Saathi

Date / /

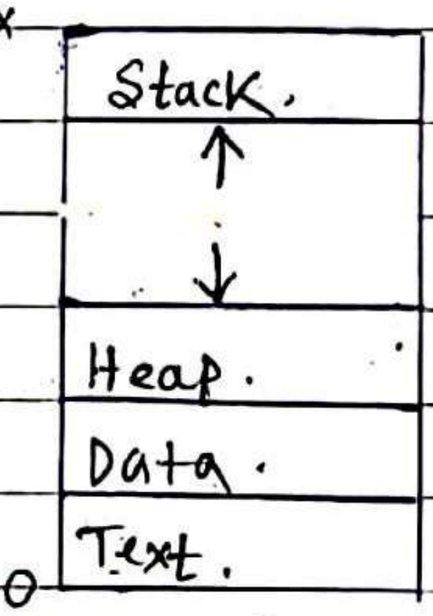
PROCESSES - (Process management)

RBR

• Process: A process can be defined in any of the following ways:
- A process is a program in execution.
- It is a asynchronous activity.
- It is the entity to which processors are assigned.
- It is dispatchable unit.
- It is the unit of work in a system.

A process is more than the program code. When we execute a program, it becomes a process which performs all the task mentioned in the program.

When a program is loaded into memory and it becomes a process, It can be divided into four sections - stack, heap, text and data. The following image shows a simplified layout of a process inside main memory -



(process in memory)

• Stack : The process stack contains the temporary data such as function parameters, return address, and local variables.

• Heap : This is dynamically allocated memory to process during its run time.
• Data : This section contains the global and static variable.
• Text : This includes the current activity represented by the value of program counter and the contents of the processo's register.
(Code section)

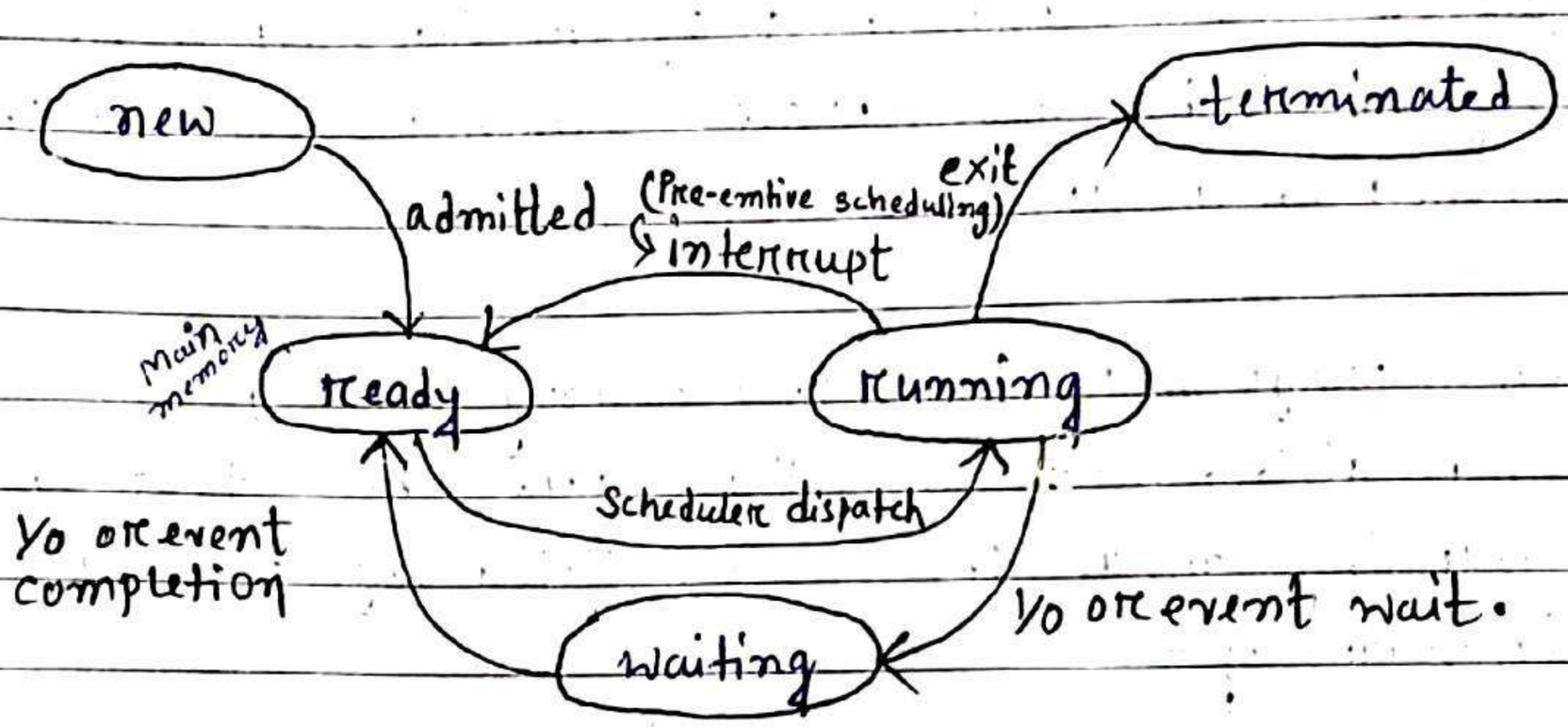
Saathi

Date ../../....

RBR

Process State Diagram:

When a process executes, it passes through different states. This stages may differ in different OS, and the name of there states are arbitrary.
A process can have one of the following five states at a time.



(Diagram of process state)

1. Start: The process is being created.
2. Running state: A process is said to be running if it has the CPU, that is, process actually using the cpu at that particulate instant.
3. Waiting (or Blocked) state: A process is said to be blocked if it is waiting for some event to happen such that as an I/O completion before it can proceed. Note that, a process unable to run until some external event happens.

4. Ready: A process is said to be ready if it use a cpu if one where available. A ready state process is runnable but temporarily stopped running to let another process run.
5. Terminated state: The process has finished execution.

Saathi

Date ___/___/___

RBR

Process Control Block (PCB):

A process in an OS is represented by a data structure known as process control BLOCK (PCB) or process despcriptor.

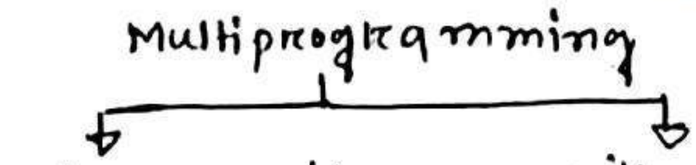
A PCB keeps all the information needed to keep track of a process.

The architecture of PCB is completely dependent on operating System and may control different information in different OS.

Process ID.
p-state
Pointer
Priority
Program counter
CPU register
I/O information
Accounting information
etc...

(PCB Diagram)

1. process State: The current state of process i.e, whether it is ready, running, waiting, or wheatever.
2. process privileges: This is required to allow/disallow access to system resources.
3. process ID: Unique identification for each of the process in os.
4. pointer: A pointer to parent process.
5. program Counter: PC is a pointer to the address of the next instruction to be executed for this process.
6. CPU registers: various CPU registers where process need to be Stored for execution for running state.
7. Accounting information: This includes the amount of CPU used for process execution, time limits, execution ID etc.
8. I/O Status information: This includes a list of I/O devices allocated to the process.
9. CPU-scheduling information: process priority and other scheduling Informatio



Date ___/___/_____

Saathi

which is required to schedule the process.

10. Memory management information: This includes the information of page table, memory limits, segment table depending on memory used by the ~~oper~~ OS.

RBR

Schedulers:

Schedulers are special system ~~@@~~ software which handle process scheduling in various way. Their main task is to select the jobs to be submitted into the system and to decide which process to run.

Schedulers are of three types -

1. long term scheduling.
2. short Term scheduling.
3. Medium Term scheduling.

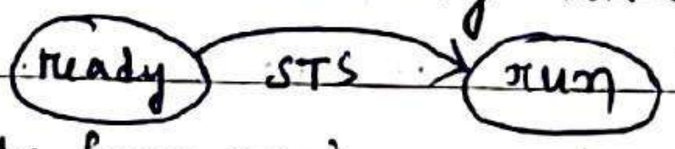
1. Long Term Scheduler or job scheduler:

LTS is responsible for creating new processes and bringing them into System. LTS select processes from job pool (where process

are kept for later execution) and load them into memory for execution. LTS control degree of Multiprogramming.

2. Short term scheduler or CPU scheduler:

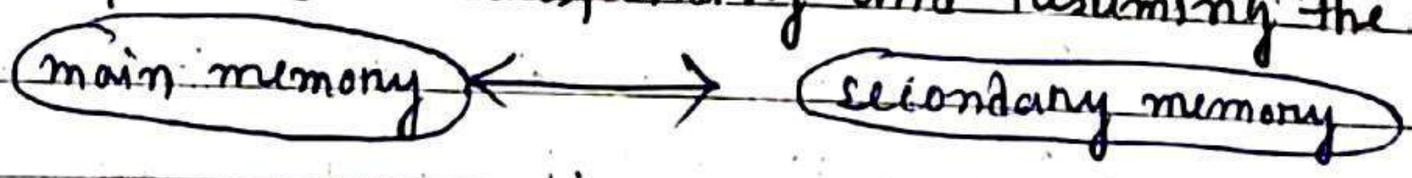
STS ~~is~~ is responsible for scheduling one of the process from ready sate to running state:



STS selects from main memory among the process that are ready to execute and allocate the cpu to one of them.

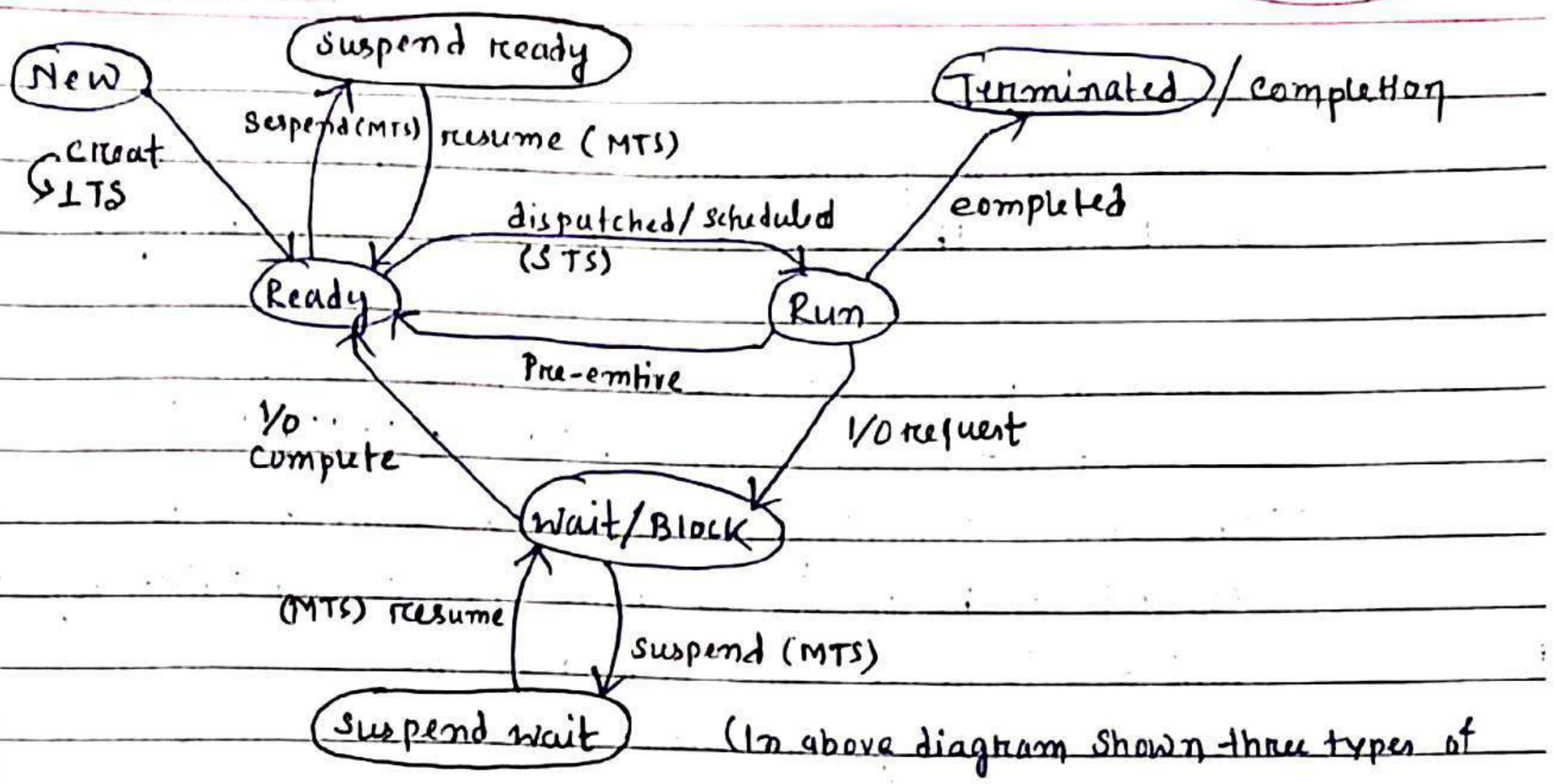
3. Medium Term scheduler:

MTS is responsible of suspending and resuming the processes.



The medium term scheduler are also called a SWAPPER.

Date __/__/____



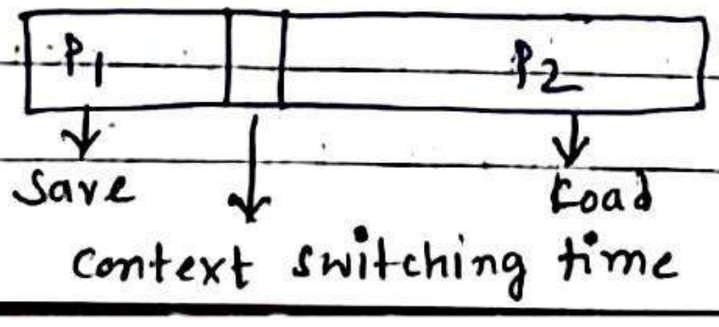
(In above diagram shown three types of schedulers)

Aspects :	LTS	MTS	STS
called as	It is Job scheduler	It is a process swapping	It is a cpu scheduler
Speed	Spee is less than STS	Sped is in between both LTS and STS.	Speed is fastest among two Schedulers.
Multiprogramming	It Control the Degree of Multipregramming	It reduce the Degree of multipregramming.	It provide less Control over degree of multiprogramming
Time-sharing system.	It is almost absent or minimal in time sharing System.	It is part of Time sharing System.	It is also ~~nominal~~ minimal. in time Sharing System.
processes	It Selects process from pool and loads them into memory for execution.	It can reintroduce the process into memory and execution can be continued.	It selects those process which are ready to execute.

RBR

Context Switching :

Saving the context of 1 process and loading the context of another process is called Context switching.



Saathi

Date ___ / ___ / _____

Some points:

→ each and every time when process is moving from one state to another, the context of the process will change.

→ Minimum process required for context switching '2'.
Exception - Round Robin '1'. (P1) [P1 || P1 || ..]

→ The context switching will also take some time To, so if the context of the process is more then context switching time will also increase which is undesirable.

→ Context switching time also is considered as overhead for the system.

• Dispatcher:
pispatcher is responsible for saving the context of 1 process and loading the context of another process. The context switching is done by dispatcher.

RBR • The fork():

→ System call fork() is used to creat processes. It takes no arguments and returns a process ID.

→ The purpouse of fork() is to create a new process, which becomes the child process of the caller.

→ After a new child processes is created, both processes will executed the next instruction following the fork() system call.

→ Therefore, we have to distinguish the parent from the child.

This can be done by testing the returned value of fork():

→ • If fork() return a negative value, the creation of a child process was unsuccessful.

→ • If fork() return a zero to the newly created child process.

→ • If fork() return a positive value, the process ID of the child process.

Date ___ / ___ / _____

to the parent. Normally the process ID is an Integer. Moreover, a process can use function ~~get pid~~ getpid() to retrieve the process ID assigned to this process.

• Conclusions -

→ If the program contains n fork() calls it will creat $2^n - 1$ child processes.

→ When the child process is created by using fork system call, both parent and children will have.

→ Relative address = Same for both parent and child process.

→ Absolute address = Different for both parent and child proces.

• fork() system call Implementation -

```
main
{
 int pid;
 pid = fork();
 if (pid < 0)
  { pf("child process creation failed");
 else if (Pid == 0)
  { pf("child process");
  }

 else
  { pf("parent process");
  }
}
```

• Example-1:

```
main()
{ fork();
 printf("Hello");
}
```

child (1)

```
main()
{
 pf("Hello");
}
```

child process

output: Hello
Hello

2 process / 2 time Hello. / 1 child ($2^n - 1 = 2^1 - 1 = 1$).

Date ___/___/_____

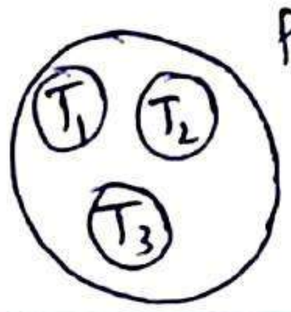
Saathi

RBR

Q. Question on process states—

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

	min	max
ready	0	M
running	0	N
block OR wait	0	M.



Date __/__/____

THREADS

Thread :

A Thread is a single sequence stream within a process. Because threads have some of the properties of processes, they are sometimes called lightweight processes.

→ A thread can be any several States (Running, Blocked, Ready, or Terminated)

→ Each thread has its own stack.

→ A thread has or consists of a program Counter (PC), a register sets, and a stack space.

→ Threads are not independent of one other like processes a result threads shares with other threads their code section, data section, OS resources also known as task, such as open files and signals.

