Processor – Intel(R) Core(TM) i5-8150U @ 2.20GHz
RAM – 4.00 GB
Windows Version – 22H2

15.7.24

OPERATING SYSTEM

Q. find configuration of system (OS, Processor, RAM, Harddisk)

Assignment-1
find all the versions of Windows. Do compasison between them

Q. Revise booting process (Warm & Cold Booting)

Operating System

Primary Job
↓
Interfacing

Secondary Job
↓
To increase system's efficiency
- → CPU Management / Process Management
- → Memory Management (RAM)
- → Disk Management
- → File Management
- → Deadlock Handling
- → Process Synchronisation
- → I/O Management
- → Security & Protection

Concept of having a feel of free main memory ⇐ Virtual Memory Management

Four types of Memory – Register
Cache
Done by O.S. { Primary
Secondary

Speed ↓

Embedded System

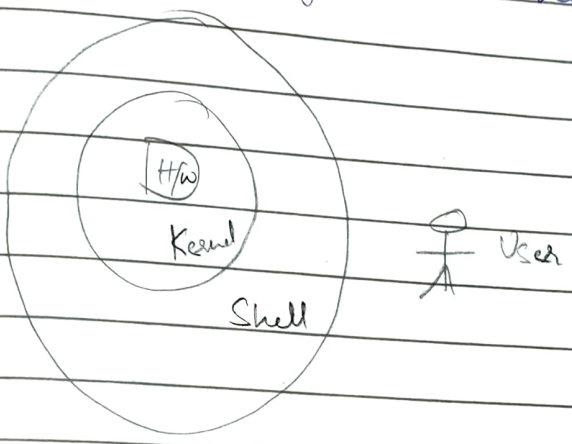
Ex - Automatic Washing Machine

All embedded systems are real time, but all real time systems are not embedded

↳ Ex - Smoke detector, Launching missile

Generally embedded systems are small task based system

IoT → embedded system + networking



Kernel ↑ then OS performance ↑

Kernel interact with H/w and shell interact with user

Shell ↑ then comfortable OS ↑

H/w → Kernel → Shell → User

Study all the basics

(1) Process Management

Which process will have CPU for how much time is process management

CPU Utilization : Idle time ↓ Utilization ↑

Also called CPU management / Short Term Scheduling

If task is in main memory then there is short term scheduling but loading task from secondary to main is long term scheduling.

Job Scheduling

It is called short term because validity of scheduling is very less, so we have to reschedule tasks frequently

Shedular → S/w in OS which performs

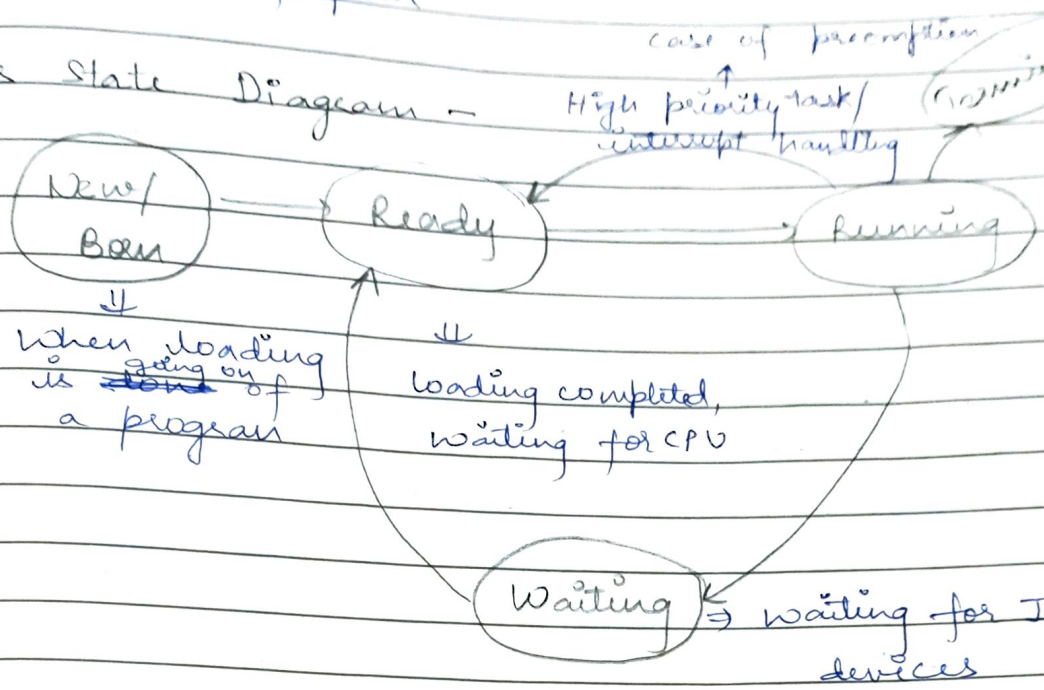
- long Term scheduling
- Short Term
- Middle Term ⇒ Used when we swap from main memory to buffer area

main memory ⇌ buffer memory

Dispatcher → which process is going to execute next is done by shedular, dispatcher dispatch task from shedular to ~~main memory~~ CPU

Time when program is in main memory is process lifespan

Process State Diagram -



Selection criteria is applicable only on ready state b/c process is making queue

We can have multiple waiting queues b/c we have multiple I/O devices.

Pre-emption → When a process is going on who needs resources but a high priority tasks occurs have resources are taken away from that process is pre-emption task.
Pre-emption is not aborting / killing of task.

PCB → Process control block is memory detail about process

Date
Page

PCB fields are - Process id
- Process State
- Process priority / Nice number
- Memory info
- CPU info
- Access info
- Other

24.7.24 Context ~~Contact~~ Switching - As soon as CPU switch to next task, the complete context of previous task will be saved.
- During the switching of CPU from a executed ~~a~~ process to another process the ~~pa~~ context of previous process gets saved
- Context switching ∝ Overhead time
- We'll consider context switching time as null.

* System Performance Measures are the parameters used to judge system's performance
As - Throughput (2)
- Processing time
- Cost efficiency
- CPU Utilization (1)
- Process Waiting Time (3)
- Process Response Time (4)
- Turn Around Time (5)

→ Throughput is the no. of ~~times~~ tasks completed in one unit of time by the processor

Throughput ∝ Utilization

→ Waiting Time, Response Time, Turn Around Time are used to check performance of system

Waiting Time	Response Time	Turn Around Time
How much time our process is waiting for CPU after getting loaded in main memory (It is in ready state)	After submitting task, how much time will it require by CPU to get a response	Time gap b/w submission of task and completion of task including RT and WT
Min. WT ⇒ Max. Utilization	Min. RT ⇒ Max. Utilization	Min TAT = Max. Ut

CPU Scheduling

When any process is in ready state we perform scheduling

New → Ready
Waiting → ~~Ready~~ Ready
After completion of I/O Task
Running → Waiting
If there is a preemption
Task termination

} Cases wh we need to perfor schedul

→ To select the next task during scheduling we will use preemptive algorithm

Scheduling Algorithms

① FCFS (First Come First Serve)

