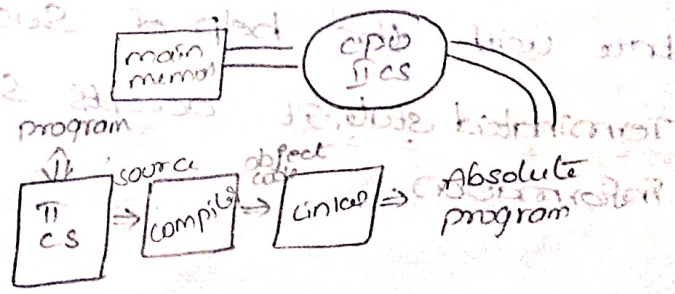
UNIT-2

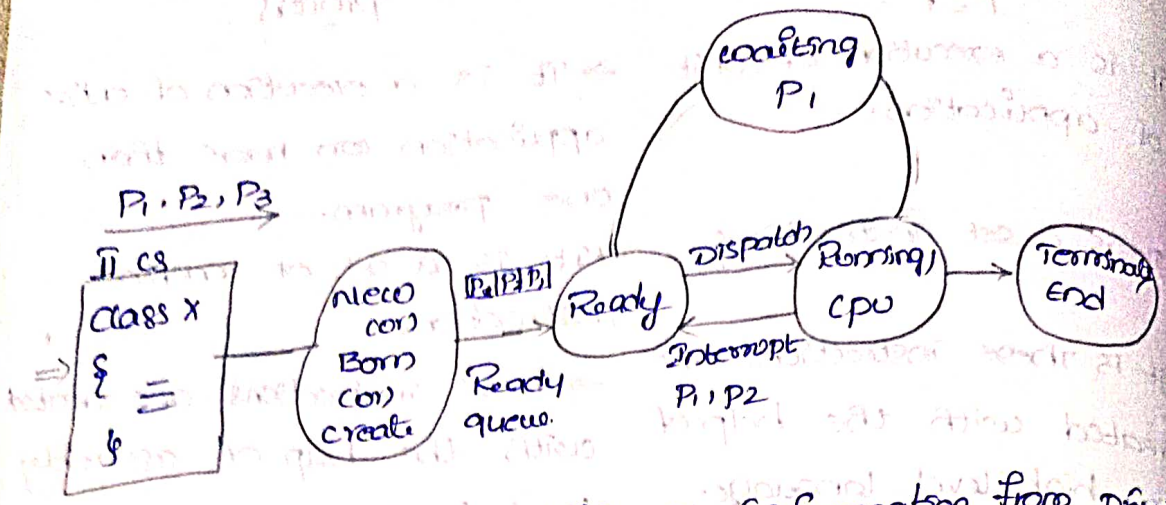
PROCESS MANAGEMENT

program	Process
→ It is a execution of unit of application.	→ It is a execution of entire application con more than one program.
→ It is a set instructions	→ It is a set of machine instructions.
→ It is these instructions are created with the help of using high level language. CC, C++, Java, dotnet -->	→ These instructions are created with the help of assembly level language (8085, 8086)
→ It is a static type	→ It is a dynamic type.
→ Static means cannot be order	→ Dynamic means can be order.
→ It is a passive type.	→ It is a Active type.
→ program stored in Secondary memory.	→ It is stored in main memory.
→ program required different functionalities are i) compiler ii) Linker iii) Loader, Assembler.	Process required different functionalities i) Linker, compiler, cpu, ii) Loader, processor, stack iii) Program counter, parameters

a) Eg:- class class name
{
}
void setdata()
{
}
void display()
{
}
P.s.v.m(String args[])
{
}
	→ Eg:-



* process state diagram (or) process life cycle



* Create state :- It accepts different information from private areas. create state is responsible for sending new state information to other state using admit operation.

* Run state :- It accepts different processes from create state. It arranges the information into queue from Run state is also responsible for sending any function use process information from ready queue to running state using dispatch function using for execution purpose.

* Running state :- It is under cpu control. It is used to properly execute the process that is fetched from the cpu to the main memory.

* wait/Block state :- If any process do not receive input information from I/O devices to running state, then the corresponding processors are sent to the wait/block state with the help of Suspend/Resume function.

* Terminated state :- It accepts successfully executed process information.

* Process control block:-

→ Each and every running process consists of information

→ Every process running information must be stored under a control unit i.e.. process control unit technically called PCB

→ PCB has different partitions. They are Process state: It maintains different states such as new / running / terminated. It consists of two other partitions. They are:

1) parent process
2) child process.

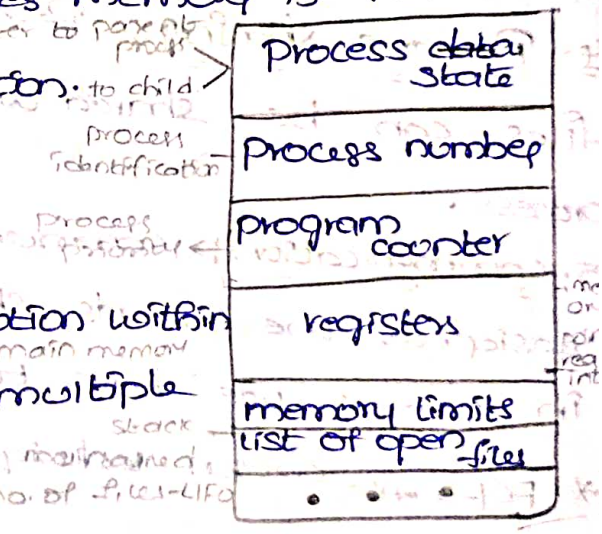
→ process hierarchy is used to prioritize different processes. process ID is called used to allocate different numbers to each process.

→ PC is used to fetch the address of the next instruct

→ Register receives process information from PC. It stores instructions temporarily.

→ Every process requires I/O information from I/O devices.

→ Memory panter stores successfully executed information under main memory. Sometimes memory is flooded. then we use swap out operation.