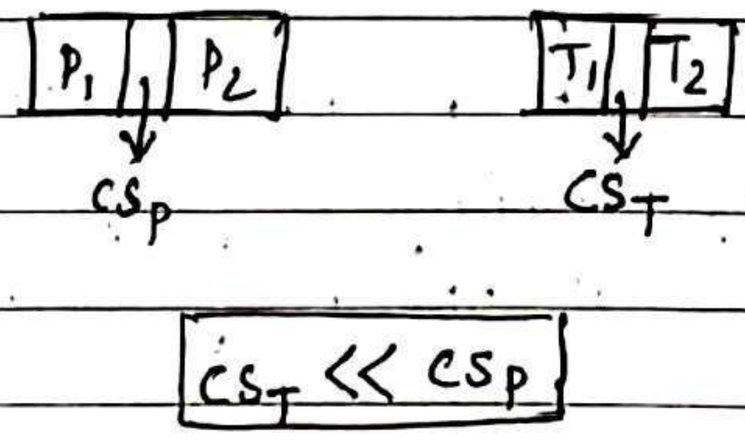
Advantages of Thread :

1. Responsivness :

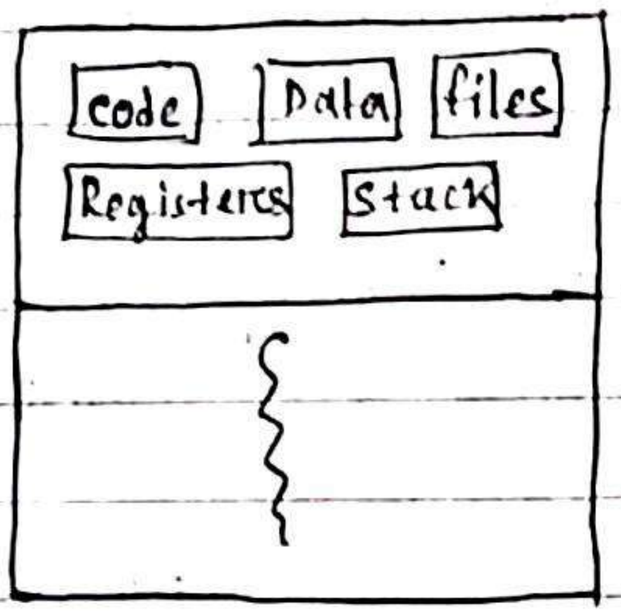
→ If the process is divided into multiple threads, then 1 thread is completed, then immediately output will be responded.

2. Faster context switches :

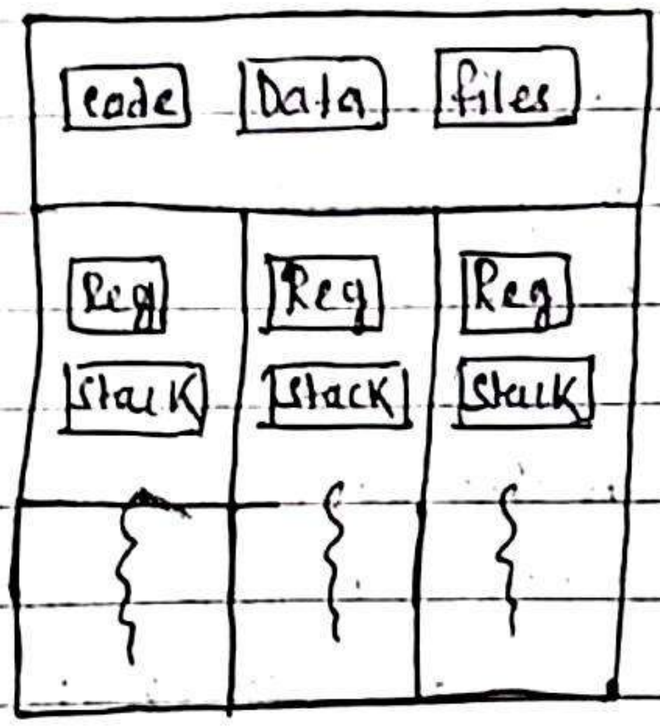
→ The context switching time between the threads is very very less as compared to context switching time between the process. Because the threads will have less context compared to process.



3. Resource sharing:



single threaded process



Multi threaded process

→ The resources like code, data, and files will be shared among all the threads with in the process but every thread will have its own stack and registers.

4. Effective Utiligation of Multiprocessor Systems:
→ If the process is divided into multiple threads, then different threads of the process can be scheduled onto a different CPU, so that the process execution will be faster.

5. Enhanced throughput of the System:
→ If the process is divided into multiple threads, if we consider 1 thread as 1 be job then number of job completed per unit time will increased and throughput of the system will be enhanced.

6. Economical:
→ The Implementation of threads does not requires any cost. There are serious programming languages API's which support Implementation of thread e.g JAVA's API, thread AI

Saathi

• Types of Threads :

Threads are categoriyed into 2 types. —

→ User level threads.

→ Kernel level threads.

• Difference between them —

User level thread	Kernel level thread
→ These are created by user or programmer.	→ These are created by operating system (OS).
→ OS can't recognige the user level threads.	→ OS recogniges the Kernel level threads.
→ If one user level thread performing Block system call, then entire process will be blocked.	→ If one Kernel level thread performing Blocking System call, then ~~oth~~ another thread will continue the execution.
→ Dependent threads.	→ Independent threads.
→ Designing user level threads is easy.	→ Designing Kernel level threads is complicated.
→ Less context.	→ More Context.
→ NO H/W support required.	→ Scheduling of Kernel level threads requires H/W support.

*. Thread Scheduling also known as GRANGE Scheduling.

• There are three common ways of establishing relationship between user threads and kernel threads —

→ Multi ... are ~~two~~ three types —

- Many-to-many model.
- one-to-one model.
- Many-to-one model.

Date ___ / ___ / _____

Saathi

1. Many-to-many model -

→ Allows many user level threads to be mapped to many kernel threads.

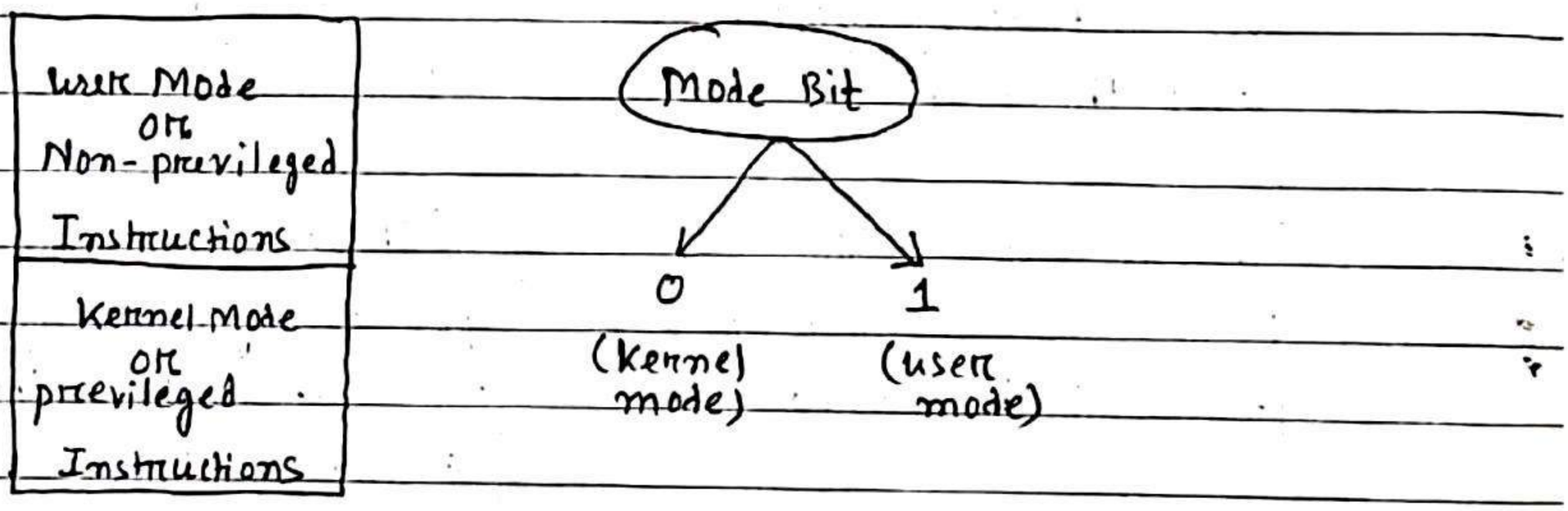
2. One-to-one model -

→ Each User level threads maps to kernel thread.

3. Many-to-one model -

→ Many user level threads mapped to single kernel thread.

Double Mode of operation :



→ The O.S executes the Instructions in 2 different modes -

1) User mode.

2) Kernel mode.

→ The dual mode of Operation is required to protection and security to the O.S and also to the user programs from Errant users (Concepted Users).

→ In which particular mode the current instruction is executing will be decided by mode bit. → 0 / 1

→ At boot time the system always starts in the kernel mode.

→ Depending upon the type of Instruction the O.S will be decide In which particular mode Instruction has to be executed.

Saathi

Date ___ / ___ / _____

→ Generally privileged instructions will be executed in the Kernel mode, and Non-privileged instructions will be executed in the User mode.

- privileged Instructions: (examples)
  1) Context Switching.
  2) Disabling Interrupts.
  3) set the time of clock.
  4) changing the memory map.
  5) I/O operation (Reading file data from disc)
- Non-privileged Instructions: (examples)
  1) Reading time of the clock.
  2) Sending the final print to the printer.
  3) Reading the status of the cpu.

(saathi)

Date

Inter-Process Communication

→ process executing concurrently in the operating system may be either Independent or cooperating processes.

→ A process is Independent; if it can't affect or be affected by other processes executing in the system.

→ A process is cooperating process, if that shares data with other processes is a cooperating proces.

→ Advantages of proces cooperation are -
- → Information sharing.
- → Computation speed up.
- → modularity and Convenience to work on many task at the same time.

→ Cooperating processes require an Inter process communication (IPC) mechanism that will allow that to exchange data and Information.

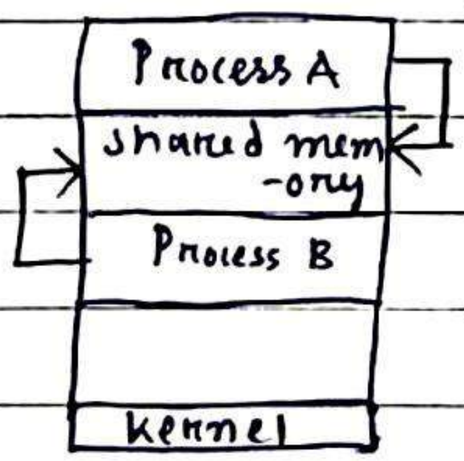
→ There are 2 fundamental models of IPC:
- I) Shared Memory.
- II) Message passing.

1. Shared memory:

→ In this method 2 or more processes shared a single chunk of memory to communicate.

→ used for exchanging larg amount of data.

→ faster than message passing.



(Shared memory)

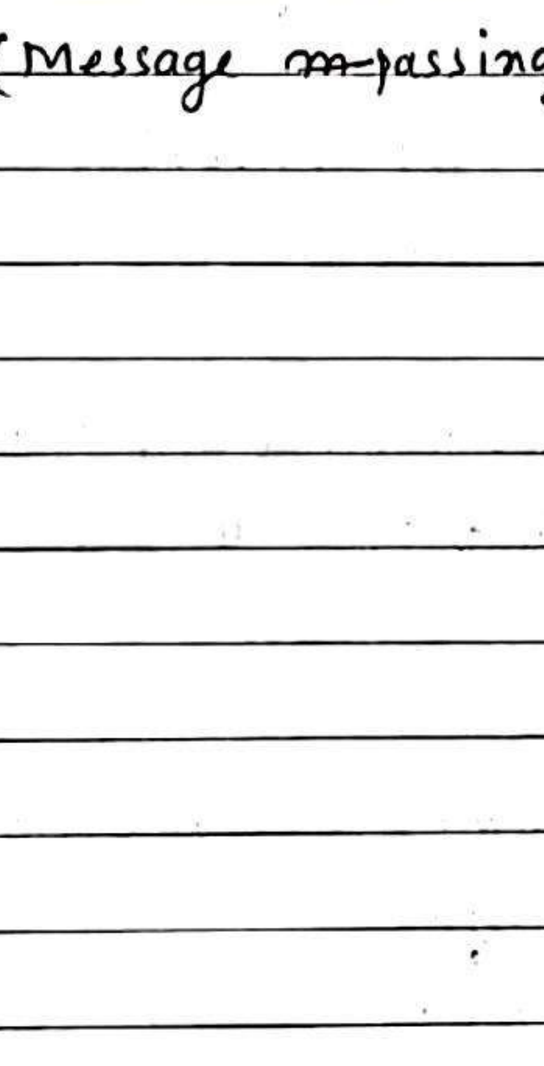
(Swadhin)

Date __/__/____

2. Message passing:

→ shared memory created in the kernel.

→ System call such as send and receive used for communication.



→ used for exchanging small amount of data.

→ slower than shared memory.

(Message passing)

Date __/__/____

PROCESS SYNCHRONIZATION

Saathi

*. The processes with respect to synchronization are of 2-types

- Co-operating process
- Independant process.

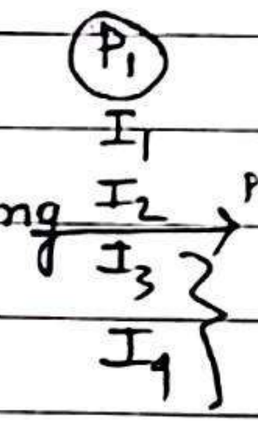
The execution of 1 process affects or affected by other process then these processes are said to be co-operative processes, Other wise they are said to be independant process.

→ Affection occurs due to shared variable, (common shared Resource / Data.

- Some Import points:

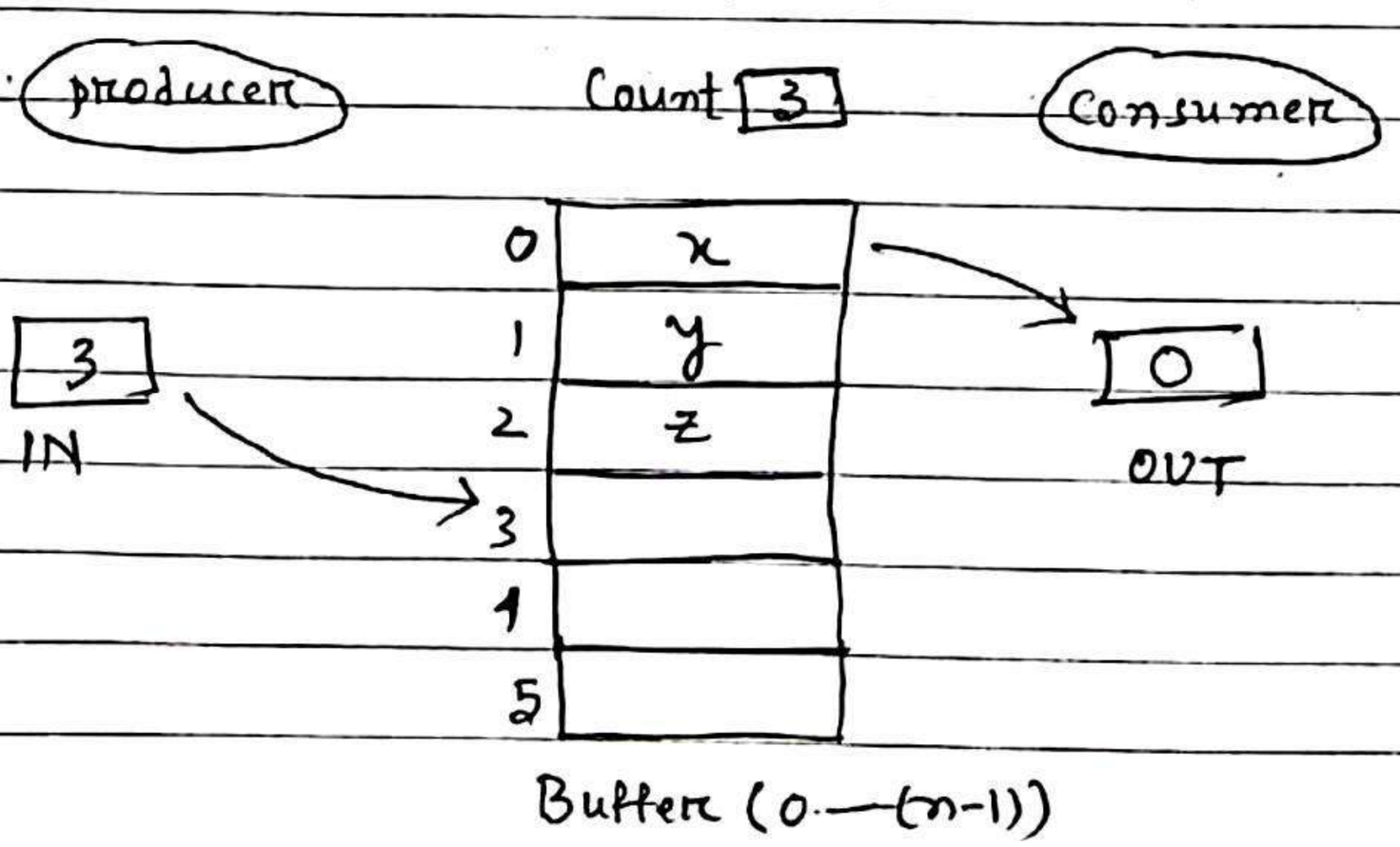
→ The pre-emption is just a temporary stop, the will come back and continue the execution.

→ If there is any possibility of solution becoming wrong by taking the pre-emption.



→ If any solution has the be deadlock, then the progress is not satisfied:

- producer-consumer problem:



Date

IN → is a variable used by producer to Identify the next empty slot in the Buffer.

OUT → is a variable used by Consumer to Identify the slot from where it has to consume the item.

Count → is a variable used by consumer and producer to identify the no of items present in the Buffer at any point of time.

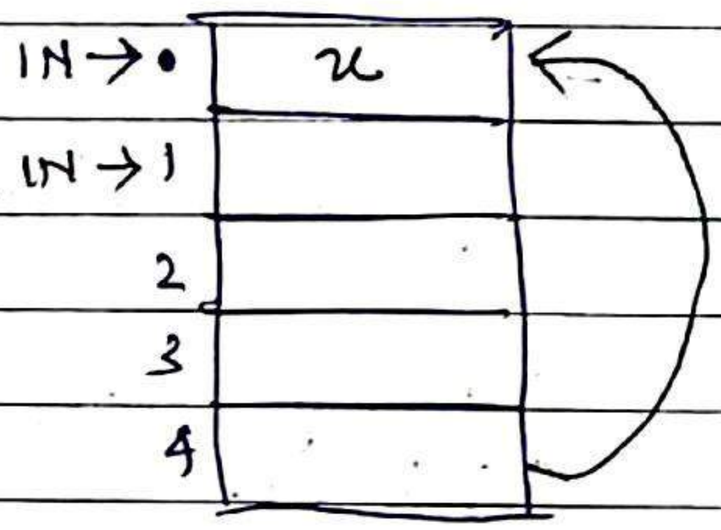
Conditions :-

1) → When the buffer is full, the producer is not allowed to produce the item.

2) → When the buffer is empty consumer is not allowed to consume the item.

