling

Ex

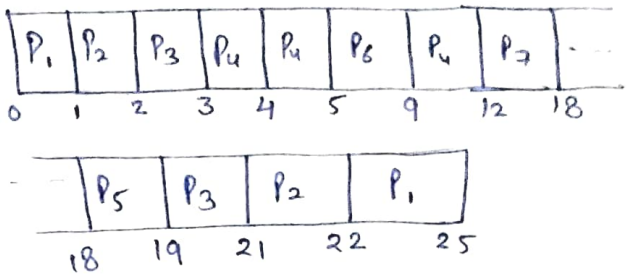
PNO	Priority	AT	BT	CT	TAT	WT	RT
1	2(L)	0	4	4	4	0	0
2	4	1	2	25	24	22	22
3	6	2	3	23	21	18	18
4	10	3	5	9	6	1	1
5	8	4	1	20	16	15	15
6	12(H)	5	4	13	8	4	4
7	9	6	6	19	13	7	7



→ Response Time = WT bcoz we follows non-premptive mode.

Premptive Priority Scheduling

PNO	Priority	AT	BT	CT	TAT	WT	RT
1	2	0	4	25	25	21	0
2	4	1	2	22	21	19	0
3	6	2	3	21	19	16	0
4	10	3	5	12	9	4	0
5	8	4	1	19	15	14	14
6	12	5	4	9	4	0	0
7	9	6	6	18	12	6	6

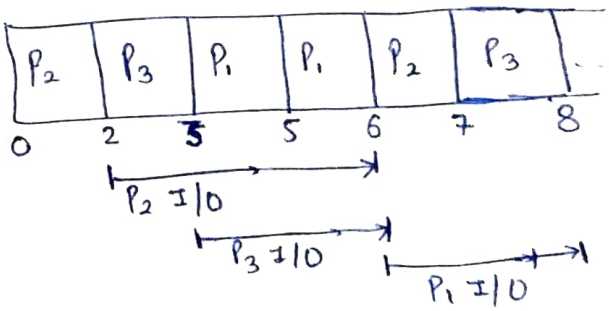


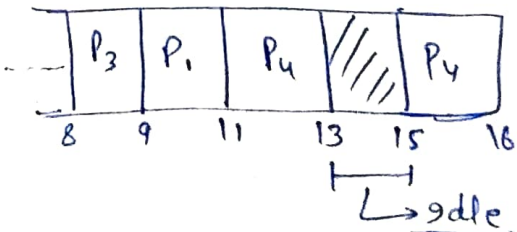
Throughput = 7/25

SRTF with processes contains CPU and I/O time (Imp.)

Ex-1

PNO.	AT	(BT	IOBT	BT)	CT	TAT	WT
1	0	3	2	2	11	11	6
2	0	2	4	1	7	7	4
3	2	1	3	2	9	7	4
4	5	2	2	1	16	11	8





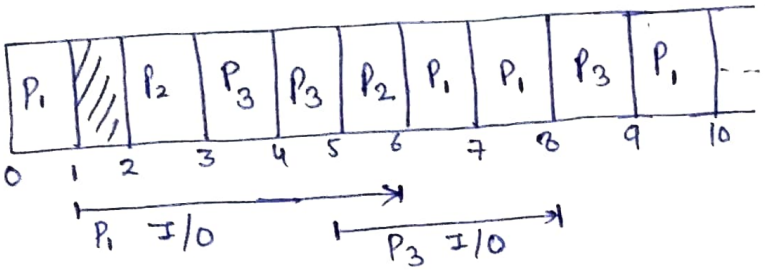
Waiting time :

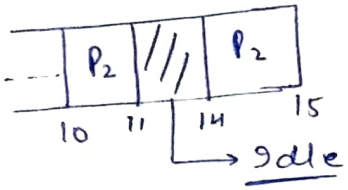
WT = TAT - BT (CPU only)

Preemptive priority with processes contains CPU and I/O Time

P.no	AT	Priority	CPU	I/O	CPU	CT	TAT	WT
1	0	2	1	5	3	10	10	6
2	2	3(L)	3	3	1	15	13	9
3	3	1(H)	2	3	1	9	6	3

Here priority are given in reverse order.





