

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 receive~~
~~(message) (message)~~

Methods for implementing - a communication link

and send() / receive() operation

- ① Direct or Indirect communication
- ② Synchronous or asynchronous
- ③ Automatic or Explicit

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

receive (id, message) — receive a message from any process.

↳ send and message to individual (id = name of process)

Example
(id)

↳ receive file name message from any process

↳ receiving message from random process

→ port might be available or free of information

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

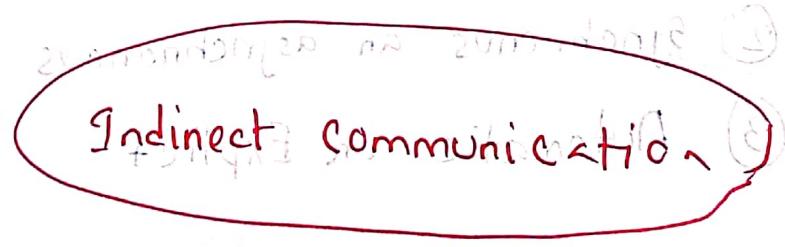
receive (Q, message) — receive message from process Q



send (A, message) — send message to mailbox on port A

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

need a common shared mailbox between two process



Indirect communication

messages are send and received from mail box or port.

Properties of communication link

- (1) Link established if there is a share 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:

- ① Process must name each other explicitly

$\text{send}(P, \text{message}) \rightarrow \text{send message}$
to process P

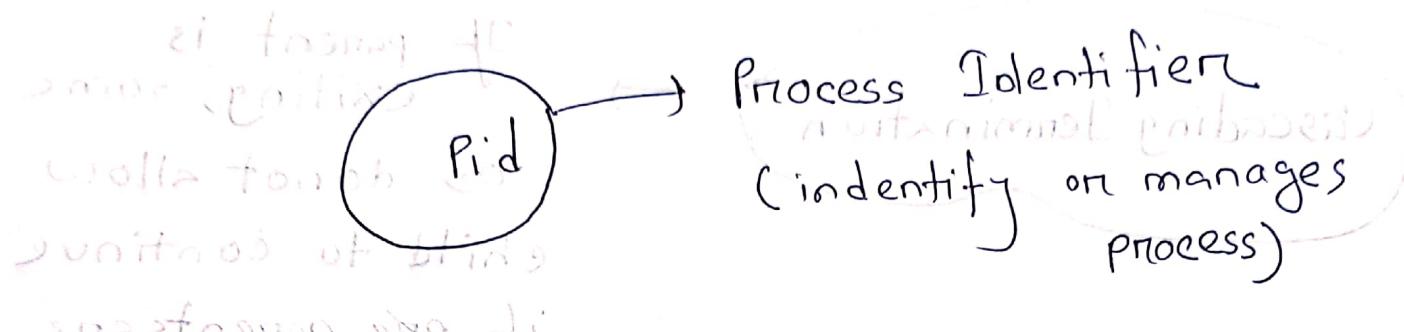
$\text{receive}(Q, \text{message}) \rightarrow \text{receive message}$
from Q process

communication link property

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

Process Creation

Parent process creates child process which in turn creates other processes forming a tree of processes.



Resource sharing

① Parent & child

share all resources

② Parent & child

share no resource

③ Child share subset

of parent resources

Execution sharing

① Parent and child

execute concurrently

② parent wait until
child terminate

③ parent finish +
child finish

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 all 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 it is allocated in CPU
it stays in CPU
until it is terminated by
CPU or switch to waiting state

(2) Preemptive :-

allows removal of process from CPU

What is short term scheduler?

→ ~~what is~~ is a scheduler which selects among processes in the ready queue and allocates the CPU to one of them.

In preemptive 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

downgrade priority

no access to shared data

preemption while in kernel mode

interrupts

occurs during crucial OS activities

emit to forums

emit hardware interrupt

(22 wrong answers)

What is dispatcher?

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

This involves activities are

context switching → switch to user mode

jumping in proper location in the user to restart execution

switching

What are scheduling criteria?

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

max ② Throughput (No. of processes 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