• Producer Code →

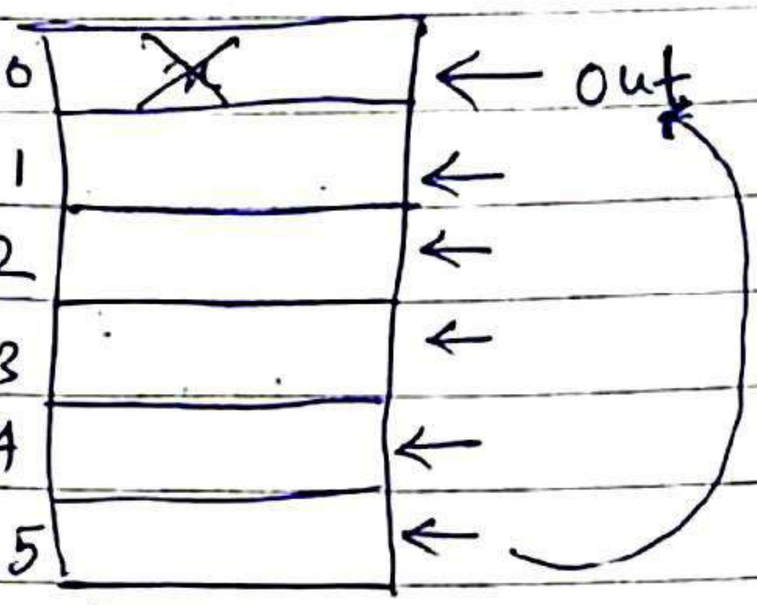
```
Int count = 0;
void producer (void)
{
   Int temp;
   While (TRUE)
   { produce_item (Itemp);
      While (count == N);
       Buffer [IN] = Temp;
       IN = (IN+1) MOD N;
       Count = count + 1;
   }
}
```



Date

- Consumer Code →

```
void consumer(void)
{
    int itemc;
    while(TRUE)
    {
        while (count==0);
        Item c = Buffer[OUT];
        OUT=(OUT+1) MODN;
        count = count -1;
    }
}
```



- producer Consumer problem Analysis:

Ⓧ producer

```
count=count+1;
```

⇓

I. LOAD R_p, m[COUNT];
II. INCR R_p
III. STORE m[COUNT], R_p

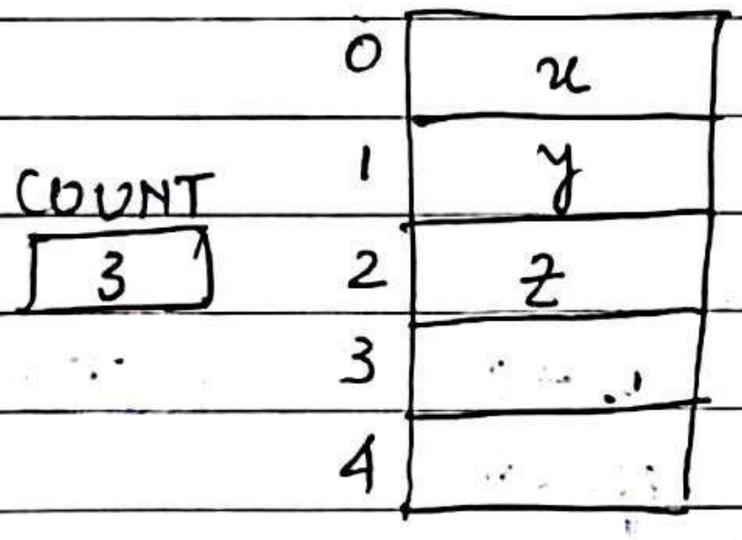
Ⓧ Consumer

```
count=count-1;
```

⇓

I. LOAD R_c, m[COUNT]
II. DECR R_c
III. STORE m[COUNT], R_c

Analysis – Race around condition



{different count value for Producer and consumer}

Now,

P → I R_p [3]
P → II R_p [4]
preempt

C → I R_c [3]
C → II R_c [2]
✓ C → III [2] ✓

✓ P → III [4] ✓

Saathi

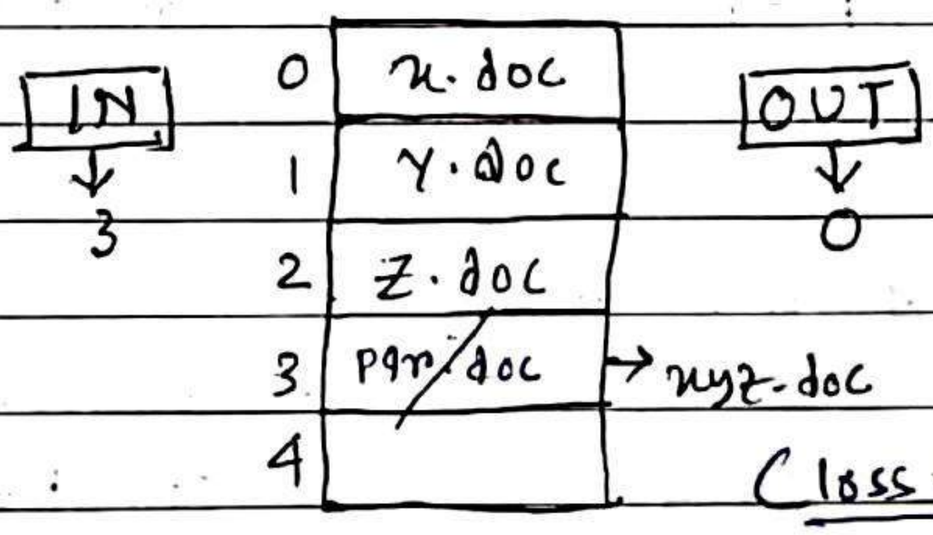
Date __ / __ / ____

- producer Consumer problem Conclusion -

→ problems of Non-Synchronigation.

Inconsistency :- The producer and consumer are not properly Synchroniged while ~~sharing~~ shearing the common variable COUNT. Hence it is leading inconsistency. No proper Synchronigation between producer and consumer

- pointer - Spooler Domain problem -



(IN) → is a variable used by all processes to identify the next empty place in the Spooler directory.

(OUT) → is a variable used ~~by~~ only by printer to identify the slot from where it has to Print the document.

Enter file -

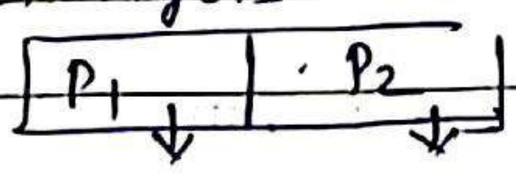
1) load R_i, m [IN]

2) STORE SO [R_i], "file name".

3) INCR R_i.

4) STORE m [IN], R_i.

Analysis -



$P_1 \rightarrow$ I R_1 [84]

$P_1 \rightarrow$ II

$P_1 \rightarrow$ III → preempt

$P_2 \rightarrow$ I R_1 [3]

$P_2 \rightarrow$ II

- problem of Non-Synchronigation:

→ Inconsistency: The producer and consumer are not properly synchroniged while sheaning the common variable COUNT. Hence it is leading to inconsistency. No proper synchronigation between producer and consumer.

→ Data loss of Data: The process are not properly synchroniged while sharing the common variable [BM], hence it is leading to lost of data.

→ Deadlock: If the process are not properly synchroniged the there is also a possibility of deadlock.

- Delinations:

1) Critical Section (C.S): The portion of program text where sheared variables are placed.

example- In producer/consumer problem:

count = count + 1 → C.S

2) Non-Critical Section: The portion of program text where the Independent code of the process will be placed.

3) Race Condition: The final value of any variable depends on the execution sequence of process. This condition is called as Race condition.

- The Critical Section problem — Structure —

```
do
{
    [entry section]
            critical section.
    [exit section]
            reminder section / NCS.
}
while (TRUE);
```

Saathi

Date __ / __ / ____

- Synchronization conditions :-

1) MUTUAL EXCLUSION: No two process may be simultenously present inside the critical section at any point of time. Only 1 process is allowed into critical section at any point of time.

2) progress : No process running outside the Critical section should block the other interested process from entering into c-s when the c-s if free.

3) Bounded Waiting : NO process should have to wait forever from entering into critical section. There should be a bound on getting chance to enter into critical section. If Bounded waiting is not satisfied, then it is possible for starvation.

- Solution Types :

1) Software Type :-

a. Lock variables.

b. strict Alternation (or) Decker's Algorithm.

c. peterson's Algorithm.

2) Hardware type :-

a. TSL Instruction set (Test & set lock)

3) O.S Type :-

a. Counting semaphore.

b. Binary semaphore.

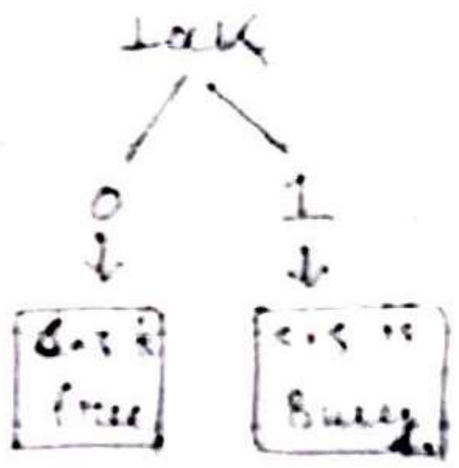
4) programming language compiler support type :-

a. Monitors.

- **Lock variables :** → [M.E → ✗, Progress → ✓, B.W → ✗]

Entry section -

I. load R_i, m[lock]
II. CMP R_i, #0
III. JNZ tostep ①
IV. STORE m[lock] #1
V. [C.S]
VI. STORE m[Lock], #0



Let, lock ⇒ 0 X 1

R.Q [P_1 | P_2]

P_1 → I — R [0].
P_1 → II
P_1 → III → Preempt.

P_2 → I — R [0]
P_2 → II
P_2 → III
P_2 → IV, V

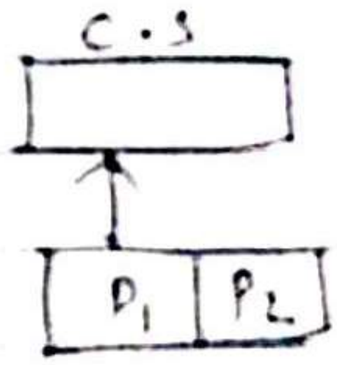
P_1 → IV,
P_1 → V

C.S
[P_2, P_1]
↓
M.E-X
✓ (Mutual exclusion not satisfy) - ✗

Entry section.

I. LOAD R_i, m[lock]
II. CMP R_i, #0.
III. JNZ tostep ①.
IV. STORE m[lock], #1.
} entry section

V. [C.S] } critical section
{ VI. STORE m[lock], #0. } remainder section



Lock=0
1

P_2 → I — R [0]
P_2 → II, III
P_2 → IV → preempt

P_1 → I — R [1]
P_1 → II, III

(Here no. of process infinite)

✓ process one satisfy ✓
✓ Bounded Waiting - ✗
(If process one time then B.W satisfy
other wise not satisfy)

Date ___/___/___

(Saathi)

- Strict Alternation (or) Decker's Algorithm :- M.E → ✓, P → X, B.W → ✓

→ process takes turn to enter into critical solution.

→ solution will work for 2 - process.

(P0) (P1)

turn

0 → P0 (entry)

1 → P1 (entry)

P0 → 0

```
While (TRue)
{
Non C.S;
While (turn != 0);
C.S
turn ⇒ 1;
}
```

P1 → 1

```
While (TRue)
{
Non C.S;
While (turn != 1);
C.S
turn ⇒ 0;
}
```

Analysis (for M.E, P, B.W) —

turn ⇒ 0

es

P0 → (M.E) ✓

for finite process → (B.W) ✓

turn → ∅, 1, 0

P0, P1, (P1)

here P0 not interested to go C.S but for turn=0

P1 can't go CS. (P0 stored P1) → (Progress) X

- 2nd Defination of progress :-

Which process will go next into C.S is decided by only those process Who want to go into CS. I this decision the process in the remainder section and the process which is not interested to go into the c.s should not take participation.

Saathi

Date

- Peterson's Algorithm :- (ME → ✓, P → ✓, B.W → ✓)
  (2 process solution)

```
# define N 2
# define TRUE 1
# define FALSE 0
```

```
int turn;
int interested[N];
void enter_Region(int process)
{
 1. int other;
 2. other = [1-Process];
 3. Interested[process]=TRUE;
 4. turn = process;
 5. while (turn == process && interested[other] == TRUE);
        CS;
}

void leave-Region (int process)
 {
   interested[process] = false;
 }
```

Initially –

```
interested[0] = false.
interested[1] = false.
```

P0	P1
↓	↓
other	other
1	0

- Analysis :-

Saathi

Date ___ / ___ / ___

P_0	P_1
1.	1. →
2. other → 1-0 = 1	2. other = 0
3. Interested[0] = True.	3. Interested[1] = True.
4. turn = 0	4. turn = 1
5. → C S P_0	5. loop CS P_0

So, M.E = ✓
progress = ✓
Bounded waiting = ✓

* peterson practise Question :-

Code –

```
void enter_Region()
{
1. int other ;
2. other = 1- process ;
3. Interested[process] = True;
4. turn = process ;
5. While (turn == process && Interested[other] == True);
6. C.S ;
}
```

P_0	P_1 ✓
1. →	1. →
2 → other = 1	2. other = 0
3 → Interested[0] = True	3. Interested[1] = True.
4. → turn = 0;	4. turn = 1;
	5. ↻
	6. S P_1

Q. If both the process P_0 and P_1 are trying to enter in CS at same time, then which process will go into CS first.

a. The process which executes statement 2 first.
b. " " " " " 3 " .
✓ c. " " " " " 4 " .
d. We cannot say.

→ Ans → (c).

Saathi

Date ___ / ___ / _____

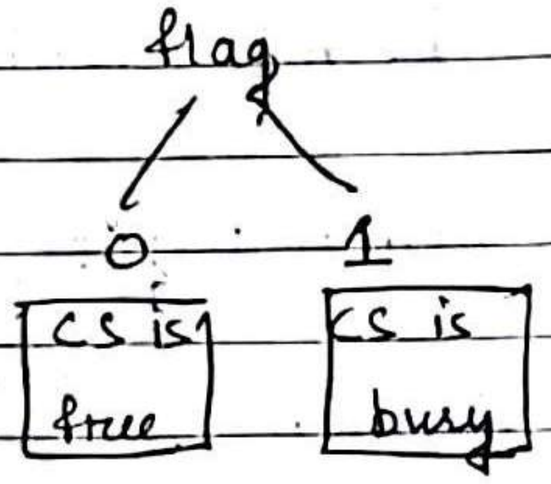
- TSL Introduction: M.E → ✓, P → ✓, B.W → X

→ TSL ~~flag~~ Register flag :- Copies the current value of flag into register and store the value of 1 into the flag in a single atomic cycle, without any preemption.

→ Disabling the interrupt is at H/W level that is why it is called H/W approch.

→ Entry section -

*1) TSL Ri, m[flag]
2) cmp Rj, #0
3) JNZ to step (1).
4) C.S
5) STORE m[flag], #0



Analysis -

flag = ∅ & 1

RQ = [P0 | P2]

P1 → I R1 [0]

P1 → II

P1 → III

P1 → IV → pre-empt CS [P1] M.E → ✓

P2 → I R2 [1]

P2 → II, III

[P1 | P2] flag = ∅ & 1

CS [P1]

P1 → I R1 [0] → preemptive

P2 → I R2 [1] progress → ✓

P2 → II, III

Where no. of process infinite so, B.W → X.
(Bound, waiting)

Saathi

Date

*. Some Important points :- (TSL)

→ In Lock variables mutual exclusion is not satisfied because loading and storing were not done in same step.

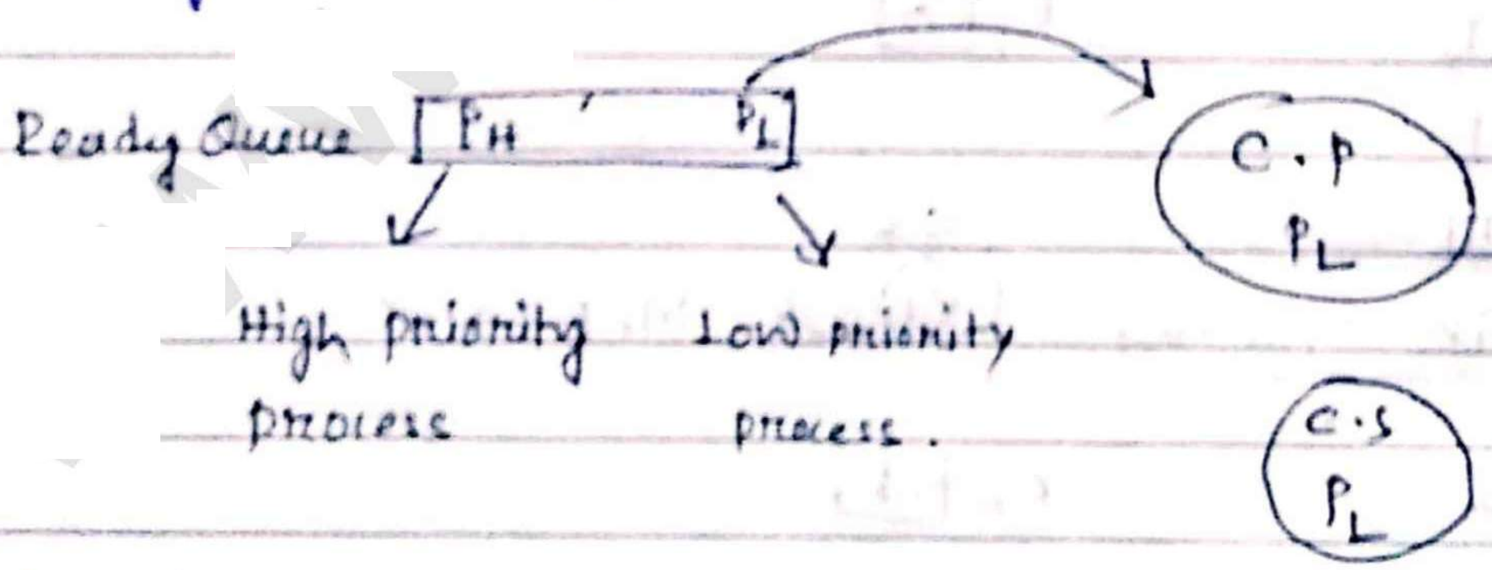
→ progress is satisfied in TSL.

→ Bounded waiting is not satisfied Because no. of processes is not countable.

*t. Summary :-

Solution	Mutual exclusion	progress	Bounded waiting
Lock variable	X	✓	X
Strict Alternation	✓	X	✓
Peterson's Solution	✓	✓	✓
TSL	✓	✓	X

- Priority Invention problem —



→ P_L is currently executing CS code. NOW suddenly P_H coms, then we are going to pre-empt P_L. ~~P_H is~~ even though P_H is scheduled by CPU but it can not enter in C.S because it is locked by P_L.

