My Notes
Module : 02

- Process
- Process Scheduling
- Operation On Process
- Co-operating Process
- Threads
- Interprocess communication
- Process Scheduling
- Scheduling Criteria
- Scheduling algorithm
- multiple process scheduling
- Real Time scheduling

Process : concept

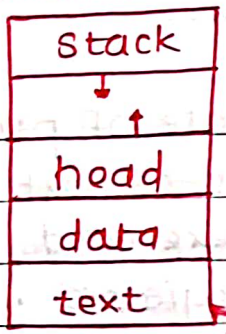
- Process is the program in the execution
- It is dynamic and running activity
- It is task performed by CPU and managed by OS.

Program is static Activity

- Program is user-written
- Process is generated by os to run small part of the program.
- Program becomes process when loaded to the memory.

Parts of Process

- Text section – program code part
- stack section – contains temporary data (local variables, function parameters return addresses)
- Data Section – Contains global variables
- Heap section – Contains memory that is allocated during run time.



Process in memory

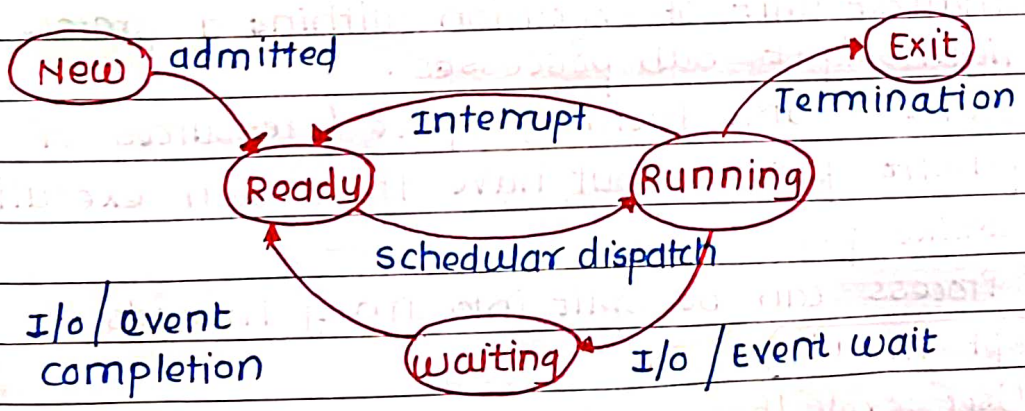
Process State

→ Process passes from various stages.
→ As the process runs it changes its state.
→ Current Activity of the process.

States

1] New : Process creation (Yet not loaded into memory)
2] Ready : Process is now ready for execution.
3] Running : In execution
4] Waiting : Process waiting for some event to occur.
5] Termination : Process is finished.

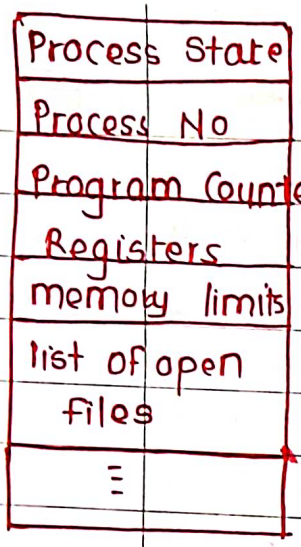
* only one process can run at a time and at the same time other processes are in waiting or ready state.



Process state diagram

Process Control Block (PCB)

→ Used to represent particular process in OS.
→ Reffered as PCB or TCB (Task Control Block)
→ Data structure that contains process information.
→ Helpful for process management.



There are following components :

1) Process state - Current state of process
2) Program counter - Address of next instruction to be executed
3) CPU Registers - various registers like Accumulator, Index reg., etc.
4) Process Number - Unique ID
5) List of open files - Files associated with process.
6) CPU scheduling info - Describes what type of CPU scheduling is done
7) Memory management info - Includes info of page tables and limit & Base register values.
8) Accounting info - Includes time limits, Acc. No, Amount CPU used, etc.
9) I/o status info - Includes list of different I/o devices allocated.

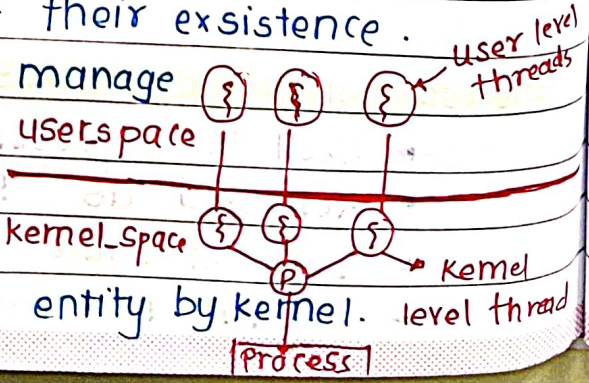
Thread

- Smallest unit of execution within a process
- lightweight sub processes.
- Shares same memory space & resources of a parent process but have their own execution context, unlike process.
- Process can be split into many threads.

Types of threads

1] User level threads :
- manage by user level thread library
- kernel is unaware of their exsistence.
- Faster to create and manage

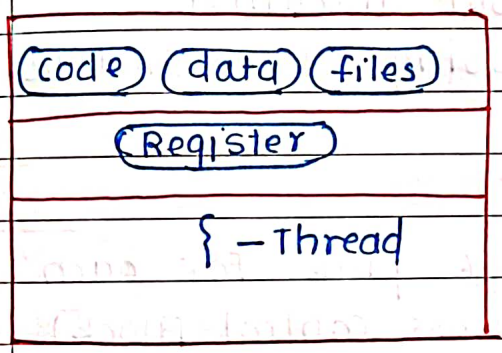
2] Kernel level threads :
- Managed by kernel
- Treated as separate entity by kernel.



