Message Passing (MP)

→ is useful for exchanging smaller amount of data

→ is easy to implement in distributed system than shared memory (SM)

→ slower than SM

MP provides a mechanism to allow process to communicate and to synchronize their actions without sharing the same address space.

∴ MP provides two operations

send (message) receive (message)

Methods for implementing a communication link and send() / receive() operation

3 methods

1. Direct or Indirect communication
2. Synchronous or asynchronous "
3. Automatic or Explict "

send (P, message) — Send a message to process P

Receive (id, message) — receive a message from any process. (id = name of process)

send (P, message) — send message to process P

receive (Q, message) — receive message from process Q

↑ Direct Communication

send (A, message) — send message to mail box on port A

receive (A, message) — receive message from mail box A

need a common shared mailbox between two process

Indirect Communication

Indirect communication

- messages are send and received from mail box or port

(

Properties of communication link

(1) Link established if there is a shane mail box between process

(2) link may associated with many processes

(3) Between two process there may have several link

(4) Link may be uni-directional or bi-directional

Direct communication

Date :

(*) Process must name each other explicitly

send ((P) , message) → send message to process P

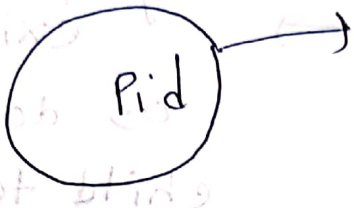
receive (Q , message) → receive message from Q process

communication link property

1. links established automatically
2. link associated exactly one pair of communication process
3. Between two process there exists exactly one link
4. link is bi-directional

Process Creation

Parent process create child proces which in turn create other process framing a tree of process



Process Identifier
(indentify or manages process)

Resource sharing

1. Parent & child share all resource
2. Parent & child share no resource
3. Child share subset of parent resource

Execution sharing

1. Parent and child executes cocurrently
2. parent wait until child terminate

Process termination

→ Process executes last statement and ask OS to delete it (exit())

Cascading Termination → If parent is exiting, some OS do not allow child to continue if parents are terminated. So all child are terminated then

For child process →

if no parent waiting → the terminated process is Zombie

if parent terminated, process are orphans

Two Types of scheduling :-

① Non-preemptive scheduling :-

once is a process is alloctacted in cpu it stays in cpu untill it is terminated by cpu on switch to waiting state

② preemptive :-

allows removal of process from cpu

what is short term scheduler ?

→ ~~when cp~~ is a scheduler which selects among process in the ready queue and allocates the cpu to one of them.

In preemitive we can context switch process if burst time process arrived next

when cpu scheduling occurs ?

→ when process

1. switches from running to waiting state
2. switches from running to ready queue
3. switches from waiting to ready
4. Terminates

Non-preemptive

Preemptive

↳ considerations → access to shared data

→ preemption while in kernel mode

→ interupts occurs during crucial OS activities

What is dispatcher?

→ is a module that gives control of the cpu to process selected by short term scheduler.

This involves activites are
- → context switching
- → switch to user mode
- → jumping in proper location in the user to restart

what are scheduling critenia?

max (1) cpu utilization (keep the cpu busy as much as possible

max (2) Throughput (No. of process that complete their execution per time unit)

min (3) Turnaround time (amount of time to execute a particular process)

min (4) waiting time

min (5) Response time

—— o ——

First come First served (FCFS) scheduling

FCFS scheduling

→ easy to implement

execution time

what is burst time

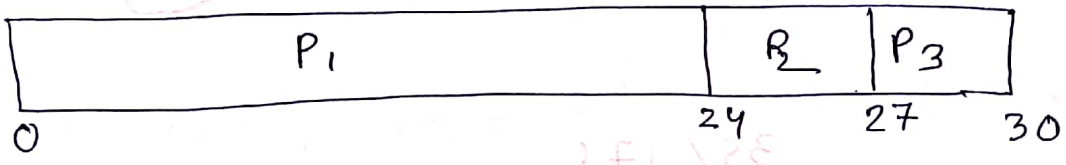
→ it is the time taking by a process on a cpu in one go

Process	Burst time
P_1	24
P_2	3
P_3	3

∴ arriving process order : P_1, P_2, P_3

P_1 comes first it will access cpu first

The Gantt chant for scheduling



∴ waiting time ; $P_1 = 0s$
$P_2 = 24s$
$P_3 = 27s$

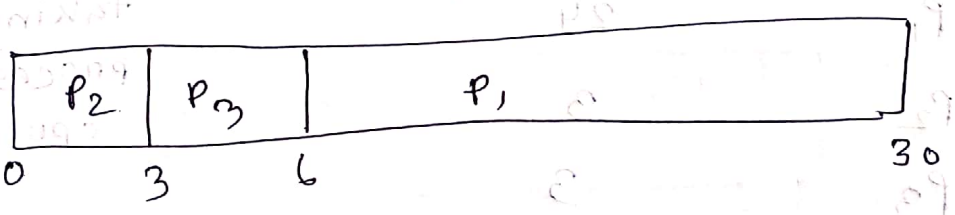
∴ avenage waiting time = $\frac{0+24+27}{3}$

= 17s

Now if we change the process onden

P_2, P_3, P_1 ← P_2 comes tinst

The Gantt chant



∴ waiting time = $P_1 = 6$
$P_2 = 0$
$P_3 = 3$

∴ avenage waiting time = $\frac{0+3+6}{3}$

= 3s

3s < 17s
this case is betten

∴ if short process comes first,
avg waiting time is less.