→ It is always non-preemptive approach

Q

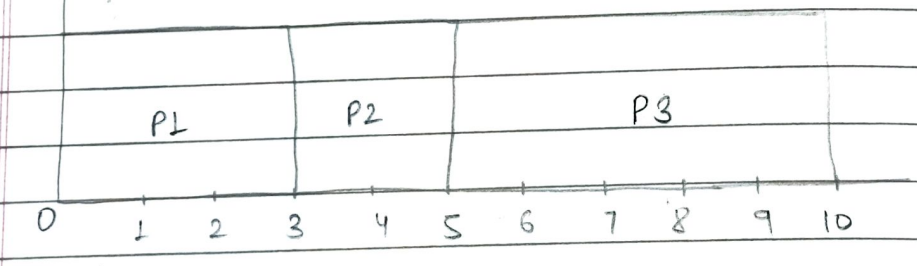
Process	Execution Time (Next required CPU time)
P1	3
P2	2
P3	5

⇒

Waiting Time
0 (0-0)
3 (3-0)
5 (10-5)

Create a schedule

GANTT Charts



Total Time = 8 sec

Avg. Waiting Time = $\frac{8}{3} = 2.\overline{6}$

Q

~~Execution~~ Arrival Time	Waiting Time
0	0-0 = 0
1	3-1 = 2
2	5-2 = 3

Avg WT = $5/3$

Q

	Arrival Time
P1	1
P2	2
P3	0

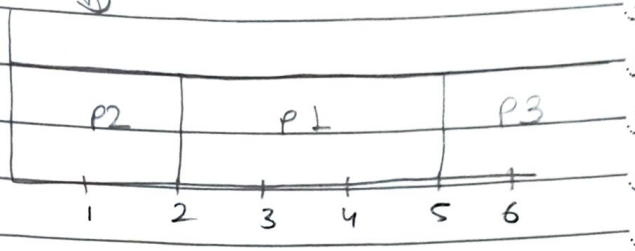
Waiting Time
5-1 = 4
8-2 = 6
0-0 = 0

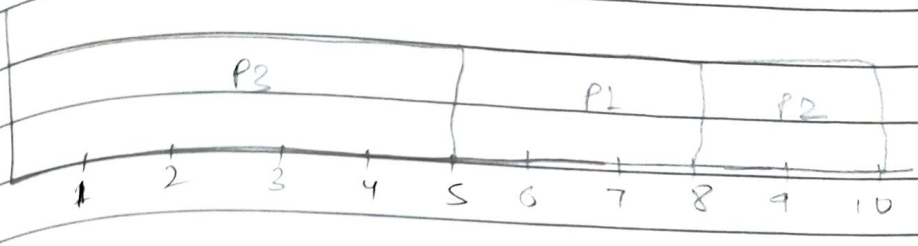
Avg WT = $\frac{10}{3}$

	Arrival Time	Waiting Time
P1	1	2 − 1 = 1
P2	0	0 − 0 = 0
P3	2	5 − 2 = 3

3+0+1 = 4

Avg W.T = $\frac{4}{3}$

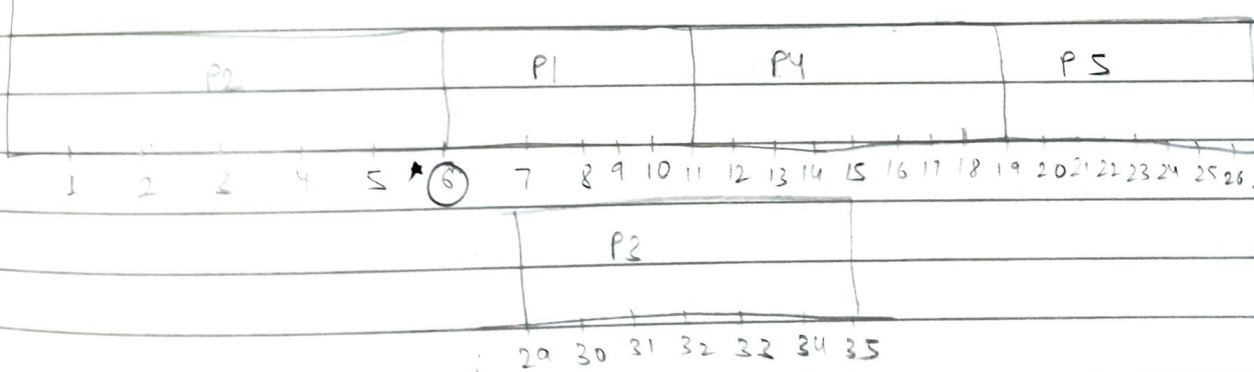




→ FCFS is starvation free algorithm
→ FCFS avg. wt fluctuates as per the arrangement of processes

P_i	A_i	E_i	Waiting Time
P1	1	5	★ (6) − 1 = 5 (− Arrival Time)
P2	0	6	0 − 0 = 0
P3	7	7	28 − 7 = 21
P4	2	8	10 − 2 = 8
P5	5	9	19 − 5 = 14

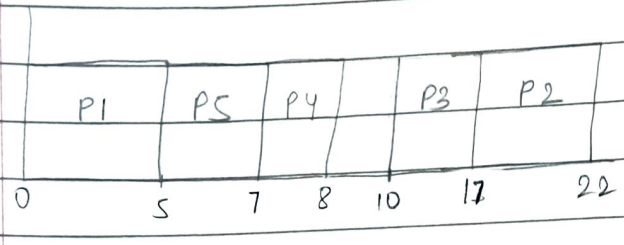
5+0+21+9+14 = 49/5



Date
Page

Q3

P_i	A_i	E_i	Waiting Time
P1	0	5	0-0 = 0
P2	15	5	17-15 = 2
P3	10	7	10-10 = 0
P4	5	1	7-5 = 2
P5	2	2	5-2 = 3



W. Time = 0+2+0+2+3 = 7

Avg WT = $\frac{7}{5}$

(2) SJF (Shortest Job first)

Pre-emptive Non-Preemptive

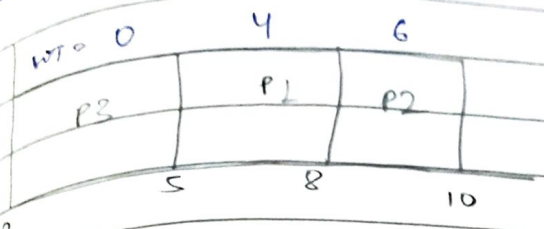
Shortest job is executed first; we prioritize tasks according to the execution time