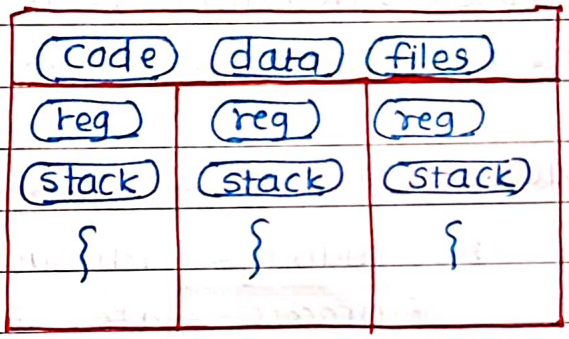
Why do we Need Thread

- Fast context switching
- Less time for termination of thread
- Can share common data.
- Less time for thread creation

Thread Components
- Program Counter
- Register set
- Stack Space



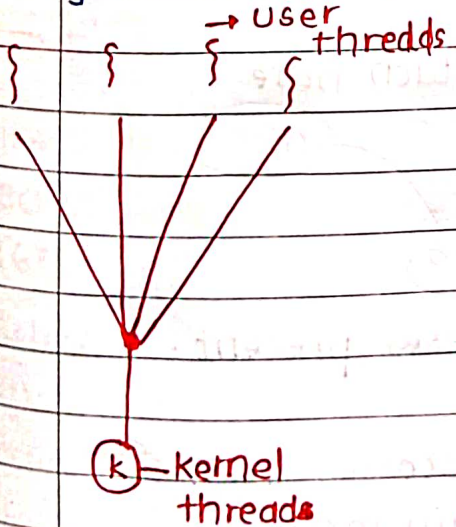
single-Threaded process



Multithreaded process

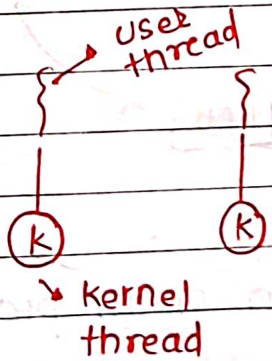
Multithreaded process Models

1] Many-to-one



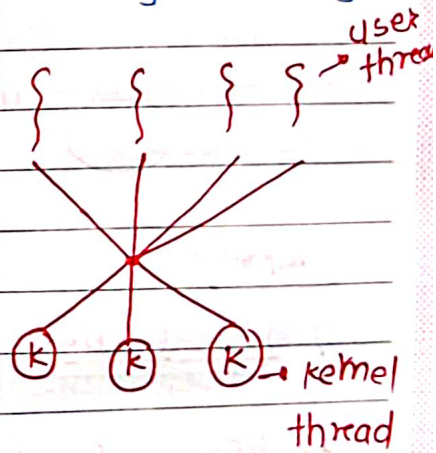
- many user level threads maps single kernel level thread
- Block of one thread leads to end

2] One-to-one



- Each user level thread maps to single kernel level thread.
- Creating kernel level thread is overhead

3] Many-to-Many



- Maps many user threads to equal or more no. of kernel level threads.

Process Scheduling

Multiprogramming objective - Loading multiple programs into main memory.

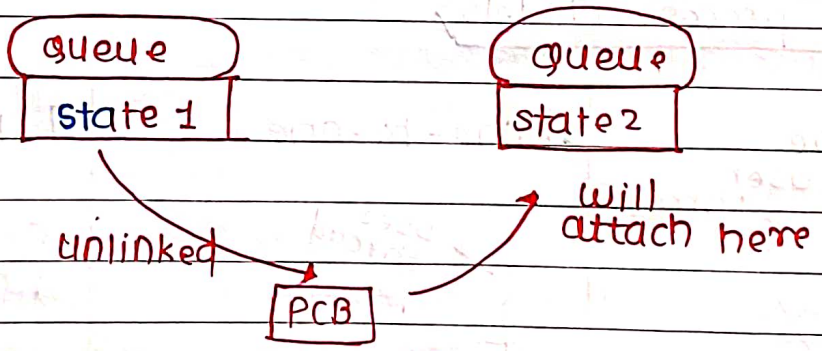
Timesharing objective - switching according to time quantum time.

→ so to maximize CPU utilization and switching CPU among processes, CPU scheduler will select appropriate process from set of processes.

→ Process scheduling is nothing but allocating the CPU for process execution using CPU scheduler.

queues of process scheduling

→ Os manages various types of queues for each of process state. PCB (Process control Block) related to process is also stored in queue of same state