↓

Problem is called live-lock

↓

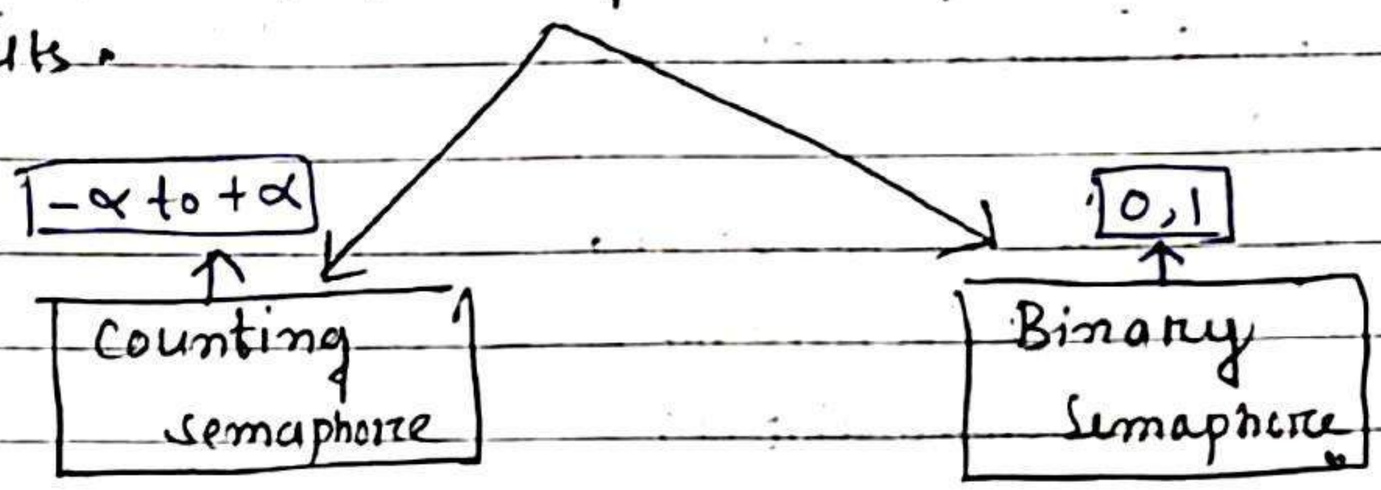
Solution → Priority Inversion / Inheritance.

Date / /

→ This allows that P_1 can Continue its execution in c.s and then Leave.

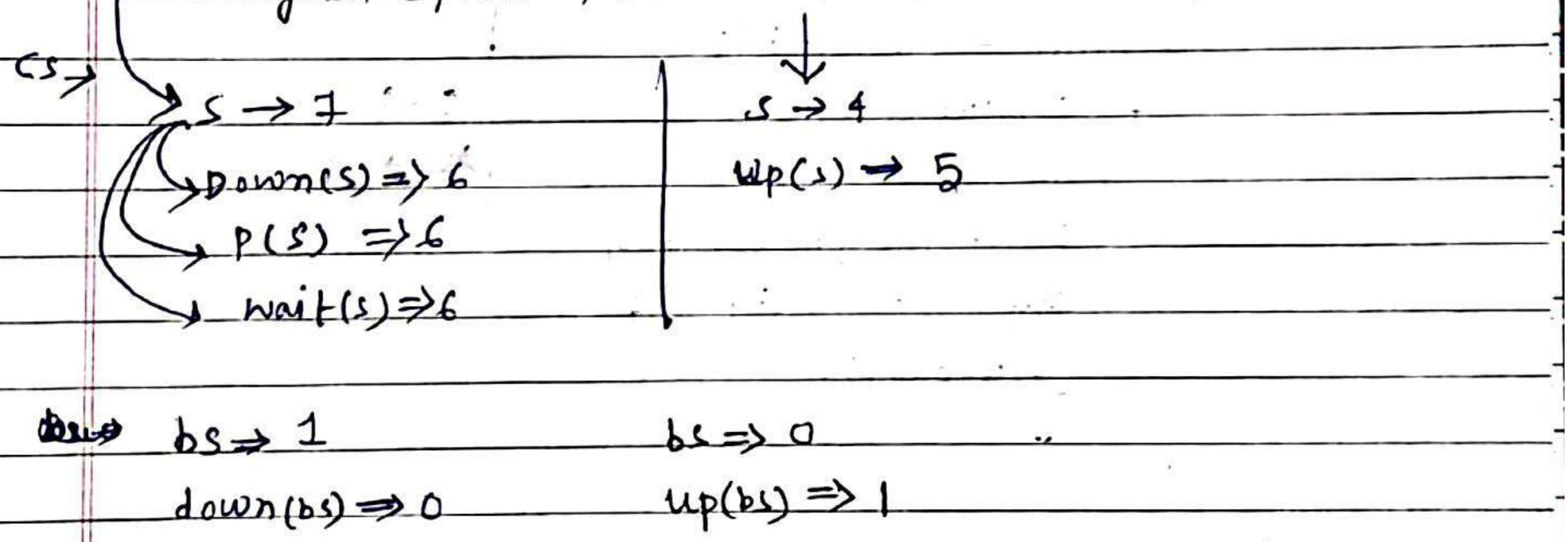
*. Semaphore :

→ is an Integer variable which is used by various process in a mutual exclusive manner to achive Synchronigation. The ~~proper~~ Improper usage of semaphore will be given Improper results.



Two different operations performed on semaphore variable are -

1) Down Operation on P() or wait().
2) signal Operation on V() or release() or UP().



Date ___ / ___ / _____

Counting Semaphore -

→ Down operation -

```
Down operation (semaphore s)
{
s.value = s.value - 1 ;
If (s.value < 0) → True
{
Block the process and
Place the PCB in the
suspended list ();
}
}
```

```
s → 2
↓
1) Down (s) → 1
2) Down (s) → 0
3) Down (s) → -1
```

→ Up operation - (always successful)

```
up operation (semaphore s)
{
s.value = s.value + 1;
if (s.value ≤ 0)
{
Select the process from
suspended list and
wakeup ();
}
}
```

```
s → 2
↓
s → 3

s → -2
↓
s → -1
```

→ After performing the down Operation, if the process is getting suspended, then it is called unsuccessful down operation.

→ After performing the down operation, if the process is not getting suspended, then it is called successful down operation.

Date ___ / ___ / _____

Saathi

→ If it is successfull down Operation, then only the process will continue the execution. The down operation on the counting semaphore is successfull only if the semaphore value is >= 1

→ There is no unsuccessfull up operation, the up Operation is always successfull because it does not suspend any process.

→ If S = +6 it means we can perform 6 successfull down Operation.

→ If S = -6 it means that there are already 6 suspended processes.

*. Question -

Consider a system where a counting semaphore value is initialized to +17.
The various semaphore Operations like -
23P, 18V, 16P, 14V, 1P are performed.
What is the final value of semaphore.

⇒ 17 - 23 + 18 - 16 + 14 - 1

⇒ 1 - 6 + 2 + 13

⇒ (9)

Binary semaphore : 0 / 1

→ Down Operation

```
Down (semaphore S)
{
  if (S.value == 1)
    S.value = 0;
  else
  { Block the process and
    place the PCB in the
    Suspended list ();
  }
}
```

→ Up Operation

```
up (semaphore S)
{
 if (suspended list() is empty)
    S.value = 1;
 else
 {
  Select a process from
 suspended list and Wakeup()
 }
}
```

→ After performing down Operation if the process is getting suspended then it is called unsuccessfull down Operation.

→ After performing down Operation, if the process not getting never suspended then it is called successfull down Operation.

→ The down Operation on the Binary semaphore is successfull only if the semaphore value is 1.

→ If it is successfull down Operation, then only the process will Continue the Execution.

→ There is no unsuccessfull up operation. The up Operation will be always successfull. The process performing the up Operation will definately continue the execution.

Question : (i) (ii) (iii)

Each process $P_i = 1$ to 9 executes the following code -

```
repeat
    P(mutex);
    c.s
    V(mutex);
forever
```

⇒ The initial value of Binary semaphore mutex = 1.

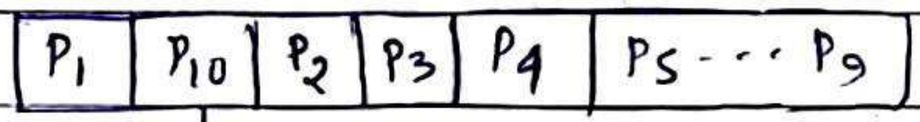
⇒ What is the maximum no. of process may be present inside the c.s at any point of time ?

mutex ⇒ 1

The process P_{10} execute the following code.

```
repeat
    V(mutex);
    c.s
    V(mutex);
forever
```

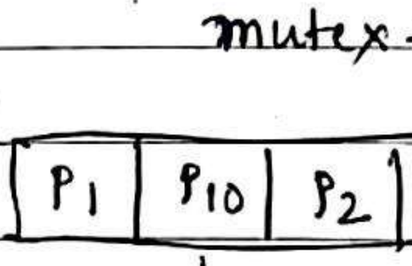
mutex → 1 0 1 0 1 0 1 0



→ Ans : (10) process.

⇒ P(mutex); then find ?

→ mutex = 1 0 1 0



→ Ans : (3) process.

⇒ P(mutex); then find ?

mutex = 1 0

→ Ans : (1) process.

classmate

- GATE-2003 Question :-

Consider the two concurrent process 'P' and 'Q' executing their ~~ex~~ respective codes.

process 'P' code	process 'Q' code

while (TRUE)
{
① W: P(T) 1
printf('0');
printf('0');
② X: V(S) 0
}
	while (TRUE)
	{
	③ y: P(S) 1
	printf('1');
	printf('1');
	④ Z: V(T) 1
	}

What should be the semaphore operations on W, X, Y, Z respectively. And What should be the initial value of Binary semaphore 'S' and 'T' inorder to get the output as 0011 0011 00 ...

(a) W=P(T), X=V(T), Y=P(S), Z=V(S), S=T=1 ;

(b) W=P(T), X=V(T), Y=P(S), Z=V(S), S=1, T=0 ;

(c) W=P(T), X=V(S), Y=P(S), Z=V(T), S=1, T=1 ;

(d) W=P(T), X=V(S), Y=P(S), Z=V(T), S=0, T=1 ;

→ PQPQ ... -

- Question :- process code same.

Which of the following will ensure that the output string never contains a substring of the form 01^n0 or 10^n1 where n is odd.

(a) W=P(S), X=V(S), Y=P(T), Z=V(T), S=T=1 ;

(b) W=P(S), X=V(T), Y=P(T), Z=V(S), S=T=1 ;

(c) W=P(S), X=V(S), Y=P(S), Z=V(S), S=1 ;

(d) W=P(S), X=V(T), Y=P(S), Z=V(T), S=T=1 ;

S →

→ 0011 0011 ...

- **Producer & consumer problem** solve with the help of **semaphore**.

semaphore S = 1

semaphore E = n here n = is of size of buffer

Semaphore F = 0

c.)

Let S = X Ø X Ø 1
E = Ø X 3
F = Ø X 2

```
* void producer()
{
    While(T)
    {
        produce()
        wait/down(E)
        wait(s)/down(s)
        append()
        signal(s)/up(s)
        signal(F)/up(F)
    }
}
```

```
* void consumer()
{
    while(T)
    {
        wait(F)/down(F)
        wait(S)
        take()
        signal(S)/up(S)
        signal(E)
    }
}
```

S = X Ø 1
E = 3 X 4
F = Ø X 1

- READ-WRITE PROBLEM:

(solve by Sipnaphore)

(there ~~using~~ used 3 semaphore).

for writer

I. wait(wrt) (1)	wrt = 1 0 ..
II. write operation	
III. signal(wrt)/up(wrt).	

for read

I. wait (mutex) (1)	I. mutex = 1 0 1 0 1.
II. readcount ++	II. readcount = 1 0
III. If (readcount == 1) wait(wrt)	III. wrt = 0 (no writer can enter in critical Section :).
IV. signal (mutex)	
V. read operation.	
VI. wait(mutex)	
VII. readcount --	
VIII. if (readcoun == 0) signal(wrt)	
IX. signal(mutex).	

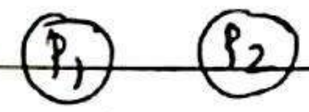
*. At a time more than one ~~Of~~ read operation done in Critical Section.

*. At a time only One write operation ~~can~~ done in C.S.

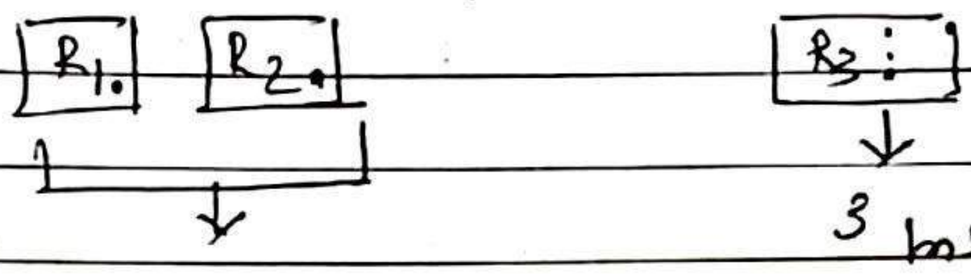
DEADLOCK

- Defination:- If two or more processes are waiting for some event to occure, which never happens, the processes are said to be involved in deadlock.
- Resource Allocation Graph (RAG):

→ Basics:- process will be represented with circle.



Resources will be represented with rectangle.



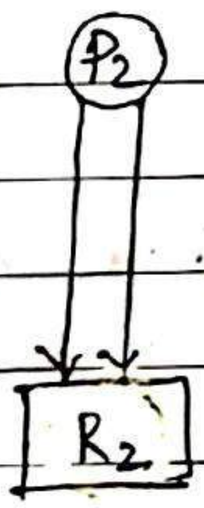
Single instance

*. Requesting edge

*. Allocation edge.

process P_1 is requesting 1 Instance of R_1

one 1 Instance of 'R1' is allocated to 'P1'.



process P_2 is requesting for 2 Instences of 'R_2'.

Two Instance of 'R_2' is allocated to 'P_2'.

→ Resource request and Resource allocation will be ~~requested in the~~ represented in the resource allocation graph (RAG).

$(R \cdot A \cdot G) = G_r(V, E)$

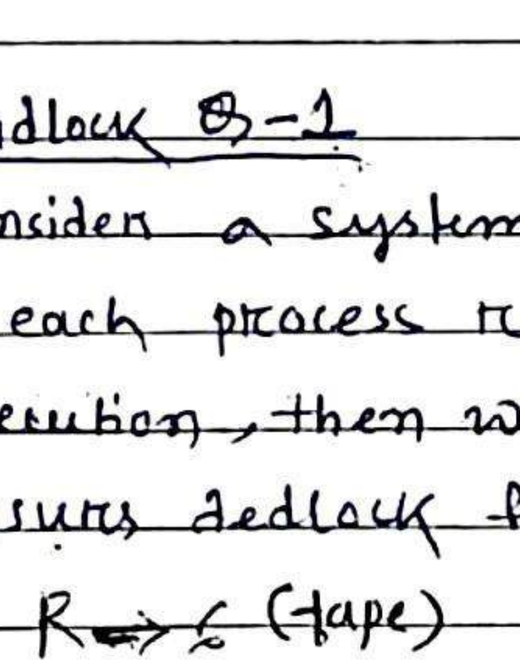
V → processes & Resources. (vertex)

E → Requesting and Allocation edge.

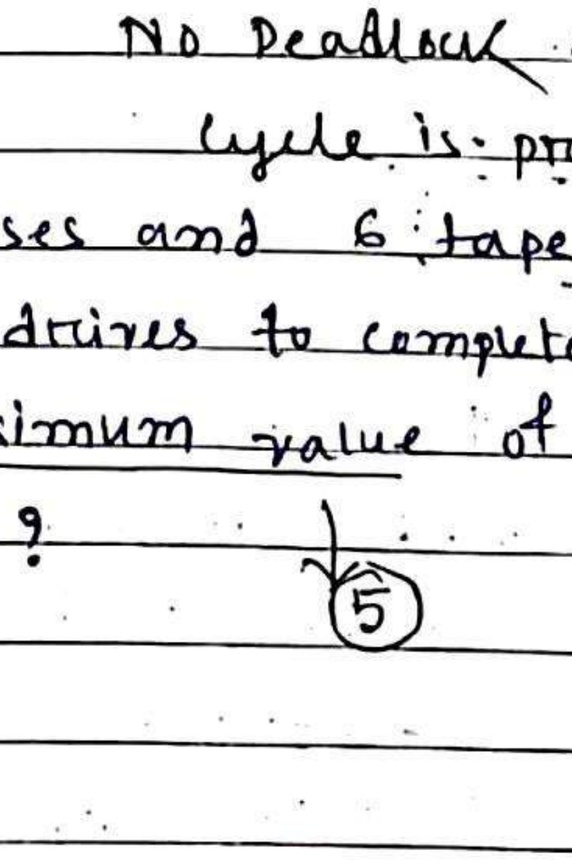
- Resource Request and Release life cycle.

→

1) The process will request for the resource. $(P_1) \rightarrow [R_1]$
2) O.S validates the request of the process.
3) If the request made by process is valid, the O.S check for availability of the resource.
4) If the resource is freely available, it will be allocated to the process, other otwise process has to wait.
5) If all the resources requested by the process are allocated then it will go into execution.
6) Once the execution of the process completed, then the process will release all the resources.



Deadlock is involved in (RAG)



No Deadlock even if cycle is present.

- Deadlock @-1

Consider a system with n processes and 6 tape drives. If each process requires 2 tape drives to complete their execution, then what is the maximum value of n which ensures deadlock free operation?

→ R ⇒ 6 (tape)

P ⇒ need {2}

* (n-1)

(5)

P_1	P_2	P_3	P_4	P_5	P_6
λ_2	1	1	1	1	1

$n = 1$

↳ execution complete.

first take, $n = 6$ (deadlock)

If $n = \underline{5}(P)$ then it will be deadlock free operation.

• Deadlock Q-2 :

Consider a system with 3 processes, where each process requires 2 tape drives to complete there execution, then what is the minimum number of resources required to ensure deadlock free execution. (1)

→ here, process = 3

P →	P_1	P_2	P_2
R →	1	1	1

$\therefore R = 3+1 = 4$

minimum number of resources required to ensure deadlock free execution are 4(R).

• Deadlock Q-3 :

Consider a system with n processes and 6 tape drives. If each process require 3 ~~process~~ tape drives to complete their execution. What is the maximum value of n which ensures deadlock free execution.

→

P_1	P_2	P_3
2	2	2
3	3	X

$R \rightarrow 6$

$n_1 \rightarrow ?$ so, n will be 2(P).

