A communication link must exist

→ Message passing is implemented using a queue.
Can go beyond Producer - consumer as it is done among multiple systems, Here we need send and receive type of fns.

⇒ Direct or Indirect → messages are send and received from mailboxes or ports

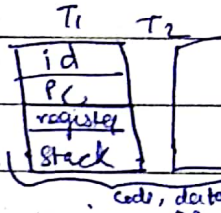
Direct: If we specify a receiver.
send(A, msg)
receive(B, msg)

Indirect: Send and receive message without any more information
send(msg)
receive(msg)

→ Direct or indirect communication
→ Sync or async comm. → Blocking and non blocking
→ Automatic or explicit buffering → Zero capacity of the queue.
Bounded Capacity
Un " "

17/01/18

Threads :- (Thread ID, P.C, a register set and a stack) Process →



→ If we perform multiple processes using Multiple processing is called heavy weight process.
→ Whenever a process makes a subprocess, some values are duplicated.
→ Increasing memory footprint of our program due to subprocess.
↳ heavy weight coz it copies its data at multiple places.
Multiple process model memory footprint will be much more than Multithread model
↳ less memory for same no. of users or processes.
→ Extra redundant data is removed in case of multithreading
→ Multi-threading model helps us in improving the performance.
↳ supported by libraries

fork
wait
exit
→ for multiprocess

Teacher's Signature.....

Threads avoid redundant informations production.

Code, Data, registers, Stack → at same place

local variables → for all process seperate

DATE : / /
PAGE NO. :

User level thread

Thread supported by a library

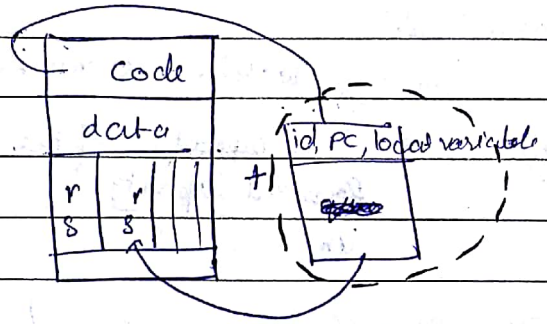
Kernel level thread

Thread supported by a O.S

→ A library can excess kernel level thread ; we can excess library directly.

→ C library for multithreading is Pthread.

→ Java Base threading library



this will be a queue of process and the one which is in execution will be their in memory space.

→ It can handle multiple threads running concurrently using time slicing.

time slicing for 3 different threads → | T_0 | T_1 | T_2 | T_0 | T_1 | T_2 | → Concurrent execution.

Multicore System

→ Devised coz of limitation of fabrication technology.

→ fabrication of multiple small process or a chip.

may be of less speed.

→ Each core will be acting as a seperate thread.

→ Generally the is a common cache shared between these multiple cores.

parallel execution :-

Each core can run one thread.

(2) Parallel → more than one process can be in execution.
(3) Internally each of the thread executes as a kernel thread.

DATE: / /
PAGE NO.:

	T_1		← can handle more than one thread.
C_0	T_0	T_4 / T_0	
C_1	T_1	T_2	T_1
C_2	T_2 / T_3	T_2	T_2 / T_4

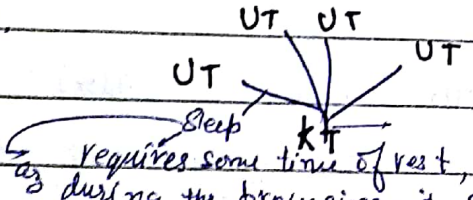
→ 4 Core with 2 threads — effectively, 8 Cores
4 processor capable of handling 2 threads each.

Each core is having 2 threads } this will be slightly performance degrade w.r.t to 8 different cores.

→ How a user model is mapped with a kernel thread?

1st model → Many to one

One kernel thread was able to manage more than one user thread.

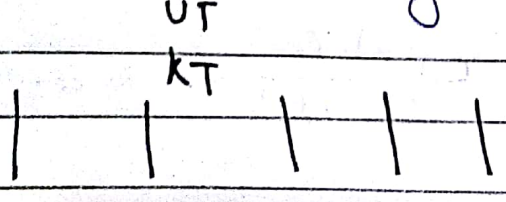


requires some time of rest, as during the browsing it requires some resources so sleep is executed.