Non-Preemptive

Q1

P_i	A_i	E_i
P1	1	3
P2	2	2
P3	0	5

from FCFS

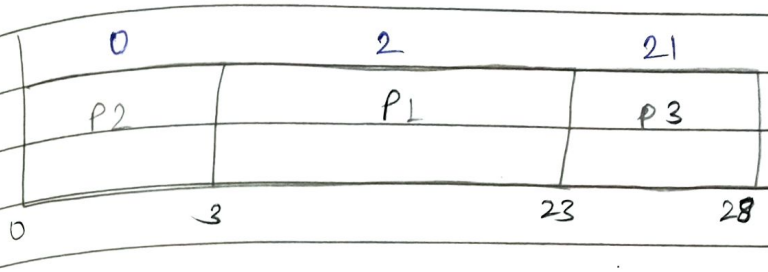


= 10 ; AWT = $\frac{10}{3}$

~~for SJF~~

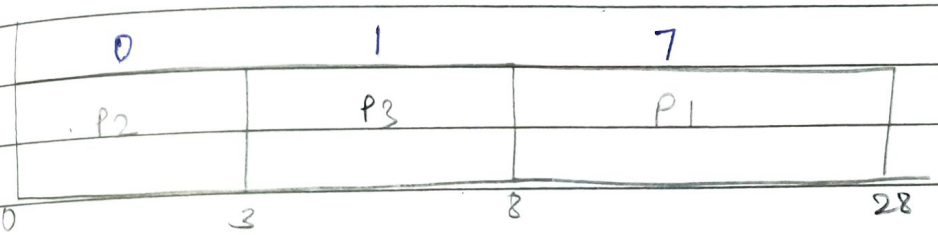
Pi	Ai	Ei
P1	1	20
P2	0	3
P3	2	5

From FCFS



= $\frac{28}{3}$

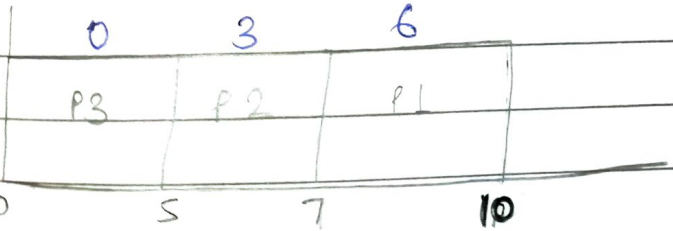
from SJF (best approach)



= $\frac{8}{3}$

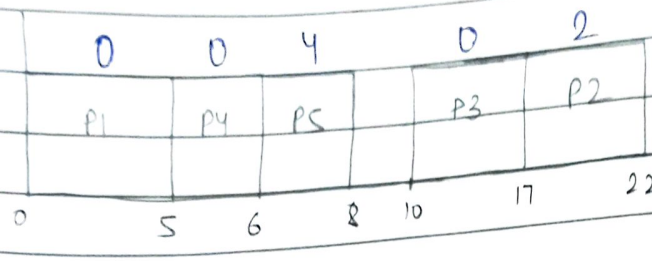
Always less waiting time

~~Ans~~ from SJF



= $\frac{9}{3}$

Ans-3 Using SJF

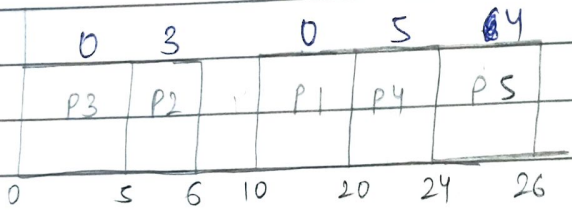


$= \frac{6}{5}$

28.7.24

Pi	Ai	Ei	Waiting Time
P1	10	10	10-10 = 0
P2	2	1	5-2 = 3
P3	0	5	0-0 = 0
P4	15	4	20-15 = 5
P5	20	2	24-20 = 4

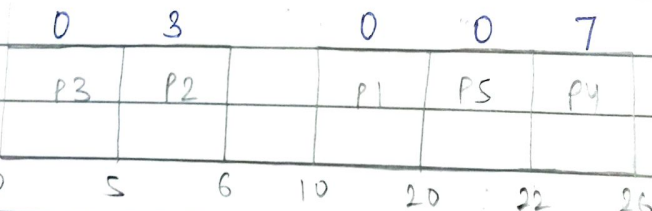
Using FCFS,



3 + 5 + 4 = 12

Avg WT = $\frac{12}{5}$

✓ Using SJF (Non-Preemptive)



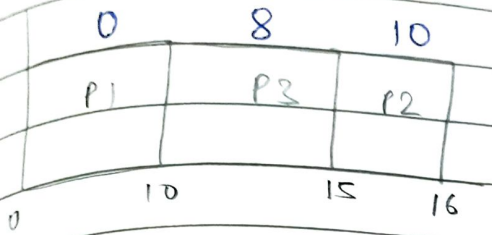
3 + 7 = 10

Avg WT = $\frac{10}{5}$ = 2

Pre-emptive SJF / Shortest Remaining Time First

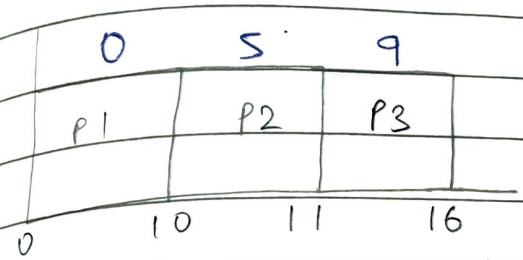
Pi	Ai	Ex:
P1	0	10
P2	5	1
P3	2	5

Using FCFS



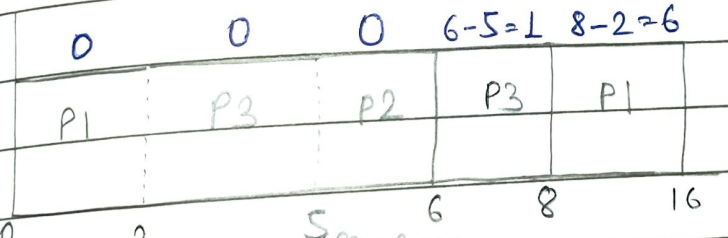
$= \frac{18}{3} = 6$

Using NP-SJF



$= \frac{14}{3}$

Using P-SJF



Waiting Time

P1 → 6

P2 → 0

P3 → 1

TWT = 6+1 = 7

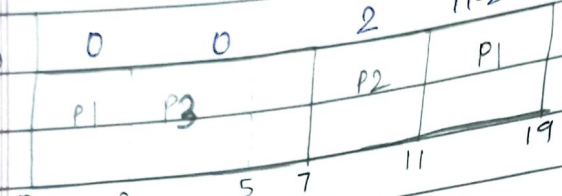
Avg = $\frac{7}{3}$

Q.

