17-10-2022

Unit-2

Process Management

*Process Management is one of the functionality of OS.

Process:- A program under a execution is called process.

(or)

Operation system will load a program from a secondary memory into main memory, then it is called Process.

For example

When we write a program in C or C++ and compile it, the compiler creates binary code. The original code and binary code are the programs. When we actually run the binary code, it becomes a process.

Program:-

Program is a set of instructions and data.

→ data means variables and declarative statements.

*→ Process is called active entity. and program is called passive entity.

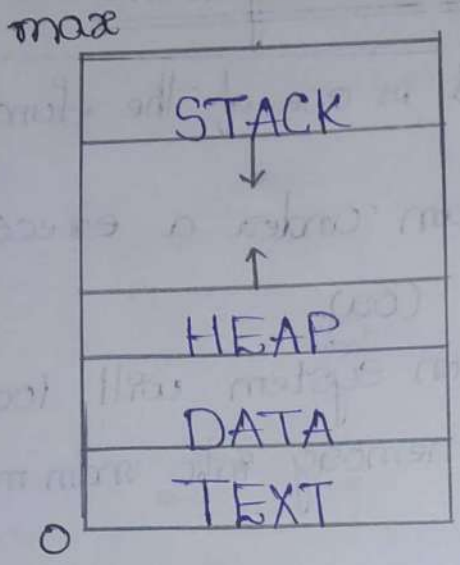
active entity:-

Process is a active entity as it is created during execution and loaded into the main memory.

Passive entity:-

Program is called a passive entity because it is simply a file containing a set of instructions that have yet to be executed. It is stored on the system's secondary memory.

Format of Process Memory:

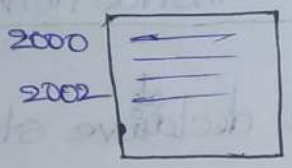


TEXT:- Text section is also called as & code section (or) code segment.

→ It stores the set of instructions and it will maintain a Program Counter.

program counter:- It stores the address of the next instruction.

Ex:-



PC stores the 2002 address.

DATA:- In this Global variables and static variables are stored.

HEAP:- Dynamically allocated memory to process during it's run time.

static allocation means it allocate the memory at the compilation.

Dynamic allocation allocate the memory at the time of execution.

FR dynamic allocation we use some algorithms.

malloc
calloc
realloc

both malloc, calloc is used for allocate the memory.
→ realloc is used, if you want to increase or decrease the memory that was allotted.
→ FR deallocation of the memory we will use a free.

STACK :- It is also called as function stack.
→ It holds the information about local variables, functions, parameters, return values and stack pointer also.

* Arrow symbols use :-

↓ → it means stack increases & decreases the memory in the downward direction.

↑ → its means stack increases & decreases the memory in the upward direction.

* 0 and max are the addresses of the memory.

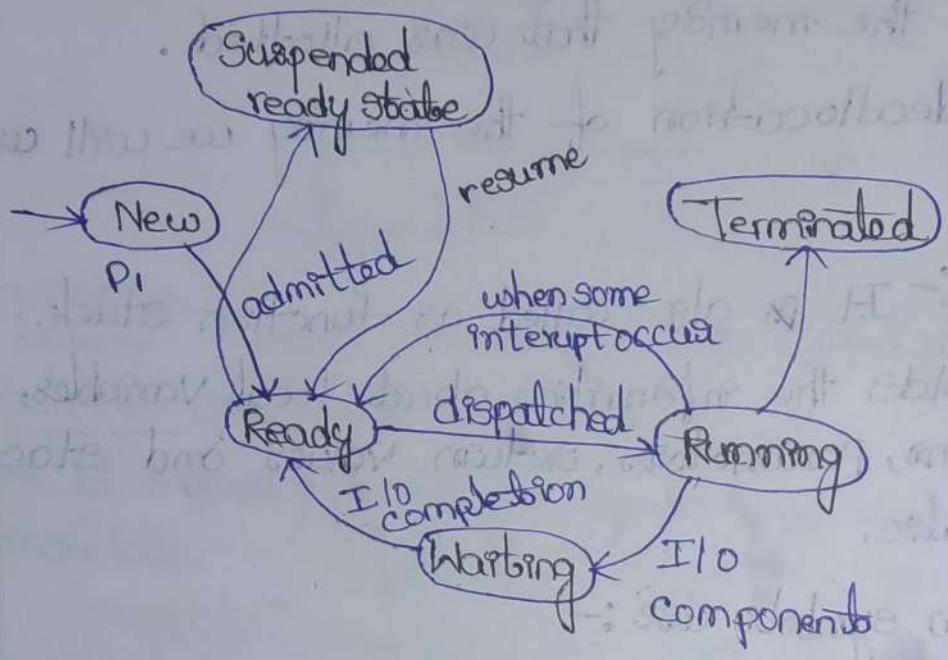
Process State :-

Process travels from one state to another state until its terminated.
→ A process is in one of the following states:
(1) New : Newly Created Process (or) being created Process.
(2) Ready
(3) Running
(4) Waiting
(5) Terminated.

(6) Suspended ready state

Process state Transition Diagram :-

Process state transition diagram is used to provide the pictorial representation for a Process.



(1) New :- It is the first state in the process state transition diagram. (Initial state).

→ When process will created, then Process is in New state.

for example P_1 is the newly created process and it is in the new state.

(2) Ready :-

After creation, a process moves to Ready state i.e, the process is ready for execution.

→ Main memory is the ready state, because the process is in main memory and it will waiting for a CPU to executes its instructions.

(iii) Running :-

When process is in the CPU then we called as Running state.

Now P_1 is in running state.

→ only one process at a time can be under execution in a single processor.

(iv) Terminated :-

After the running state, Process will goes into terminated state. That means after completion of it's execution in the Running state (CPU) Process will terminated or OS will terminate the process.

Now P_1 will terminated after it's execution.

(v) Waiting :-

When P_1 goes to the input then CPU will at the idle state. So OS will scheduling the P_2 to the running state. Now P_2 will at the running state. After some time P_1 will complete it's input then, it will go to the waiting state and then it has to go to the ready state after that P_1 will again schedule to the CPU after that P_1 will executed and terminated.