Drawback → if one of the thread in executing goes to blocked state than all the other threads will also go into blocked state, and as only one thread can access the kernel at a time, multiple threads are unable to run in parallel on multiple core systems.

2nd model → One to One

For each user thread internally a kernel thread is formed.



→ Each user thread can be scheduling on seperate KT.

was capable of managing the blocking problem of many to one model.

→ The no. of kernel threads in many to many is decided in regard with a particular application or a particular machine.
→ either one to one or many to many is preferred

DATE : / /
PAGE NO. :

Drawback → extra burden, for each thread you need to create another kernel thread. Sometime the no. of threads that can be made is restricted.

3rd mode → Many to many model

① Less kernel thread as compared to user thread.
② developers can create as many threads as they want. + Two level model.

Next class → How to create these threads in our program?

22/01/18

Multi Threading Programs :→

```
#include <pthread.h>                                  % a.out 10
int Sum = 0;
int main (int ac, char * av[])
{
    pthread_t tid;
    pthread_attr_t attr;
        |
    attribute
    pthread_attr_init (& attr);
To initialize the variables with default values, this is a fun →
    // Now we are ready to create a thread
    pthread_create (& tid, & attr, computeSum, av[1]);
```

per this time two
e will be
eads in the
ogram, main thread
and child thread.

4 arguments →

during creation time, if some priority is changes then by this address we have to change the associated attributes.

→ `void * computeSum (void *p)`
restriction

```
{   int i, n = atoi (p);
    for (i = 1; i <= n; i++)
        sum += i;
    pthread_exit (0);
}
```

→ To convert a string representation of a no. into a numeric form.

exit status of our thread

Teacher's Signature.....

The main thread will be block with this line and the main will not Proceed unless the computeSum fxn is exited.

```
pthread_join (tid, NULL);
```
→ thread represented by the tid has to be completed first.

```
printf ("the sum of %d natural no. = %d ", atoi(av[1]), sum);
return 0;
}
```

We can make available and here pass the address of the variable to store the exist status of the computeSum fxn.

Multiple threads program :- Each thread has to display its no. as many times as its number.

```
# include <pthread.h>
int main()
{ pthread_t tid[5];
  for (i=1; i<=5; i++)
{ pthread_create (&tid[i], NULL, display, &i);
  pthread_join (tid[i], NULL);
}
  void * display
{ int i, n = atoi (p);
  For (i=1; i<=n; i++)
  { printf (" in thread %d", n);
  }
  pthread_exit (0);
}
printf ("All Childs are finished");
  Return (0);
}
```

→ A condition when the parent thread gets terminated and child thread is still executing, the child thread is called a Orphan thread and thread will go in some unusual state.

Teacher's Signature.....

Date / /
Page No.

Thread

A multi-core processor is a single computing system with two or more independent processing units called core.
A multi core processor implements multiprocessing in a single physical package.

→ Benefits of multithreading :→

1. Responsiveness
2. Resource sharing :→ Threads share memory and resources of the process to which they belong.
3. Economy
4. Scalability :→ benefits of multithreading can be even greater in a multiprocessor architecture.

Challenges

1) identifying tasks
2) Balance
3) Data splitting
4) Data dependency → synchronization
5) Testing and debugging

Thread management is done by the thread library in user space.

A thread library provides the programmer with an API for creating and managing threads.

→ POSIX Pthread, Window and Java.

→ Pthreads refers to POSIX standard defining an API for thread creation and synch. This is a specification for thread behavior, not an implementation.

→ int pthread_join (pthread_t thread, void ** val_ptr);

① the pthread_join() function suspends execution of the calling thread until the target thread terminates.

② If value_ptr is not NULL, then the return value of thread is stored in the location pointed to by value_ptr.

③ If successful this fun returns 0 else an error no. is returned to indicate the error.