First come first served (FCFS) scheduling

FCFS scheduling

→ easy to implement

execution time

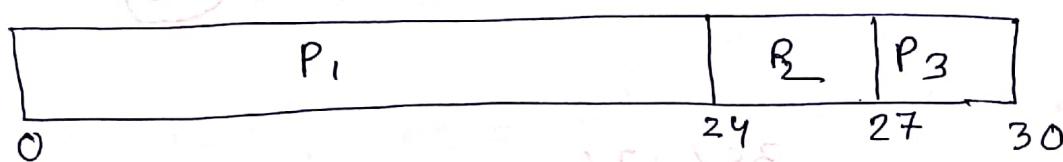
what is burst time

→ 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

p_i comes first
it will access CPU first
∴ arriving process order: P₁, P₂, P₃

The Giant chant for scheduling

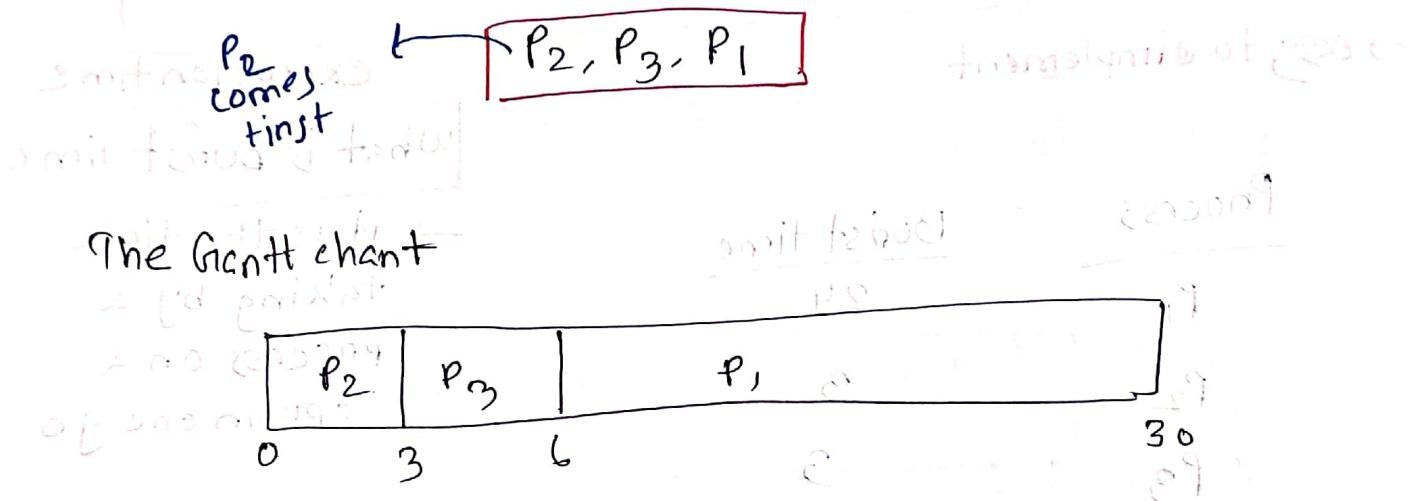


gianted if burst

$$\therefore \text{waiting time} ; \quad P_1 = 0s \\ P_2 = 24s \\ P_3 = 27s$$

$$\therefore \text{average waiting time} = \frac{0+24+27}{3} \\ = 17s$$

Now if we change the process order



$$\therefore \text{waiting time} = P_1 = 6s \\ P_2 = 0 \\ P_3 = 3$$

$$\therefore \text{average waiting time} = \frac{0+3+6}{3} \\ = 3s$$

$$3s < 17s$$

this case is better

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

\therefore convoy effect :- short process first,
long process behind.

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

Shortest Job first (SJF) scheduling

Upnos

most optimal
better than FCFS

cons

difficult
to know
the length
of

next CPU request

Process	burst time
P ₁	6
P ₂	8
P ₃	7
P ₄	9

non-preemptive
Scheduling

SJF scheduling chart

P ₄	P ₁	P ₃	P ₂
0	3	9	16

3 sec, 4.16 sec, 5.5 sec, 7.9 sec

P₄, P₁, P₃, P₂

$$\therefore \text{avg. waiting time} = 0 + 3 + 5 + 16 / 4 = 7.5$$

shortest remaining time first

Process	Arrival	Completion time	Burst time
P ₁	0	4	4
P ₂	2	6	5
P ₃	3	12	5
P ₄	5	10	3

Preemptive scheduling

Hence P₁ comes first (as arrival time)

after 1s P₂ comes. In that time P₁ has done 1s time. Now P₁ burst time (8-1) 7 > P₂ burst time. 4

short burst time process first

∴ P₂ will context switch P₁.

∴ P₁ have to wait until P₂ done

$$ef - rf + et + os = \text{unit turnaround}$$

after 2^s P_3 comes
 P_3 burst time (9) $>$ P_1 burst time $\rightarrow P_2$ burst time
 $\rightarrow P_1$ scheduling $(4-1)/3$
 (7)

$\therefore P_2$ will continue. P_1 have to wait still. then P_3 at P_3 will start c2 high last burst time

after $\frac{3s}{as}$ P_4 comes; P_3 burst time $> P_1$ burst time \Rightarrow P_4 burst time
 $(optimal) \Rightarrow P_2$ burst time
 $(3-1)/2$

\therefore now P_2 will still continue. 29

: Now P_2 will still continue
 \therefore Then P_4 will start at $(P_4 - L P_1)$ ~~to end at time~~