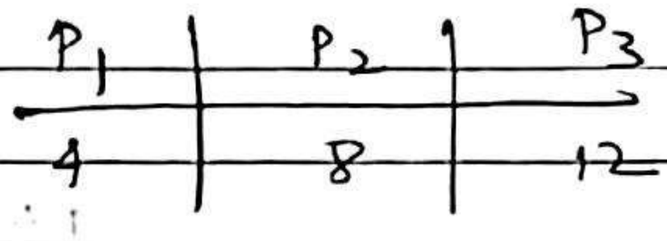
need(3)

• Deadlock Question-1 :

Consider a system with 3 proces P_1, P_2, P_3. The peak demand of every process is 5, 9, 13 respectively for the Resource (R). Then for what minimum no of resources, the execution is deadlock free.

(25)

→ $P_1 \rightarrow 5$
$P_2 \rightarrow 9$
$P_3 \rightarrow 13$
} $(n-1)$



$R \rightarrow 4+8+12$

$= 24$

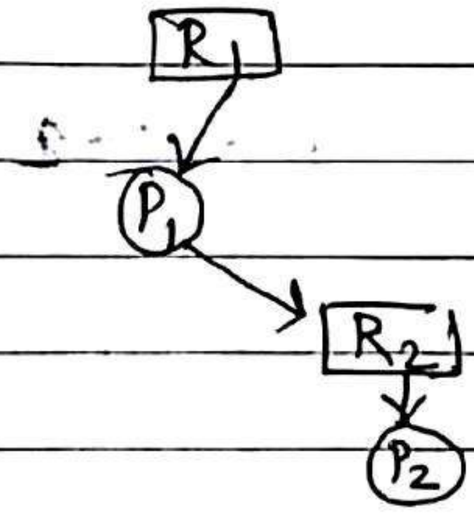
$+1$

$R = 25$ ✓

• Deadlock Characteristics Introduction –
(Behaviour of Deadlock)

(I) Mutual exclusion.
(II) Hold and wait.
(III) NO – Preemption.
(IV) circular wait.

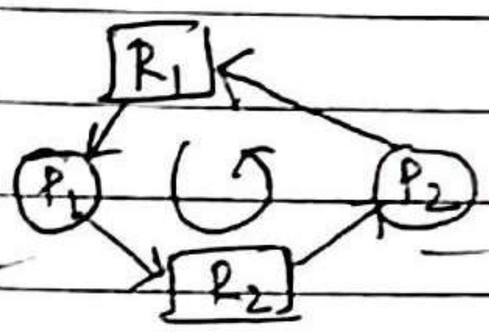
1. Mutual exclusion :- The resource has to be allocated to only one process or it is freely available. There should be a one to one relation ship between the resource and process.
2. Hold and wait :- The process is holding some resources and waiting on some other resource simultenously.



3. No- Preemption :- The resource has to be voluntarily released by the process after the completion of execution. It is not allowed to forcefully pre-empt the resources from process.

4. Circular Wait :-

The process are circularly waiting on each other for the resources.



→ R1 → P1 → R2 → P2

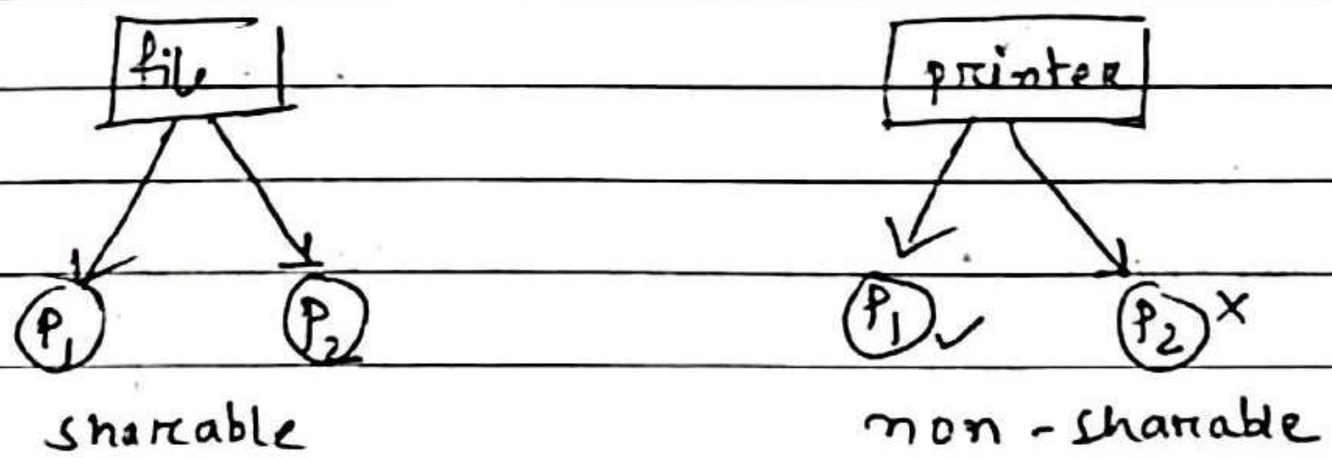
All 4 MUST occur simultaneously. Then Delinately Deadlock is there.

All the above 4 conditions are not truely independent because circular wait includes hold & wait.

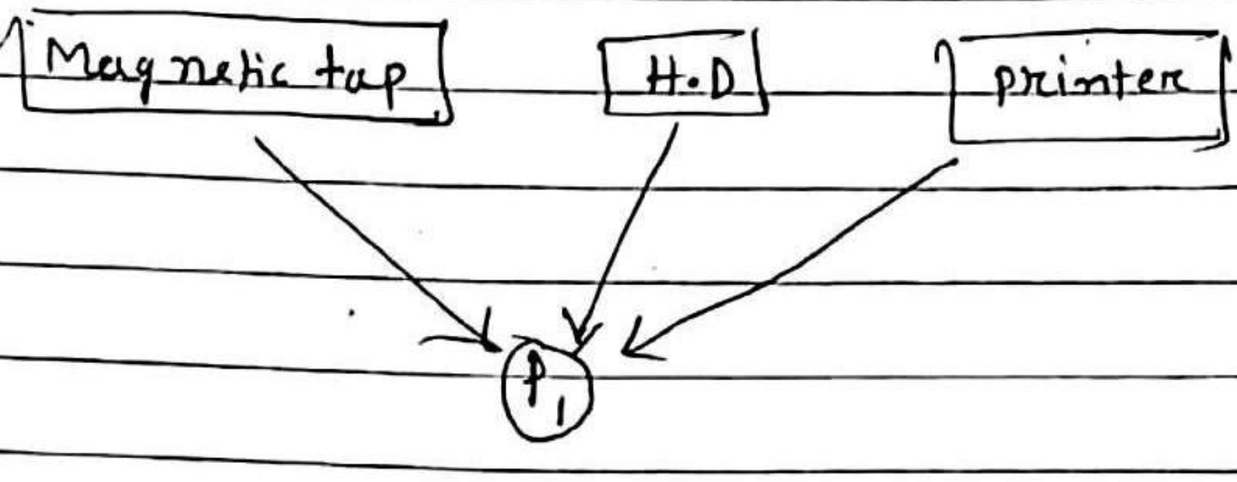
- Deadlock prevention schemes :

(1) Deadlock (M.E) prevention : It is not always possible to dissatisfy mutual exclusion because of sharable and non-sharable resource.

→ If there are sharable resources then mutual exclusion can be dissatisfy and if they are non sharable resource then Mutual exclusion cannot be dissatisfied.



(2) Hold and wait :- (i) Allocate all the ~~process~~ Resources required by the process before start of execution.



→ Low device utiligation.

(ii) The process should release all existing resource before making new request.

$P_1 \Rightarrow$ 100 R

1) $P_1 \rightarrow$ 100R	$P_1 \leftarrow$ 85 R
2) $P_1 \rightarrow$ 100 R	$P_1 \leftarrow$ 40 R
3) $P_1 \rightarrow$ 100R	$P_1 \leftarrow$ 100R

→ (starvation).

(3) No preemption : The process 'P' is requesting for resource 'R'. $(P_1) \rightarrow [R_1]$

If resource R_1 is free then it will be allocated to process 'P_1'.

$[R_1] \rightarrow (P_1)$

If the process P_1 is in the execution then process P_1 has to wait.

The Resource R_1 is not free, and it is allocated to process P_2.

$[R_1] \rightarrow (P_2)$

The process P_2 is not executing and is waiting for the other resource R_2.

Prempt the resource R_1 from process P_2 and Asrign it to process P_1.

(4) Circulare Wait : The resources will be assigned with a numerical numbers. The process can request for the resource only in Increasing order of enumeration.

$R_1, R_2, R_3, \cdots R_{12}$

$P_1 \rightarrow R_5$ ✓

$P_1 \rightarrow R_7$ ✓

$P_1 \rightarrow R_9$ ✓

$\boxed{P_1 \rightarrow R_6}$ ✗

$P_1 \rightarrow R_9$ ✓

Bankers Algorithm :

- Deadlock Avoidance (BANKERS ALGORITHM)

Total Resources ⇒

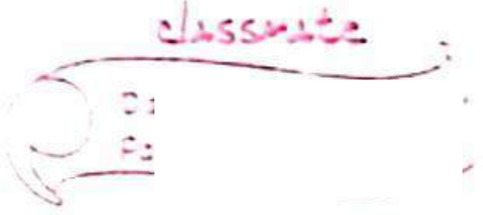
A	B	C
10	5	7

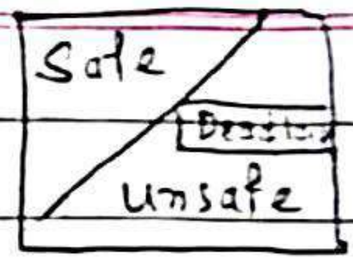
= (1 instance)

	Max Need			current Allocation			Remaining Need			Current Available		
	A	B	C	A	B	C	A	B	C	A	B	C
(III) P_0	7	5	3	0	1	0	7	4	3	(3)	(3)	(2) Ans
(I) ✗ P_1	3	2	2	(2)	0	0	1	2	2	5	3	2
(IV) P_2	9	0	2	3	0	2	6	0	0	7	4	3
(II) ✗ P_3	2	2	2	2	1	1	0	1	1	7	5	3
✗ P_4	4	3	3	0	0	2	4	3	1	10	5	5
				7	2	5				10	5	7

$\boxed{P_1, P_3, P_0, P_2, P_4}$

safe sequence





- Deadlock Avoidance Conclusion:

*. Remaining need = Max - Current Allocation.

→ If we can satisfy the remaining need of the process with the current available resources, then system is said to be in the safe state other wise, the system is said to be in the unsafe state.

→ If the system is in the unsafe state, then it is possible for deadlock.

→ The order in which we satisfy the remaining need of all the process is called safe sequence.

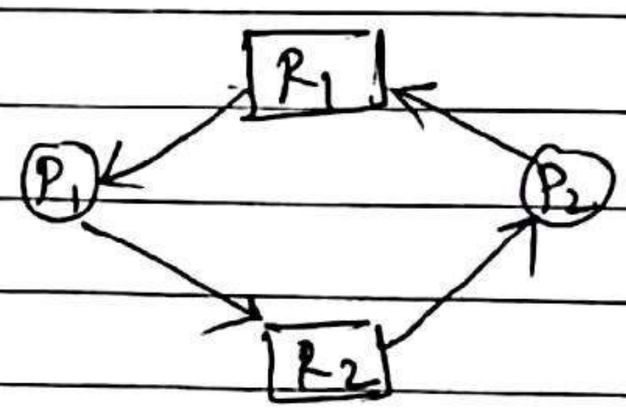
→ The safe sequence may not be unique. We can have multiple safe sequence.

→ Deadlock Avoidance is more restrictive than deadlock pre perention.

→ The unsafe state purely depends on the behaivour of the process.

- Deadlock Detection:

(i) All the resources are of single Instance type.



If all the resources are of single Instance type, then cycle in the RAG is nessery and sufficient condition for occuring of

deadlock. If all the resources are of more than single Instance type, then cycle in the RAG is just a necessary condition but not sufficient condition for occuring of deadlock.

(ii) The Resources are of multiple Instance type.

total resources

A	B	C
7	2	6

	Current Allocation			Remaining need			current Available		
	A	B	C	A	B	C	A	B	C
X (P0) →	0	1	0	0	0	0	0	0	0
X3 (P1) →	2	0	0	2	0	2	0	1	0 ✓
X2 (P2) →	3	0	3	0	0	0	3	1	3 ✓
X4 (P3) →	2	1	1	1	0	0	5	1	3 ✓
X5 (P4) →	0	0	2	0	0	2	7	2	4 ✓
	7	2	6				7	2	6

→ $P_0 \rightarrow P_2 \rightarrow P_1 \rightarrow P_3 \rightarrow P_4$ safe sequence. ✓

* What happens if P_2 is demanding for 001 resources?

total resources

A	B	C
7	2	6

	Current Allocation			Remaining need			current Available		
	A	B	C	A	B	C	A	B	C
X (P0)	0	1	0	0	0	0	0	0	0
P1	2	0	0	2	0	0	0	1	0
P2	3	0	3	0	0	1			
P3	2	1	1	1	0	0			
P4	0	0	2	0	0	2			
	7	2	6						

→ deadlock

(P0) → (P1, P2, P3, P4) → deadlock.

- Deadlock Recovery :-

i) Killing the process :

(a) Killing all the processes involved in the deadlock :

(b) Kill one by one till the deadlock is cracked/resolved.

1) low priority process.

ii) % of process completion $P_1 \rightarrow 95\%$ ~~$P_2 \rightarrow 5\%$~~ ✗

iii) no of resources the process is holding.

$P_1 \leftarrow 20R$ $P_2 \leftarrow 2R$ ✓

iv) no of resources the process is requisting.

$P_1 \rightarrow 1R$ $P_2 \rightarrow 20R$ ✓

ii) Resource pre-emption : The resources from the process which are involved in the deadlock will be preempted and the pre-empted resources will be allocated to other process so that there is a possibility of ~~resource~~ recovering the system from deadlock.

→ If resource pre-emption is followed, there is a possibility of starvation.

1) selection of victim
2) Roll back.
3) starvation.

Date ___ / ___ / ___

CPU SCHEDULING - (numerical)

RBR • Basic Introduction:

Where? → In the ready state to running.

Who? → Short term Scheduler / CPU scheduler.

When? →

1) Run → termination, Run → wait, Run → Ready.

2) Wait → Ready.

3) New → Ready.

• Functionality: To allocate the CPU to the process.

• GOAL: 1) Increase the CPU Utiligation and throughput of the System.

2) Minimige the Average waiting time, Averag turn around time of the process.

RBR • Some Basic terms:

① Arrival time (A.T) → The time when the process is arrival into ready state is called as Arrival time.

② Burst time (B.T) → The time is required By the process for its execution is called as Burst time of the process.

③ Completion time (C.T) → The time when the process is completed or completes its execution is called completion time.

④ Turn around time (T.A.T) → The time difference between completion time and arrival time is called turn around time of process.

$$T.A.T = C.T - A.T$$

⑤ waiting time (W.T) → The difference between T.A.T and B.T

$$W.T = T.A.T - B.T$$

⑥ Responce time (R.T) → The difference between 1st response and arrivel time is called as Respons time.

Date __/__/____

RBR

- First Come First Served (FCFS) Scheduling:

selection criteria: Arrival time.

Mode: Non-pre-emptive.

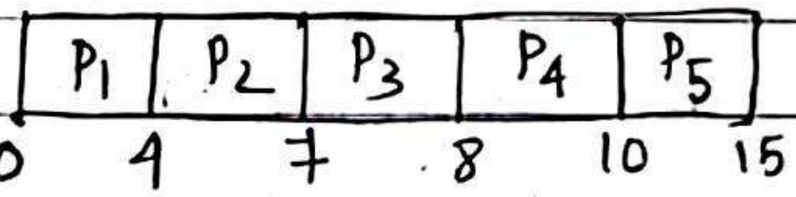
T.A.T = C.T − A.T
= W.T + B.T

- Question-1:

process no	A.T	B.T	C.T	T.A.T	W.T
1	0	4	4	4	0
2	1	3	7	6	3
3	2	1	8	6	5
4	3	2	10	7	5
5	4	5	15	11	6

W.T = T.A.T − B

Gcuantt chart



Average waiting time = $\frac{0+3+5+5+6}{5}$

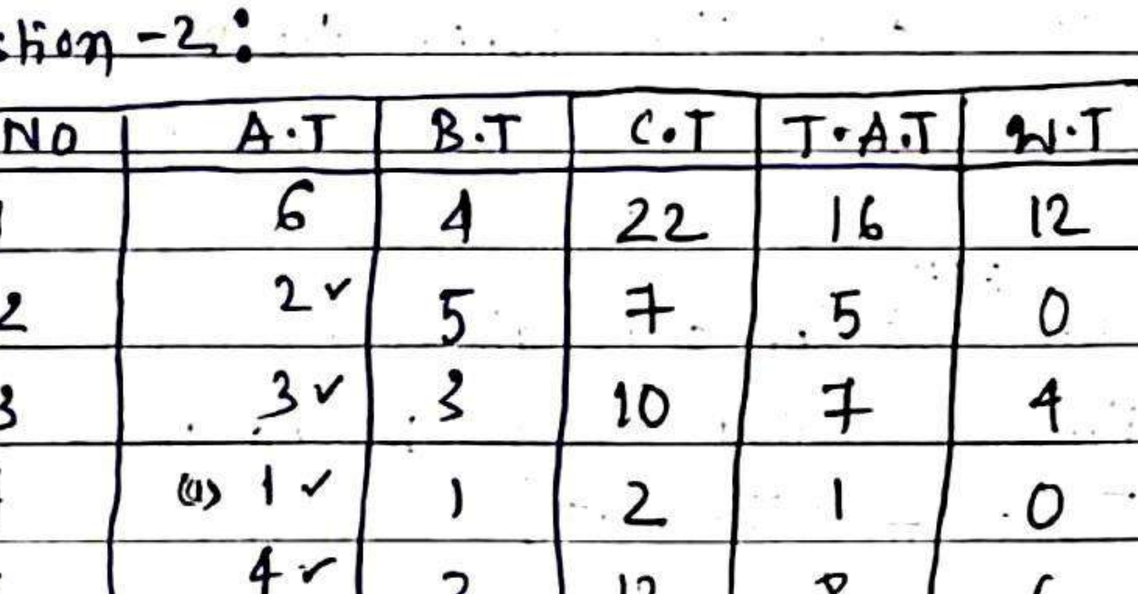
$= \frac{19}{5}$

$= 3.8 \text{ ms}$.