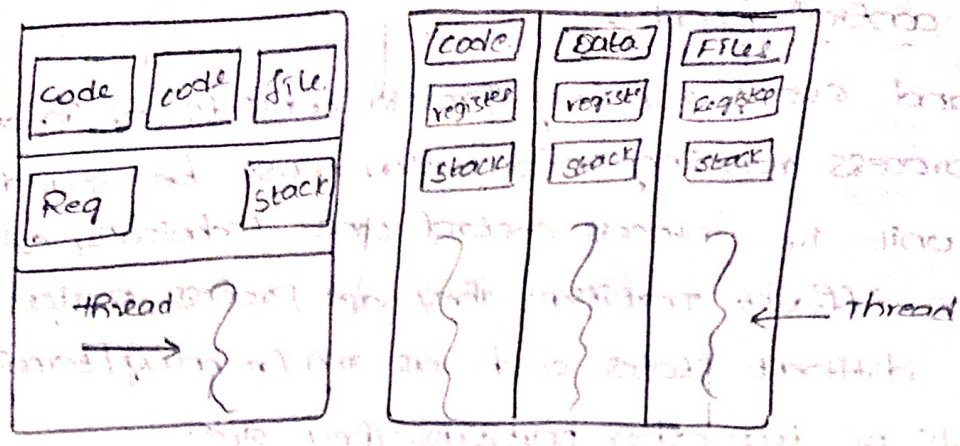
* Threads

1) what is a thread?

→ A Thread is a path of execution within a process. A Process can contain multiple threads.

2) Why multithreading?

→ A Thread is also known as Light weight process. The idea is to achieve parallelism by dividing a process into multiple threads. For eg. in a browser, multiple tabs can be different threads. MS word uses multiple threads: one thread to format the next, another thread to process inputs, etc.



* Advantages of Thread over process :-

1. Responsiveness :- If the process is divided into multiple threads, if one thread completes its execution, then its output can be immediately returned.

2. Faster context Switch :- context Switch time between threads is lower than compared to process context Switch process context switching requires more overhead from cpu.

3. Effective utilization of multiprocessor system :- If we have multiple threads in a single process, then we can schedule multiple threads on multiple processor. This will make process execution faster.

4. Resource Sharing :- Resources like code, data and files can be shared among all threads within a process.

5. communication :- communication b/w multiple threads is easier, as the threads shares common address space, while in process we have to follow some specific communication technique for communication b/w two pro

* FCFS Allocation technique for communication b/w two pro

* User Threads

Thread management done by user-level threads library.

* Three primary thread libraries.

→ POSIX Pthreads - pthreads, the threads extension of the posix standard, may be provided as either a user or kernel-level library.

→ Win 32 Threads: The Win32 threads library is a kernel-level library available on windows systems.

→ Java Threads - The Java thread API allows threads to be created & managed directly in Java programs.

* Multithreading Models

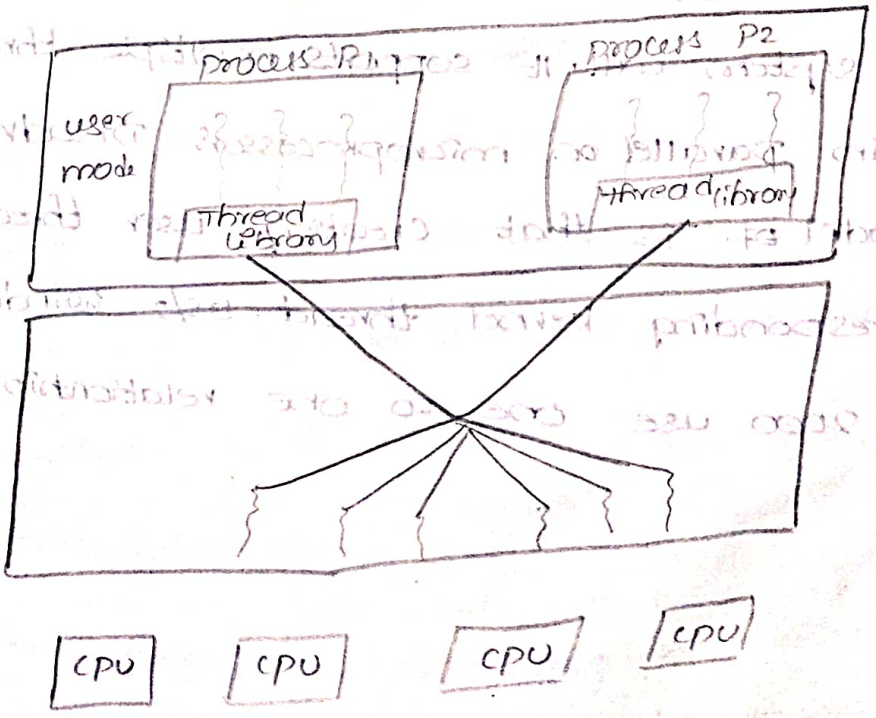
→ Many-to-one

→ one-to-one

→ Many-to-Many

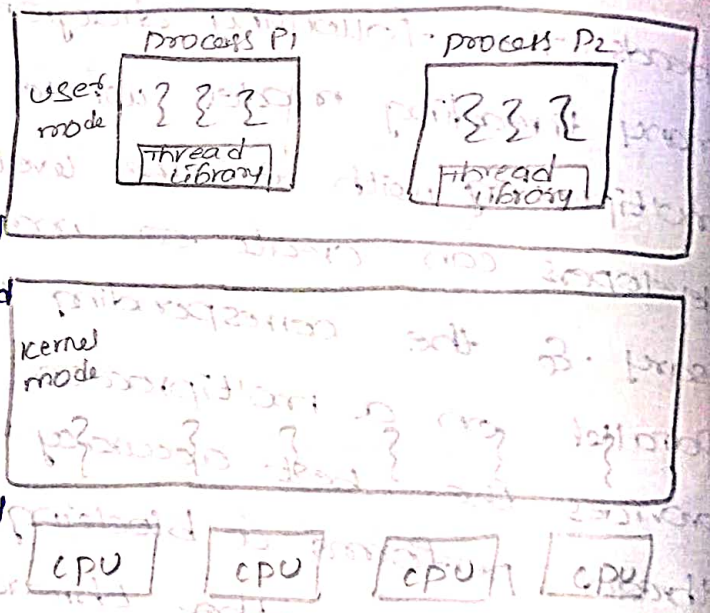
* Many to many model

→ The many-to-many model multiplexes any number of user threads onto an equal or smaller number of kernel threads. The following diagram shows the many-to-many threading model where 6 user level threads are multiplexing with 6 kernel level threads. In this model developers can create as many user threads as necessary & the corresponding kernel threads can run in parallel on a multiprocessor machine. This model provides the best accuracy on concurrency & when a thread performs a blocking system call, the kernel can schedule another thread for execution.