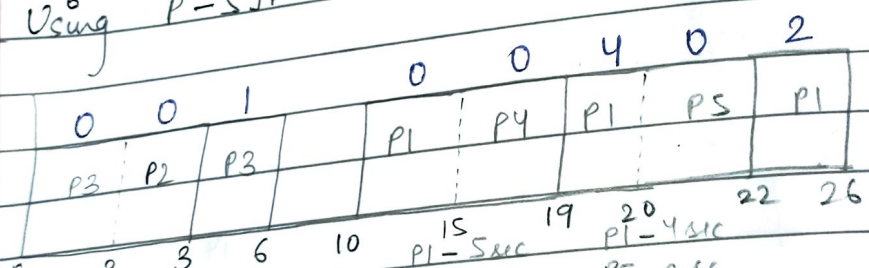
Pi	Ai	Ei
P1	0	10
P2	5	4
P3	2	5

Avg WT = 11/3

P-SJF →



Ans-4 Using P-SJF



P1 → 0+4+2 = 6

P2 → 0

P3 → 1+0 = 1

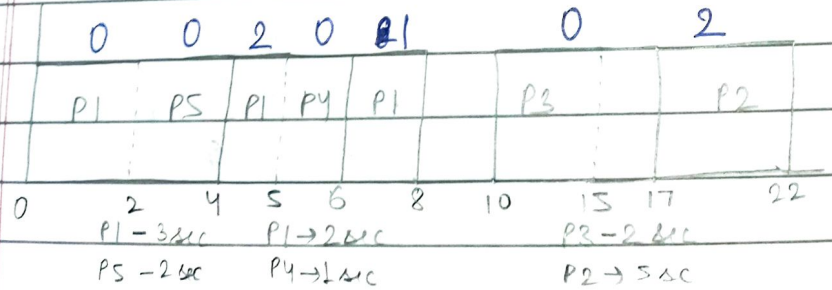
P4 → 0

P5 → 0

WT = 7

A.WT = 7/5

Ans-3 Using P-SJF



= 5/5

= 1

- SJF algorithm is the most appropriate/optimal algorithm for both preemptive and non-preemptive case.
- More complex than FCFS
- Starvation (It is with every algorithm)
  1) We apply starvation handling algo along with SJF
- The practical implementation of this algorithm is not possible for CPU scheduling, only theoretical is possible b/c for applying this we need next CPU burst time of any task before executing the task which is not possible
  Before execution of task we need execution time which aint possible (It is based on futuristic data)

SJF is used to set standard for other algorithms

2) Priority Based Scheduling

Preemptive Non-Preemptive

→ Only applicable where priority values are given
→ Priority are preference number of every process written in PCB which tells abo importance of tasks

	Pi	Ai	Ei	Pr
	P1	0	10	1
	P2	5	4	3
	P3	2	5	2

Use any one convention

Numbering Convention → Higher no. Higher priority

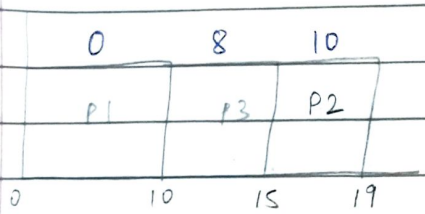
Ranking Convension → lower no. Higher priority

= most common approach

(Use this mostly)

Non-Preemptive

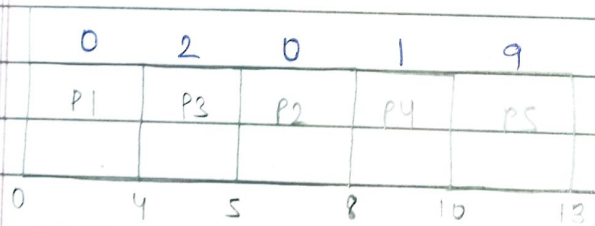
Sol-1



$\frac{18}{3} = 6$

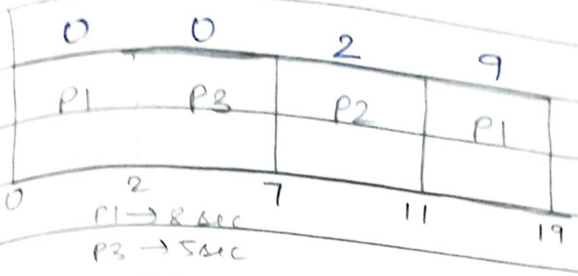
Q2

Pi	Ai	Ei	Pr.
P1	0	4	2
P2	5	3	1
P3	2	1	0
P4	7	2	1
P5	1	3	5

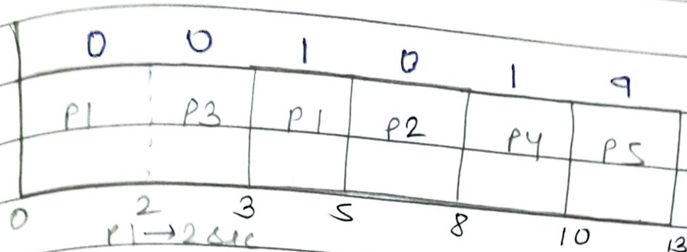


$= \frac{12}{5}$

Pre-emptive



= 11/3



= 11/5

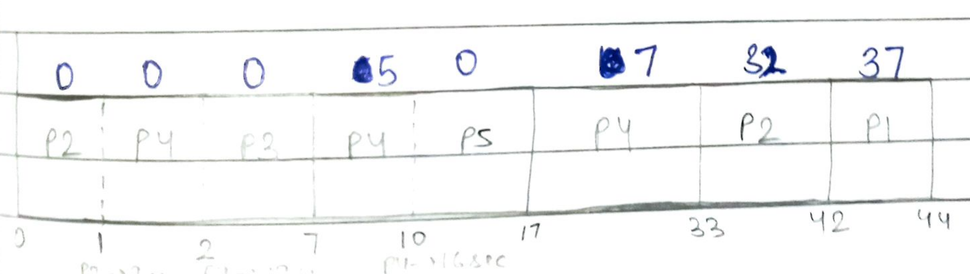
- A → It is best for application ~~priority~~ based system (It is only algo. used)
- → Implementation is possible

- D → Applicable only in specific cases
- → User Requirement needs to be there
- → High complexity
- → Starvation

Q3

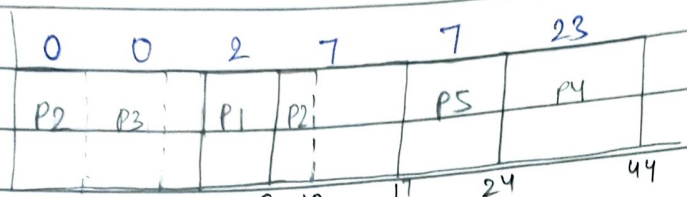
Pi	Ai	Ei	Pr
P1	5	2	7
P2	0	10	5
P3	2	5	2 ✓
P4	1	20	4 ✓
P5	10	7	1 ✓

81/5



Date
Page

Using P-SJF,



39/5

P2-8sec P3-2sec P2-7sec
P3-5sec P1-2sec P5-7sec

31.7.24

① CPU Utilization Time = 100%
= CPU Working Time / Total Time x100

② Throughput - No. of tasks per unit time

③ Response Time - Initial Waiting Time when task arrived and when its execution started.

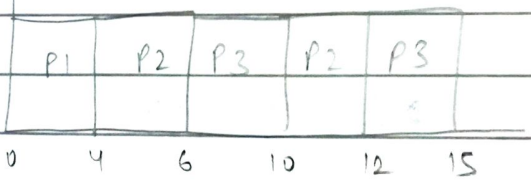
execution start - arrival

④ Turn Around Time

Task completion time - task arrival time

Ex

Pi	Ai	Response	Turn Around
P1	0	0-0=0	4-0=4
P2	1	4-1=3	12-1=11
P3	2	6-2=4	15-2=13



RR Scheduling (Round Robin)