\therefore after P_4 , remaining P_1 start with ~~(coincide)~~

at last p₃ starts (highburst time)
compared with a following regeneration - if so (final)

$$\begin{array}{r}
 P_1 \quad P_2 \quad P_3 \\
 (10-1) + (1-1) + (17-2) \\
 \hline
 P_4 \quad (0) + (5-3) \\
 \hline
 4
 \end{array}$$

$$89 \text{ mslh} - 111.2 \text{ ft. } = 6.58$$

less waiting time
less complex

Priority scheduling

PS

→ Priority numbers are associated with each process (Integer)

\rightarrow smallest integer (highest priority)

Problem of PS

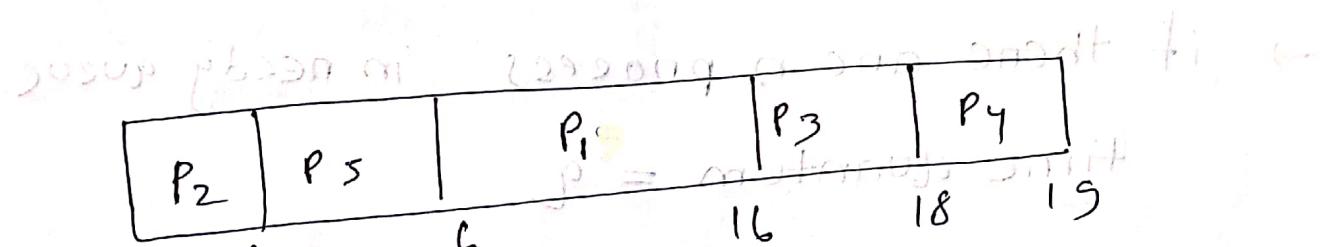
con^s
(starvation)

→ low priority problem may never execu

(caging) Sol :- increase priority as time progresses
of process

<u>Process</u>	<u>Burst time</u>	<u>Priority</u>	<u>Given</u>
P ₁	10	3	
P ₂	1	1	
P ₃	2	2	Using
P ₄	1	4	Non
P ₅	5	5	Preemptive
		1	(No context
		5	switch,
		2	Wait until
			a process
			finishes)

Grant chart



Priority goes to P1 & P3 because P1 has lowest priority

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

$$= 8.2$$

Time taken for 21 units execution

$\lceil \frac{x}{m} \rceil$ result

Waiting time for each process = $\frac{\text{burst time}}{\text{processes}}$

Round Robin | RR

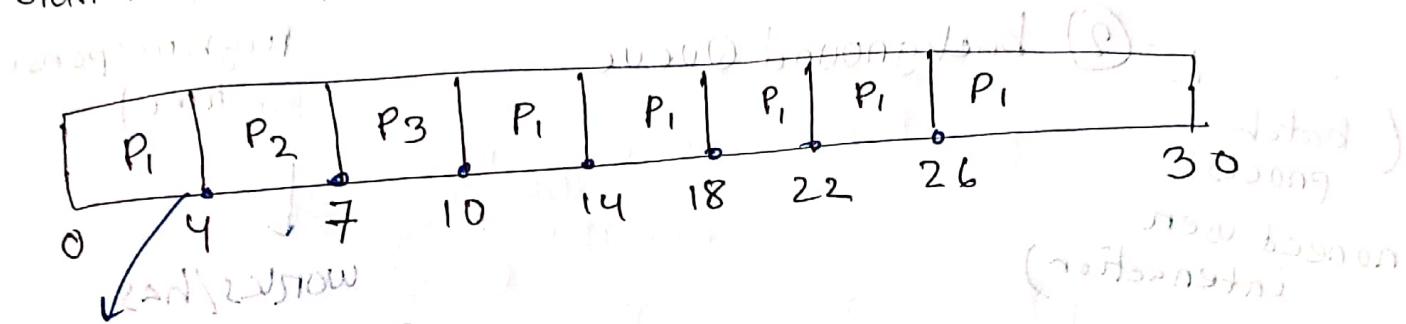
- Each process gets a small unit of CPU with time (time quantum, q) (10-100 milliseconds)
- After this time passed, process will stop and moved to end of ready queue / added
- if there are n processes in ready queue.
 - ∴ time quantum = q
 - ∴ each process gets $\frac{1}{n}$ of CPU time in chunk at most q times
 - ∴ each process gets q unit of CPU time at once
 - ∴ waiting time is not more than $\lceil (n-1) \times q \rceil$

cons:- spend more time context switching

<u>Process</u>	<u>Burst time</u>	<u>Time quantum</u>
P ₁	24	q=4
P ₂	3	Hence, 11
P ₃	3	

Pneemptive

Gantt chart is:



Although higher avg. time (30)
more context switches
but faster response time.