Compilation :→ gcc thread1.c -lpthread

If we put the join statement in a seperate for loop in main

```
For (i=1 ; i<=5 ; i++)
↳ pthread_join (tid[i], null);
```

→ the sequence of the threads will be random.

```
↳ Printf ("All childs are finished")
  return (0);
  }
```

→ If we do not want to change the attr variable we can replace it by NULL

```
Pthread_create (&tid[i], NULL, display, i);
                    ↓
              inplace of attr.
```

→ Process Synchronization
Can be done for both multiple process and multiple threads.

01/18 Write End and Read End

→ We will be have two different variable, one to write and the other to read.

→ A Pipe can be unidirectional pipe or bidirectional pipe.

Half → Duplex → If A is writting then B cannot write at the same time.

full Duplex → Both can write at the same time

Different types of pipes :→

The process should be the parent one.

Ordinary pipe → multiple process should show parent child relationship

Named pipe → the processes are independent.
(FIFO, exist after the process termination also, bi directional)

Teacher's Signature...

Pipes mostly used in the process for doing, the o/p of one command is the i/p of another program.

DATE : / /
PAGE NO. :

→ ls -l → gives out all the information abt the process in different columns.

→ ls -l ; cut -d" " -f4,5- → A child process automatically inherits a pipe of parent.

o/p of one command passed to another command.

-d → delef

-f → 5 → colum → assumed.

fn fs

filename file size.

fd → file discriptor

System call → to create a pipe :→

pipe (fd)

fd[0] → read end of file → used by the process which is going to read.

fd[1] → write_end

→ Read here is a blocking call, as the Process has to wait unless it gets the data from the file.

handled by the O.S in the time out way.

Code :→

```
#define buffer_size 30   → Represents size of each message.
#define read_end 0
#define write_end 1
int main()
char write_msg[buffer_size] = "dummy msg";
char read_msg[buffer_size];
int main ()
{
  int fd[2];
  pid_t pid;   (fork part)
  if (pipe(fd) == -1)   → the cond: when the system is not able to open a file.
  { printf("error in pipe open");
    exit(1);
  }
```

Teacher's Signature .....

Ideally we should close the file handles we open

```
pid = fork();   → pid < 0 → error
                  pid = 0
if (pid == 0)   → for child and else is for parent
{
write(fd[write_end], write_msg, strlen(write_msg));
close(fd[write_end]);
}
else
{
read(fd[read_end], read_msg, buffer_size);
close(fd[read_end]);
printf("The message is %s", read_msg);
}
}
```

once you finish writing then only you should close the write end.

release the space occupied by the pipe.

If the write_msg is greater than the buffer size then, read_end will read the first 20 characters.

→ If we apply a loop (for) the condition should be applied on length of read_msg as the read msg length is 0 then this is no more execution.

```
if (strlen(read_msg) == 0)
    break;
```

→ A two way communication can be implemented using multiple pipes

Teacher's Signature

Under non-preemtive scheduling, once the CPU has been allocated to a process, the process keeps the CPU until it releases the CPU either by terminating or by switching to the waiting state.

DATE: / /
PAGE NO.:

5/9/18

CPU Scheduling :-

→ In each state there is a queue, and this scheduling activity corrdinates the allocation of a processes to a particular processor.

Four conditions when the CPU Scheduling decision will be taken :→

1) Running → waiting
The process voluntarily leaves the CPU. After it realeas from waiting state, it will go to ready state and wait for it's chance.

2) Running → ready
It is cg of some external condition, cg of some interrupt or else. The processor deals with interrupt service routine and then again process start to run.

3) waiting → ready (Rare case)
This will be the case when there is one process, and CPU is waiting. (for eg, at completion of I/O)

(4) Termination
Since the current running process completes, the process need to be terminated.

There are two types of scheduling algorithms :-

① Non-Primitive Preemptive or cooperative scheduling
If scheduling decision is taken out of 1st or 4th condition then it is non-primitive algo. will be applied.

Once CPU is allocated to a process, it will release the CPU only in 1st or 4th condition.
For case 1) → the process on their own only releases the processor.

② ~~Primitive~~ Preemptive
We can take CPU based on some calculations or criteria.
May be based on Priority.

Teacher's Signature.....

Gives a particular time slice to each process

DATE: / /
PAGE NO.:

The bases may be priority or time slice bases etc.
Only for a particular time a process is runned.