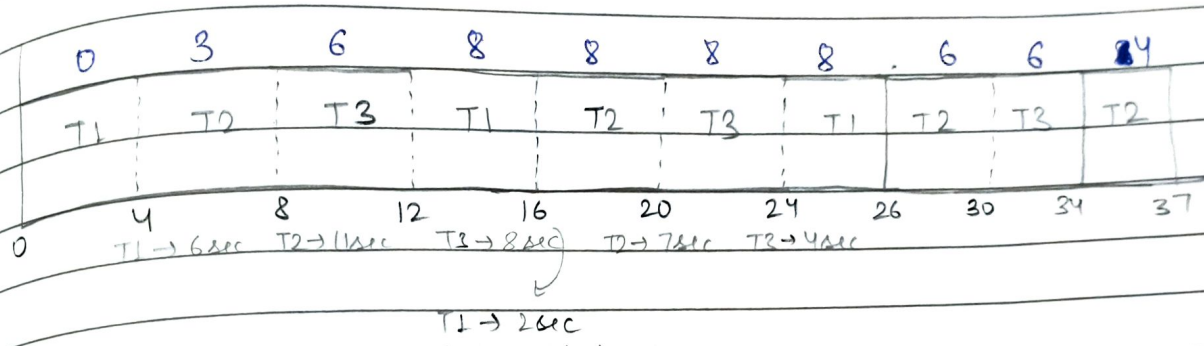
OR

(Rajasthan Royals)

- If priorities are not given then we use RR Scheduling

T_i	A_i	E_i	P_i
T1	0	10	3
T2	1	15	2
T3	2	12	1

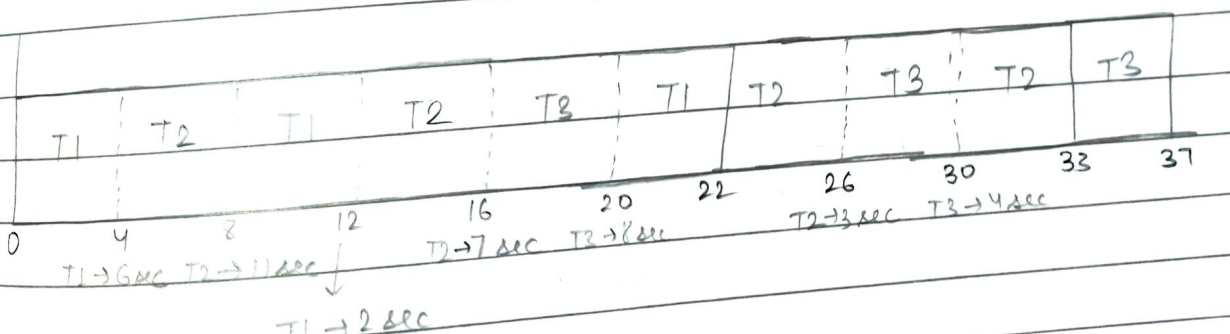
Time Slice = 4 sec



$$WT = 3 + 6 + 8 + 8 + 8 + 8 + 6 + 6 + 4$$

T_i	A_i	E_i	P_i
T1	0	10	3
T2	1	15	2
T3	10	12	1

T.S = 4 sec

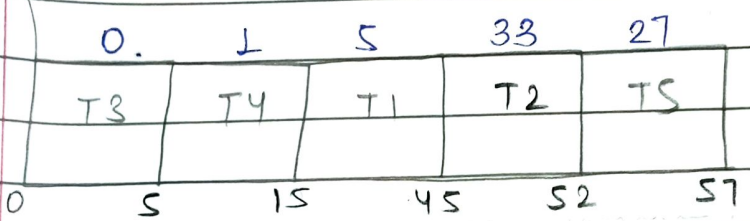


Q. Apply FCFS, NP-SJF, P-SJF / SRJF, P-Pr.B, NP-Pr.B, RR

T_i	A_i	E_i	P_i
T1	10	30	2
T2	12	7	3
T3	0	5	1
T4	4	10	0
T5	25	5	4

T.S = 4 sec

(1) FCFS

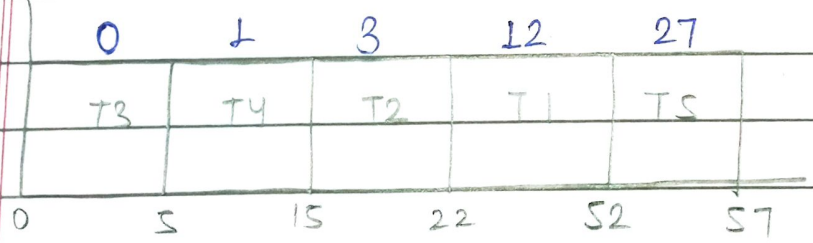


$WT = 1 + 5 + 33 + 27$

$= 66$

$Avg.\ WT = \frac{66}{5}$

(2) NP - SJF

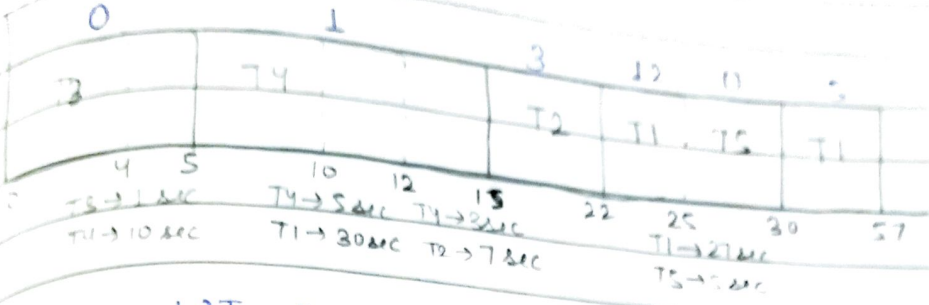


$WT = 1 + 3 + 12 + 27$

$= 43$

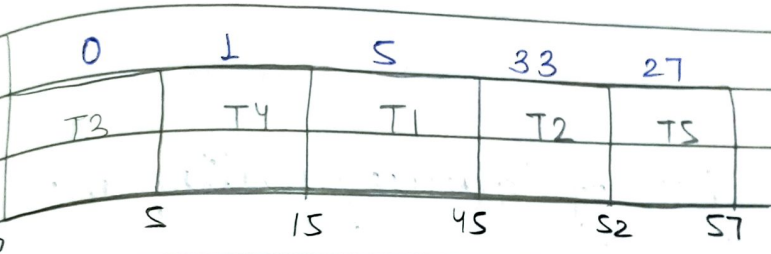
$Avg.\ WT = \frac{43}{5}$

SJF



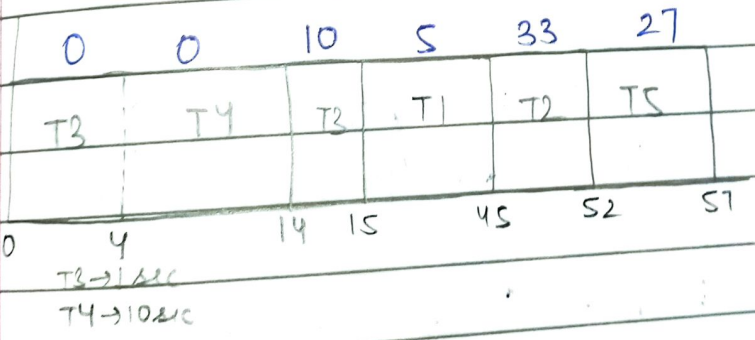
$$WT = 1 + 3 + 12 + 5$$
$$= 21 \Rightarrow \frac{21}{5}$$