- Question-2:

P.NO	A.T	B.T	C.T	T.A.T	W.T
1	6	4	22	16	12
2	2	5	7	5	0
3	3	3	10	7	4
4	1	1	2	1	0
5	4	2	12	8	6
6	5	6	18	13	7

Gcuantt chart



average waiting time = $\frac{12+4+6+7}{6}$

$= \frac{29}{6} = 4.8$

Saathi

Date __/__/____

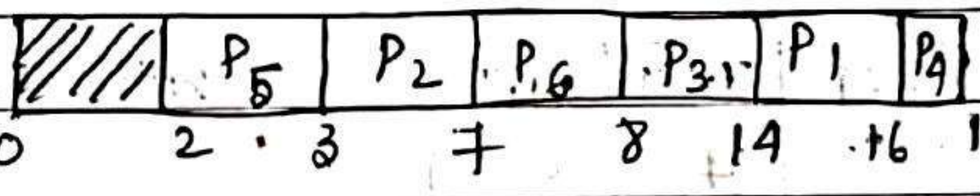
(*) If the Arrival time of the process are same/matching then schedule the process which has lowest process id.

- Question no-3:

P.NO	A.T	B.T	C.T	T.A.T	W.T
1	8 ✓	2	16	8	6
* 2	3 ✓	4	7	4	0
3	7 ✓	6	14	7	1
4	10 ✓	3	19	9	6
5	2 ✓	1	3	1	0
* 6	3 ✓	1	8	5	4

T.A.T = C.T − A.T
W.T = T.A.T − B.T

Gnantt chart



So, average waiting time = $\frac{6+1+6+4}{6}$

$= \frac{17}{6}$

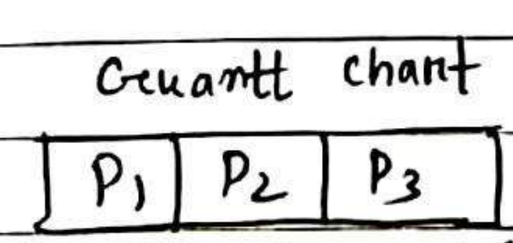
$= 2.83$ ms.

- Question no-4:

(a)

P.NO	A.T	B.T	W.T
1	0	20	0
2	1	2	19
3	2	2	20

Gnantt chart



CONvoy Effect

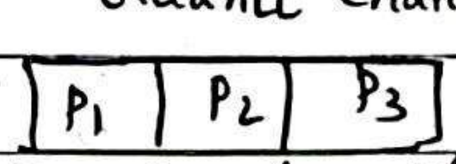
average waiting time,

$\frac{0+19+20}{3} \Rightarrow (13)$ ✓

(b)

P.NO	A.T	B.T	W.T
1	0	2	0
2	1	2	1
3	2	20	2

Gnantt chart



average waiting time,

$\frac{0+1+2}{3} = (1)$ ✓

Saathi

Date / /

RBR ✱ In FCFS, if the 1st process is having large burst time then it will be have drastic effect on average waiting time of all the process. This effect is called convoy Effect.

RBR Shortest Job first (SJF) ~~Sh~~re scheduling:

selection Criteria :- Burst time

Mod :-

→ Non-pre-emptive.

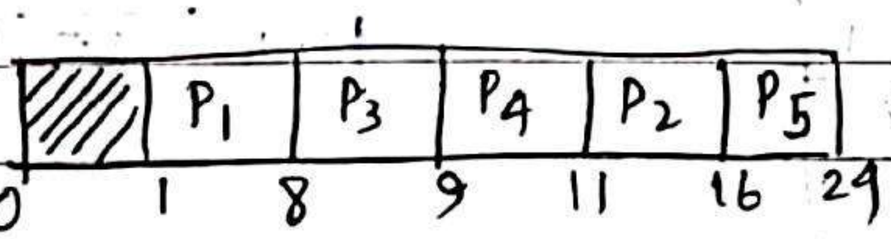
*→ ✓pre-emptive. * (important for gate)

• Question 1:- SJF Non pre-emptive.

P.NO	A.T	B.T	C.T	T.A.T	W.T
✓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

W.T = T.A.T - B.T

Gnantt Chart



average T.A.T. (Turn around time) :-

$\Rightarrow \frac{7+14+6+7+19}{5}$

$\Rightarrow \frac{53}{5}$

average W.T (waiting time),

$= \frac{0+9+5+5+11}{5} \Rightarrow \frac{30}{5} \Rightarrow 6$

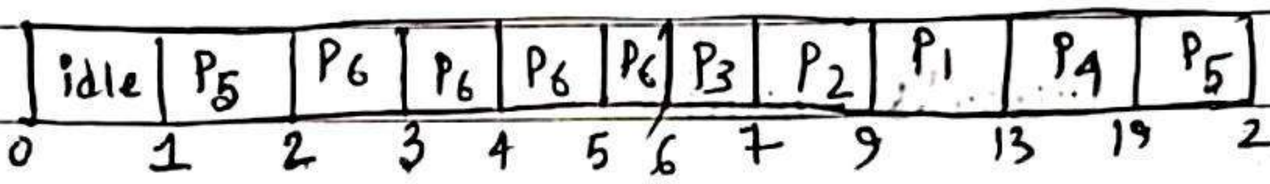
Saathi

Date / /

• Question-2 :- SJF Non pre-emptive.

P.NO	A.T	B.T	C.T	T.A.T	W.T
1	3	4	13	10	6
2	4	2	9	5	3
3	5	1	7	2	1
4	2	6	19	17	11
5	1	8	26	25	17
6	2	4	6	4	0

Gantt chart



Average T.A.T = $\frac{10+5+2+17+25+4}{6}$

$= \frac{63}{6} \Rightarrow 10.5$

Average W.T = $\frac{6+3+1+11+17+0}{6}$

$= \frac{38}{6}$

$= 6.33$.

RBR

• Question-1 :- SJF preemptive.

*. SJF is also called SRTF.

SRTF ⇒ Shortest Remaining time first.

P.No	A.T	B.T	C.T	T.A.T	W.T
1	0	7	19	19	12
2	1	5	13	12	7
3	2	3	6	4	1
4	3	1	4	1	0
5	4	2	9	5	3
6	5	1	7	2	1

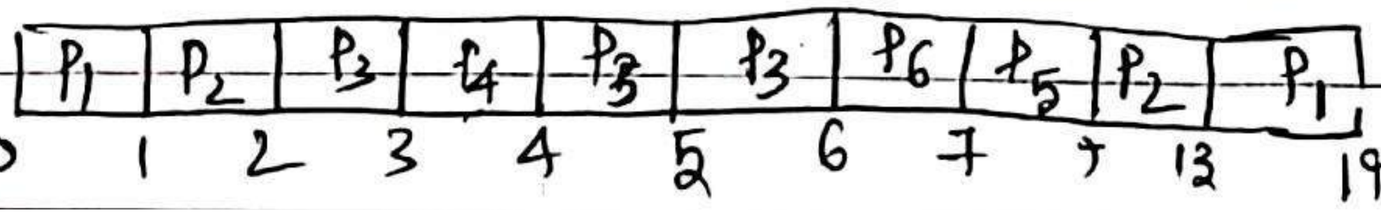
avg T.A.T =
$= \frac{19+12+4+1+5+2}{6}$

$= \frac{43}{6} \Rightarrow 7.16$

avg W.T,
$= \frac{12+7+1+0+3+1}{6}$

$= 4$.

Ganttt Chart :-



Saathi

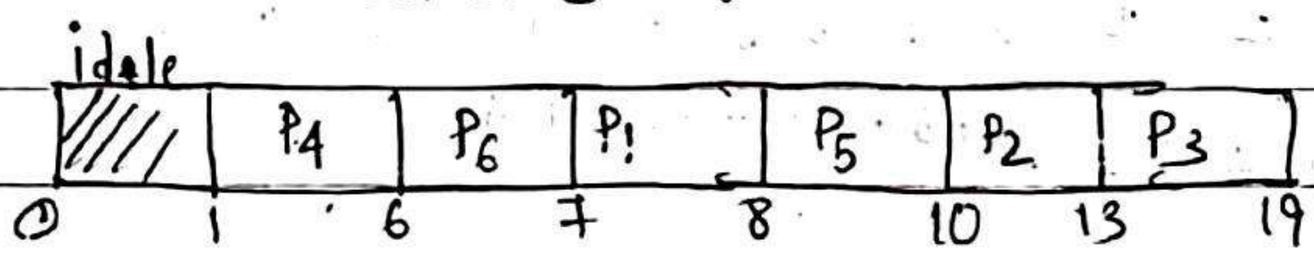
Date ___ / ___ / _____

*. Note:- If the Burst time of the process are same/matching then schedule the process which has lowest Arrival time.

• Question: SJF → Non-pre-emptive.

P.NO	A.T	B.T	C.T	T.A.T	W.T
1	6	1	8	2	1
2	5	3	19	10	7
3	4	6	19	15	9
4	1	5	6	5	0
5	2	2	10	8	6
6	5	1	7	2	1

Gruantt Chart



$$\text{avg T.A.T} = \frac{2+10+15+5+8+2}{6} \Rightarrow \frac{42}{6} \Rightarrow 7$$

$$\text{avg W.T} = \frac{1+7+9+6+1}{6} \Rightarrow \frac{24}{6} \Rightarrow 4$$

W.T = T.AT - B.T

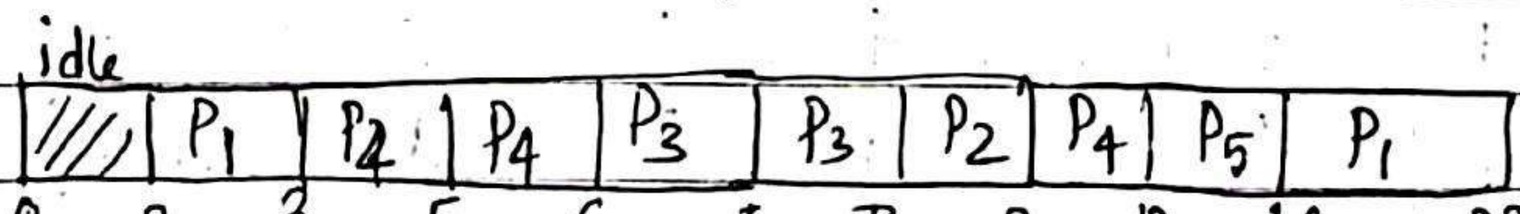
• Question :- SJF pre-emptive.

P.NO	A.T	B.T	C.T	T.A.T	W.T
1	2	8	23	21	13
2	7	1	9	2	1
3	6	2	8	2	0
4	3	6	12	9	3
5	5	4	16	11	7

$$\text{Avg T.A.T} = \frac{21+2+2+9+11}{5} = 9$$

$$\text{Avg W.T} = \frac{13+1+0+3+7}{5} = \frac{24}{5} = 4.8$$

Guantt chart



Saathi

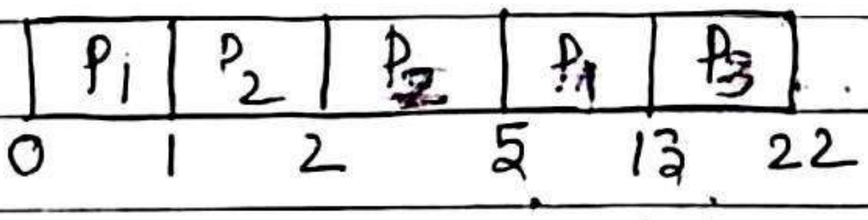
Date __/__/__

PBR

- GATE 2011 Question :- SJF pre-emptive.

P.NO	A.T	B.T	C.T	T.A.T	W.T
1	0	9	13	13	4
2	1	4	5	4	0
3	2	9	22	20	11

Gantt Chart



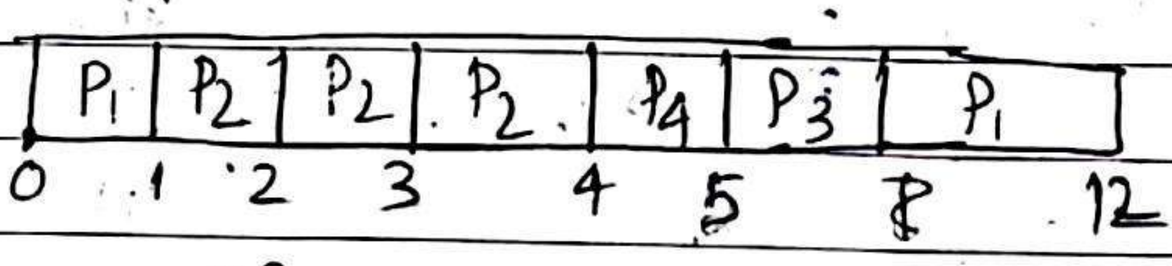
avg T.A.T = $\frac{13+4+20}{3} \Rightarrow \frac{37}{3} \Rightarrow 12.33$.

avg W.T = $\frac{15}{3} = 5$.

- GATE 2004 :→ SJF pre-emptive.

P.NO	A.T	B.T	C.T	T.A.T	W.T
1	0	5 1	12	12	7
2	1	3 2	4	3	0
3	2	3	8	6	3
4	3	1	5	2	1

Gantt chart



Avg T.A.T = $\frac{23}{4} \Rightarrow 7.72$

Avg W.T = $\frac{11}{4} = 2.75$

Throughput = $\frac{\text{No. of Process}}{\text{C.T}}$

Date __/__/____

- GATE 2006 Question :- SJF pre-emptive.
  What is the waiting time of process P_2?

P.No	A.T	B.T	C.T	T.A.T	W.T
1	0	20	20	20	0
2	15	25	55	40	(15) → Waiting time of (P_2)
3	30	10	40	10	0
4	45	15	70	25	10

Gnantt chart :

P_1	P_1	P_2	P_3	P_2	P_2	P_4

0 15 20 30 40 45 55 70

So, W.T of P_2 process is 15.

RBR

- priority Based Scheduling :
  Selection criteria :- priority. (high-priority)
  Mode :- Non-pre-emptive.

Q.1 →

P.No	A.T	B.T	priority
1	0	4	4
2	1	5	5
3	2	1	(7) → high priority
4	3	2	2
5	4	3	(1) → low-priority
6	5	6	6

C.T	T.A.T	W.T
4	4	0
16	12	10
5	3	2
18	13	13
21	17	14
11	6	0

Gnantt chart

P_1	P_3	P_6	P_2	P_4	P_5

0 4 5 11 16 18 21

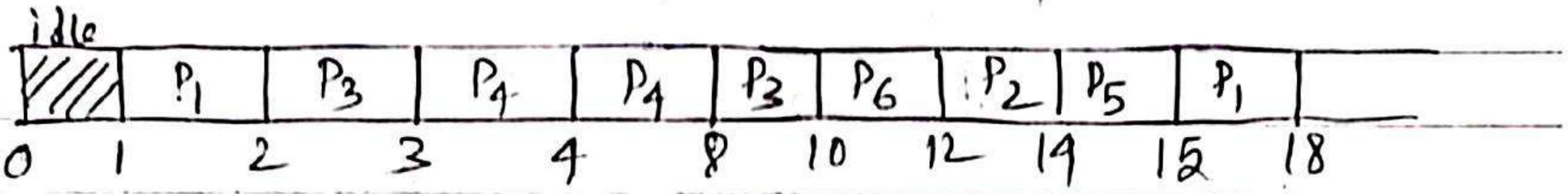
Avg Waiting time (W.T) = $\frac{0+10+2+13+14+0}{6}$

$= \frac{39}{6} \Rightarrow 6.5$

- Priority Based : pre-emptive. (Q1)

	Priority	P.NO	A.T	B.T	C.T	T.A.T	W.T
low ←	(4)	1	1	7	13	12	13
	5	2	2	2	14	12	10
	4	3	2	3	10	8	5
high ←	(8)	1	3	5	8	5	0
	2	5	3	1	15	12	11
	6	6	4	2	12	8	6

Gruantt Chart



Avg W.T, $= \frac{13+10+5+0+11+6}{6}$

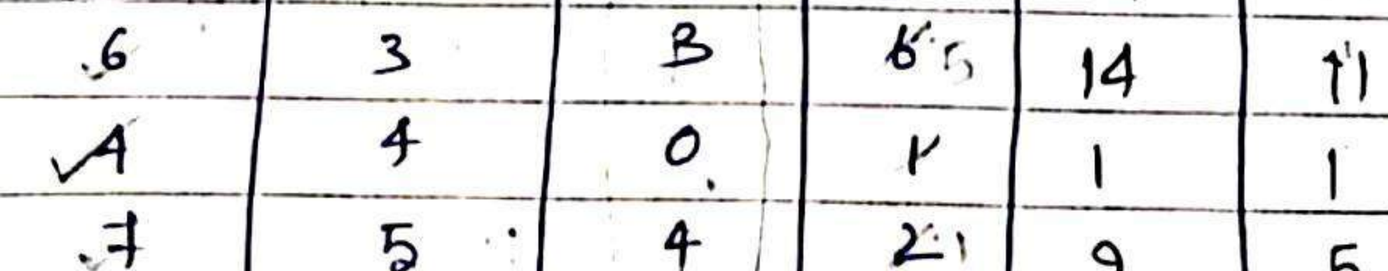
$= \frac{45}{6} \Rightarrow 7.5$.

- Priority Based : pre-emptive. (Q2)

T.A.T = C.T − A.T
W.T = T.A.T − BT

	Priority	P.NO	A.T	B.T	C.T	T.A.T	W.T
	5	1	1	4	16	15	11
low priority ←	(2)	2	2	5	21	19	14
	6	3	3	6	14	11	5
	4	4	0	1	1	1	0
	7	5	4	2	9	5	3
high priority ←	(8)	6	5	3	8	3	0

Gruantt Chart



So Avg W.T = $\frac{33}{6}$

$= 5.5$.

Saathi

Date

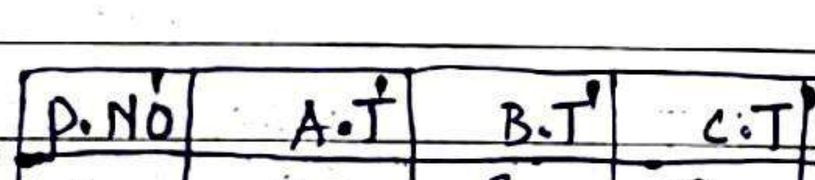
RBR

- Largest Job First : opposit of SJF.
  Selection Criteria → Brust / Mode → Non-preemptive and preemptive.

- Q-1 (Non-preemptive).

P.No	A.T	B.T	C.T	T.A.T	W.T
1	0	3	3	3	0
2	1	2	20	19	17
3	2	4	18	16	12
4	3	5	8	5	0
5	4	6	14	10	4

Gnantt chart



Avg T.AT = $\frac{53}{5}$ ⇒ 10.6 Avg W.T = $\frac{33}{5}$ = 6.6.

- Q-2 SJF → Non-preemptive.