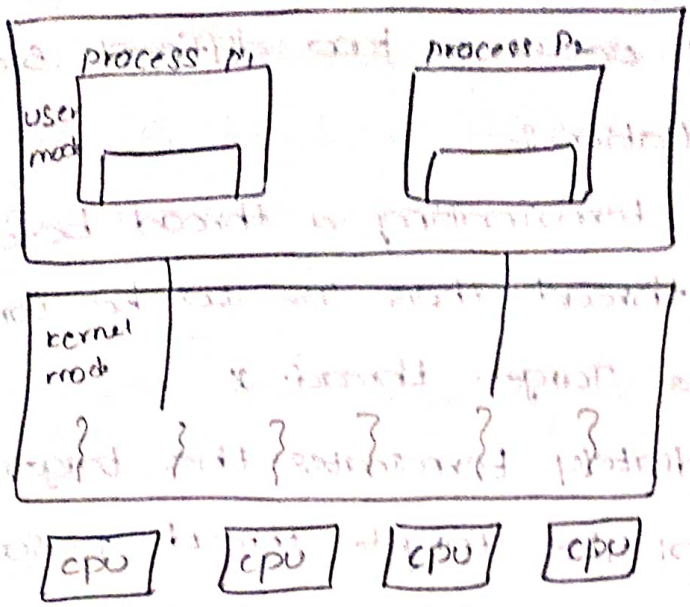
* many to one model

Many to one model maps many user level threads to one kernel-level thread. thread management is done in user space by the thread library. when thread makes a blocking system call, the entire process will be blocked. only one thread can access the kernel at the time, so multiple threads are unable to run in parallel on multiprocessors. If the user-level thread libraries are implemented in the operating system in such a way that the system does not support them, then the kernel threads use the many-to-one relationship modes.



* one to one Model

There is one-to-one model relationship of user-level thread to the kernel-level thread. This model provides more concurrency than the many one-to-one model. It also allows another thread to run when a thread makes a blocking system call. It supports multiple threads to execute in parallel on microprocessors. Disadvantage of this model is that creating user thread requires for corresponding kernel thread. OS/2 windows NT & windows 2000 use one to one relationship model.



* Threading Issues :-

Different Threading issues are :

→ The fork() & exec() System calls :-

The fork() and exec() call creates a duplicate process of the process. The new duplicate process is called child process & process invoking the fork() will duplicate all the threads of the parallel process or the duplicate process would be single-threaded. The excel() system call when invoked replaces the program along the all its threads with the program that is specified in the parameter to exec(). The issue in exec() system call is if the exec() system call is lined up just after the fork() system call then duplicating all the threads of parent process in the child process by fork() is useless.

* Thread cancellation :-

→ It is the task of terminating a thread before it has completed. A thread that is to be canceled is often

referred to as Target thread. The Cancellation of a target thread may occur in two different scenarios.

Asynchronous Cancellation:-

* It is a task of terminating a thread before it has completed. A thread that is to be canceled is often referred to as Target thread. *

One thread immediately terminates the target thread.

Deferred cancellation: The target thread periodically checks whether it should terminate. allowing it an opportunity to terminate itself in an orderly fashion.

* Process Scheduling:-

→ The Act of determining which process is in the ready state, and should be moved to be the [Routing] running state is known as process scheduling.

→ The main aim of the process scheduling system is to keep the cpu busy all the time & to deliver minimum response time for all program.

→ scheduling is of two types.

1, Pre emptive scheduling.

2, Non-preemptive scheduling.

* Pre-Emptive Scheduling:-

→ It is a cpu scheduling techniques that works by dividing time slots of cpu to a given process.

2, Non-preemptive scheduling :-

→ It is a cpu scheduling technique the process take the resource & holds it till the process gets terminated (or) is pushed to the waiting state.

~~x~~ Scheduling queue :-

→ All processes upon entering into the system, are stored into the Job queue. process in the Ready state are placed in Ready queue. process waiting in Device queues.

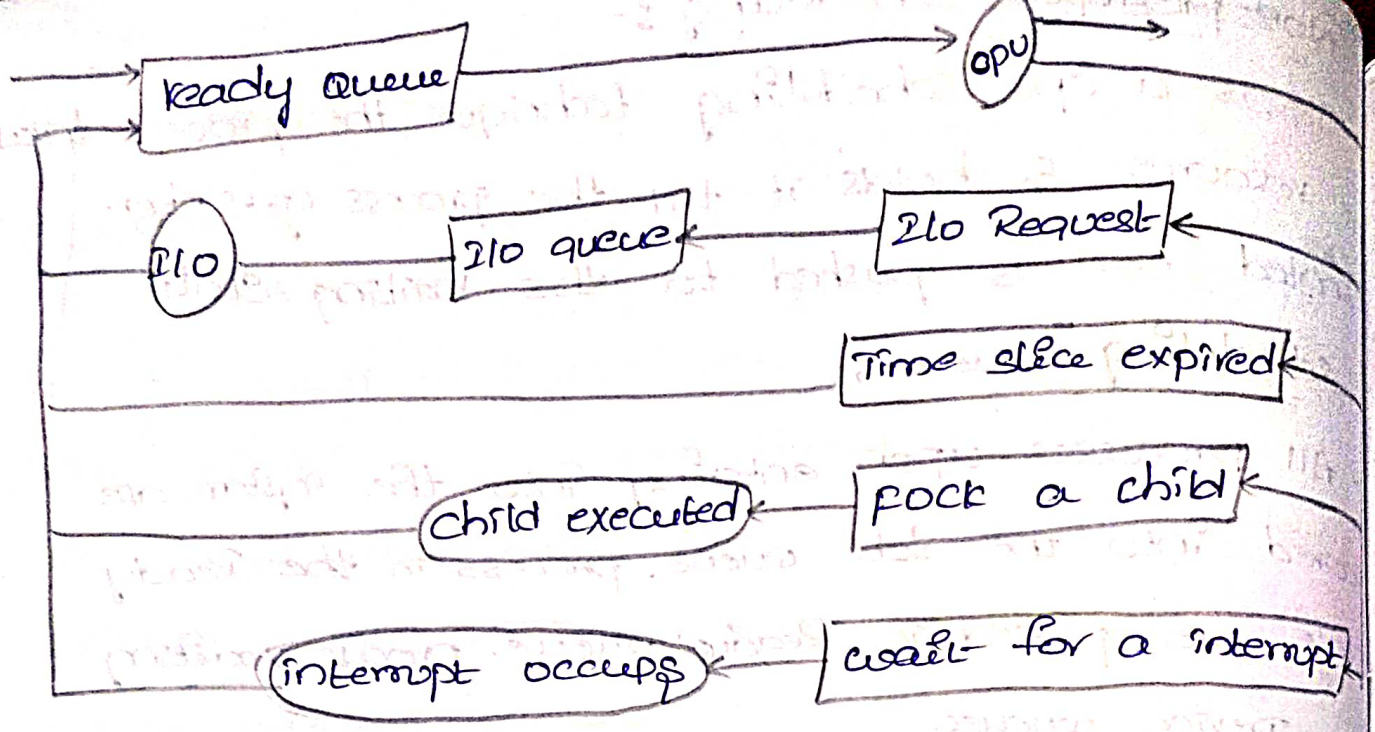
→ A new process in is initially put in Ready queue. It waits in the ready queue until it is selected for execution.