→ There should be some criteria for selection of processes to allocate a processor.

Content Switch →

Whenever we allocate process 1 → process 2 it is Context Switch.

→ for Content Switch we need to save the current state of process 1, value of variables, line no., PC, memory situation, CPU registers so that we can resume a process again afterwards.

→ First we will save the current info of process 1. than retrive the state of next process and then start execution.

Dispatcher → Deals with the activity associated with Content switch. A dispatcher is the module that gives control of the CPU to the process selected by short-term scheduler.

→ We will not see any time lag b/w this switch.

→ If this Content Switch is not very frequent, then this delay is called as Dispatch Latency → Stop one Start another

→ The time taken by Content Switch is called delay latency

μsec ↔ few sec

→ There are many parameters in which we can decide that which Scheduling algo. is better for our system:→

1. CPU utilization (0 – 100%)

↑ CPU utilization much better scheduling.

Throughput → It is the measure of work performed by a processor. Number of processes completed is a time unit

Teacher's Signature.....

CT - AT = BT + WT
(turnaround time)

① It is desirable to ↑ CPU utilization & throughput ↓ TAT, WT, RT.

② For interactive sys ↓ variance in Response time

Submission time – completion time.

3. Turnaround time → This refers to the difference of a Submition time to its completion time. (AT)

→ Its best possible value → Actual processor time of a process. (Execution)

→ Smaller the value of turn around time better is the algo.

4. Waiting time :→ the time spent in non-execution activity.
Waiting time is the sum of periods spent waiting in the ready queue.

* Turn around time – actual execution time of process ⇒ waiting time
(B.T)

5. Response Time :- From submission time to the 1st response time.
First – AT
Response time

The last 3 points are all process dependent parameters, thus we can take the best their ~~avg~~ average.

Non primitive algorithms :-

① FCFS (first come first serve)

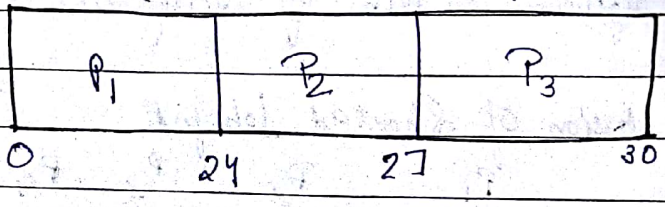
Assuming there is no waiting, then will be starting and ending of a process.

P_1	P_2	P_3
24 unit	3 unit	3 unit

→ Total (execution time / burst time) → actual time required by the process to execute.

Gantt

$(0 + 24 + 27)/3$ → waiting time. $= 17$

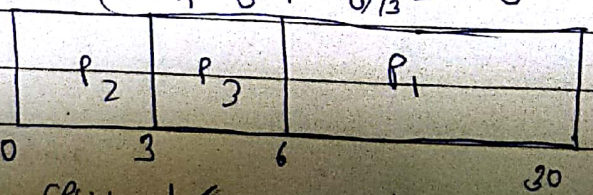


Influenced by sequence of processes

P_2 P_3 P_1

$(0 + 3 + 6)/3 = 3$ → waiting time.

Gantt Chart



Process eats up CPU bound time. ← Convoy effect → All the other processes wait for the one big process to get off the CPU.

Teacher's Signature.....

DATE : / /
PAGE NO. :

Algorithm (shortest next CPU burst)

② SJF → Shortest job first

Processes are sorted on the bases of their execution time, etc.

Execution time :→ (burst time)

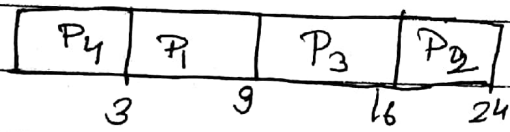
The CPU time time the process requires to complete its processing

→ The process with least burst time will be executed first

Eg:→

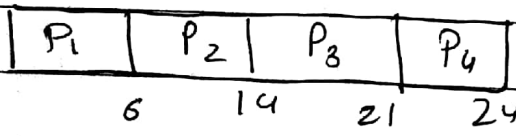
	P_1	P_2	P_3	P_4
burst time :→	6	8	7	3

At the time we are just focusing about the sequence.