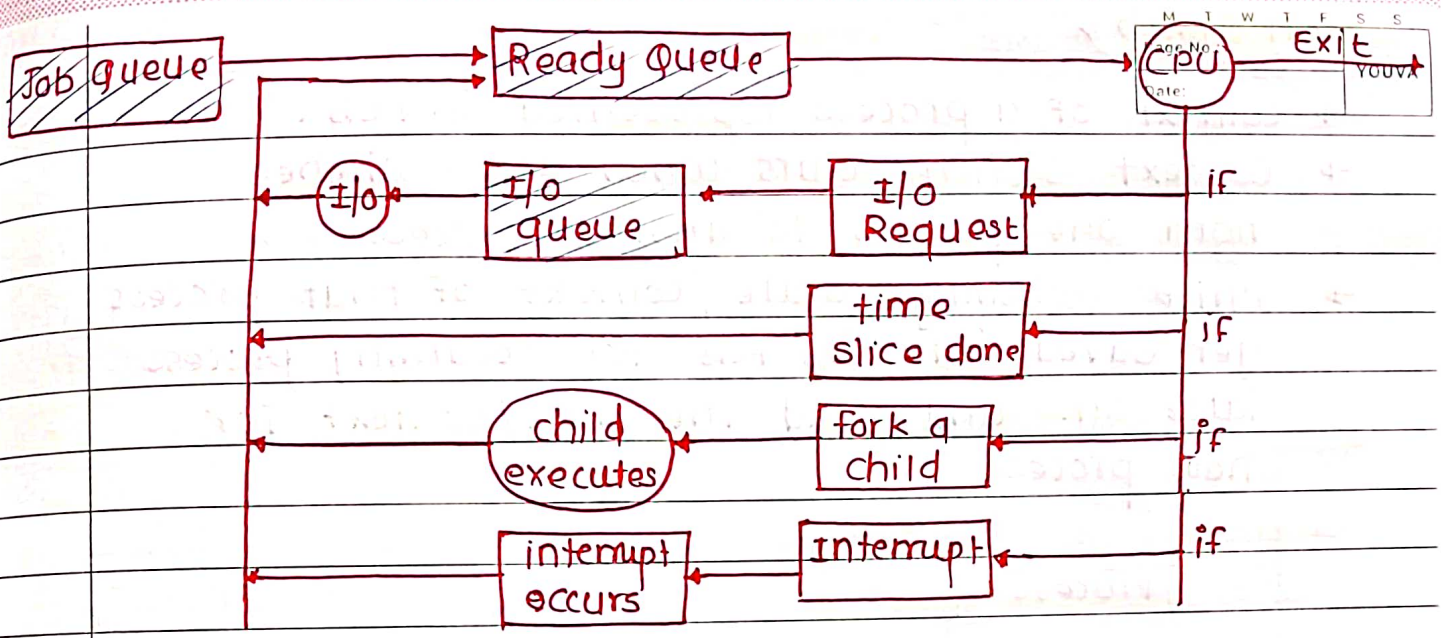


Types

1) JOB queue - collection of all processes present.

2) READY Queue - collection of all processes that are stored in main memory.
- Process which are in ready or wait state

3) Device Queue - List of process that are waiting for I/o device.



queueing diagram

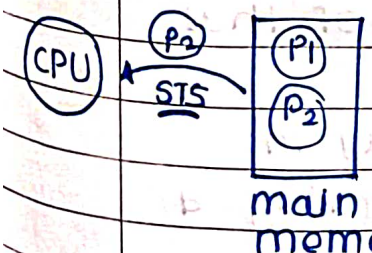
schedular

- During process lifetime it travels from various queues.
- OS have to select processes for scheduling purposes from queues.
- This is done by schedular.

Types

(CPU schedular)
1) SHORT TERM SCHEDULERS (STS)

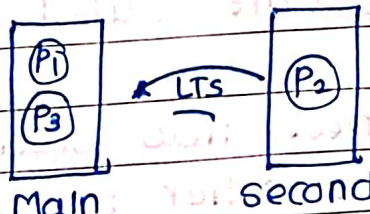
- Selects processes that are ready to execute & allocates CPU to one of them



- Ready state to running state

(Job schedular)
2] LONG TERM SCHEDULERS (LTS)

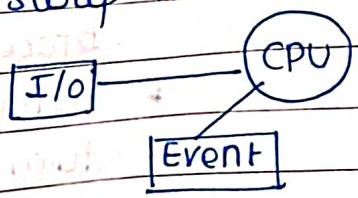
- Selects processes from secondary memory to main memory



- new to ready state

(swapper)
3] MEDIUM TERM SCHEDULERS (MTS)

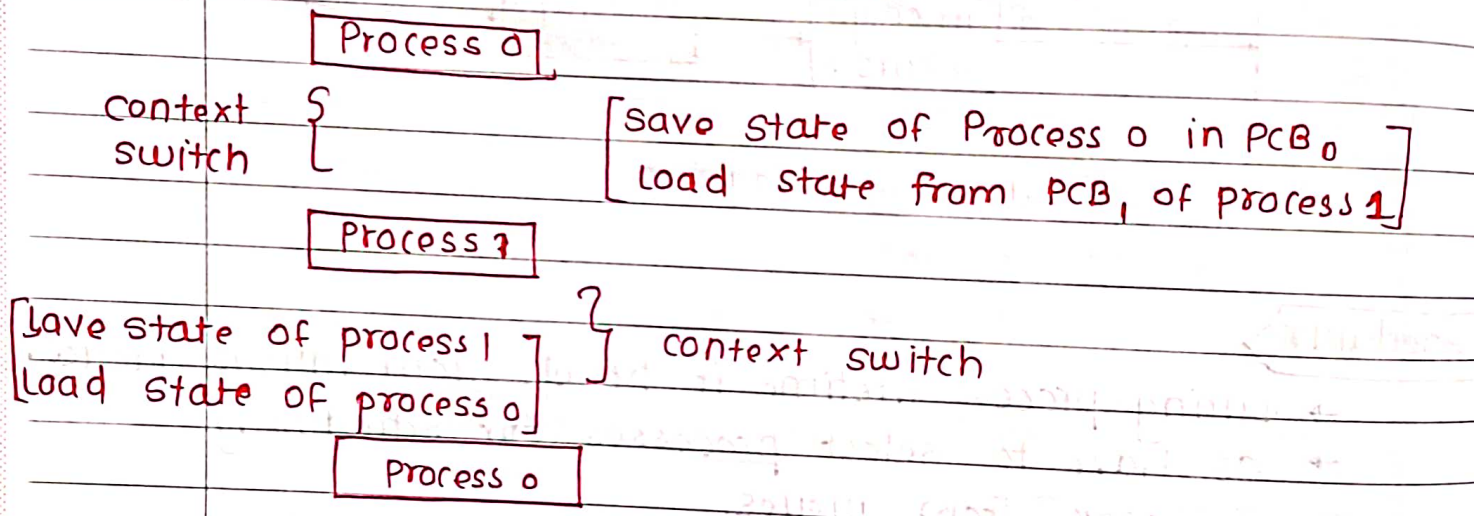
- It is part of swapping.
- Here process is swap out & then swap in later



- Running to wait to ready

Context Switch

- context of a process represented in PCB.
- Context switch occurs when CPU switches from one process to another process.
- while switching state / context of that process get saved into its PCB for resuming process later on. and load the state/context for new process



Operation on Processes

Includes i] process creation
ii] Process termination

Process Creation

- A process may create several new processes.
- Creating process is called a parent process new processes are called children of that process.
- Each of these new processes may in turn create other processes, forming a tree of processes.

possibilities after new process creation : ① Resource sharing

1] Parent and Children share all resources.
2] Children share subset of parent's resources.
3] parent and child share no resources

② Execution option :

1] Parent & child execute concurrently.
2] Parent waits until child terminate.