→ once the process is assigned to cpu. one of the following events may occur.

(i) The process could issue an I/O request & then be placed in the I/O queue.

(ii) The process could create a new sub process and waits for its termination.

(iii) The process could be removed forcibly from the cpu, as a result of interrupt, & put back in the ready queue.



* Schedules :-

→ three types of schedules are present.

1) Long Term Schedule :-

→ Long-Term scheduler, (or) job scheduler, selects processes from this pool & loads them into memory for execution.

2) Short-Term Schedule :-

→ short-Term scheduler, (or) CPU scheduler, selects from among the processes that are ready to execute & allocate the CPU to one of them.

3) Middle-Term Schedule :-

→ This scheduler removes the processes from memory and thus reduces the degree of multiprogramming At some later time, the process can be reintroduced into memory & its execution can be continued where is left off.

this is called swapping. the process is swapped out, and it is later swapped in, by the medium term scheduler.

* Context Switch

→ Switching the cpu to another process requires performing a state save of the current process and a state restore of a different process. this task is known as a context switch.

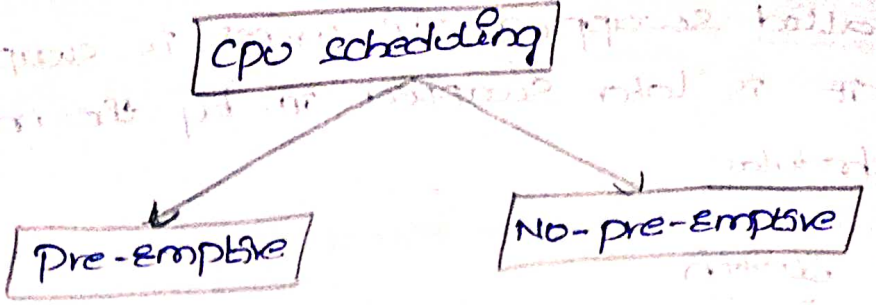
→ when a context switch occurs, the kernel saves the context of the old process in its PCB and loads the saved context of the new process scheduled to run.

→ context-switch time is pure overhead, because the system does no useful work while switching.

→ its speed varies from machine to machine, depending on the memory speed, the number of registers that must be copied, & the existance of special instruction

* what is cpu scheduling?

cpu scheduling is a process of determining which process will own cpu for execution while another process is on hold. the main task of cpu scheduling is to make sure that whenever the cpu remains idle, the os at least select one of the processes available in the ready queue for execution. the selection process will be carried out by the cpu scheduler. It selects one of the processes in memory that are ready for execution.



* Preemptive scheduling :-

In preemptive scheduling, the tasks are mostly assigned with their priorities. Sometimes it is important to run a task with a higher priority before another lower priority task, even if the lower priority task is still running. The lower priority task holds for some time and resumes when the higher priority task finishes its execution.

* Non-preemptive scheduling :-

→ In this type of scheduling method, the CPU has been allocated to a specific process. The process that keeps the CPU busy will release the CPU either by switching context or terminating. It is the only method that can be used for various hardware platforms. That's because it doesn't need special hardware like preemptive scheduling.

When scheduling is preemptive or Non preemptive?

To determine if scheduling is preemptive or non preemptive, consider three four parameters:

→ A process switches from the running to the waiting state.

→ Specific process switches from the running state to the ready state.

→ Specific process switches from the waiting state to the state.

process finished its execution & terminated. only condition 1 & 4 apply. the scheduling is called non preemptive.

* dispatcher :-

→ the dispatcher is the module that carries any information from run to running state. this function involves the following:

1, Switching context

2, switching to user mode

3, Jumping to the proper location in the user program to restart that program.

* scheduling criteria :-

The criteria include the following

→ cpu utilization : It is the process of reducing the ideal time during process execution.

→ cpu throughput : It is the process of calculating of number of processors successfully executed under the processor.

→ cpu throughput ranges from 0 - 100

→ Turnaround Time : It is the process to find out difference between arrival time & finishing time.

* formulas :-

Turn around Time (TA) = finishing time - arrival Time.

Total Turnaround time = $\frac{(P1 * TA) + (P2 * TA) + (P3 * T3) + \cdots + (Pn * TA)}{\text{Number of processors.}}$

waiting time:- It is the process to find out diff between starting time and arrival time.

formulae:-

waiting time (WT) = starting time - arrival time

Total waiting time = $\frac{(P_1 * WT) + (P_2 * WT) + \cdots + (P_n * W}{\text{No. of processors.}}$

* Response time:-

It is the process of to find out difference bw response time and arrival time.

* formulae:-

Response time (RT) = first response time - arrival time

Total response time = $\frac{(P_1 * RT) + (P_2 * RT) + \cdots + (P_n * R}{\text{No. of Processors}}$

* Scheduling algorithms:-

→ Different scheduling Algorithms are:-

→ FCFS Algorithm

→ SJF (Shortest Job First

→ Algorithm Priority Algorithm

→ Round Robin Algorithm.

* FCFS :- It stands for first come first serve.

It is a non primentive scheduling algorithm.

→ Non primitive means any process successfully executed under the processor without any time duration.

→ main Algorithm of FCFS

1, Start the process

2, Declare the Array

3, Requi Declare the no. of process

4, Input the values of given burst time

5, FCFS scheduling using burst time.

* To find out turn around time, waiting time & response time using mathematical formulae.

1, Turn Around time = Finishing Time − Arrival Time

2, waiting Time = Starting Time − Arrival Time.

3, Response Time = First Response Time − Arrival Time.

(In non-premenptive, waiting Time & Response time are same)

→ finally, to calculate total Average waiting time & total Average response time using mathematical formula

1. Total Average Turn Around Time = (Turn Around time of P_0 + Turn Around time of P_1 + ---- Run Around time of P_n) / Total No. of Processes.