Again P_3 in the queue datastructure. Then process is same for P_3.

(vi) Suspended ready state :-

P3 Process will go for suspended ready state, when OS haven't enough resources then OS will suspended the process then it will go for suspended ready state.

Incase after terminate the P_1 and P_2 then if resources will available in terminated then P_3 will resume and comes into ready state from suspended ready state. Otherwise P_3 will keep in suspended state.

Incase after terminate the P_1 and P_2 then if resources will available in terminated then P_3 will resume and comes into ready state from suspended ready state. Otherwise P_3 will keep in suspended state.

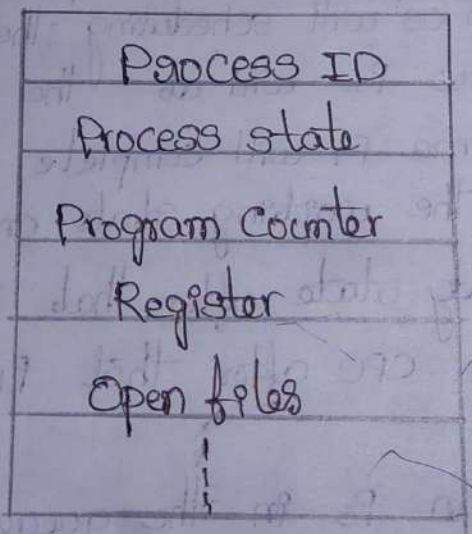
19/10/2022

Process Control Block

Process control Block is called as Task control Block.

→ Process control block is a data structure used by computer operating systems to store all the information about a process.

→ When a process is created, then the operating system creates a corresponding process control block.



PCB

Let us discuss one by one.

(1) Process ID:- Process ID shows the unique number or ID of a particular process. So, every process has to be represented by a unique ID, which will identify the process.

(5) CPU scheduling Information :-

It is also stored in the Process control Block. It contain the information about different scheduling algorithms include other scheduling parameters.

(6) Memory related information :-

It represent the memory that is being used by a particular process.

→ For using main memory in efficients manner we use like Base, limit, Page and segmentation etc.,

(7) Accounting information :-

It is stored information related to the CPU like

* how much time allocate to each process by CPU.
* how many number of processes are waiting for a CPU.

(8) Input / output status :-

It represents the input / output device that are being assigned to a particular process.

* Scheduling Queues :

Scheduling Queues are 3 types. They are

1. JOB QUEUE
2. READY QUEUE
3. DEVICE QUEUE.

① JOB QUEUE :-

As processes enter the system, they are put into a job queue which consist of all processes in the system.

→ It is used to maintain the list of processes that are in the system. (Secondary memory).

② READY QUEUE :-

The processes that are residing in main memory and are ready and waiting to execute are kept on a list. and that list is called the ready state.

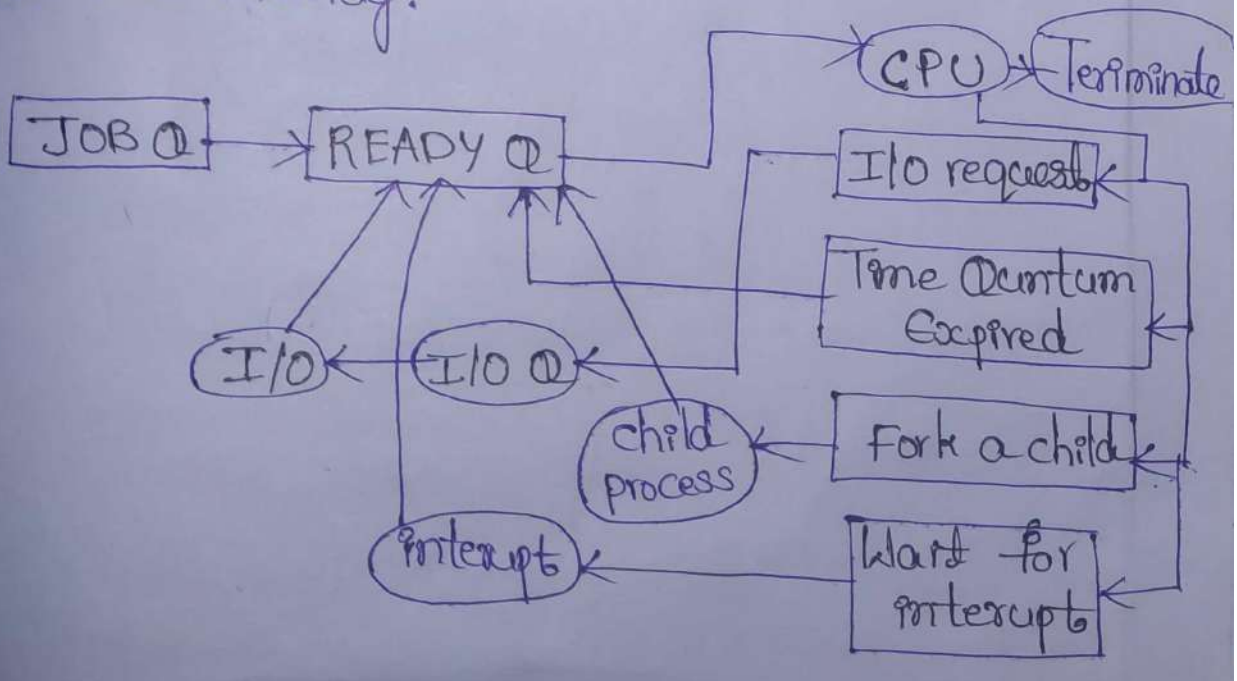
→ Ready state queue is used to maintain the list of processes which are waiting for a CPU, and this is used in ready state.

→ Ready Queue will store in the main memory.

③ Device Queue :-

If a process is waiting for a I/O, then we have to store that processes in the queue. That time we use Device Queue.

* Device Queue will stored in device that contains it's own memory.



SCHEDULERS :-

Schedulers are used to schedule the process to the CPU.

→ schedulers are 3 types

(1) Long term scheduler

(2) short term scheduler

(3) medium term scheduler.

(1) Long-term scheduler :-

It is used to scheduling the process from JOB QUEUE.

→ It is used to achieve the multiprogramming OS.