③ Address Space :

1] Duplicate of parent
2] New program loaded into it.

Process Termination :

- A process terminates when it finishes executing its final statement and asks the os to delete it by using exit() system call.
- via exit() system call pro child process returns status value to its parent process.
- all resources are then deallocated

- Process can be terminated by other process (but only if that other process is parent process)

The parent is exiting & if os doesn't allow a child to continue after parent terminates

parent process can terminate its children process for

- Task assigned to child is no longer required.
- Child has exceeded allocated resources

Interprocess Communication (IPC)

- The way in which processes communicate with each other
- Process may be
  1] Independent
  2] Cooperating

Independent : - Can't affect or be affected by any other process running in system.
- Do not share anything with other processes.

Cooperating : - get affected and also affects other processes running in the system
- It shares data.
- So IPC is required

Need of Co-operation

① **Information sharing** - when several users interested in same piece of data, there should be environment to access that data.

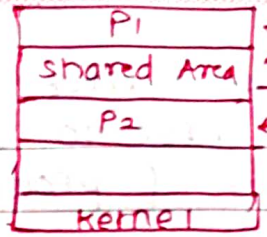
② **Computation speed up** - Divide task into subtasks, each of which will be executing in parallel, for speed up.

③ **Modularity** - Dividing

④ **Convenience** - User can work on multiple tasks at a same time.

IPC models
1] shared memory
2] Message passing

1] Shared Memory :
- Here An area of memory shared among the processes wish to communicate



} In this case

- Here P_1 will write data to shared memory and P_2 will read that data and vice versa
- Shared memory region resides in the address space of process creating shared-memory segment.
- Other process that wish to communicate using this shared memory segment must attach it to their address space.
- OS don't interfere here.

- **Producer - Consumer Problem**

Producer: Produces info

Consumer: Consumes info produced by producer.

Problem: Producer and consumer should work concurrently so that consumer will only consume what is produced.

Solution: - using shared memory.
- i.e We should have buffer (like shared mem)

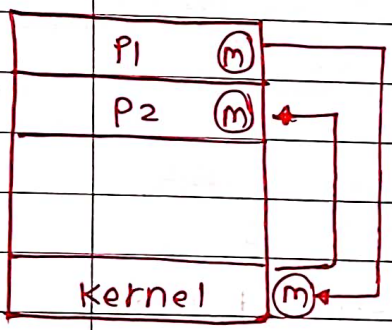
buffer:
- where it will filled by producer's production
- It will be empty by consumer consumption

- working concurrently.
- synchronization is must

Buffer — Bounded Buffer
→ limited size buffer
→ so both have to wait (consumer & producer)

Unbounded Buffer
→ unlimited size
→ thus producer will always produce new item without waiting buffer to be free.
→ consumer have to wait only if buffer is empty.

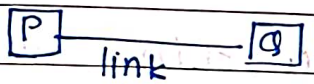
2] Message Passing:



→ Communication in distributed environment, where communicating processes may located on diffnt computers of same N/w.
→ Communication happen with messages.

Two operation:
- send (msg)
- receive (msg)

→ Message can be of fixed or variable size
→ If P and Q wants to communicate with each other then there should be communication link betn them.



→ Link implementation [logically]:
1] direct / indirect communication
2] Synchronous / asynchronous communication

3] Automatic / Explicit buffering

M T W T F S S
Page No.:
Date:
YOUVA

(1) Direct and Indirect Communication :

- Direct communication : Name should be specified.
  * send (P, message) - send to P
  * receive (Q, message) - message from Q

→ Communication link properties of direct comm.:
  • Automatically link established betn pair of processes wants to communicate. (Identity should be known to each other)
  • link is only betn two

- Indirect Communication :
  → mailbox is used for send & receive the msg
  → Each mailBOx has unique id.
  → Two process can communicate if the process have a shared mailbox.

send (A, msg) } A is mailbox here.
receive (A, msg)

→ Communication link properties
  • link will established only if both processes have shared mailbox
  • link may be associated with more than two processes.

Synchronization : Either blocking or non-blocking msg passing is there.

→ Synchronous and asynchronous communication

- Blocking (Synchronous)
- Non-Blocking (Asynchronous)

Blocking :

⊙ Blocking send - sending process is blocked until the message is received by receiving process or by the mailbox.

⊙ Blocking receive - Receiver is blocked until a msg. is available.

Non-Blocking :

⊙ Non-blocking send : Sending process sends the message and resumes operation.

⊙ Non-Blocking receive :- receiver retrieves either a valid message or a null.

Buffering

- msgs are resides in temporary queue.

Three ways

1] zero capacity ÷ no messages are queued on link.

2] Bounded -

3] unbounded -

Client-server Based communication -

Socket

→ endpoint for communication

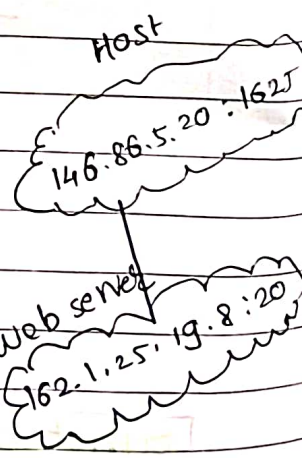
→ Pair of processes communicating over a N/w consist Sockets.