2. Total Avg waiting Time = (waiting time of P_0 + waiting time of P_1 + --- waiting time of P_n) / Total No. of processes

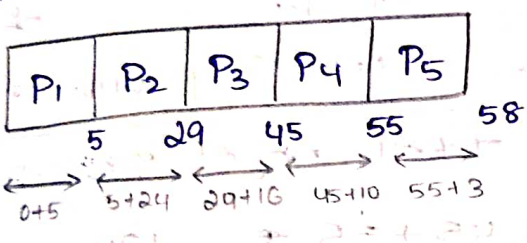
Total Avg Response time = $\frac{\text{Response time of } P_0 + \text{Response time of } P_1 + \cdots + \text{Response time of } P_n}{\text{Total no. of Processes.}}$

→ display the result.

Example:-

Process	Burst time
P_1	5
P_2	24
P_3	16
P_4	10
P_5	3

→ Create a Gantt chart firstly



To find out turn around time for each & every process

Turn around time = Finishing time − Arrival time.

Turn Around time for P_1 = 5 − 0 = 5 m sec

Turn Around time for P_2 = 29 − 0 = 29 m sec

Turn Around time for P_3 = 45 − 0 = 45 m·sec

Turn Around time for P_4 = 55 − 0 = 55 m·sec

Turn Around time for P_5 = 58 − 0 = 58 m·sec

Total Average turn around time

= $\frac{\text{Turn Around time } P_1 + \text{Turn Around time for } P_2 + \cdots + \text{Turn Around time of } P_n}{\text{Total no. of Processes}}$

= $\frac{5 + 29 + 45 + 55 + 58}{5} = \frac{192}{5} = 38.4$ m sec.

Total Average turn Around time. = 38.4 m. sec

To find out waiting time for each & every process

Waiting Time = Starting Time - Arrival time

Waiting Time for P_1 = 0 - 0 = 0 m·sec

Waiting time for P_2 = 5 - 0 = 5 m·sec

Waiting Time for P_3 = 29 - 0 = 29 m·sec

Waiting time for P_4 = 45 - 0 = 45 m·sec

Waiting time for P_5 = 55 - 0 = 55 m·sec

= 58 - 0 = 58

Total Average waiting Time = (waiting time of P_1 + waiting time of P_2 + waiting time of P_3) / Total no. of processors

$= \frac{0+5+29+45+55}{5} = \frac{124}{5} = 26.8$ m·sec

Total Average waiting time = 26.8 m·sec

→ To find out Response time for each & every process

Response Time = First response time - Arrival time.

Response time of P_1 = 0 - 0 = 0 m·sec

Response time of P_2 = 5 - 0 = 5 m·sec

Response time of P_3 = 29 - 0 = 29 m·sec

Response time of P_4 = 45 - 0 = 45 msec

Response time of P_5 = 55 - 0 = 55 m·sec

Total Average Response time = (Response time of P_1 + Response time of P_2 + Response time of P_3) / No. of processors

$= \frac{0+5+29+45+55}{5} = \frac{124}{5} = 26.8$ m·sec

SJF scheduling

→ SJF stands for shortest job First

→ It is a non-premeptive algorithm

→ Non premeptive means any process successfully executed without any time duration.

* Main algorithm of SJF

1) Start the process
2) Declare the Array.
3) Declare the no: of Processes
4) Inputs are given based on burst time.
5) Start the sort the burst time in increasing order.

→ To find out turn Around time, waiting time, and response time.

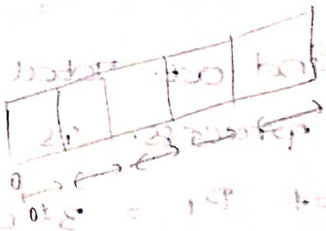
(Refer T.A.T, W.T, R.T. formulas from above algorithm)

→ finally to calculate the total Average turn around time. total Average waiting time & total Average response time. (using mathematical formulas same as the above)

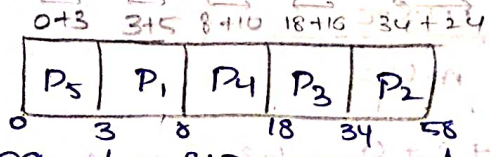
→ Display the resultant values.

Example:

Process	Burst time
P_1	05
P_2	24
P_3	16
P_4	10
P_5	3



→ First prepare a Gantt chart



→ According to SJF, we should select the Least burst time

→ once the select the least burst time then the process is taken in Gantt chart to execute the process

Now, we find out each & every process turn around time.

Turn Around time for P_1 = 8 − 0 = 8 m sec

Turn Around time for P_2 = 58 − 0 = 58 m sec

Turn Around time for P_3 = 34 − 0 = 34 m sec

Turn Around time for P_4 = 18 − 0 = 18 m sec

Turn Around time for P_5 = 3 − 0 = 3 m sec

Total Average turn around time = $\frac{8 + 58 + 34 + 18 + 3}{5} = \frac{121}{5} = 24.2$ m

To find out Total Average turn Around time for each & every processes is

W.T of P_1 = 3 − 0 = 3 m sec

W.T of P_2 = 34 − 0 = 34 m sec

W.T of P_3 = 18 − 0 = 18 m sec

W.T of P_4 = 8 − 0 = 8 m sec

W.T of P_5 = 0 − 0 = 0 m sec

Total Average waiting time = $\frac{3 + 34 + 18 + 8 + 0}{5} = \frac{63}{5} = 12.6$ m sec

→ To find out Total Average Response time for each & every process. is

R.T of P_1 = 3 − 0 = 3 m sec

R.T of P_2 = 34 − 0 = 34 m sec

R.T of P_3 = 18 − 0 = 18 m sec

R.T of P_4 = 8 − 0 = 8 m sec

R.T of P_5 = 0 − 0 = 0 m sec

Total Average R.T = $\frac{30 + 34 + 18 + 8 + 0}{5} = \frac{63}{5} = 12.6$ m sec

* Priority Algorithm

→ It is a non-preemptive scheduling algorithm but sometimes it also act as preemptive algorithm.

1) start the process
2) declare the Array
3) declare the no. of processors
4) input the processors along with their burst time and priority.

Sort the priority, burst time, processors according to their priority.

If any two processors has the same priority then break them using FCFS technique.

For each & every process calculate tornaround time & waiting time.

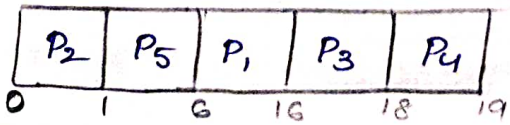
Finally calculate total waiting time and total turn-around time.