③ NP – Pr · B



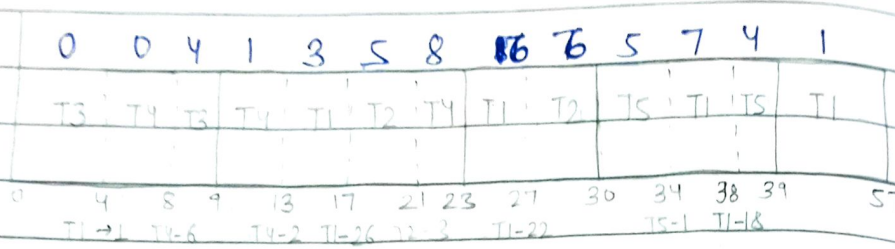
$$WT = 1 + 5 + 33 + 27$$
$$= \frac{66}{5}$$

P – Pr · B



$$WT = 10 + 5 + 33 + 27$$
$$= \frac{75}{5} \quad 15 = 15$$

(4) RR



Waiting Time = 4+1+3+5+8+6+6+5+7+4+1

= 50/5

= 10

4 8 24 For n - numbers of queues / Multiqueue

R_1 T1 T2 T3 —— Algo (1)

R_2 T4 T5 —— Algo (2)

R_3 T_6 T_7 T_8 —— Algo (3)

We need n+1 no. of algorithms

for deciding which queue to

→ After all Tasks all R_1 gets completed, then only will shift to queue R_2

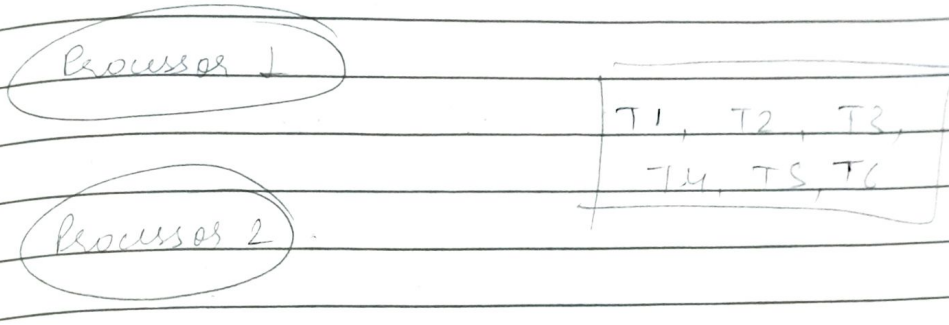
Multiqueue feedback - Under certain circumstances we can shift tasks among queue which is not possible in simple multiqueue

Multilevel feedback queue decreases the chances of task starvation by upgradation of queue.

Multiprocess Scheduling

Two methods - Common Queue Implementation
- Distinct Queue Implementation

Common Queue



Whoever is free from the two processor will handle the tasks

(A) → No selection criteria

~~Distinct Queue~~ (D) → Messy approach, lengthy task will get starved mostly

(D) → Management is tuff

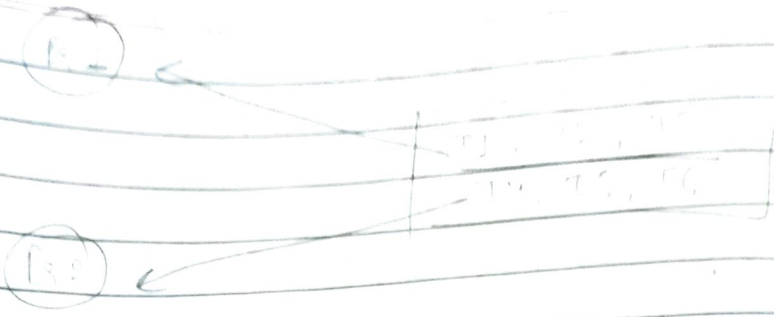
(A) → Faster b/c no initial seggregation

(A) → Better CPU utilization

Distinct Queue

(D) → Processor utilization may not be very good, load balancing (work division) is not good

→ In this particular, tasks are assigned to both the processors



Multiprocessor Arrangement

Symmetric

All the processors are treated equally, they are at same level

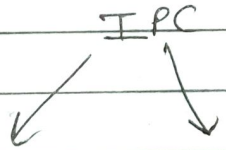
Asymmetric / Master-slave

One master processor will handle the other processors. Task is assigned to master processor then it will divide tasks among other processors.

Process Synchronisation

→ Whenever multiple processes are comm. with each other it is known as Interprocess Communication (IPC)

→ If there is some data, one process needs to notify other one before changing it.



Shared Memory

All changes made by a processor are implicitly reflected to other processors

Message Passing

Working is done in local processors are distributed memory then notify

(E) Shared memory is faster b/c we need to notify it

(A) → IPC has lower complexity
→ Depending upon situation & application we can choose any one of them

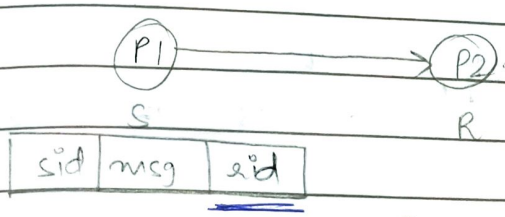
(F) → In shared memory security, deadlock, & concurrency control issues will be there

(P) → Anomalies will be there and race condition occurs

(P) → Shared memory can be executed in a tightly coupled system only

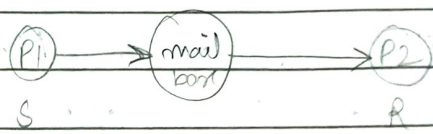
5.8.24 Message Passing Methods

(1) Direct Communication



If we specify this then it is direct comm. / naming convention

Indirect Communicat



P2's id is unknow to P1. P2 is s to receive the m P2 has some mai box whose no. is shared with other processors

(2) Blocking / Synchronous
- Sender will send the msg and before sending another msg it will

Non-
Non-Blocking / Synch
- Sender will not for acknowledge from user.

Multiprocessor Arrangement

Symmetric

All the processors are treated equally, they are at same level

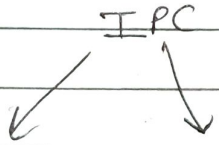
Asymmetric / Master-slave

One master processor will handle the other processors. Task is assigned to master processor then it will divide tasks among other processors.

Process Synchronisation

→ Whenever multiple processes are comm. with each other it is known as Interprocess Communication (IPC)

→ If there is some data, one process needs to notify other one before changing it.