→ Socket = IP address + port No.

→ Communication consist a pair of socket

→ 0-1024 well known

→ Loopback IP - 127.0.0.1



146.86.5.20:1625

IP → port

RPC

Remote Procedure Call :

- Protocol that one program can use to request a service from a program located in another computer on a N/w without having N/w details.
- As the programs are present in two different systems which are connected by same N/w the "message based communication scheme" is used.
- It uses ports.

Stubs : Client side proxy for actual procedure on server.

- client side stubs : locates the server and sends parameters.
- Server side stubs : • receives the message,
  • unpacks the parameters,
  • Performs the procedure, on the server.

Tricks ⊙ when arrival time not given only burst time is given → (calculation of waiting time is normal)

⊙ when arrival time is also given with burst time (Non-preemptive scheduling) → (calculation of waiting time is Turnaround time − Burst time)

Turnaround time = completion − Arrival

⊙ when arrival time is also given with burst time (Preemptive scheduling)

→ waiting time = (Total WT) − (How much executed) − (Arrival Time)

→ Turnaround = completion − Arrival

RPC

Remote Procedure Call :

- Protocol that one program can use to request a service from a program located in another computer on a N/w without having N/w details.
- As the programs are present in two different systems which are connected by same N/w the "message based communication scheme" is used.
- It uses ports.

Stubs : Client side proxy for actual procedure on server.

- client side stubs : locates the server and sends parameters.
- Server side stubs : • receives the message,
  • unpacks the parameters,
  • Performs the procedure, on the server.

Tricks ① when arrival time not given only burst time is given → (calculation of waiting time is normal)

② when arrival time is also given with burst time (Non-preemptive scheduling) → (calculation of waiting time is Turnaround time − Burst time)

Turnaround time = completion − Arrival

③ when arrival time is also given with burst time (Preemptive scheduling)

→ waiting time = (Total WT) − (How much executed) − (Arrival Time)

→ Turnaround = completion − Arrival

⊙ CPU Scheduling ⊙

- Basis of multiprogrammed operating systems
- By switching the CPU among processes, the OS can make the computer more productive.
- maximize CPU utilization.
- Several processes kept in memory at one time.

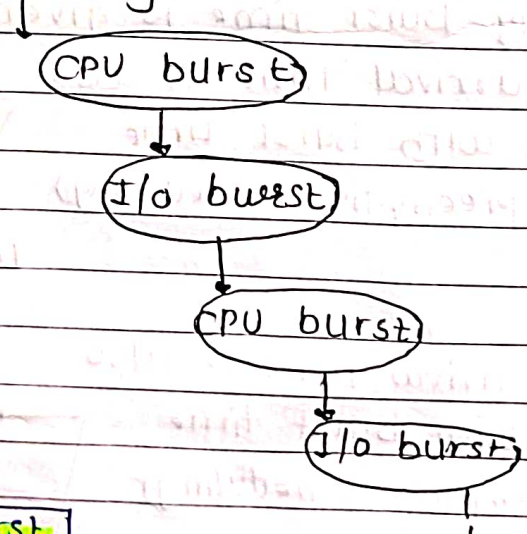
CPU and I/O Burst Cycles:

- Process execution consists of a cycle of CPU execution and I/O wait

↓

Processes alternate betn these two states.

- Process execution begins with



CPU burst is when process being executed in CPU

I/O burst when CPU waits for I/o -for further operation

final CPU burst ended with a system request to terminate execution.

CPU Scheduler :

→ Selects the processes from ready queue and allocates a CPU core to one of them.

→ CPU scheduling takes place when process goes from
① running to waiting state
② running to ready state (eg. Interrupt)
③ Waiting to ready state
④ Termination.

- **Preemptive and Non-Preemptive Scheduling :**

Preemptive : case ② and ③ ;
- CPU can be given to any other process present in ready queue.
- Here, multiple process can run, one process can be preempted to run another

Non-preemptive : case ① and ④
- when process is assigned to CPU, it ~~takes~~ keeps process busy, till it gets terminate.

- **Dispatcher :**
  - Gives control of CPU to the process selected by CPU scheduler.
  - involves → switching context
    → switching to user mode
    → Jumping to proper location in user program.

Dispatch latency : time taken by to stop one process and start another.

Scheduling Criteria:

Scheduling criteria involves

1] CPU utilization :
- keep CPU as busy as possible

2] Throughput : (max)
No. of processes completed within given time frame.

3] Turnaround time : (min)
Time taken by process to complete the execution.

4] Waiting time : (Min)
Amount of time a process has been waiting in ready queue.

5] Response time : (Min)
- Time taken from when req. submitted until the first response.

Scheduling Algorithms.

1] First - Come, First - Served (Non-premptive)
- FCFS
- Easy
- Process which requests the CPU first is allocated to the CPU First.
- Follows FIFO
* - Not good for time shared os.
* - CPU and I/o are not utilized properly.
→ Avg. waiting time is long.