Display the result.

Example:

process	Burst time	priority
P_1	10	3
P_2	1	1
P_3	2	3
P_4	1	4
P_5	5	2

* First prepare a Gantt chart for given problem.



(0+1) (1+5) (6+10) (16+2) (18+1)

Calculate tornaround time for each process.

1. TA for P_1 = finishing time − arrival time
= 16 − 0 = 16

T.A for $P_2 = 1-0 = 1$

T.A for $P_3 = 18-0 = 18$

T.A for $P_4 = 19-0 = 19$

T.A for $P_5 = 6-0 = 6$ msec

To find out the turnaround time

Total turnaround time $= \frac{P_1 TA + P_2 TA + \cdots + P_n T_A}{\text{No: of Processes}}$

$TAT = \frac{16+1+18+19+6}{5} = \frac{60}{5} = 12$ m.sec

Calculate the waiting time for each & every Process.

W.T for P_1 = Starting time - arrival time.

$= 6-0 = 6$

w.t for $P_2 = 0-0 = 0$

w.t for $P_3 = 16-0 = 16$

w.t for $P_4 = 18-0 = 18$

w.t for $P_5 = 1-0 = 1$

To find out the waiting time

Total waiting time $= \frac{P_1 WT + P_2 WT + \cdots + P_n W_T}{\text{No: of processors}}$

$W.T = \frac{6+0+16+18+1}{5} = \frac{41}{5} = 8.2$ m.sec

Calculate the Response time for each & every process

R.T for P_1 = First response time - Arrival time

R.T for $P_1 = 6-0 = 6$

R.T for $P_2 = 0-0 = 0$

R.T for $P_3 = 16-0 = 16$

R.T for $P_4 = 18-0 = 18$

R.T for $P_5 = 1-0 = 1$

To find out the Response time

Total R.T $= \frac{P_1 R_T + P_2 R_T + \cdots + P_n R_T}{\text{No: of Processors}}$

$W.T = \frac{6+0+16+18+1}{5} = 41/5 = 8.2$ m.sec

* Round Robin Algorithm

→ It is a preemptive algorithm.

→ preemptive means any process executed successfully under the process based on time quantum (or) timeslice

→ Timeslice means each & every process allocated to time

→ If Burst time less than equal to timeslice then any process successfully executed under the processor.

→ If Burst time greater than time slice then any process not successfully executed under the processor.

* ALGORITHM :-

1, Start the process

2, Declare the array

Declare the no. of processes

Input the process along with their burst time & time slice.

To find turn around time, waiting, & response time.

(using some mathematical formulae)

Turnaround time = Finishing time - Arrival time.

waiting time = (w)starting time - Arrival time

Response time = First sta response time - Arrival time.

Finally to calculate total average turn around time, total average waiting time, total average response time;

1) Total Average turn around time. $= \frac{P_1 TA + P_2 TA + --- + P_n T_n}{\text{NO. of Processors.}}$

2) Total Average W.T $= \frac{P_1 W_T + P_2 W_T + --- + P_n W_T}{\text{NO. of process}}$

3, Total Average Response time $= \frac{P_1R_T + P_2R_T + \cdots + P_nR_T}{\text{No. of process}}$

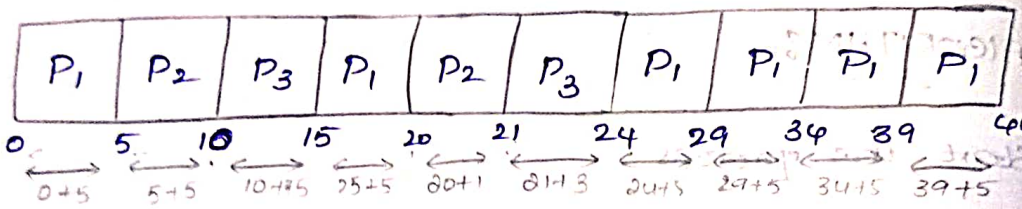
4, Display the resultant values.

5, Stop the process

Eg:

Process	Burst Time	Timeslice
P_1	30 (5,5,5 5,5,5)	5
P_2	6 (5,1)	5
P_3	8 (5,3	5

→ First prepare a Gantt chart



Now we find out the turn around time per for each & every process

Turnaround time = Finishing time - Arrival time

T.A.T for P_1 = 44 − 0 = 44

T.A.T for P_2 = 21 − 0 = 21 (0-0+15-5+24-20)

T.A.T for P_3 = 24 − 0 = 24

Total Average T.A.T $= \frac{P_1TA + P_2TA + \cdots + P_nTA}{\text{No. of process}}$

$= \frac{44 + 21 + 24}{3} = \frac{89}{3} = 29.6$ m.sec

Calculate each & every process for waiting time:

Waiting time = Starting time - Arrival time

W.T for P_1 = 0-0+15-5+24-20 = 14

W.T for P_2 = 5-0+20-10 = 15

W.T for P_3 = 10-0+21-15 = 16

T.A W.T = $\frac{14 + 15 + 16}{3} = \frac{45}{3} = 15$ msec

To find out Response time for each & every process

Response time = First Response time − Arrival time

R.T for P_1 = 0 − 0 = 0

R.T for P_2 = 5 − 0 = 5

R.T for P_3 = 10 − 0 = 10

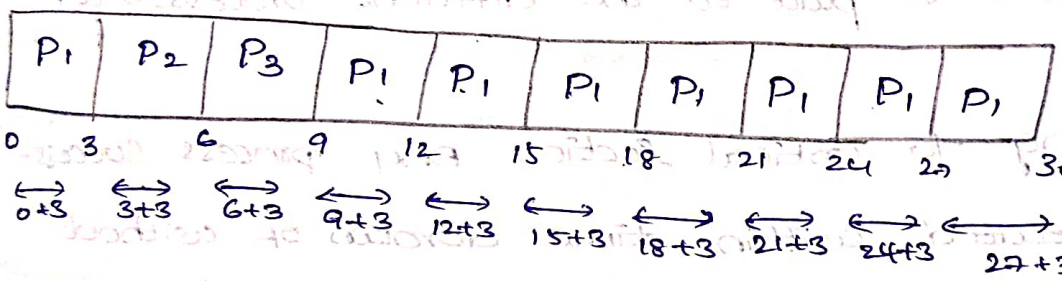
total average Response time = $\frac{10+5}{3} = \frac{15}{3} = 5$ m.sec.

According to preemptive algorithm waiting time & response times are different.