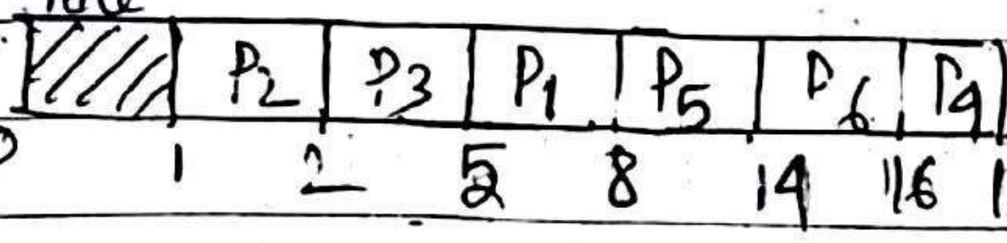
* → If the Brust time of the process are matching, then schedul the process which has the lowest arrival time.

P.NO	A.T	B.T	C.T	T.A.T	W.T
1	3	3	9	5	2
2	1	1	2	1	0
3	2	3	5	3	0
4	4	2	18	14	12
5	6	6	14	8	2
6	2	2	16	14	12

Avg T.A.T = $\frac{45}{6}$ ⇒ 7.5

Avg W.T = $\frac{28}{6}$ = 4.66

Gnantt chart



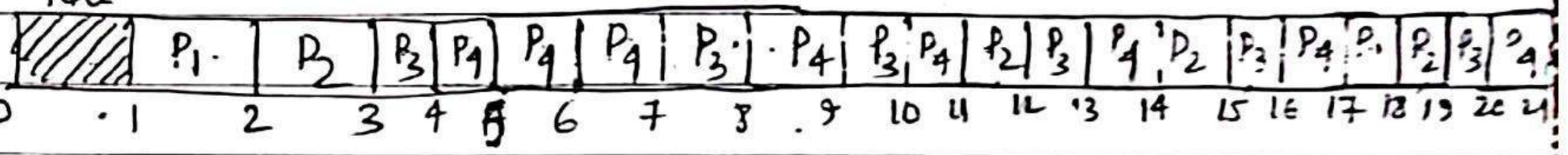
Date

* (LRTF)

- LJF → pre-emptive ⇒ (Largest Remaining time first).

P.No	A.T	B.T	C.P	T.A.T	W.T
1	1	2	13	17	15
2	2	4	19	17	13
3	3	6	20	17	11
4	4	8	21	17	9

Gruantt time



Avg T.A.T = 17

Avg W.T = $\frac{48}{4}$ ⇒ 12

(2006)

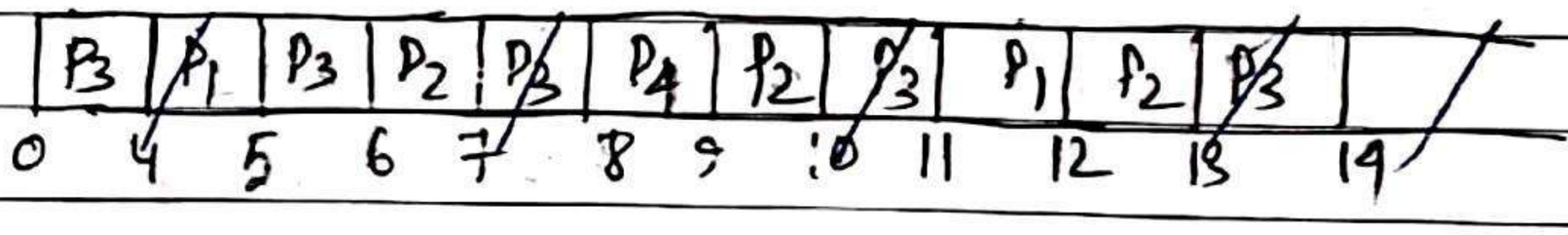
* • GATE Problem :- LRTF.

* =

What is Avg T.A.T using LRTF.

P.NO	A.T	B.T	C.T	T.A.T	W.T
1	0	21	12	12	10
2	0	4	13	13	9
3	0	8	19	19	6

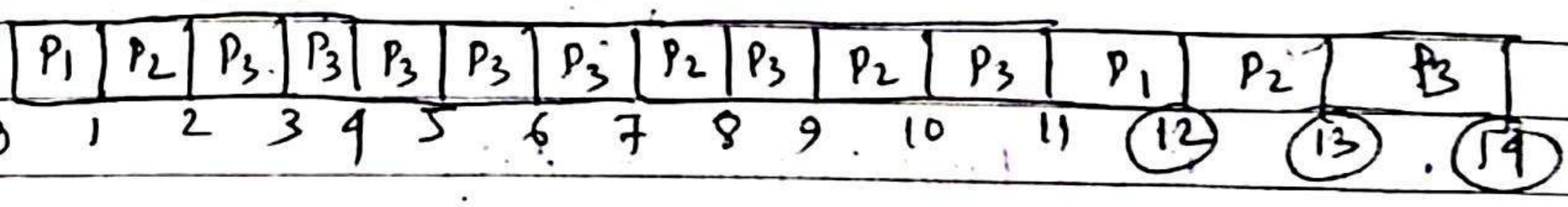
Gruant chart



Avg T.A.T = $\frac{39}{3}$ ⇒ 13.

Avg W.T = $\frac{25}{3}$ ⇒ 8.3.

Gruantt chart

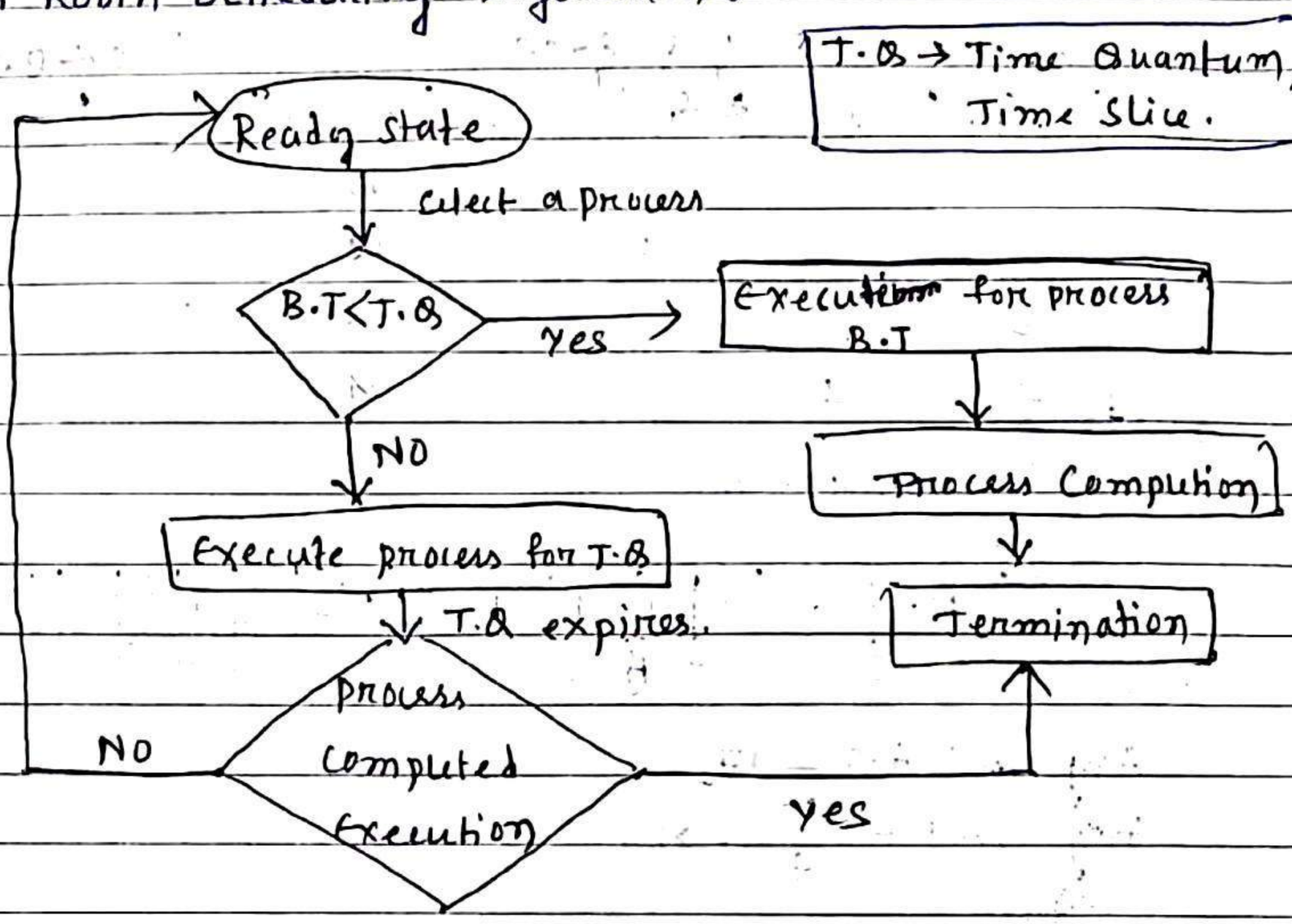


Saathi

Date ___/___/_____

RBR

Round Robin Scheduling Algorithm:



- Round Robin scheduling :- (Q-1)

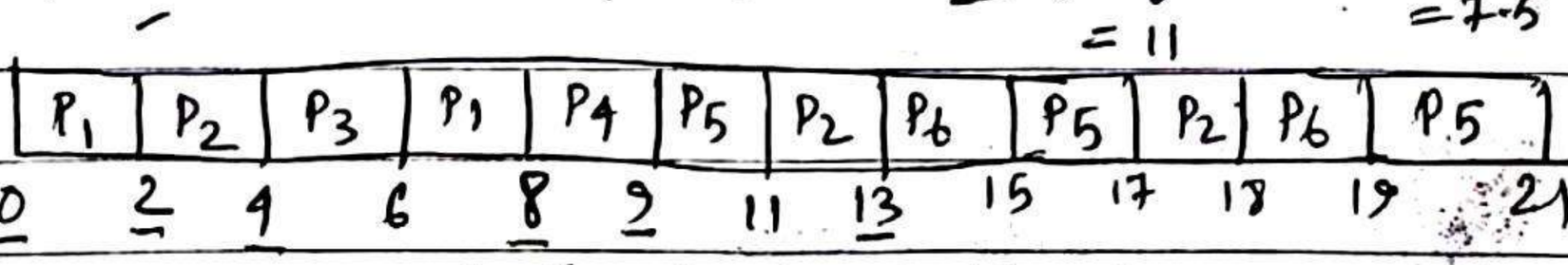
*. Selection criteria: Time Quantum or Time slice.

*. Mode : pre-emptive.

(T.Q = 2)

P.No	A.T	B.T	C.T	T.A.T	W.T	R.T
1	0	4	8	8	4	0
2	1	5	18	17	12	1
3	2	2	6	4	2	2
4	3	1	9	6	5	5
5	4	6	21	17	11	5
6	5	3	19	14	11	10

Avg R.T = $\frac{23}{6} \Rightarrow 3.8$ Gruantt chart [Avg = $\frac{66}{6}$ = 11 [Avg = $\frac{45}{6}$ = 7.5



Ready Queue → P1, P2, P3, P1, P4, P5, P2, P6, P5, P2, P6, P5

(flow chart)

Response time = First Response = A.T

- Round Robin Scheduling :- (Q-2)

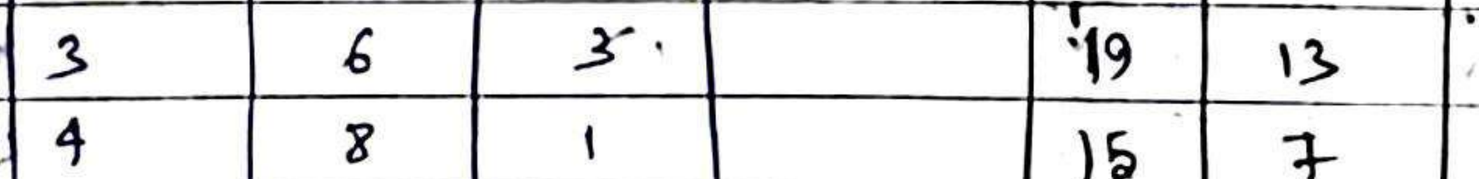
T.Q = 3

P.NO	A.T	B.T	C.T	T.A.T	W.T
1	5	5	32	27	22
2	4	6	27	23	17
3	3	7	33	30	23
4	1	9	30	29	20
5	2	2	6	4	2
6	6	3	21	15	12

(time sharing process)

Ready Queue → $\underset{\checkmark}{P_4}, \underset{\checkmark}{P_5}, \underset{\checkmark}{P_3}, \underset{\checkmark}{P_2}, \underset{\checkmark}{P_4}, \underset{\checkmark}{P_1}, \underset{\checkmark}{P_6}, \underset{\sim}{P_3}, \underset{\sim}{P_2}, \underset{\sim}{P_4}, \underset{\sim}{P_1}, \underset{\sim}{P_3}$

Guantt chart



Avg-T.A.T = $\frac{128}{6} \Rightarrow 21.3$

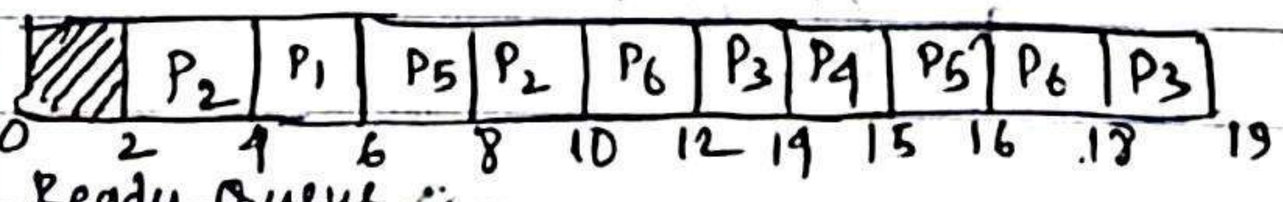
Avg-W.T = $\frac{96}{6} \Rightarrow 16$

- Round Robin scheduling :- (Q-3)

TQ = 2

P.NO	A.T	B.T	C.T	T.A.T	W.T
1	3	2	6	3	1
2	2	4	10	8	4
3	6	3	19	13	10
4	8	1	15	7	6
5	4	3	16	12	9
6	5	1	18	13	9

Guantt chart



Ready Queue :

→ $\underset{\checkmark}{P_2}, \underset{\checkmark}{P_1}, \underset{\checkmark}{P_5}, \underset{\checkmark}{P_2}, \underset{\checkmark}{P_6}, \underset{\checkmark}{P_3}, \underset{\checkmark}{P_4}, \underset{\checkmark}{P_5}, \underset{\checkmark}{P_6}, \underset{\checkmark}{P_3}$

avg = $\frac{54}{6}$ = 9.3

avg = $\frac{39}{6} \Rightarrow 6.5$

Saathi

Date ___ / ___ / _____

(1)

• Question :- Consider a system with 4 proceses P_1, P_2, P_3, P_4. The Burst time Requirements of these procers are 4, 1, 8, 1 respectively. Then what is the Completion time of procers P_1 assuming Round Robin scheduling.

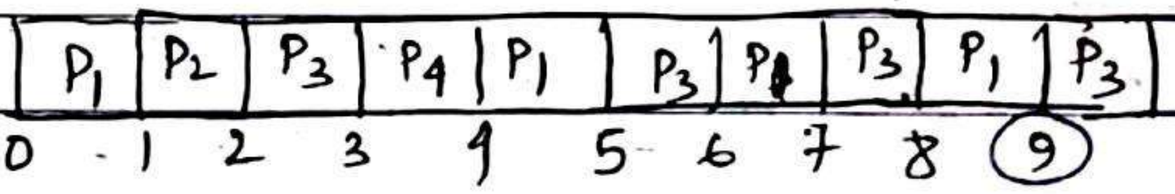
(T.Q | Ts = 1)

→ $P_1 \rightarrow 4$ ✓ Let, Arrival time of all procers '0'.

$P_2 \rightarrow 1$ ✓ then,

$P_3 \rightarrow 8$ ✓ Ready queue → $P_1, P_2, P_3, P_4, P_1, P_3, P_1, P_3, P_1, P_3$

$P_4 \rightarrow 1$ ✓



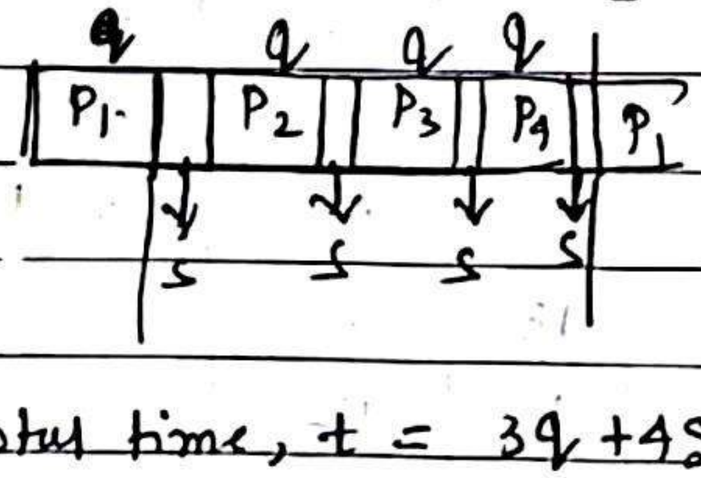
So, the Completion time of P_1 is = 9.

(2)

• Question :- Consider the process which has 'n' processes sharing the cpu in Round Robin fashion. The Context switching time is 's' units. Then what must be the time Quantum 'q' such that each process is guaranted to get its turn at the cpu for every t seconds of time.

(i) $q = \frac{t-ns}{n+1}$ (ii) $q = \frac{t+ns}{n-1}$ (iii) ✓ $q = \frac{t-ns}{n-1}$ (iv) $q = \frac{t-ns}{n+1}$

→ Let, n=4 (P_1, P_2, P_3, P_4)

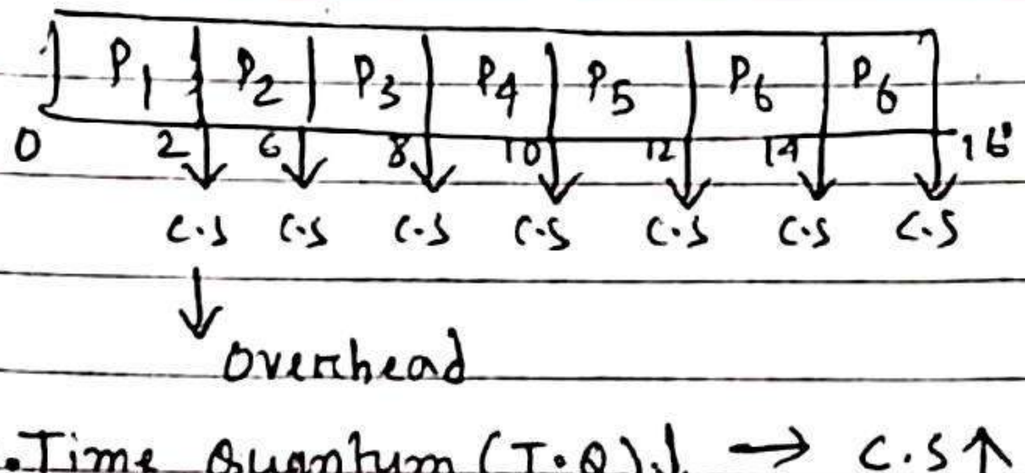


total time, $t = 3q + 4s$

So, for n processes - $t = ns + (n-1)q$

$$q = \left(\frac{t-ns}{n-1}\right)$$

Date ___ / ___ / ___



*. Time Quantum (T.Q) ↓ → C.S ↑

→ Response time decrease.