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

For FCFS



41

$(0+6+14+21)/4 = 10.25$

→ SJF - best algorithm in terms of waiting time.

Eg:→ Preemptive Version of Shorted Job first

	P_1	P_2	P_3	P_4
Arrival time :→	0	1	2	3
burst time :→	8	4	9	5

① P_1 arrives at 0, But at 1 P_2 arrives, P_1 need 7 more time point

② At t=2 P_3 arrives P_2 needs 3 sec.

9 sec

③ At t=3 P_4 arrives, 9 sec.
↳ 5 sec

P_1	P_2	P_4	P_1	P_3
1	5	10	17	26

Preemptive version of SJF is called SRTN (Shortest Remaining time next)

First we will determine the turnaround time in this case and then after we will talks about waiting time.

			WT
P_1 Turnaround time =	(17 − 0)	= 17	T.T − B.T = 17 − 8 = 9
P_2 " =	(5 − 1)	= 4	= 4 − 4 = 0
P_3 " =	(26 − 2)	= 24	= 24 − 9 = 15
P_4 " =	(10 − 3)	= 7	= 7 − 5 = 2

(termination time, AT)

Response time = 1st Response time − AT

P_1 R.T = 0 − 0 = 0

P_2 R.T = 1 − 1 = 0

P_3 R.T = 17 − 2 = 15

P_4 R.T = 5 − 3 = 2

③ **Priority Scheduling :-** (Have both versions preemptive + non preemptive)

Priority is a no., and depending on the system we can decide whether the lower no. or higher no. has more priority.

→ Higher priority will be considered for a lower no.

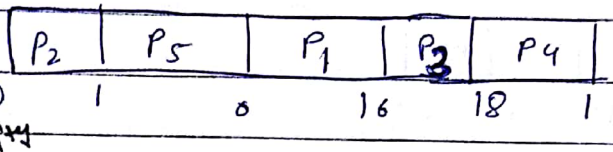
Eg →

	B.T	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

Teacher's Signature.....

DATE : / /
PAGE NO. :

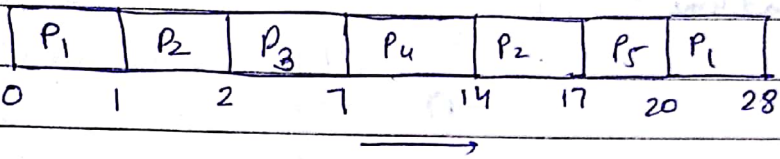
non preemtive.



2nd priority bases → 1st Priority

Preemtive Version →

	A.T	B-T	Priority
P_1	0	9	5
P_2	1	4	3 → ↑ Priority.
P_3	2	5	1 → ↑ Priority.
P_4	3	7	2
P_5	4	3	4



Now on the basis of priority.

→ Turnaround time → P_1 (28-0), P_2 (17-1), P_3 (7-2), P_4 (14-3), P_5 (20-4)

→ Waiting time →

④ :→ Totally Preemptive algorithm
Round Robin Scheduling algorithm
(Used alot)
Here the basis is :→ There is a fixed time slice, after which the CPU will give time to next process from queue.

Next process will be Scheduled 1. → Time quantum is over
2. → If during that time quantum, the process is finished

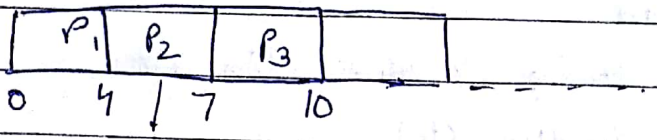
Preempt process on fixed time slic.
Once the time quantum is decided, it will be fixed for all process.

Teacher's Signature

RR → time quantum should be large compared with the context switch, it should not be too long.
80% of the CPU bursts should be shorter than the time quantum.

Ready queue	B.T
P_1	24
P_2	3
P_3	3

Eg → Time quantum = 4



② case

The process which is preemtive will go to the end of the queue. If after 10 there is no process, P_1 will keep running till it finishes execution.

Problem of Starvation: → for which time process has been in the system.

Ageing → Along with the Age of the process, the priority has to be increased, eg → after 5 time unit we will ↑ the priority.