Eg:-

process	burst time	timeslice
P_1	24	3
P_2	3	3
P_3	3	3

Q:- First prepare a Gantt chart.



Now, find out the turn around time for each & every process

T.A.T = R.T − A.T

T.A.T for P_1 = 30 − 0 = 30

T.A.T for P_2 = 6 − 0 = 6

T.A.T for P_3 = 9 − 0 = 9

T.A.TAT = $\frac{P_1TA + P_2TA + P_3TA}{\text{No. of process}}$

= $\frac{30+6+9}{3} = \frac{45}{3} = 15$ m.sec.

Find out the waiting time for each & every process

W.T = S.T − A.T

W.T for P_1 = 0 − 0 + 9 − 3 = 6

W.T for $P_2 = 3 - 0 = 3$

W.T for $P_3 = 6 - 0 = 6$

T. Average W.T = $\frac{P_1WT + P_2WT + \cdots + P_3WT}{\text{No. of process}}$

$= \frac{6 + 3 + 6}{3} = \frac{15}{3} = 5$ m.sec.

Find out response time for each & every process

R.T = First R.T − Arrival time

R.T for $P_1 = 0 - 0 = 0$

R.T for $P_2 = 3 - 0 = 3$

R.T for $P_3 = 6 - 0 = 6$

Total Average Response Time $= \frac{0 + 3 + 6}{3} = \frac{9}{3} = 3$ m.sec

Critical section

→ It provides a place to the different processors for execution.

→ According to critical section any process successfully executed within time duration of without time duration.

→ Critical section depends on 3 factors

1, Mutual execution

2, Progress

3, Bound

1, Mutual execution

→ It supported to different processor for execution

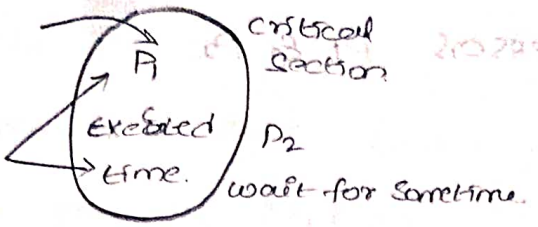
→ According to mutual execution property only allow one process into critical section.

→ When first process successfully executed and next process allows into critical section for execution.

→ Any process executed under the critical section at the time doesn't allows any another process into critical section.

→ According to mutual execution P_1 executed time wait for sometime of the P_2 process

→ consider a two process P_1 & P_2



* progress

→ It Initially maintained a '0' value.

→ progress property wishes to the any one of the process from private area into critical (section) section for execution

→ one process successfully executed under the critical section & gives some chance to next process of the private area.

* Bound

→ It initially maintained time slots or time duration.

→ According to bound property assigns a time slots to the each & every process.

→ Based on time slots allows any process into critical section for execution.

* Peterson's solution

→ It is a software technique.

→ Peterson's solution is implemented with the help of using different factors.. on.

i, critical section

ii, minimum 2 processors

→ Two variables they are.

1, flag

ii, Turn

→ Flag variable supported two values are true/false (0 or 1).

→ Turn variable supported two variables 0 or 1

→ Consider a two processors P_1 & P_2

→ Procedure for P_1

1, Start the process

2, Declare the variables

3, flag[0] = true

turn = 1

3, set the values of flag and turn == 1 & flag[1]

4, Above condition satisfys P_1 enter into critical section for execution.

5, If flag[0] false then P_2 remainder section enter into critical section.

6, Display the resultant values

7, Stop the process

Eg:- implementation code:-

```
void P1()
{
 while (true)
 {
    flag [0] = True;
    Turn = 1;
 while (flag [1] && Turn == 1) // do nothing
 {<P1 enter into critical section for execution>
 flag [0] = false;
```

< P_2 remainder section enter into critical section for execution>

}}} Procedure for P_2 :-

* Start the process

1, Declare the variables flag [1] = true, turn = 0;

2, Set the value of flag[0] and turn = = 0;

3, Above condition satisfys P_2 enter into critical section for execution.

4, If flag [0] = false then P_1 remainder section enter into critical section.

5, Display the resultant values

6, Stop the process.

Eg :- Implementation code

```
void P2 ()
{
 while (true)
 {
  flag [1] = True ;
   TURN = 0 ;
 while flag [0] && TURN = = 0) // do nothing
 {
 < P2 enter into critical section for execution>
 flag. [1] = false;
 < P2 remainder section enter into critical section
   for execution >
 }
 }
  }
```

* Main process for P_1 & P_2

```
void main()
{
  flag[0] = false;
  flag[1] = false;
  Process(P1, P2);
}
```

* Hardware techniques

→ To implements synchronization (or) concurrency with help of using 3 hardware techniques.

1) Interrupts
2) Test set approach
3) Exchange method.

* Interrupts :- It is a hardware technique

2) Interrupt technique provides two conditions to the user.

1) Disable condition
2) Enable condition

* Disable condition

→ It is used to maintaining of the temporary null value of the under the critical section.

* Enable condition

→ It is used to maintaining of inactive state to Active state under the critical section.

→ To properly implementing of interrupt techniques minimum required two processors (P_1, P_2)

→ According to interrupt technique executed successfully P_1 & P_2 processors under the critical section (using disable & enable conditions)

Eg:- Implementation code

procedure for P_1

```
void P1()
{
  while (true)
  {
    <P1 disable condition>
    <P1 enter into critical section>
    <Enable condition>
    <P2 remainder section enter into critical section>
  }
}
```

* procedure for P_2

```
void P2()
{
  while (true)
  {
    <disable condition>
    <P2 enter into critical section>
    <Enable condition>
    <P1 remainder section enter into critical
                                  section>
  }
}
```

* Test Set Approach

→ It is to be initially maintained zero values.

→ It is supported boolean type.

→ Boolean type means to maintaing of two values of 1, 0 true (or) false.

→ It also supported any one integer value.

→ According to test set Approach integer value initally is equicated to 1.

→ when integer variable assigned to 1 at the time process information written to true value (process P1 successfully executed) under the critical section)

→ when integer variable assigned to zero at that time process information return to false (process P2 remainder section successfully executed critical section)

Eg:- Implementation code:

```
Boolean Test Set (int i)
{
 If (i==1) then
 {
  r=1;
  return true;
 }
 else
 {
  r=0
  return false;
 }
}
```

, It is a hardware technique.

* Exchange method :-

→ It is a hardware technique

→ It is implemented with the help of using two main variables are 1, register 2, main memory.