(2) Short-term scheduler :- It is used to scheduling the process from the ready queue.

~~→ It is used for swapping techniques.~~

~~Two types~~ → ~~swap in~~

→ ~~swap out~~.

(3) Medium-term scheduler :-

If there is no long term scheduler in the system then system will uses the medium ~~then~~ term scheduler.

→ It is used in the swapping technique.

swapping is two types

(1) swap in

(2) swap out.

Process based on execution

I/o bound

CPU bound.

I/o bound :- If process will spend all it's time in the I/o then it is called I/o bound.

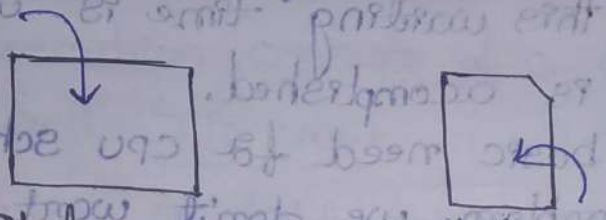
CPU bound :- If process is going to spends all its time in CPU execution, then it is called a CPU bound.

→ ~~Both the~~

→ Combination of both Ilo bound and CPU bound will increase the CPU performance.

Context switching :-

It is a process of saving the context (state) of the old process (suspend) and loading it into the new process (resume). It occurs whenever the CPU switches between one process and other.



The entire schema is called context switching

27-10-2022

CPU Scheduling Concept

CPU scheduling is the basis of multi programming operating system.

→ By switching the CPU among processes, the operating system can make the computer more productive.

* In a single-processor system, only one process can run at a time.
* Any other processes must wait until the CPU is free and can be rescheduled.

This is happened in the single processor system.

→ So, Multi-processprogramming os is much better for no. of processes and have there can be more no. of processes are running at simultaneously at the * same time.

So, the objective of the multiprogramming os is to have some ~~process~~ processes running at all time, to maximize cpu utilization.

→ A process is executed until it must wait, typically for the completion of some I/O request.

→ In a single computer system, the cpu is just sits idle. All this waiting time is wasted, no useful work is accomplished.

The basic need for cpu scheduling is to avoid this problem. we don't want to the cpu is sit idle at any time.

→ With multiprogramming we try to use cpu more productively.

example :-

Several processes are kept in memory at one time. instead of having one process we will have several processes.

→ When one process has to wait, the os takes the cpu away from that process and gives the cpu to another process and this pattern will continues. So, the cpu can't in idle position. It is always utilized by the other processes. when doing this, how do we determined which process should get the cpu or which process should wait or after how long should a process been given to the cpu all these things are going

to be done by CPU scheduling.

→ Generally scheduling means, we are trying to assign a particular time for doing a particular time.

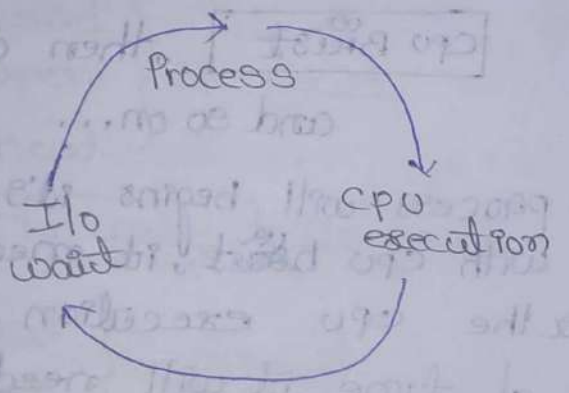
→ In the same way the operating system will prepare a schedule as which process should get the CPU at what time, for how long should be wait and so on...

→ By using cpu scheduling we can access maximum utilization of the CPU.

CPU and I/O Burst cycles :-

Process execution consists of a cycle of CPU execution and I/O wait.

The processes are alternative between these two states.



→ When a process will begins it execution, then it is under the CPU execution state, that means it is making use of the CPU for its execution. So, if this is that state, we call it as CPU execution state and after that the process will need to wait for an I/O Operation to be complete in order to continue its execution in that time we need to waiting for I/O operation to be complete, then

we can say that the process is in I/O wait state, these are the two states in which a process can be in ones it has begin it's execution.

CPU Burst :- It is the time when the process is under the CPU execution. so when the the process under the CPU execution we say that, that is CPU Burst.

I/O Burst :- How much time your process takes I/O time.

→ When process execution begins with a CPU Burst That is followed by an I/O Burst, which is followed by another CPU Burst, then another and so on...

When a process will begins it's execution it's start with CPU burst, it means the process will under the CPU execution and after some point of time it will need a some I/O operation to be done. so, at that time it will wait for the I/O to be completed and that is we called as I/O Burst. So, after the I/O burst completed, it will again case a CPU for its execution so hence we have CPU burst again, then again it need to wait for some I/O operation and hence we have I/O burst and so on. This process will continue until a process will terminated or it completes its execution.

* CPU burst is when the process is being executed in the CPU

* I/O burst is when the CPU is waiting for I/O for further execution

example:-

If you want to read from one file, then we have to open that file, then the file loaded from secondary memory into main memory. In order to write or read we have to open the file. These set of instructions are executed in CPU. So, the process uses the CPU time. So, we called it as CPU Brust. In order to read file, it asks some output files, then the process goes to I/O and uses I/O time. So that is called I/O burst. These procedure continuous until the operation is completed.

Load, Open, Read } CPU Burst

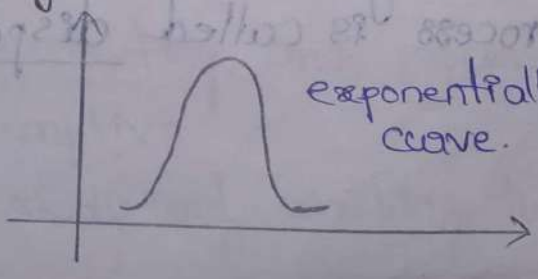
I/O wait } I/O burst

load, open, write } CPU burst

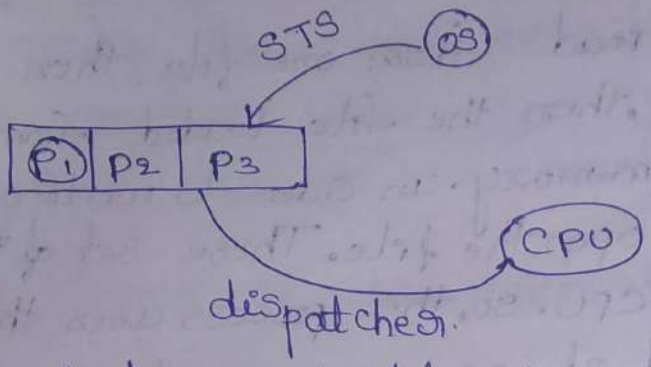
:

{until it's completion.}

* Diagramatically representation of I/O Brust and CPU brust.