Problem with Priority Scheduling :→ Starvation.
Priority Scheduling can leave some low priority process waiting indefinitely.

→ **Thread Scheduling:** →

many to one - many - many → The thread library schedules user level threads called PCS since scheduling takes place among threads belonging to the same process.

→ SCS Scheduling → All threads compete with each other.
↓
One - to one model systems use only SCS.

for response time - Round Robin best.
for waiting time - SJF best.

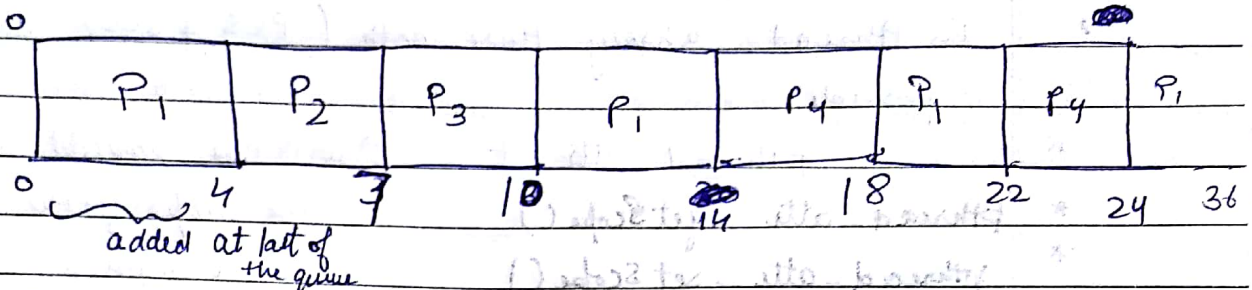
9/02/18

Time slicing

Eg:→ Round Robin time quantum = 4

P_1	24
P_2	3
P_3	3

Soln.



added at last of the queue

If P_4 occurs in the system after 14

P_4 6

- There should be neither too short nor too long time quantum.
- As due to very short time slice we have to perform a lot many of content switches. These unnecessary content switch will degrade the performance of our process.
- Time quatum is decided by system administrator → for different type of queue give optimal result for diff. Process.

Multiqueue algorithm:→ This will be different queue of process for different algorithms depending on the bases of execution → response time, waiting time etc.

- we can adopt differesit algorithms for different queue.

Advanced:→ If required we can move the process in b/w the groups.

Thread Scheduling

Which thread to be sheduled for execution.

The decision of which thread has to be scheduled is done at kernel level called **System Contention Scope.**

Teacher's Signature

→ In case of many-to-one mapping, ~~[illegible]~~ Here the scheduling decision has to be taken that from many users we have to select one. This decision from user level to Kernel level is called Process Contention scope

→ In Pthread library these both (PCS & SCS) scopes are possible.

* pthread_attr_t → with this variable we can set
* pthread_attr_getScope() the scope for the thread which
* pthread_attr_setScope() we require.

which of the thread will be entred to the ready queue first than the usual scheduling will be done.

→ By Default the scope is PCS

Multiprocessor Schedulars :→

→ Who will be doing the scheduling decision? How that decision will be taken etc.

Two Scheduling model → asymmetric
→ Symmetric

Asymmetric :→
→ 1 processor will be the master of processor will coordinate the scheduling ~~process~~ activity.

Symmetric :→
→ For each process there is only one common queue. Centralized queue. Each processor maintain their ready queue and fetch their processes from these common queue. The entry of the processes is done in common queue.

* We need some sort of coordination here, for managment of moving processes to one processor from another processor.

Cache memory → P_1 → P_2 (new process)

Other storage info. → costly operation.

SHREE
DATE : / /
PAGE NO. :

→ **Processor affinity** → Each process have Processor affinity for the on which it is cur... Cache memory and the executing etc.

(For SMP) Symmetric multiprocessor.

Load Balancing :- System tries to balance the load of different proc... Since their are sometime unusual loading as their are some other processor which are ideal at that time.

Eg → If all of the processes are allocated to a signal processor then we are using only 50% of its efficiency thus process are migrated to over processors.

Push migration / Scheduler pushes the processes to the processor which have less load.