→ Total time = 15

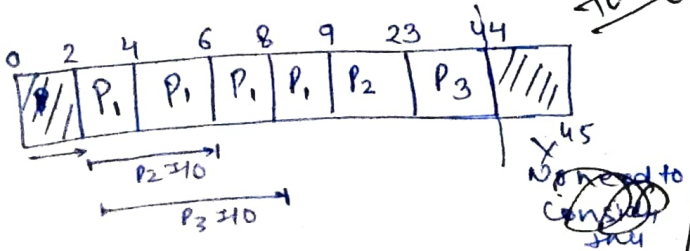
→ Used time = 15 - (1+3) = 11

efficiency = $\left(\frac{11}{15} \times 100\right)$

Q 3 processes arriving at time zero with total execution time of 10, 20 and 30 units. Each process spends the first 20% time doing I/O, 70% time in computation and last 10% in doing I/O again. Compute % idle time for CPU.

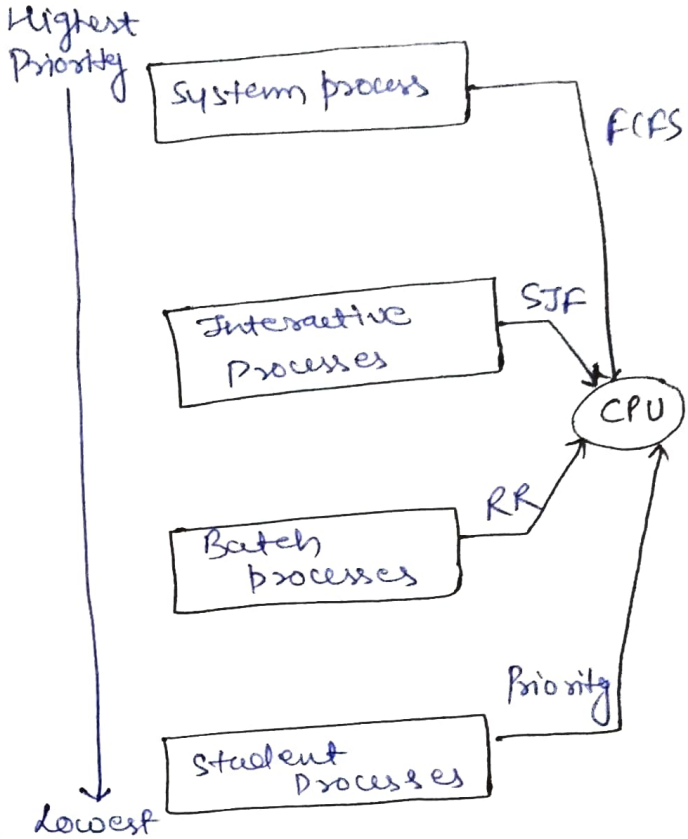
	AT	I/O	CPU	I/O
→ P_1	0	2	7	1
P_2	0	4	14	2
P_3	0	6	21	3

we will complete the I/O too



% Idle = $\left(\frac{2}{44} \times 100\right)$

Multilevel Queues and multilevel feedback Queues



→ Starvation for lower level queues. (Disadvantage)

→ We are applying different scheduling for different queues. (Advantage)

→ To avoid starvation we have 'multilevel feedback queues'. (Just remember)

↳ we can move processes up and down in it.

Operating System

→ OS is interface btw user & hardware, also provides system calls, works as resource allocator, manages m/m, process & security.

→ PCB attributes are → Process Id, Program Counter, Process state, Priority, General Purpose register, List of open files/devices, Protection.

→ If one executable code tries to access other's space called segmentation fault.