• Example 1]

Process	Burst Time
P_1	24
P_2	3
P_3	3

① Corder of arrival (P_1, P_2, P_3)

Gantt chart

waiting time for $P_1 = 0$

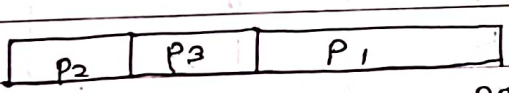
$P_2 = 24$

$P_3 = 27$

Avg. WT = $0+24+27 / 3 = 17$ ms

② order of arrival (P_2, P_3, P_1)

Gantt chart



WT = $P_2 \rightarrow 0$

$P_3 \rightarrow 3$

$P_1 \rightarrow 6$

Avg WT = $0+3+6 / 3 = 3$ ms

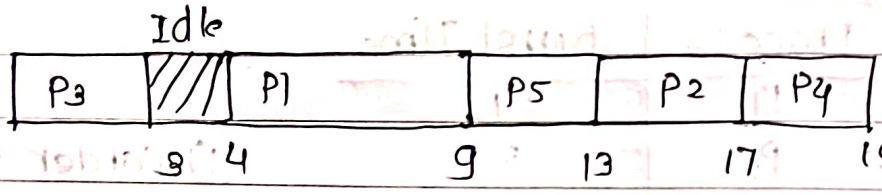
• Example 2]

Process	AT	BT
P_1	4	5
P_2	6	4
P_3	0	3
P_4	6	2
P_5	5	4

calculate Avg. waiting & Avg. turnaround time.

* Turnaround time = completion time - arrival
* waiting = Turnaround - Burst time.

Gantt chart

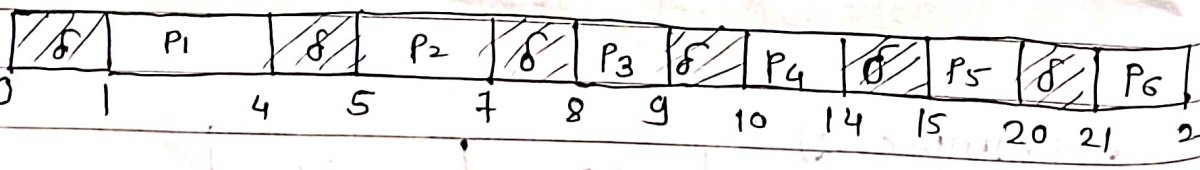


process	Completion	Turnaround	waiting
P1	9	5	0
P2	17	11	7
P3	3	3	0
P4	19	13	11
P5	13	8	4

Avg. Turnaround = 40/5 = 8
Avg. waiting = 22/5 = 4.4

Example 03

Process	Arrival	Burst	
P1	0	3	
P2	1	2	there is 1 unit of overhead find efficiency of algo.
P3	2	1	
P4	8	4	
P5	4	5	
P6	5	2	



δ delay

Useless time = 6 × δ
= 6 units.
Total time = 23
Useful time = 23 - 6 = 17
efficiency = 17/23 × 100 = 73.91 %

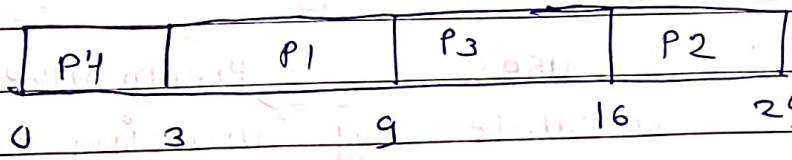
2] Shortest Job First (SJF)

- → CPU is allocated to the process having smallest CPU burst.
- → If same burst time for two processes then FCFS is used to solve that problem.
- → Also refered as shortest Next-CPU burst.
- → Can be Preemptive / Non Preemptive.
- * → burst size should be known in advanced.
- * → Cannot be implemented at short-term CPU scheduli

Example :01] Non-preemptive (SJF)

Process	Burst time	Arrival
P1	6	0
P2	8	0
P3	7	0
P4	3	0

Gantt chart



WT = $P_1 = 3$

$P_2 = 16$

$P_3 = 9$

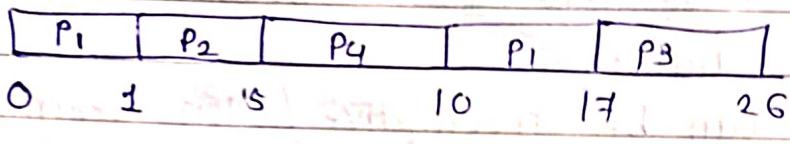
$P_4 = 0$

Avg WT = $(3+16+9+0)/4 = 7$ ms

Example 02 : Preemptive (SJF)

Process	Arrival	Burst
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

Gantt chart



- waiting time = (Total waiting time) − (How much process executed) − (Arrival Time)

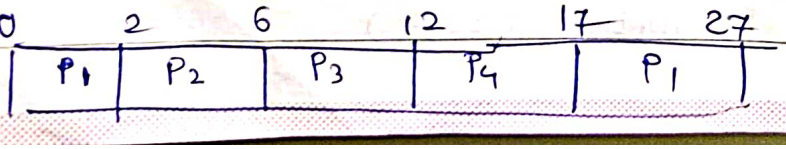
WT = P_1 → 10 − 1 − 0 = 9
P_2 → 1 − 0 − 1 = 0
P_3 → 17 − 0 − 2 = 15
P_4 → 5 − 0 − 3 = 2

Avg WT = 9 + 15 + 0 + 2 / 4 = 6.5 ms

Example 03] : OS uses SJF − Preemptive process. Calculate Avg. waiting time.