∴ convoy effect :- short process first,
long process behind.

con:- FCFS suffers from convoy effect & avg waiting time higher

— o —

Shortest Job first (SJF) scheduling

↓ pros

most optimal
better than FCFS

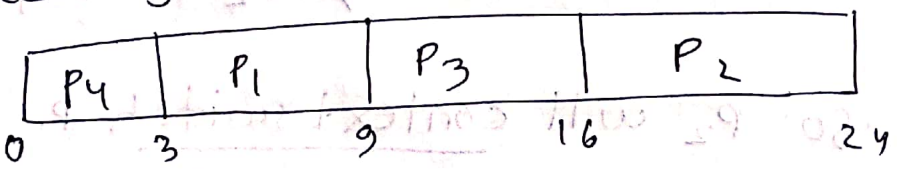
↓ cons

difficult to know the length of ~~next~~ next cpu request

Process	burst time
P_1	6
P_2	8
P_3	7
P_4	3 → shortest first

non-preemptive scheduling

SJF scheduling chart



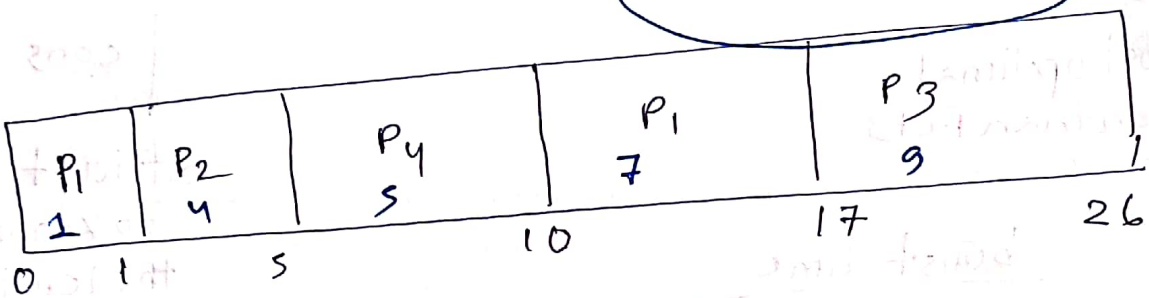
P4, P1, P3, P1

∴ avg. waiting time = 0 + 3 + 9 + 16 / 4 = 7s

shortest remaining time first

Process	arrival time	Burst time
P_1	0	8 (1+7)
P_2	1	4
P_3	2	9
P_4	3	5

Preemtive scheduling



∴ Here P_1 comes first (as arrival time)

∴ after 1s P_2 comes ∴ In that time P_1 has done 1s time. Now P_1 burst time (8-1) 7 > P_2 burst time.

∴ So P_2 will context switch P_1.

∴ P_1 have to wait untill P_2 done

short burst time process first

after 2s P_3 comes;

P_3 burst time (9) > P_1 burst time (7) > P_2 burst time (4-1)/3

inscheduling

∴ P_2 will contine. P_1 have to wait still. then P_3 at last will start cz high burst time

after 3s P_4 comes;

P_3 bursttime (9) > P_1 bursttime (7) > P_4 bursttime (5) > P_2 burst time (3-1)/2

∴ NOW P_2 will still continue.

∴ then P_4 will start cz (P_4 < P_1 bursttime)

∴ after P_4, remaining P_1 start

at last P_3 starts (high burst time)

$$\therefore \text{avg. waiting time} = \frac{\overbrace{(10-1)}^{P_1} + \overbrace{(1-1)}^{P_2} + \overbrace{(17-2)}^{P_3} + \overbrace{(5-3)}^{P_4}}{4}$$

$$= 6.5_s$$

---

less waiting time
less complex

Priority scheduling PS

→ Priority number is associated with each process (Integer)

→ smallest integer (higest priority)

Problem of PS

cons

(Starvation) → low priority problem may never execute

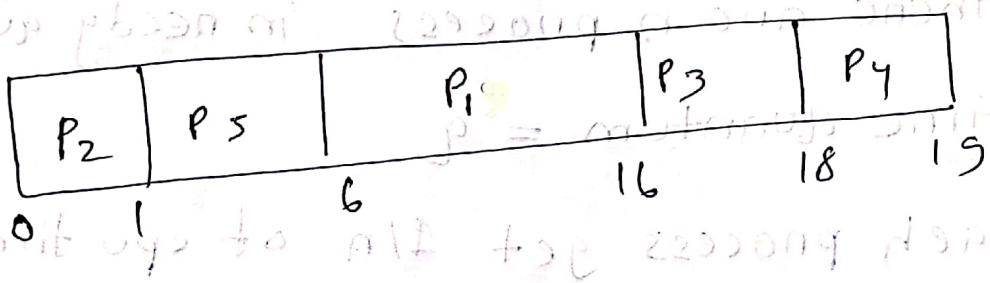
(aging) Sol:- increase priority as time progress of process