* If process is in cpu bound then it will have "few" long cpu bursts.
* If process is in IIo bound then it will have "many" short cpu bursts.



short term scheduler (STS) will select the process, after selecting the process, the OS will scheduling the selected process to the CPU. By using dispatcher os will schedule the selected process by STS to CPU.

→ Dispatcher will give the control of the process to the CPU.

Functionalities of Dispatcher :-

1. Context switching.
2. Switching to usea mode.
3. Restoat the program.
4. Dispatch latency.

Dispatch Latency :- It is a time interval in between stoping one process and stoating the anothea process is called dispatch latency.

Under four conditions your OS will take decision for scheduling the scheduling algorithms.

(1) when a process switches from the running state to the waiting state.

(2) when a process switches from the running state to the ready state

(3) when a process switches from the waiting state to the ready state.

(4) when your process will get terminated.

(1) Running to waiting

when an process wants I/O request (or) kernel invoke system call i.e wait().

* It went to waiting to running state.

wait system call :- Parent process will wait until child process get terminated. for this purpose it use wait() system call.

(2) Running to ~~wait~~ ready (Interupt)

when an interupt (not only access, accessory permission) occur, then

→ It went to ~~ready~~ to running state to ready state.

(3) Waiting to Ready - Completion of I/O request.

(4) Process get terminated.

* Running to waiting and process get terminated are non-preemptive.

* Running to ready and waiting to ready are Preemptive.

→ For preemptive and non-preemptive we use different algorithms.
OS will use some creteria for selecting the scheduling the algorithms to schedule the processes.

(i) CPU utilization :- We want to keep the CPU as busy as possible. Conceptually, CPU utilization can range from 0 to 100 percent. In a real system, it should range from 40 percent to 90 percent.

(ii) Throughput :-
No. of processes that are completes its execution per unit time.
→ The throughput will always have maximum.

(iii) Turn around time :-
Time interval in between submission to completion.
∴ Turn around time = Completion time − arrival time.

(iv) Waiting time :-
If process are waiting in ready queue, that time is called waiting time.
∴ Waiting time = Turn around time − Burst time.

(v) Response time :- When the process has been allocated to the CPU for the first time.

→ The algorithm have to follow turnaround time, waiting time and response time is low.

→ The algorithm have to follow cpu utilization and Throughput is maximum.

Shedeling Algorithms

(1) FCFS

"Fiast Come Fiast Serve."

→ FCFS schedeling algorithm is uses FIFO (Fiast In Fiast out) queue.

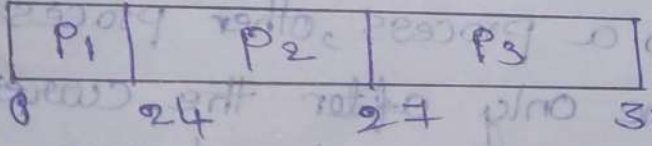
→ In this whatever process will request the CPU fiast, then it will schedule to the CPU.

→ OS will schedule the processes based upon least arrival time.

Example:-

Processes	arrival time	Brust time
P_1	0	24
P_2	1	3
P_3	2	3

Here we use gantt chart. {Non-preemptive}



Completion time for P_1 is 24 ms.

Completion time for P_2 is 27 ms.

Completion time for P_3 is 30 ms.

* Turn around time = Completion time - arrival time.

for $P_1 = 24 - 0 = 24$

for $P_2 = 27 - 1 = 26$

for $P_3 = 30 - 2 = 28$

* Average turn around time = $\frac{24 + 26 + 28}{3}$

$= \frac{78}{3} = 26$ ms

* Waiting time = Turn around time - Brust time.

for $P_1 = 24 - 24 = 0$

for $P_2 = 26 - 3 = 23$

for $P_3 = 28 - 3 = 25$

Average waiting time = $\frac{0 + 23 + 25}{3}$

$= \frac{48}{3}$

$= 16$ ms

In FCFS, there is one drawback. i.e, Convoy Effects.

Convoy Effect:-

FCFS algorithm is non-preemptive in nature, that is, once CPU time has been allocated to a process, other processes can get CPU time only after the current process has finished. This property of FCFS scheduling leads to the situation called Convoy Effect.

→ Small cpu brust process can wait until the completion of long cpu brust process. This is called Convoy Effect.

25/10/2022

2> SJF

"Shortest Job First"

→ SJF is Applicable for both non-preemptive and also preemptive. SJF have both natures.

~~SJF for Non preemptive~~ :-

→ This algorithm associates with each process the length of the processe's next CPU burst.

→ When the CPU is available, it is assigned, it is assigned to the process that has the smallest next CPU burst.

→ If the next CPU bursts of two processes are the same, FCFS scheduling is used to break the tie.

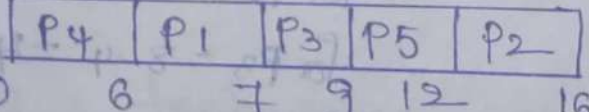
→ It depends on the length of the next CPU burst of a process, rather than its total length.

Example of SJF scheduling (Non-preemptive):-

①

Process	Arrival time	Burst time
P_1	3	1
P_2	1	4
P_3	4	2
P_4	0	6
P_5	2	3

Gantt chart



Completion time

$P_1 = 7$ $P_3 = 9$ $P_5 = 12$

$P_2 = 16$ $P_4 = 6$

Turn around time = completion time − arrival time

$P_1 = 7-3 = 4$

$P_2 = 16-1 = 15$

$P_3 = 9-4 = 5$

$P_4 = 6-0 = 6$

$P_5 = 12-2 = 10$

Average Turn around time = $\frac{4+15+5+6+10}{5}$

$\Rightarrow \frac{40}{5} \Rightarrow 8$ ms.