Process	Arrival	Burst
P_1	0	12
P_2	2	4
P_3	3	6
P_4	8	5

Gantt chart



WT : P_1 = 17 - 2 - 0 = 16

P_2 = 2 - 0 - 2 = 0

P_3 = 6 - 0 - 3 = 3

P_4 = 12 - 0 - 8 = 4

$$\text{Avg WT} = \frac{16+0+3+4}{4} = 5.5 \text{ ms}$$

Example 04] Use preemptive, calculate avg. turnaround time

Process	Arrival	Burst
P_1	0	10 → 7
P_2	3	6 → 2
P_3	7	1
P_4	8	3

Gantt chart :

P_1	P_2	P_3	P_2	P_4	P_1
0	3	7	8	10	13	20

Turnaround time = Completion - Arrival

P_1 = 20 - 0 = 20

P_2 = 10 - 3 = 7

P_3 = 8 - 7 = 1

P_4 = 13 - 8 = 5

$$\text{Avg TT} = \frac{20+7+1+5}{4} = 33/4$$

$$= 8.2 \text{ ms}$$

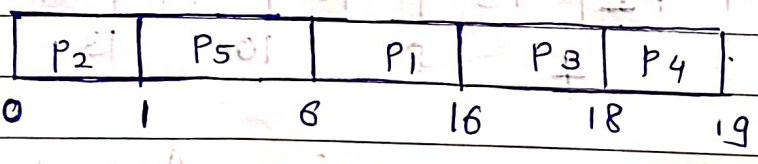
3] Priority ~~Queue~~ Scheduling Algorithm :

- Priority is associated with each process, and the CPU is allocated to the process with the highest priority.
- Equal priority = Follow FCFS
- Can be preemptive / Non-preemptive.

Example 1

Here arrival time = 0.

Process	Priority	Burst time
P_1	3	10
P_2	1	1
P_3	4	2
P_4	5	1
P_5	2	5



WT = $P_1 \Rightarrow 6$

$P_2 \Rightarrow 0$

$P_3 \Rightarrow 16$

$P_4 \Rightarrow 18$

$P_5 \Rightarrow 1$

Avg WT = $\frac{6 + 16 + 18 + 1 + 0}{5} = 8.2$ ms

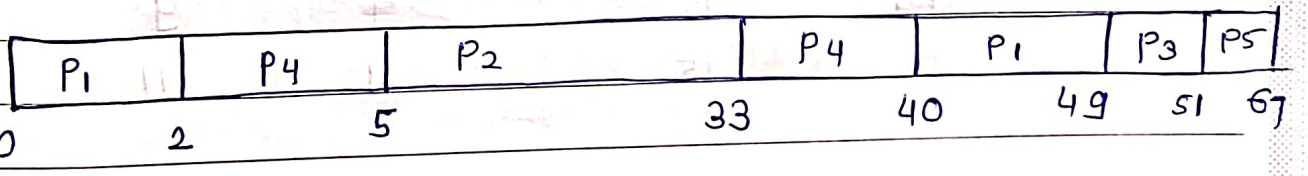
Problems :

- Indefinite blocking and Starvation.
  - Solution aging
    (Gradually increasing priority of process that wait in system for long time)

Example 2 : 0 is with high priority
Use preemptive priority scheduling.

Process	Arrival	Burst time	priority
P_1	0	11	2
P_2	5	28	0 (Highest)
P_3	12	2	3
P_4	2	10	1
P_5	9	16	4

Gantt chart :



WT = $P_1 \Rightarrow 40 - 2 - 0 = 38$
$P_2 \Rightarrow 33 - 0 - 5 = 0$
$P_3 \Rightarrow 49 - 0 - 12 = 37$
$P_4 \Rightarrow 33 - 8 - 2 = 28$
$P_5 \Rightarrow 51 - 0 - 9 = 42$

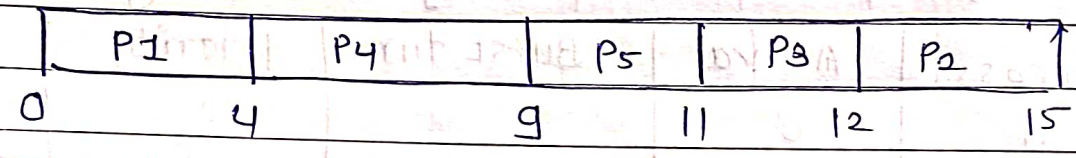
Avg = $(38 + 37 + 28 + 42 + 0)/5 = 29$ ms

Example 3 :- Higher No represents higher priority
use non-preemptive priority scheduling

Process	Arrival	Burst	Priority
P1	0	4	2
P2	1	3	3
P3	2	1	4
P4	3	5	5
P5	4	2	5

Find avg WT and avg TT.

⇒



WT ⇒ TT - Burst

TT ⇒ Completion - Arrival

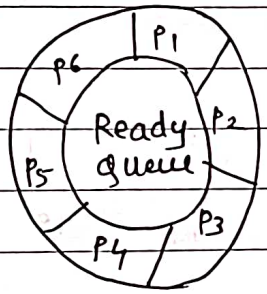
Process	Completion	WT	TT
P1	4	0	4
P2	9	5	8
P3	11	8	9
P4	12	4	9
P5	15	9	11

$$\text{Avg WT} = \frac{0+5+8+4+9}{5} = \frac{26}{5} = 5.2\text{ms}$$

$$\text{Avg TT} = \frac{4+8+9+9+11}{5} = \frac{41}{5} = 8.2\text{ms}$$

4] Round Robin Scheduling :-

- Especially designed for timesharing system.
- Similar to FCFS scheduling, but preemption is added to switch processes in betn
- Small unit of time called time quantum or time slice (10 to 100 milliseconds)



- treated as circular queue.
- Preemptive

Implementation :

- Ready queue treated as FIFO principle
- New processes are added to tail
- Scheduler picks process from first, set timer and dispatches the process.

Conditions

when cpu burst less than 1 time quantum
↓
Process will release CPU voluntarily
↓
CPU scheduler will proceed to Next

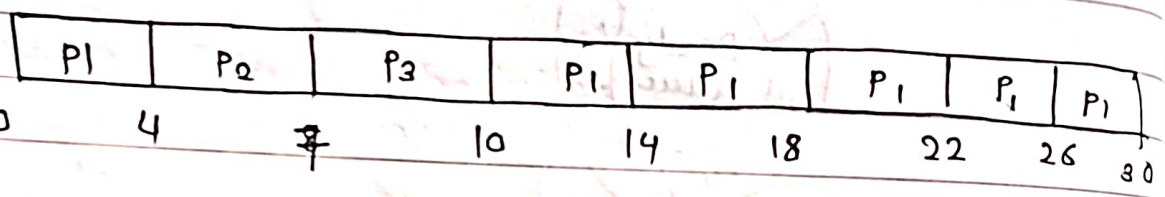
when CPU burst more than 1 time quantum
↓
Context switch will be executed and process will be put at tail
↓
CPU scheduler will then select Next processes.