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

lowest priority first

Using Non Preemptive (No context switch, wait until a process finish)

Gantt chart



$$\therefore \text{avg. waiting time} = \frac{6+0+16+18+1}{4+1}$$

$$= 8.2$$

Round Robin | RR

→ Each process gets a small unit of cpu time (time quantum, q) (10-100 miliseconds)

∴ After this time passed, process will stop and moved to end of ready queue /added

→ if there are n process in ready queue.

∴ time quantum = q

∴ each process get 1/n of cpu time in chunk at most q times unit at once

∴ waiting time is not more than $(n-1) \times q$

cons :- spend more time context switching

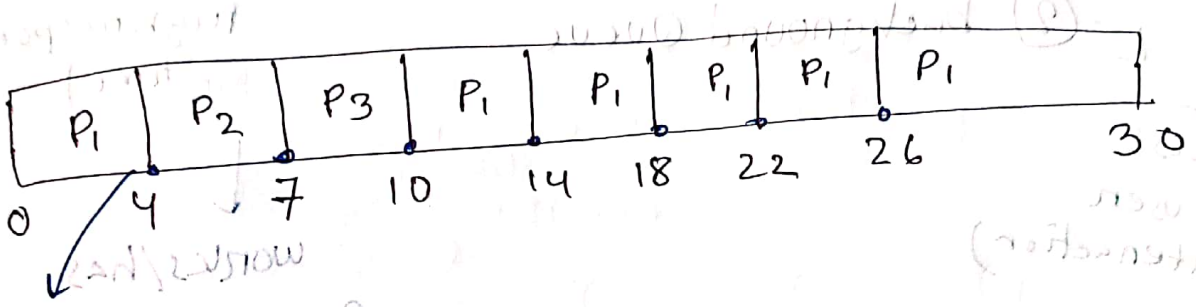
Process	Burst time
P_1	24
P_2	3
P_3	3

time quantum $q = 4$

Here,

Preemptive

Gantt chant is:



context switching

∴ Althrooght, higher avg. time

∴ but faster response time.

∴ q should be > context swtich time.

(but not too large)

↓

then it becoms FcFS

18 Final

Multilevel Queue

Ready Queue is ~~partitioned~~ partitioned into two queue

① foreground Queue (interactive process which have high response time)

② background Queue.

(batch process no need user interaction)

(FCFS algorithm)

works/has Round Robin algorithm

(better response time)

scheduling between queue

① Fixed priority scheduling

② Time Slice

80% to foreground in RR

20% to background in FCFS

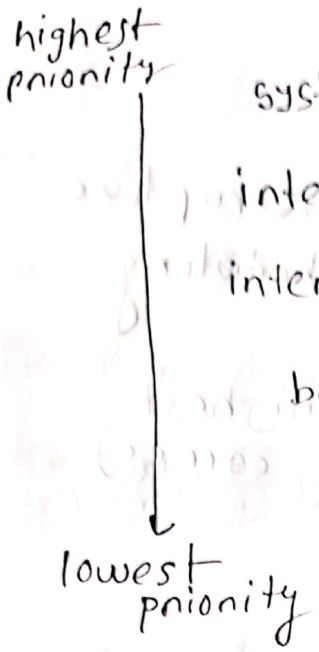


Fig:- Multilevel Queue

multilevel feed back Queue

→ three Queue.

① Q_0 — RR with time quantam = 8

② Q_1 — RR " " " = 16

③ Q_2 — FCFS

window scheduling

→ uses prionity based preemptive scheduling

(context switches when another highest prionity comes)

Multiple Processor Scheduling

(1) Asymentric multipnocesing :- only one processon acess the system data structen alleviating the need of kernel data shaning

(2) Symentric :- each processon is self scheduling each has its own private queue of ready process

what is time quantum?

→ a preemptive scheduler will allow a process to run for a short amount of time. this is called time quantum.

lower time quantum → more context switching → less performance

What is preemptive scheduling?

→ PS is when a process transitions from running state to ready state or from waiting state to a ready state.

→ it runs a task with higher priority before another lower priority task, even if the lower priority task is still learning.

→ better cpu utilization

→ less waiting & response time

→ RR, shortest Remaining time first

what is non-preemptive?

→ Non preemptive when, once a process starts its execution in cpu, it must finish it executing other, it cannot be paused in middle
(but preemptive it can be paused)

→ less optimization

→ more response/waiting time

→ SJF, FCFS

→ cpu is allocated to process untill it terminates or switches to waiting state

[In preemptive; cpu is allocated to process for short amount of time]

what is cpu scheduling

→ cpu scheduling is a process of determing which process will own cpu for execution while another process is on hold.