Waiting Time = Turn around time − Burst time

for $P_1 = 4-1 = 3$

for $P_2 = 15-4 = 11$

for $P_3 = 5-2 = 3$

for $P_4 = 6-6 = 0$

for $P_5 = 10-3 = 7$

Average waiting Time = $\frac{3+11+3+0+7}{5} = \frac{24}{5}$

$= 4.8$ ms

For Preemptive (SRTF) :-

Gantt chart

P4	P2	P2	P1	P2	P2	P3	P5	P4
0	1	2	3	4	5	6	8	11	16

completion time

$P_1 = 4$

$P_2 = 6$

$P_3 = 8$

$P_4 = 16$

$P_5 = 11$

Turn around time

for $P_1 = 4-3 = 1$

for $P_2 = 6-1 = 5$

for $P_3 = 8-4 = 4$

for $P_4 = 16-0 = 16$

for $P_5 = 11-2 = 9$

average Turn around time = $\frac{1+5+4+16+9}{5}$

$= \frac{35}{5} = 7$ ms.

Waiting time

for $P_1 = 1-1 = 0$

$P_2 = 5-4 = 1$

$P_3 = 4-2 = 2$

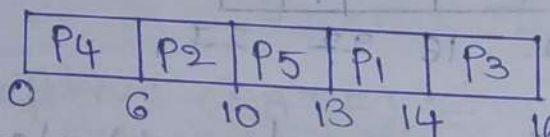
$P_4 = 16-6 = 10$

$P_5 = 9-3 = 6$

average waiting time = $\frac{0+1+2+10+6}{5} = \frac{19}{5}$

$= 3.8$ ms.

For FCFS :-



Completion time	Turn around time	Waiting time
$P_1 = 14$	$P_1 = 14-3 = 11$	$P_1 = 11-1 = 10$
$P_2 = 10$	$P_2 = 10-1 = 9$	$P_2 = 9-4 = 5$
$P_3 = 16$	$P_3 = 16-4 = 12$	$P_3 = 12-2 = 10$
$P_4 = 6$	$P_4 = 6-0 = 6$	$P_4 = 6-6 = 0$
$P_5 = 13$	$P_5 = 13-2 = 11$	$P_5 = 11-3 = 8$

$\therefore$ average turn around time = $\frac{11+9+12+6+11}{5}$

$= 9.8$ ms

$\therefore$ average waiting time = $\frac{10+5+10+0+8}{5}$

$= \frac{33}{5} = 6.6$ ms.

Among 3 techniques SRTF has less waiting time. So, this is the Best technique.

Assignment

Process	Arrival time	Burst time
P1	0	12
P2	2	4
P3	3	6
P4	8	5

(1) FCFS :-

Gantt chart

P1	P2	P3	P4

0 12 16 22 27

Completion time	Turn around time	Waiting Time
P1 = 12	P1 = 12-0 = 12	P1 = 12-12 = 0
P2 = 16	P2 = 16-2 = 14	P2 = 14-4 = 10
P3 = 22	P3 = 22-3 = 19	P3 = 19-6 = 13
P4 = 27	P4 = 27-8 = 19	P4 = 19-5 = 14

Average Turnaround time = $\frac{12+14+19+19}{4}$

= 16 ms.

average waiting time = $\frac{0+10+13+14}{4}$

= $\frac{37}{4}$

= 9.25 ms.

(2) SJF (Non-preemptive)

Gantt chart

P1	P2	P4	P3

0 12 16 21 27

Completion Time	Turn around Time	Waiting Time
$P_1 = 12$	$P_1 = 12-0 = 12$	$P_1 = 12-12 = 0$
$P_2 = 16$	$P_2 = 16-2 = 14$	$P_2 = 14-4 = 10$
$P_3 = 27$	$P_3 = 27-8 = 24$	$P_3 = 24-6 = 18$
$P_4 = 21$	$P_4 = 21-8 = 13$	$P_4 = 13-5 = 8$

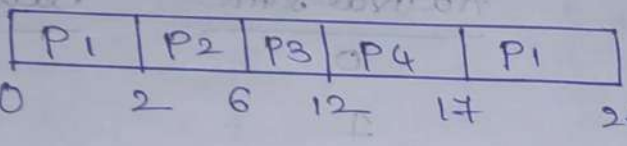
* average turn around time = $\frac{12+14+24+13}{4}$

$= \frac{63}{4} = 15.75$ms

* average waiting time = $\frac{0+10+18+8}{4} = \frac{36}{4} = 9$ms

(iii) Preemptive (SRTF) :-

Gantt chart



Completion time	Turn around time	Waiting time.
$P_1 = 27$	$P_1 = 27-0 = 27$	$P_1 = 27-12 = 15$
$P_2 = 6$	$P_2 = 6-2 = 4$	$P_2 = 4-4 = 0$
$P_3 = 12$	$P_3 = 12-3 = 9$	$P_3 = 9-6 = 3$
$P_4 = 17$	$P_4 = 17-8 = 9$	$P_4 = 9-5 = 4$

* Average Turn around time = $\frac{27+4+9+9}{4}$

$= \frac{49}{4} = 12.25$ms

* Average Waiting time = $\frac{15+0+3+4}{4} = \frac{22}{4} = 5.5$ms

∴ Among SRTF (Preemptive) is the best because it's have less waiting time i.e. 5.5ms

31-10-2022

Round Robin scheduling Algorithm :- (RR)

→ Round Robin scheduling Algorithm consists only preemptive nature.

→ Round Robin uses time quantum (or) time slice criteria.

→ It is simple and easy to implement.

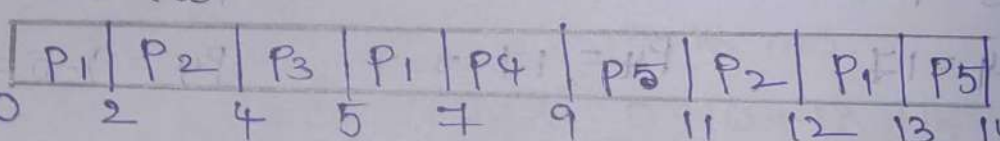
→ It is the one of the most commonly used technique in CPU scheduling.