Shared Memory

All changes made by a processor are implicitly reflected to other processors

Message Passing

Working is done in local processors are distributed memory then notify

(C) → Shared memory is faster b/c we need to notify it

(A) → It has lower complexity
→ Depending upon situation & application we can choose any one of them

(B) → In shared memory security, deadlock, & concurrency control issues will be there

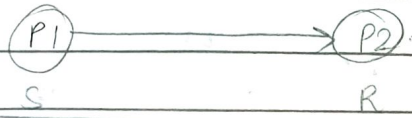
(D) → Anomalies will be there and race condition occurs

(E) → Shared memory can be executed in a tightly coupled system only

8-24
Qsns

Message Passing Methods

(1) Direct Communication



sid	msg	rid

If we specify this then it is direct comm. / naming convention

Indirect Communication

P2's id is unknown to P1. P2 is ready to receive the msg P2 has some mail box whose no. is shared with other processors

(2) Blocking / Synchronous
- Sender will send the msg and befor sending another msg it will

Non-Blocking / Synchronous
- Sender will not wait for acknowledgement from user.

will be acknowledged
when receiver sends

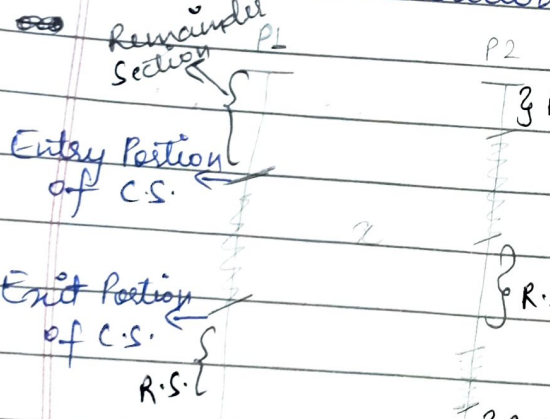
(3) Buffering Category

Zero	Finite	Infinite
Sender is sending data but receiver is processing 1st data only, it has no free time. All the other msgs will get lost, this is known as Buffer. Zero Buffer - we cannot store anything	Some finite storage area is there. Ex - mailbox it has some limit, after that limit messages will get lost	Infinite storage area is there.

4.8.24

Shared Memory Approach

→ There are always chances of race-around conditions.



In these portions we are accessing common variable
⇒ Critical Section

→ We don't want two process to run their critical section at same time & access same variable at same time

→ If this kind of problem occurs then it is known as process synchronisation issue / critical section problem
→ To solve this we have three cond" :-
- Mutual Exclusion
- Bounded Waiting
- Progress Cond"

If we satisfy all three of them our processes will work properly

Mutual Exclusion
→ Only one can happen at a particular time
→ Either head or tail ; both can't occur at same time
→ If some process is running its critical section then no other proce can run their critical section

Bounded Waiting
→ Waiting should be finite
→ Here waiting means wating for critical section to run
→ There should be finite / bounded no. of process that can run b/w the process so that there is no starvation
- P1 is executing and P2 requests then there should be finite no. of processes that can run between P1 & P2

Progress Cond^n

→ Whenever we consider any process to enter in its critical section, we apply some selection criteria and select only those process that are not running in their remainder section

→ Then we'll never have stuck cond^n

Algorithm to ensure critical section b/w two processes

Boolean T = 0;

P_0	P_1
while(T == 1); (Infinite loop (waiting till value of T changes))	while(T == 0);
C.S.	C.S.
T = 1;	T = 0;

Mutual Ex. ✓
Bound Value = 1 ✓
Progress Cond^n ✗

→ If Turn = 0 then P_0 will enter in its critical section.

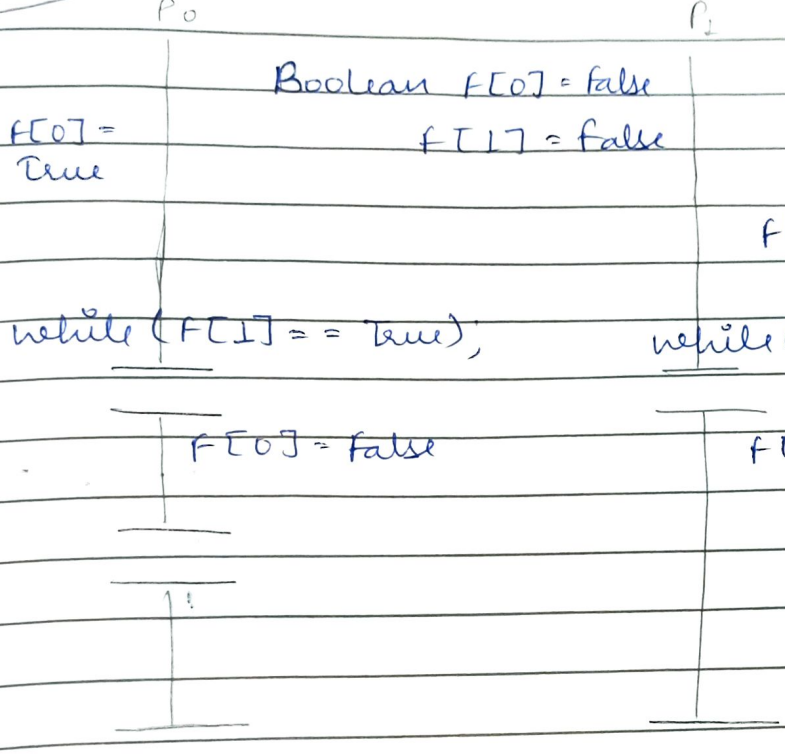
→ After P_0's C.S. ends we will toggle value of Turn = 1 so that P_1's C.S. can execute

→ Process has to check value of Turn before entering C.S.

→ If initial value of T = 0 then P_0 is also working in R.S. therefore progress

cond^n fails. The problem is if we have 5 C.S. in P_0 and 1 C.S. in P_1, after completing 1 1 C.S of each proces it will get stuck for next 4 C.S.

Algorithm 2

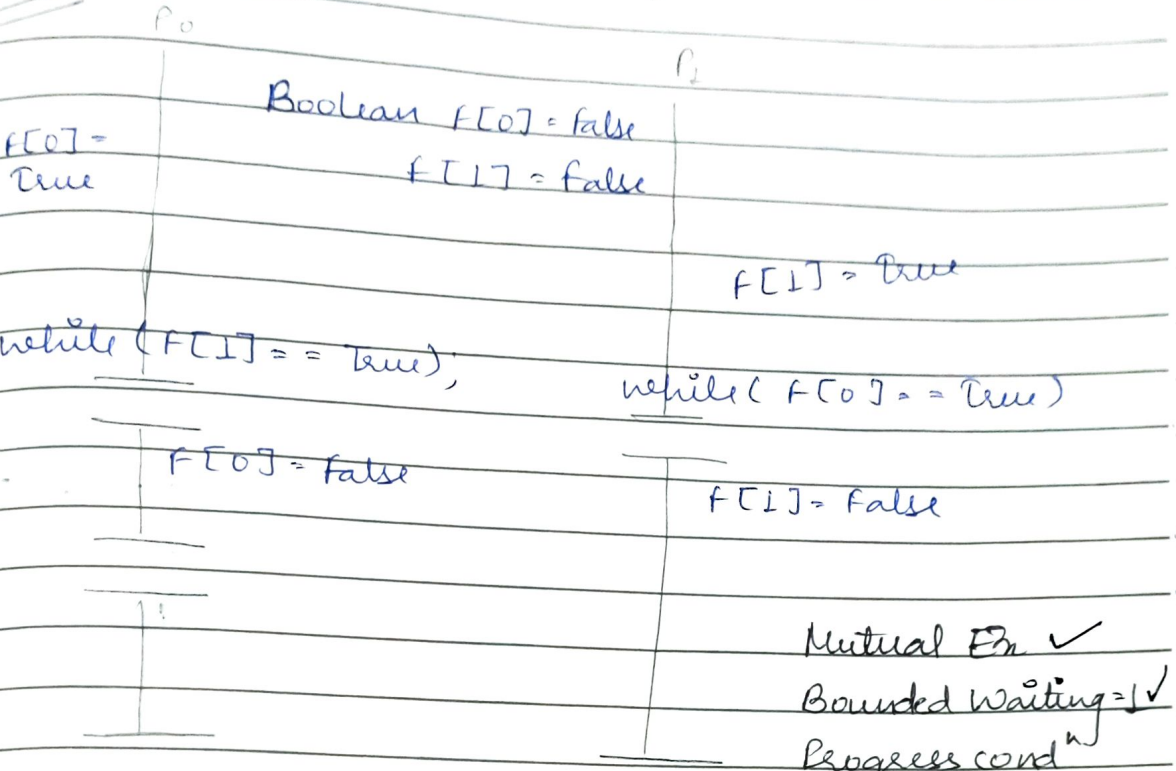


Mutual Ex ✓
Bounded Waitin
Progress cond^n
(It is sati
a/c to th
definition
in one case it
not satisfied