*. T.Q ↑ → C.S ↓

→ Response time Increase.

*. T.Q ↑↑↑ → FCFS.

- Round Robin scheduling conclusion:

→ If the time Quantum is less, then the no of context switching -es will Increase and response time will be less.

→ If the time Quantum is large, then number of context switches will decrease and response time will be more.

→ If the time Quantum is very very large, the algorithm degenerates to FCFS Algorithm.

→ Round Robin is used to decrease the Response time.

RBR

Highest Response Ratio Next (HRRN)

selection criteria → Response Ratio. (high)

Mode → Non-pre-emptive.

$$\text{Response Ratio (RR)} = \frac{W+S}{S}$$

W → waiting time.

S → service time or Burst time.

The HRRN favours the shortest jobs and also limits the waiting time of largest jobs.

Saathi

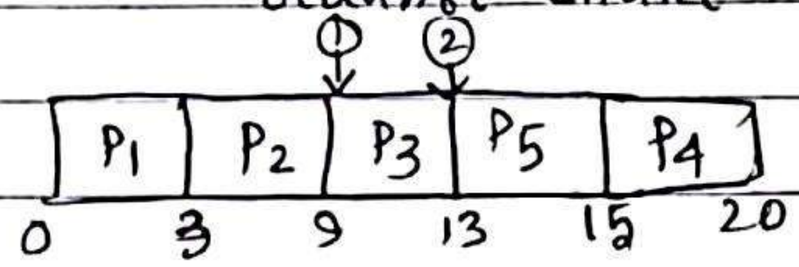
$W.T = T.A.T - B.T$

- Question 1: ~~HNA~~ HRRN.

P.NO	A.T	B.T
✓1	0	3
✓2	2	6
✓3	4	4
4	6	5
✓5	8	2

C.T	T.A.T	W.T
3	3	0
9	7	1
13	9	5
20	14	9
15	7	5

Gruantt chart



avg $= \frac{20}{5} \Rightarrow 4$

(i) $RR_3 = \frac{W+S}{S} = \frac{(9-4)+4}{4} = 2.5$

$RR_4 = \frac{W+S}{S} = \frac{(9-6)+5}{5} = \frac{8}{5} \Rightarrow 1.6$

$R.R_5 = \frac{(9-8)+2}{2} = \frac{3}{2} \Rightarrow 1.5$

(ii) $RR_4 = \frac{(13-6)+5}{5} = \frac{12}{5} = 2.$

$RR_5 = \frac{(13-8)+2}{2} = \frac{7}{2} = 3.5$

So, Average waiting time (W.T) = 4.

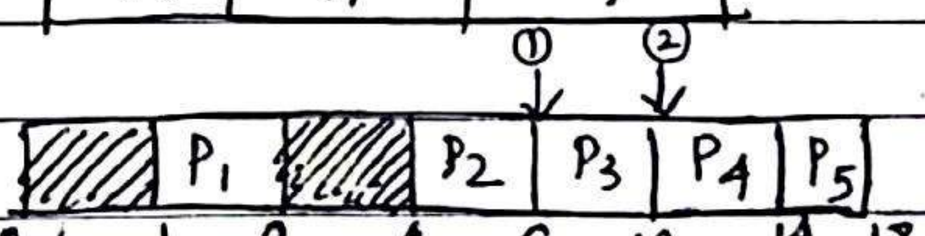
- Question 2: HRRN.

P.NO	A.T	B.T
✓1	1	1
✓2	4	2
✓3	5	6
4	6	2
5	7	9

C.T	T.A.T	W.T
2	1	0
6	2	0
12	7	1
14	8	6
18	11	7

avg W.T $= \frac{14}{5}$
$= 2.8.$

Gruantt chart



(i) $RR_3 = \frac{(6-5)+6}{6} = \frac{7}{6} = 1.$

$RR_4 = \frac{(6-6)+2}{2} = 1$

(ii) $RR_4 = \frac{(12-6)+2}{2} = 4$

$RR_5 = \frac{(12-7)+4}{4} = \frac{9}{4} = 2.1$

Saathi

Date

RBK

CPU Scheduling & I/O Scheduling -

• Question :-) what is completion time of process P_1, P_2, P_3 using SRTF.

P.NO	A.T	cpu time	Yo time	cpu time
1	0	1	2	2
2	1	2	4	5
3	2	3	6	8

Note :-

(1) The process first spends cpu time followed by I/O time, followed by cpu time.

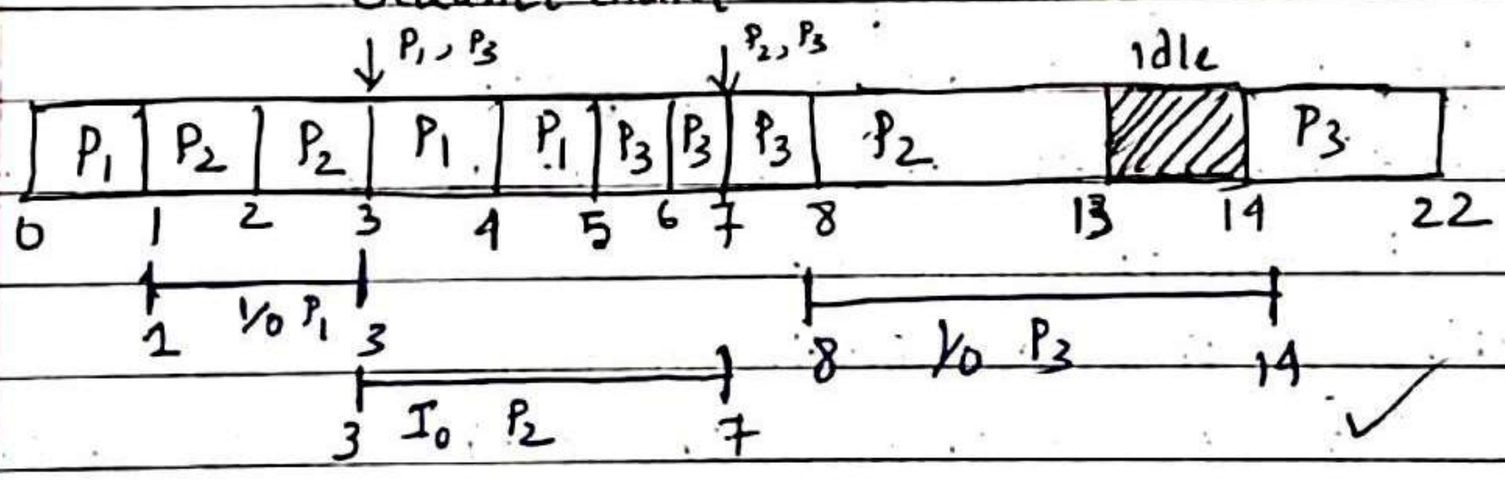
(2) I/O time of the process can be overlapped as much as possible.

→

P.NO	A.T	cpu time	Yo time	cpu time	C.T
1	0	1	2	2	5
2	1	2	4	5	13
3	2	3	6	8	22

← Completion time

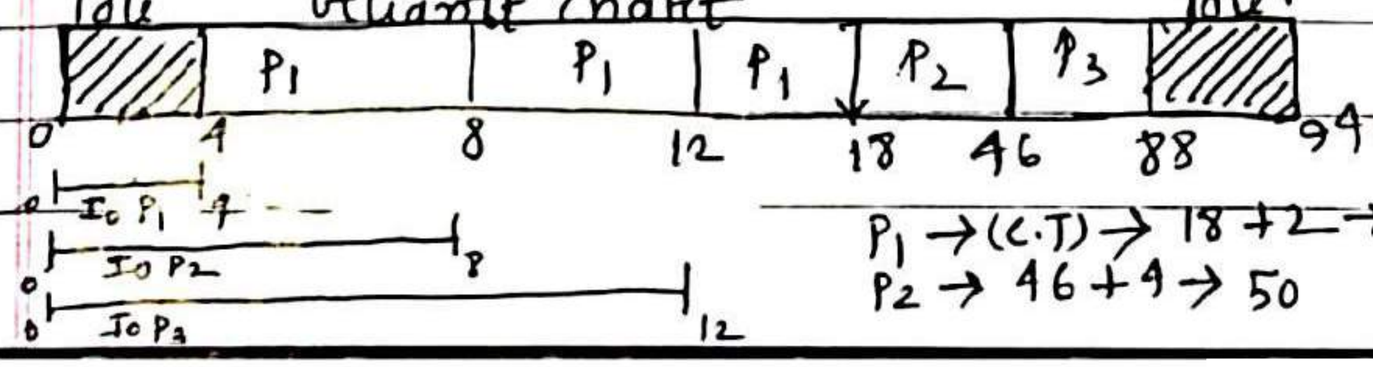
Gruantt chart



• Question 2 :-) Using SRTF

P.NO	A.T	I/O time	CPU time	Yo time	C.T
1	0	4	14	2	
2	0	8	28	4	
3	0	12	42	6	

Idle Gruantt chart Idle



$P_1 \rightarrow (C.T) \rightarrow 18+2 \rightarrow 20$

$P_2 \rightarrow 46+4 \rightarrow 50$

$P_3 \rightarrow 88+6 \rightarrow 94$

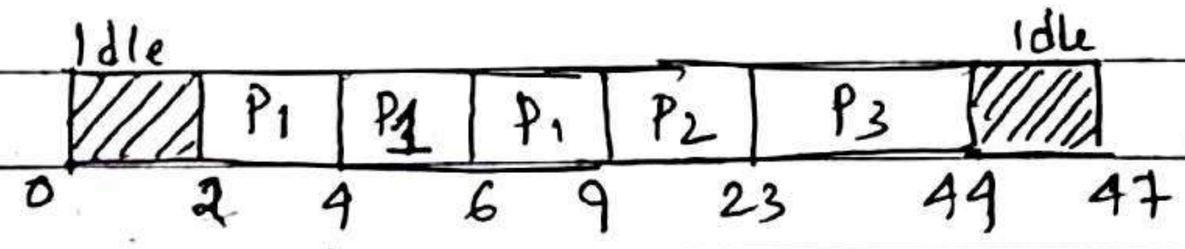
Saathi

Date: / /

GATE problem: 2006 Question. → SRTF.

P.NO	A.T	B.T Total	I/O time(20%)	CPU time(70%)	Yo time(10%)
P_1	0	10	2	7	1
P_2	0	20	4	14	2
P_3	0	30	6	21	3

Gruant chart



So,

completion time of $P_1 = 9+1 \Rightarrow 10$

$P_2 = 23+2 \Rightarrow 25$

$P_3 = 44+3 \Rightarrow 47$

Total ~~time~~ C.P.U idle time, $\Rightarrow 2+3$

$\Rightarrow 5 \Rightarrow \frac{5}{47} \times 100 \Rightarrow$ 10.6.%

Multi processors Scheduling:

Question:- Consider the dependancy graph between the process. At what time all the processes complete there execution using 2CPU's (i.e two processors).

Note: (1) Use Non-preemptive mode.

(2) one process cannot share the 2 cps at the same time.

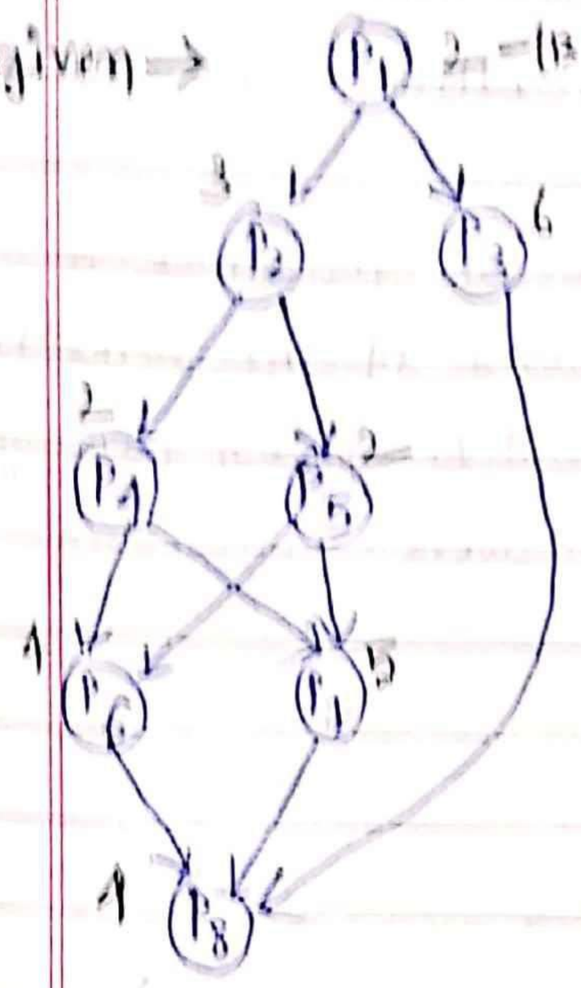
(a) 17 (b) 18 (c) 19 (d) 20.

given -

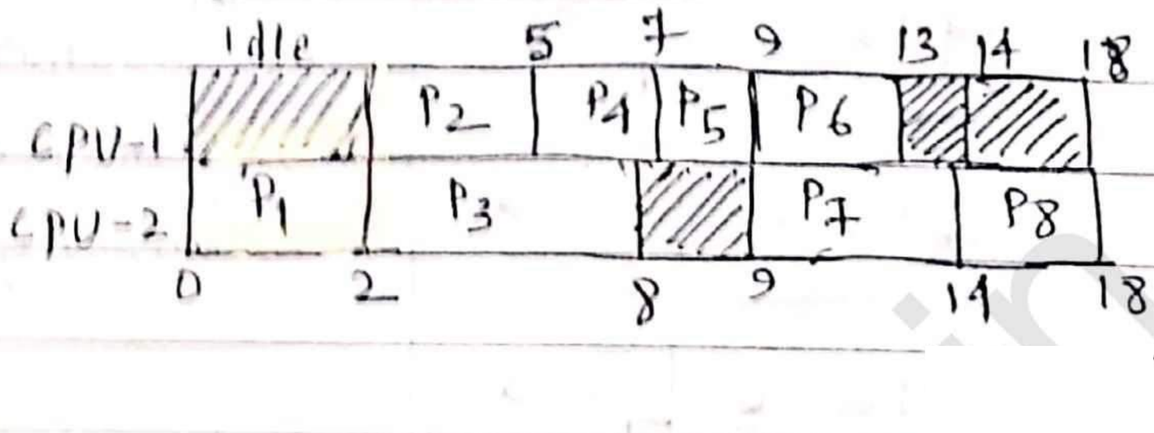
(Next page).

Date

given →



Gantt chart



So, Completion time → 18.

Multilevel Queue Scheduling :

(Ready Queue)

RQ1 →

RQ2 →

RQ3 →

RQ4 →

→ Depending on the proprity of the process, In which particular Queue, the process has to be placed will be decided.

→ The high priority process will be placed in the tope level ready Queue and low priority process will be placed in the Bottom level ready queue.

→ only after completion of all the process-es from the top level ready queue, the further level ready queue process will be scheduled.

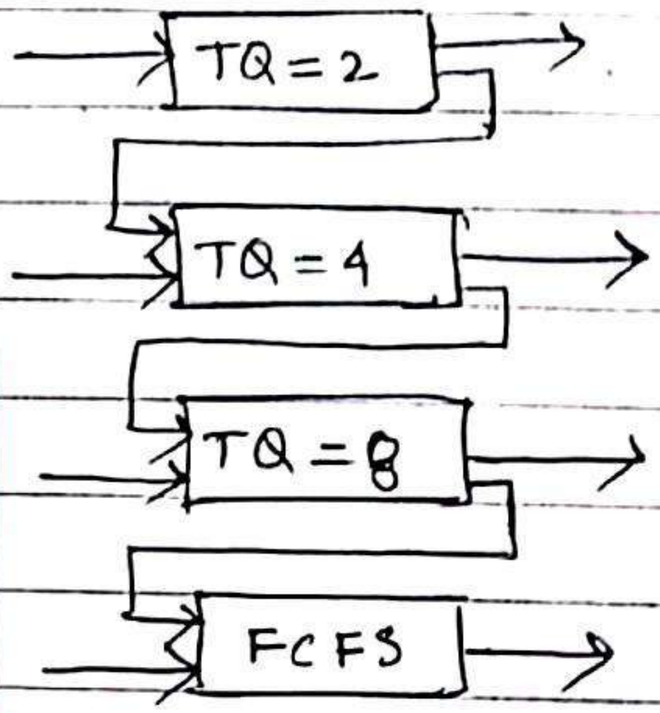
→ If this is the strategy which is followed then the processes which are placed in the bottom level ready queue will suffer from STARVATION.

STARVATION : The Indefinate waiting of a process is called as STARVATION.

→ It is also possible to use different scheduling algorithms in the different ready queue.

RBR

MULTILEVEL FEEDBACK QUEUE SCHEDULING –



→ Algorithm avoids the problem of starvation and at the same time preference will be given to high priority process.

• Question :- Consider the system which has CPU Bound ~~Bound~~ process which requires Burst time of 40 time Units. Multilevel feedback queue scheduling is used. The time quantum is 2 units and it will be Incremented by 5 units in each level. How many time the process will be Interrapted and in which queue, process will complete the execution ?

(a) 4, 5 (b) 5, 6 (c) 3, 4 (d) 5, 5.

→

		remain
→ TQ-2 — ①	2	38
→ TQ-7 — ②	9	31
→ TQ-12 — ③	21	19
→ TQ-17 — ④	38	2
→ TQ-22 — ⑤	40	0

• 5th level complete.
• 4 time Interrapted.

Date / /

- Conclusion :-

Algorithm	Starvation
1) FCFS	NO
2) SJF (NP)	yes
3) SRTF	yes
4) R.R	NO
5) LJF (NP)	yes
6) LRTF	yes.
7) HRRN	NO
8) priority (NP)	yes.
9) priority (P)	yes
10) MLQ	yes
11) MLFQ	NO