→ It is preemptive as processes are assigned CPU only for a fixed slice of time.

Problem 1 :- Time quantum = 2 ms.

Process	Arrival time	Burst time
P_1	0	5
P_2	1	3
P_3	2	1
P_4	3	2
P_5	4	3

Gantt chart



0 : P_1

2 : P_2 P_3 P_1

4 : P_3 P_1 P_4 P_5 P_2

5 : P_1 P_4 P_5 P_2

7 : P_4 P_5 P_2 P_1

9 : P_5 P_2 P_1

11 : P_2 P_1 P_5

12 : P_1 P_5

13 : P_5

Completion time	Turn-around time	Waiting time
$P_1 = 13$	$P_1 = 13 - 0 = 13$	$P_1 = 13 - 5 = 8$
$P_2 = 12$	$P_2 = 12 - 1 = 11$	$P_2 = 11 - 3 = 8$
$P_3 = 5$	$P_3 = 5 - 2 = 3$	$P_3 = 3 - 1 = 2$
$P_4 = 9$	$P_4 = 9 - 3 = 6$	$P_4 = 6 - 2 = 4$
$P_5 = 14$	$P_5 = 14 - 4 = 10$	$P_5 = 10 - 3 = 7$

→ average waiting time = $\frac{8+8+2+4+7}{5} = \frac{29}{5}$

$= 5.8$ ms

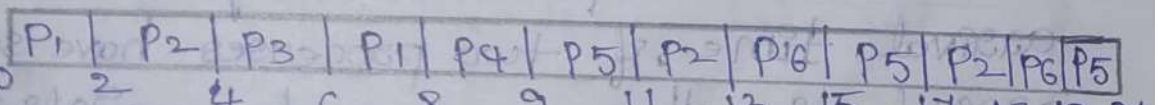
→ Throughput = 14 ms.

Problem-2:-

Process	arrival time	Burst time
P_1	0	4
P_2	1	5
P_3	2	2
P_4	3	1
P_5	4	6
P_6	6	3

→ Given time-quantum = 2 ms.

Gantt chart:-



0 : P_1

2 : P_2 P_3 P_1

4 : P_3 P_1 P_4 P_5 P_2

6 : P_1 P_4 P_5 P_2 P_6

8 : P_4 P_5 P_2 P_6

9 : P_5 P_2 P_6

11 : P_2 P_6 P_5

13 : P_6 P_5 P_2

15 : P5 P2 P6

17 : P2 P6 P5

18 : P6 P5

19 : P5

*→ Throughput is 21.

Completion time	Turnaround time	Waiting time
$P_1 = 8$	$P_1 = 8 - 0 = 8$	$P_1 = 8 - 4 = 4$
$P_2 = 18$	$P_2 = 18 - 1 = 17$	$P_2 = 17 - 5 = 12$
$P_3 = 6$	$P_3 = 6 - 2 = 4$	$P_3 = 4 - 2 = 2$
$P_4 = 9$	$P_4 = 9 - 3 = 6$	$P_4 = 6 - 1 = 5$
$P_5 = 21$	$P_5 = 21 - 4 = 17$	$P_5 = 17 - 6 = 11$
$P_6 = 19$	$P_6 = 19 - 6 = 13$	$P_6 = 13 - 3 = 10$

→ average waiting time is $\frac{4+12+2+5+11+10}{6}$

$= \frac{44}{6} = 7.3$ ms

Priority scheduling Algorithm :-

→ Priority scheduling is a non-preemptive algorithm and one of the most common scheduling algorithm.

→ In this Each process is assigned first arrival time; if two processes have same arrival time, then compare to priorities (highest priority process first).

→ Also, if two processes have same priority then compare to process number (less process

number first).

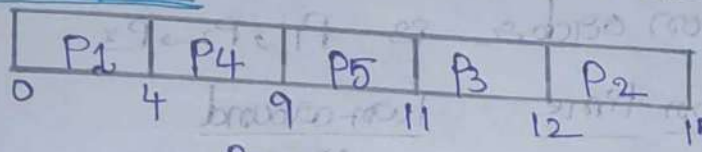
→ This process is repeated while all processes get executed.

Problem 1 :-

Process	arrival time	Burst time	Priority
P_1	0	4	2
P_2	1	3	3
P_3	2	1	4
P_4	3	5	5
P_5	4	2	5

Both Preemptive and non-preemptive

Non-preemptive



0 : P_1

P_5

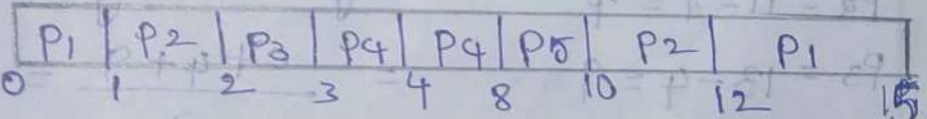
4 : P_2 P_3 P_4 (P4 and P5 has highest priority, same but we take P4 first)

9 : P_2 P_3 P_5 (P_5 has highest priority)

11 : P_2 P_3 (P_3 has highest priority)

12 : P_2

Preemptive :-



0 : P_1

1 : P_1 P_2 P_3

2 : P_1 P_2 P_3 P_4

3 : P_1 P_2 P_3 P_4 P_5

4 : P_1 P_2 P_3 P_4 P_5 (P_4 has highest)

8 : P_1 P_2 P_5

10 : P_1 P_2

12 : P_1

15 : P_5

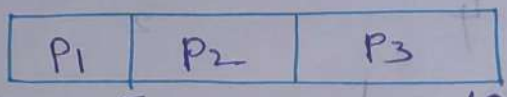
Assignment Problem :-

Process	Arrival time	Burst time
P_1	0	5
P_2	1	7
P_3	3	4

Completion order FCFS and round robin.
Time quantum = 2 ms.

* FCFS

Gantt chart



completion order is P_1, P_2, P_3.