One cond^n –
If flag of both processes becomes tr
at the same time then infinite
waiting or deadlock is there.
∴ Progress cond^n ✗

cond^n fails. The problem is if we have 5 c.s in P_0 and 1 c.s in P_1, after completing 1 c.s of each process it will get stuck for next 4 c.s

Algorithm 2



Mutual Ex ✓
Bounded Waiting ✓
Progress cond^n
(It is satisfied a/c to the definition but in one case it is not satisfied)

One cond^n –
If flag of both processes becomes true at the same time then infinite waiting or deadlock is there.
∴ Progress cond^n ✗

Algorithm 3 (Peterson's Solution for two process critical sol^n problem)

Pi

Boolean F[0] = false
F[1] = false
T = 0;

P0

F[0] = true
T = 1
while (F[1] == true && T == 1);
F[0] = false;

Pi

F[1] = true;
T = 0;
while (F[0] == true && T == 0);
F[1] = false;

Mutual Ex ✓
Bound Value = 1 ✓
Progress cond^n ✓

This will allow other process to run if it is waiting

22.8.24

Bakery Algorithm / Lamport Algorithm

int token [n] = 0;
Boolean flag [n] = false;

Pi
flag [i] = true;
token [i] = 1 + max (token [0] ---- token [n-1]);
flag [i] = false;
for (j = 0 to n-1) do
while (token [j] < token [i] && token [j] != 0);

All the tokens given bef

while (flag[j] == true) {

{

If both cond^n are true then Pi will wait

token [i] = 0 ;

Tools which provide support to process synchronisation

- Hardware Synchronisation Tools
- Software Synchronisation Tools

Hardware Synchronisation

```
C = A + B ;
```

```
L    1, A
ADD  1, B
ST   1, C
```

→ In H/w code, a single line code is broken into multiple lines of code.
→ In this, if a high level code has single line code and during execution this single line code is converted multiple lines of code, we need apply process synchronisation tools order to avoid any synchronisation issue.

Tools - Swap f^n, increment f^n, decrement
Test and set f^n

H/w solⁿ is to provide some low level f^n to high level codes and run atomically
as a whole no interleaving

token [i] = 0 ;

Tools which provide support to process synchronisation

Hardware Synchronisation Tools

Software Synchronisation Tools

Hardware Synchronisation

C = A : B ;

```
L    1, A
ADD  1, B
ST   1, C
```

→ In H/w code, a single line code is broken into multiple lines of code.

→ In this, if a high level code has a single line code and during execution this single line code is converted into multiple lines of code, we need to apply process synchronisation tools in order to avoid any synchronisation issue.

Tools - Swap f^n, increment f^n, decrement f^n, Test and set f^n

H/w sol^n is to provide some low level f^n to high level codes and run them atomically

as a whole no interleaving

Date
Page

Software Synchronization

Semaphores

It is a kind of data type

semaphores s1 = 0;

- Counting (0, 1, ----, n)
- Binary (can contain only two values 0 or 1)

They always contain non negative value

All the semaphores variables can be accessed through two fn –

Wait (s1)

```
{
    while (s1 <= 0);
        s1--;
}
```

Signal (s1)

```
{
    s1++;
}
```

Use of semaphore to ensure mutual exclusion –

P1 — Semphore S=1; — P2

wait (s); ← locking a data item

wait (s);

C.S

signal (s); ← Unlocking a data item

signal (s);

P1 S = 0 P2

Wait (S);

Signal (S);

P2 will always run after P1. This will ensure sequencing

Drawbacks –

→ Deadlock can occur

P1 Semaphore S1=1, S2=1 P2

Wait (S1);
Wait (S2);
Wait (S2); They both will get stuck here
Wait (S1);

Signal (S1);
Signal (S2);

Signal (S2);
Signal (S1);

→ They implicitly don't handle starvation, we need to apply methods explicitly

→ Situation of busy waiting i.e. cpu will run continuously without any output (Situation of spin lock)

Three Problems

1. Finite Buffer / Bounded Buffer / Producer and Consumer Problem

In this, we use three common semaphores

i) full } Counting Semaphores
ii) empty }
iii) mutex } Binary Semaphores
used for mutual exclusion

Producer

Semaphore full = 0;
empty = n;
mutex = 1;

wait (empty);
wait (mutex);

signal (mutex);
signal (full);

Consumer

wait (full);
wait (mutex);

signal (mutex);
signal (empty);

(2) Reader – Writer Problem

Reading → sharing mode
writing → exclusive mode

At a time only one writer will work and along with writer reader can't work also multiple writers can't work together.
Readers can work along with another readers but they won't work with writers.

Writer

semaphore mutex1 = 1;

wait (mutex1);

signal (mutex1);

Reader

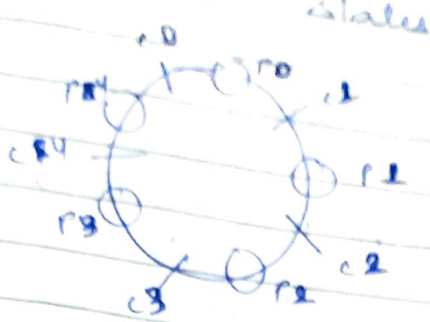
semaphore mutex2 = 1;
int rc = 0;

wait (mutex2);
rc++; if (rc == 1)
wait (mutex1);
signal (mutex2);

wait (mutex2);
rc--;
if (rc == 0)
signal (mutex1);
signal (mutex2);

3. Dining Philosophers Problem

There are two states with thinking



They use chopstics to eat, generally they are in thinking state but after sometim they feel hungry so they want to eat at any random times.

If P1 is in eating state, P2 & P5 also want to eat but P2 only has 1 chopstick so it will wait

Chopsticks - Resources

Person - Program

P0

Semaphore m[5]=1;

wait(m[i]);

wait(m[i+1] % 5);

signal(m[i]);

signal(m[i+1] % 5);

This situation can lead to deadlock which is a disadvantage of semaphore. When all programs pick ith chopstick, each will wait for (i+1)th chopstick infinitel

To handle this -

(1) Before acquiring any chopstick, check for availability of both the chopstick first. If available then only assign.

(2) Even - odd.
Even no. will pick right hand side chopstick first
Odd no. will pick LHS chopstick first

(3) If possible use an extra chopstick.
If not then reduce 1 process.