→ Multiprogramming with preemption: Multitasking

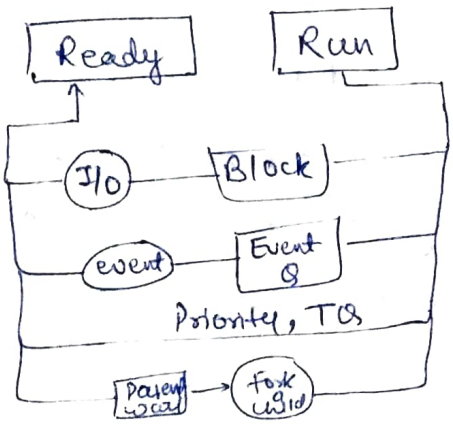
→ States of Process: New -- Ready -- Run -- Terminate -- Block/Wait -- Suspend ready -- Suspend block/Wait

→ Long term scheduler do swap in/out between secondary m/m to New Queue.

→ Short term scheduler do context switching work as dispatcher.

→ Medium term scheduler decided about suspend wait/block.

→ After Suspension process stay in secondary memory.



Consider a system with 'N' CPU's & 'M' process than,

	min	max
ready	0	M
run	0	N
block	0	M

Step: find completion Time

Step: TAT = (Completion - Arrival)

Step: WT = (TAT - BT)

Step: Response Time → process gets scheduled first time.

① FCFS: Non-preemptive & criteria is arrival time.

→ In case of non-preemptive Algo. 'WT & response time is same'.

Convoy effect is disadvantage of both FCFS and SJF

T(n) = O(n) DS = Queue

② SJF: Non preemptive, shortest job first.

Throughput → No. of process / Total burst time

BT prediction Technique

- Static
  - ① Process size
  - ② Process Type
- Dynamic
  - ① Simple avg.
  - ② Exponential avg or ageing.

exp. avg → $\tau_{n+1} = \alpha t_{n-1} + (1-\alpha)\tau_{n-1}$

③ SRTF → Preemptive and selection criteria is BT.

④ Round Robin → Queue is used to make process sequence.

⑤ HRRN → Better as compare to SJF.

→ At the time when all process gets inside start calculate Response ratio $\left(\frac{w+BT}{BT}\right)$ for each remaining process. Starts from shortest job available.

→ Using Multilevel feedback queue, we avoid starvation.

→ ~~Process Synchronization~~

User level thread → Implemented by users, doesn't recognized by OS, implementation is easy, less context switching time, no hardware support, all threads are independent of one thread gets block complete process will get blocked.

Kernel level thread → Implemented by OS and complicated, recognized by OS, hardware support needed, more context switch time, independent threads, if one thread blocked another thread can continue.

→ Each thread shares code section, data section and OS resources but don't share its program counter, registers set & stack sets. Also shares message queue and file table.

Advantages of thread over process → Responsiveness, fast context switching, effective utilization of (MP), resource sharing.

→ Types of System Call: Process control, file management, Device management, Information maintenance, comm^n.

Spooling → Simultaneous peripheral operation online.

Application: ① Used by I/O device like like keyboard, printer etc
② A batch OS used to maintain a queue of ready → sean jobs
③ E-mail is deliver by Mail transfer agent (MTA) to a temporary storage where it waits to be picked up by MAC (Mail User agent).

Inter Process Comm^n → IPC may be either independent or cooperating. In independent one process can't effect other's environment.

Advantage of process Co-operation is → sharing info, computation speed up, modularity and convinient to work on same task at a same time.

Two models of IPC → (1) Shared m/m → All the process Shared the m/m space.

(2) Message passing → Communication takes place by message exchanged mechanism, requires Kernel support.

→ In Batch OS coe group all the jobs with same requirement

→ In timesharing os the main aim is to reduce response time instead of Increase in CPU utilization. (Round Robin)

→ Kernel mode : Prevliged mode ; User mode → Non-privileged

→ User level threads are transparent to kernel levels

Hardware Interrupt : Generated by external device or hardware, don't increment Program Counter, invoked with some external device, low priority than SI, asynchronous event. Types – Maskable or Non maskable.

Software Interrupt : Generated by Internal System of computer, increment program counter, invoked by INT instruction, higher priority among all interupts, it is a way to trigger system call or contact with kernel, synchronous, [Normal Interrupts & Exceptions] are its types.

→ In both Synchronous | asynchronous ISR invoked after completion of I/O.

→ fork() → return 0 for child
→ return(+ve) for parent.