Completion time	Turnaround time
$P_1 = 5$	$P_1 = 5 - 0 = 5$
$P_2 = 12$	$P_2 = 12 - 1 = 11$
$P_3 = 16$	$P_3 = 16 - 3 = 13$

average turnaround time is $\frac{5+11+13}{3}$

$= \frac{29}{3} = 9.66$ ms.

Waiting time

$P_1 = 5 - 5 = 0$

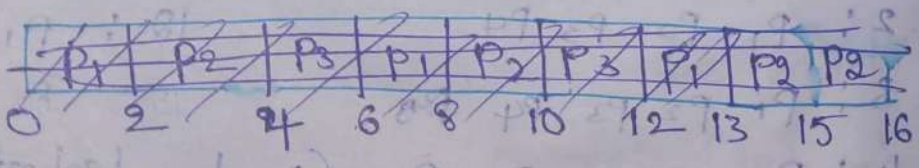
$P_2 = 11 - 7 = 4$

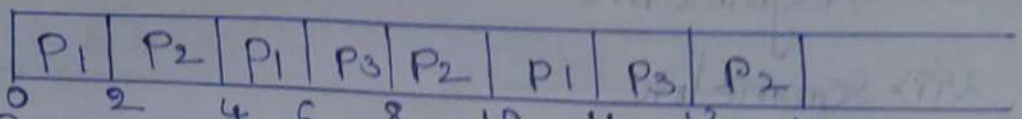
$P_3 = 13 - 4 = 9$

average waiting time $= \frac{0+4+9}{3} = \frac{13}{3} = 4.33$ ms

~~9.66 ms.~~

* round robin :- Time - quantum = 2 ms





0 : P_1

2 : P_2 P_1

4 : P_1 P_3 P_2

6 : P_3 P_2 P_1

8 : P_2 P_1 P_3

10 : P_1 P_3 P_2

11 : P_3 P_2

13 : P_2

16 : ~~P_2~~

completion order is

P_1 : 11

P_2 : 16

P_3 : 13

P_1	P_3	P_2

Turn around time

$P_1 = 11 - 0 = 11$

$P_2 = 16 - 1 = 15$

$P_3 = 13 - 3 = 10$

$$\text{Average Turn-around time} = \frac{11+15+10}{3} = \frac{36}{3} = 12 \text{ ms}$$

waiting time

$P_1 = 11 - 5 = 6$

$P_2 = 15 - 7 = 8$

$P_3 = 10 - 4 = 6$

Average waiting time is $\frac{6+8+6}{3} = \frac{20}{3} = 6.6$ ms

3/11/2022

Multiple - Processor scheduling :-

→ Multiple - processor scheduling consists of multiple processors (CPU's)

→ Multiple - processor scheduling is more complex than single processor scheduling.

* Approaches to multiple processor scheduling :-

There are two types of approaches to multiple processor scheduling in the operating system.

(i) Asymmetric

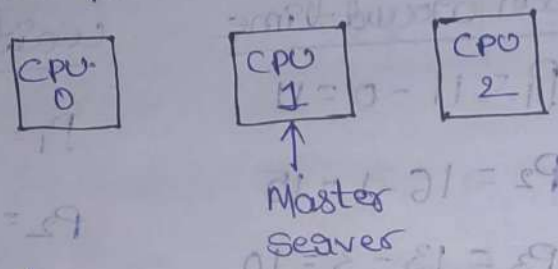
(ii) Symmetric

(i) Asymmetric multi processing:-

Foa asymmetaic multi paocessing we use one system with moa than 1 paocessoa / CPU.

→ In this paocessoas one will acts as like a master seavea and aemaining paocessoas aae acts as slave seaveas

foa example



in the above example CPU 1 acts as a mastea seaver and CPU 0, CPU 2 aae slave seaveas.

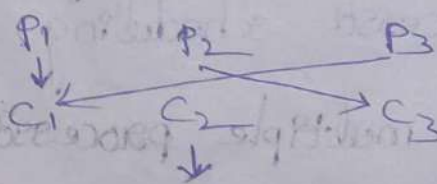
→ Slave seaveas will followo the commands which aae povided by the mastea seavea.

→ only mastea seavea has a scheduler, but oo othea paocessoas doesn't have the schedulea.

→ Mastea seavea can decide the which paocess should gets done thaough the schedulea.

→ Mastea paocessoa have a aeady queue.

ex:-



P1 and P3 aae scheduling to the C1 and P2 will be

scheduling to the C_3. This can be done by master server.

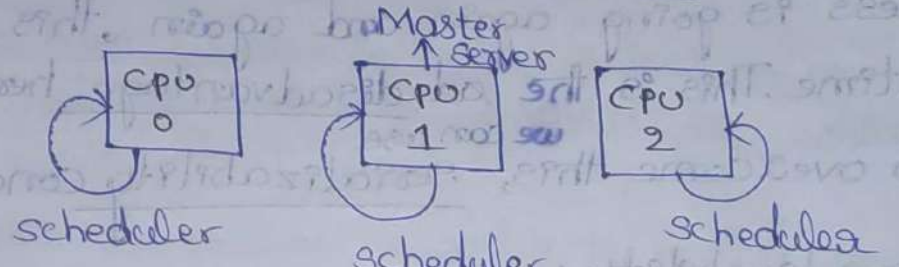
Disadvantage:-

* In this other processes (slave servers) are waiting for commands from master server.
* This can may lead "time-over head."
* "Performance degradation".

(ii) Symmetric multi-processing:-

→ In symmetric multiprocessing approach every processor has it's own scheduler.

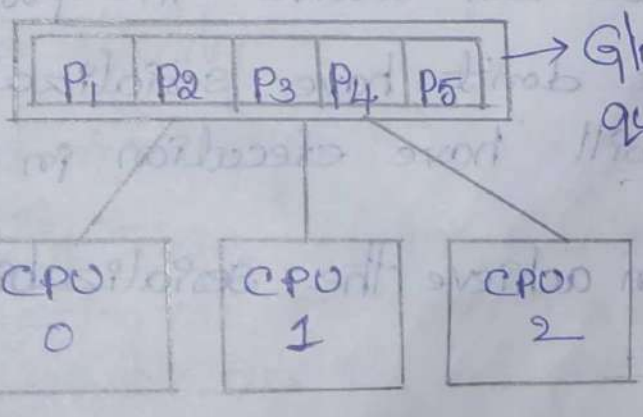
Ex:-



→ In this all processes may be in a common ready queue or each processor may have its private queue for ready processes.