Example 1 : amival time = 0

quantum time = 4ms

Process	Burst
P1	24
P2	3
P3	3

Gantt chart



TTumaround tim. = Completion - Arrival

$P_1 = 30 - 0 = 30$

$P_2 = 7 - 0 = 7$

$P_3 = 10 - 0 = 10$

$\text{Avg TT} = \frac{30+7+10}{3} = \frac{47}{3} = 15.6\text{ms}$

Waiting time =

method (1)

WT = TT − Burst time

$P_1 = 30 - 24 = 6$

$P_2 = 7 - 3 = 4$

$P_3 = 10 - 3 = 7$

$\text{Avg} = \frac{7+4+7}{3} = \frac{17}{3} = 5.6\text{ms} = 5.66\text{ms}$

method ②

WT = last start time − Arrival time − (preemption × Time quantum)

$P_1 = 26 - 0 - (5 \times 4) = 6$

$P_2 = 4 - 0 - (0 \times 4) = 4$

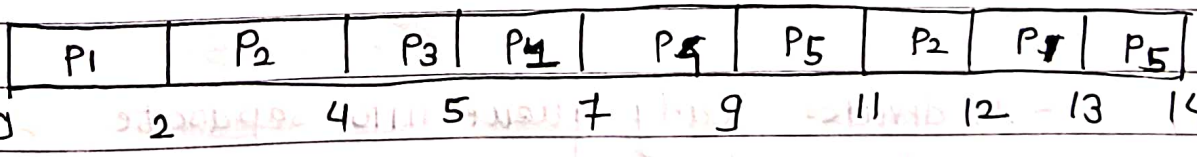
$P_3 = 7 - 0 - (0 \times 4) = 7$

$\text{Avg WT} = \frac{6+4+7}{3} = \frac{17}{3} = 5.66 \text{ ms}$

Example 2 : time quantum = 2
find avg WT and avg TT.

Process	Arrival	Burst
P_1	0	5 - 3 - 1 - 0
P_2	1	3 - 1 - 0
P_3	2	1 - 0
P_4	3	2 - 0
P_5	4	3 - 1 - 0

Gantt chart



Turnaround Time = completion − Arrival

$P_1 = 13 - 0 = 13$

$P_2 = 12 - 1 = 11$

$P_3 = 5 - 2 = 3$

$P_4 = 9 - 3 = 6$

$P_5 = 14 - 4 = 10$

$\text{Avg TT} = \frac{13+11+3+6+10}{5}$

$= 8.6 \text{ ms}$

waiting time :

$P_1 = 12 - 0 - (2 \times 2) = 8$

$P_2 = 11 - 1 - (1 \times 2) = 8$

$P_3 = 4 - 2 - (0 \times 2) = 2$

$P_4 = 7 - 3 - (0 \times 2) = 4$

$P_5 = 19 - 4 - (1 \times 2) = 13$

$$\text{Avg. WT} = \frac{8 + 8 + 2 + 4 + 13}{5} = \frac{29}{5} = 5.8$$

5] Multilevel Queue Scheduling :

- scheduling among queues.
- Class of scheduling algorithm has been created for situations in which processes are easily classified.

processes →

Foreground

(interactive
direct interaction with users)
- quick
- High priority

Background

(Batch
- Not interactive
- Indirect interaction
- Not quick)

- It divides ready queue into separate queues and processes are permanetly assigned to ~~those~~ one queue (done on the basis of some properties)
- Each queue has its own algorithm

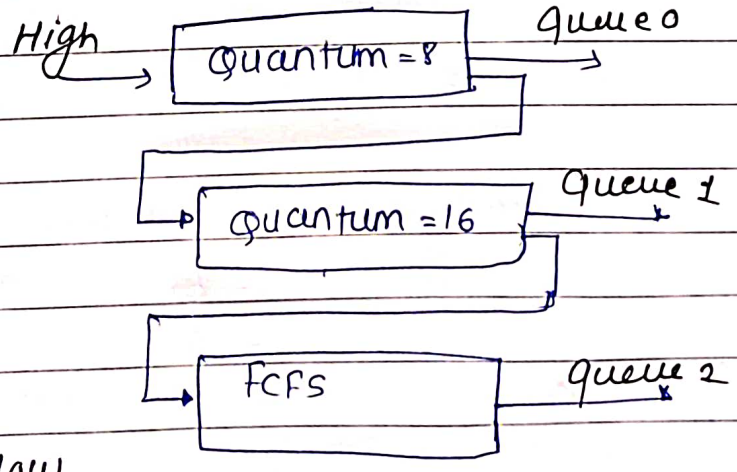
Example :

High → System Processes →
↑ → Interactive Process →
→ Interactive editing Process →
→ Batch Process →
→ Student process →

low priority

6] Multilevel Feedback Queue Scheduling Algo.

- It allows processes to move betn queues
- Separation of processes according to characteristics of their CPU bursts.
- If process uses too much CPU time, it will moved to lower-priority queue
- Leaves I/O bound and Interactive processes in higher priority queues.
- process waiting in lower priority from so long time then move it to high priority queue
- It prevents starvation.



Parameters

- No. of queues
- Scheduling algorithm for each queue
- Method to assign queue to process
- Method to determine moving process among queue.
- Method to determine when to upgrade process or demote process