Pull migration the processor which are ideal, pull the processes

---

⇒ In real time Operating System the deadline is the bases for the process to be executed first. These types of OS are only focused to minimize any further delay.

Main focus → To finish the process by its deadline.

Real time Operating System :→

1. Hard realtime system → It is essential to finish the process by deadline.
2. Soft real-time system
   - The process can be preponed or postponed by few time units

2/01/18

Real time Scheduling :→

Here before executing the process first Scheduler verify that whether it will be able to meets its deadline (through some mechanism) called _____

→ Deadline will be having a value in time units.
(Need burst time, deadline etc)

→ These process are generally periodic process, they have fixed frequency after which next instance will execute.

→ There may be cases that user is not able to fulfill deadline due to some fault policies.

(static)

① Preemtive Priority based Scheduling algorithm :→

Eg:→

	P_1	P_2
Period	50	100
Burst time $= t_i$	20	35

↳ this P_1 instance should finish its processing in time unit 50

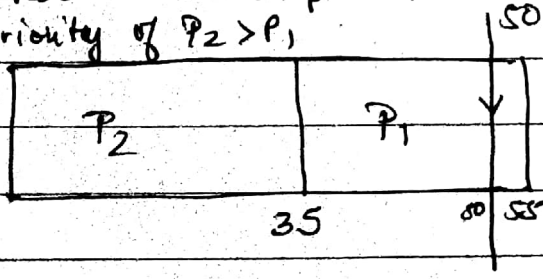
$\frac{20}{50} = 0.4$ $\quad$ $\frac{35}{100} = 0.35$ → ① Utilization is determined.

done by CPU → if these process come in CPU in this way then will be 75% of CPU utilization. $(0.4+0.35)$

Assumption for deadline :→

Deadline :→ Before arrival of another instance of P_1, the first one should be completed.

① If Priority of $P_2 > P_1$



→ so if we schedule in this way we will miss the deadline of P_1

② If priority of $P_1 > P_2$

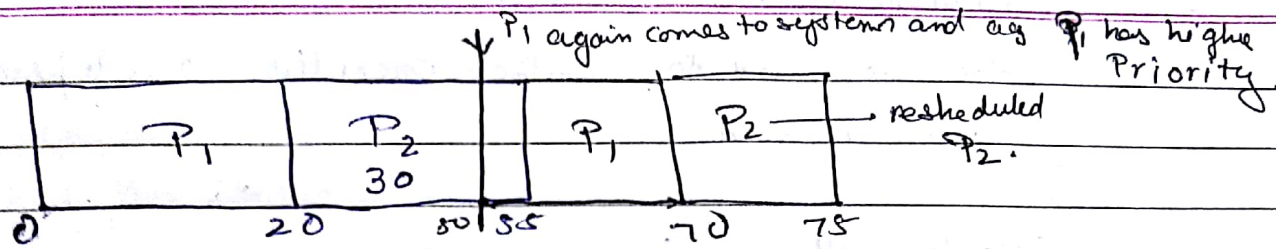
Teacher's Signature.....

If we slightly change the no. then is a possibility to miss the deadline.

→ Highest levels → priority values 16 to 31.

$0 \le t \le d \le p$

rate of periodic task → $\frac{1}{p}$



Using this algorithm we are able to meet its deadline.

(2) Priority → decided by the rate which process are coming.

The process Arrival rate → The shorter the Period of the process, that should be allocated a higher priority.

→ Same as the idea of second chart when Priority of $P_1 \ge P_2$

Rate monotonic Priority Algorithm: →

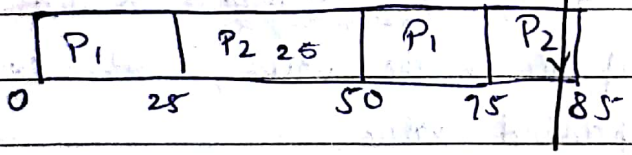
Eg: →

	P_1	P_2
↑ Priority		
	50	80
	25	35

→ 0.5 + 0.46

= 0.96 → Close to 100%

Gantt Chart



This algorithm is not suitable as we missed the deadline.

(3) Earliest deadline first: →