→ To properly exchanging any process information from register to main memory (or) main memory to register. with the help of using "temp" variable

→ Eg:- implementation code

```
void exchange (int register, int main memory)
{
  int temp;
    temp = main memory;
    main memory = register;
    register = temp;
}
```

* producer consumer problem :-

→ It is a hardware technique.

→ This problem is implemented with the help of using that are

1, wait operation (represented as 'n' (or) 'o')

2, Signal operation (represented as 'S' (or) '1')

→ To be filling of any information into buffer queue requires a variable.

* variable names assigned as in)

→ To be fetch out any value from buffer queue using a information variable (variable name assigned as out)

→ Buffer queue specified as a same represented variable is b.

* producers

→ producers created number of items using produce method / function.

→ According to producer to add no. of items into buffer queue using add method (or) append meth

→ Buffer items are arranged into array queue (or) array manner. (Array type is represented in the form of p(in)

→ Temporarily buffer items are assigned into co

→ Finally produce side decremented (in--)

* consumers

→ consumers retrieving no of items from the buffer queue based on initial condition. (in ≥ 1)

→ Above condition satisfys retrieving any item from the buffer queue using "take()" function (or) method.

→ These items are temporarily stored into buffer item array

Buffer array items are represented in the form of b[out].

→ b[out] information assigned into 'i' variable as 'v'

V = b[out]

→ consumer received no. of items from the buffer queue. [represented as (out++)]

→ Finally all items are received with the help of using consume() method

Implementation code

consumer producer - consumer structure

```
{
    int n=0;
    int S = 1;
    int in, out;
    int w, r;
    int b;
}
```

* procedure for producer

```
void producer()
{
 while (true)
 {
   produce();
   add();
   w = b[in]
   in--;
 }
}
```

* procedure for consumer

```
void consumer()
{
 while (true)
 {
   consumer();
   take();
   r = b[out];
   out++;
 }
}
```

```
while (in >= 1) do
{
 take();
 r = b[out];
 out++;
 consume();
}
```

* Semaphores

→ Semaphores is a hardware technique.

→ Semaphore is maintained non-negative variable.
(positive variable)

→ Semaphore is maintained integer variable range values are 0 to 9.

→ Semaphore prouides two operations are.

1, wait operation

→ It is used to temporarily any process hold to block under the to ready queue.

→ wait operation (invented) denoted as 'p' operation

→ Sometimes wait operation specifies another names are 1, sleep operation
2, down operation
3, decrease operation

* Signal operation :-

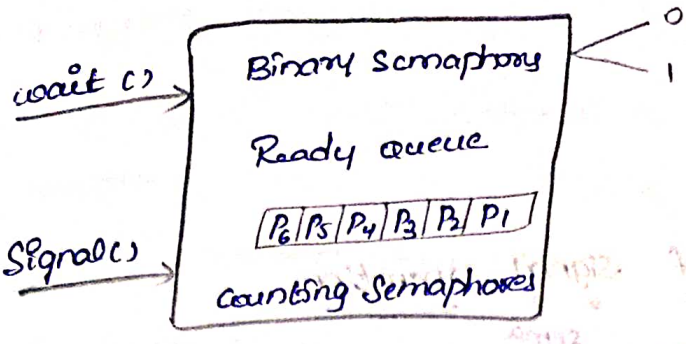
→ It is used to activating of any process from the ready queue and any process send into critical section for the purpose of execution.

→ Signal operation denoted as 'v'

→ It specifies another names are
1, wakeup operation
2, up operation
3, Increase operation

→ Semaphore is again divided into 2 types
1, Binary semaphores
ii, Counting semaphores

Semaphores represented structure operation.



* Binary Semaphores

It supported two values are 1 & 0

Sometimes binary Semaphore are used true or false conditions.

Binary Semaphore is applied on critical section for the purpose of processors execution

Semaphores is executed with the help of using wait, signal & binary Semaphores

Implementation of wait() operation

```
void semaphores (binary semaphores BS)
{
while
void wait (binary semaphore BS)
{
while (true)
{
if (BS.value == = 1) then
{
BS value = 0
< process enter into critical section >
}
else
{
BS value = 1
< process enter into ready queue & block the process >
```

```
        }
      }
    }
  }
```

* Implementation of signal operation

```
void Semaphores (binary semaphore BS)
{
  void signal (binary semaphore BS)
  {
  while (true)
  {
  if ( BS queue == empty ) then
  {
  then BS queue =0
< process enter into Critical Section & executed successfully
  }
  else
  {
   BS queue = 1
   < Remove old process from the ready queue and
   insert new process into ready queue>
    }
   }
  }
}
```

Counting Semaphores

Counting Semaphores provides two operations to the user:

1, counting Semaphore wait

2, counting Semaphore Signal.

3) Counting Semaphore wait operations is technically called decrement operator.

4) Counting Semaphore signal operation is technically called increment operator.

* Structure of counting Semaphore :-

```
Struct Sem ( countsem cs )
{
 int count;
 queue type queue;
}
```

Implementation of counting Semaphore-wait()

```
void cs. wait( )
{
 cs. count--;
 if (cs.value <0)
 {
  any process send into ready queue and block the process corresponding process under ready queue
 }
}
```

* Implementation of counting Semaphore_signal() :-

```
void cs.signal ( )
 {
   cs. count++;
   if (cs. value <=0)
   {
    remove old processes from ready queue and insert new process for the purpose of execution.
   }
 }
```

* monitors

→ monitor is a software technique.

→ TO properly implementing of any synchronization, concurrency using monitor mechanism.

→ monitor mechanism supported different conditions ex. Ready queue, minimum 2 processors (or) more than one process

(i) Ready queue

(ii) Critical section.

(iii) Local variables

(iv) Procedures

(v) Initial conditions

→ monitor provides mainly two operations to the user

→ control wait (or) conditional wait

→ control (or) conditional signal

→ condition wait() operation is used to blocking of the any process under the monitor using C control.

→ Blocking process is sending monitor into blocking queue always

2) control signal

→ It is used to retrieving is use of any process from urgent queue into monitor for the purpose of execution.

→ According to monitor to enter any process from ready queue into monitor using entry exit.

→ Any process properly using local variables, procedures & initial conditions.

→ Any process successfully executed under the monitor must & should send out into outside. using out exit / end exit.

→ Incase of any process not executed under the monitor needed to send into from monitor to blocked queue using conditional wait operation.

→ Incase of any process needed to execute urgently under the monitor using urgent queue, retrieving require process from urgent queue into monitor using conditional signal operation.

→ Finally all processes are more than one process successfully completed under the monitor.

* Structure of the monitor