∴ a should be $>$ context switch time.
(but not too large)

if it becomes FCFS

18 tiny

Multi level Queue

Ready Queue is partitioned into two queue

① foreground Queue (interactive process which have time)

② background Queue. (batch process, none need user interaction)

(batch process
none need user interaction)

(FCFS algorithm)

Round Robin algorithm

(better response time)

scheduling between two queues

① Fixed priority scheduling

② Time Slice

80% to foreground in RR

20% to background in FCFS

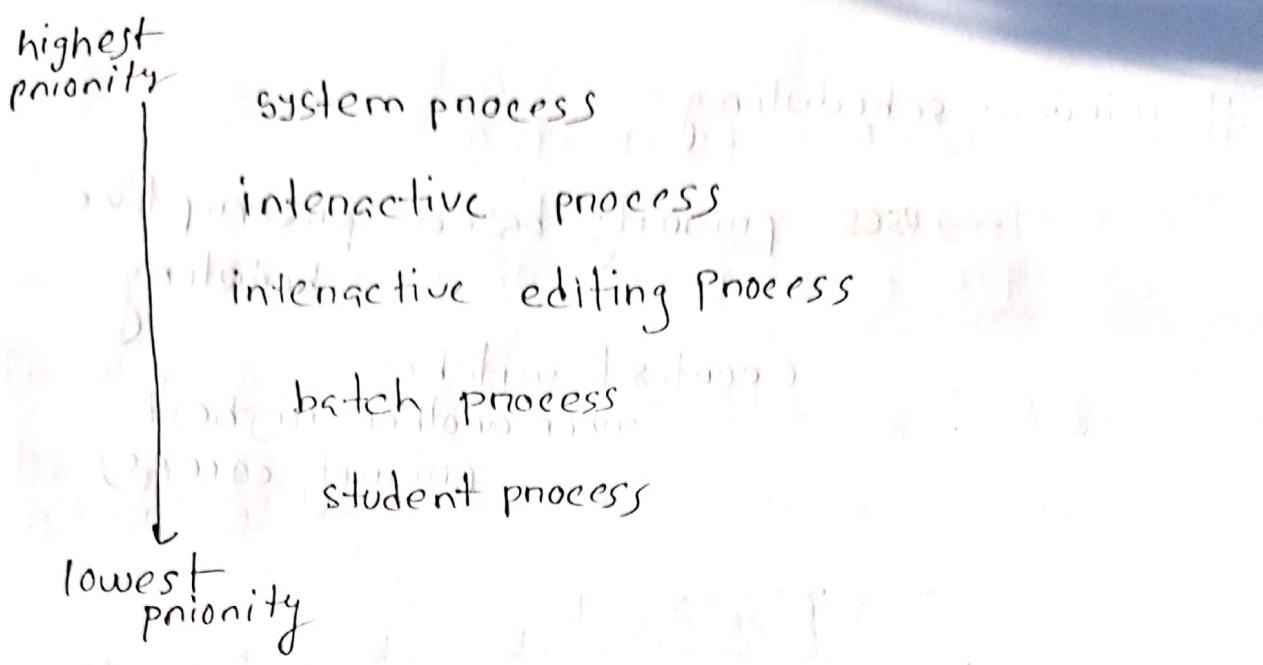
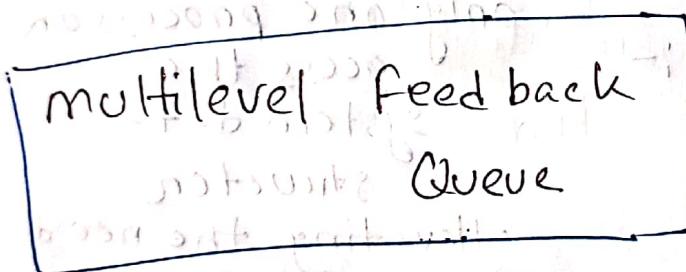


Fig:- Multilevel Queue



→ three Queue.

① Q_0 — RR with time quantum = 8

② Q_1 — RR if situation = 16

③ Q_2 — FCFS

window scheduling

→ uses priority based preemptive scheduling

(context switches when another highest priority comes)

multiple Processor Scheduling

- ① Asymetric multiprocesing :-
 - only one processor access the window based system data
 - would structure alleviating the need of kernel data sharing
- ② Segmentic :- each processor is self scheduling
 - each has its own private area of memory

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 ready state.
- it runs a task with higher priority before another lower priority task, even if the priority task is still in state critical state.
- better CPU utilization
- more less waiting at
- 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, reduction of state transitions
 - more personnel waiting time, patients

\rightarrow SJF; FCFS

- CPU is allocated to process until it terminates or switches to waiting state

In preemptive scheduling, CPU is allocated

to process for short

amount of time]

what is cpu scheduling

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