→ Symmetric approach can be implemented by global queue and per-cpu queue.

Global queue:-



All the CPU's shares a common queue, that is called Global queue.

Advantages:-

* all the other CPU's are not waiting for commands
* CPU utilization is maximum.
* fair distribution of work.

Disadvantages:-

① In this queue CPU 0, CPU 1 selecting the same process P_1 at a time, the execution for same process is going again and again, this will give overatime. This is the ~~a~~ disadvantage here.

→ To overcome this, we can use serializability concept.

→ Serializability:-
Giving serial number for every process

* By serializability CPU 0 will alloted to P_1 and it goes from ready state to running state. So, P_1 is scheduled to CPU 0.

→ Now, CPU 1 can select the Process P_2 because P_1 is already is on executing. In this manner we can resolve the problem.

→ If we don't have serializability, same process will have execution in multiple times.

→ We can achieve the serializability by using Locking.

Locking :-

→ We can implement the serializability by using of "Locking". If one process is accessed by or scheduled to one processor, then the same process will not be executed by other processor.

→ With the help of Locking we can stop the process which is currently allotted to the CPU not to allot/schedule to other processor.

→ Locking is not optimal solution because we can stop the minimum processes but when the processes in Large, we can't stop them. It's take so much of time

(2) Processor Affinity :-

It provides a chance to select ~~a processor from user~~ which process is executed by which processor from users.

→ Suppose the process migrates to another processor the data and instruction of process which is present in processor will be call again carried to migrated processor.

→ processor affinity avoid process migration.

→ It try to avoid migrating processes from one processor to another and keep a process running on the same processor.

→ Processor Affinity is two types.

<i> soft Affinity.

<ii> Hard Affinity.

<i> Soft Affinity :- When os has a policy of keeping a process running on the same processor, it will try to don't migrate, but it not give 100% guarantee.

(2). Hard Affinity :-

It contains a set of processes. A process will migrate to the set of processors only.

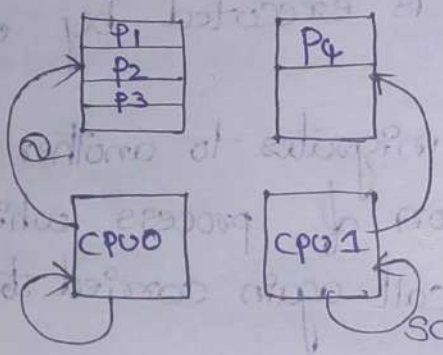
Ex:- Linux system contain soft affinity and hard affinity.

→ There is a system call in linux sched set affinity () it support soft and hard affinity.

Per CPU Queue (Local Queue) :-

At the starting of execution of a process the os decides which CPU the process needs to be executed.

→ Here every process has it's own queue.



Advantage :-

<1> Easy to implement

<2> Conflict-not arise

<3> no locking mechanism

<4> Scalable

<5> Locality (own CPU queue) not bothered about other cpus.

Disadvantages:-

→ Fair distribution not occur in each and every process.

→ Load imbalance:- It occurs when some cpus queue have a lot of processes to execute in a queue and some CPU has less process to execute in queue.

→ To reduce we follow 2 methods.

(1) Push migration:-

It will monitor each and every process load if one processor with less load it will distribute to make balance.

(2) Pull migration:-

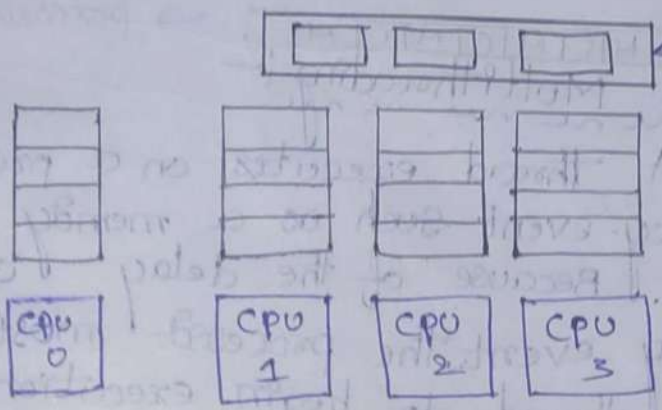
Idle processor will get the process from busy processor.

Hybrid approach:-

→ It uses local and global queues.

Local queue ensure locality.

global queue maintain fair distribution ensure load balance.



Multi-core Processor:-

on a single chip multiple processors are placed

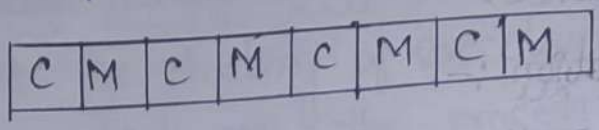
→ It is faster access.

→ less power consumes.

Memory stall :-

If CPU fetching the process, if the data doesn't available in the memory we need to transfer from the disk. It take some time to place data in memory.

Thread :- It is a single instruction of a program. In such cases; the processor can spend upto 50% of it's time waiting for data to become available from memory. To solve this problem, recent hardware design have implemented multithreaded processor.

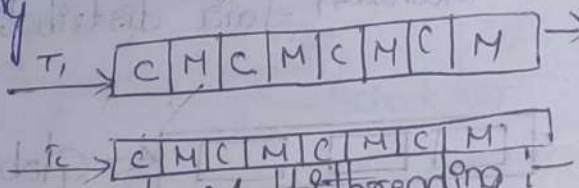


on thread
C - execution done by CPU (compute cycle)

M - Memory stall.

Multi-threaded :-

If one thread stalls while waiting for the memory



(1) Coarse Grained Multithreading :-

A thread executes on a processor until a long latency event such as a memory stall occurs in CGMT. Because of the delay caused by the long latency event, the processor must switch to another thread to begin execution.

The cost of switching blw threads is high as the instruction pipeline must be terminated before the other thread can begin execution on the processor core. once this new thread begins execution it begins filling the pipeline with its instruction.

(2) Fine-Grained Multithreading:-

This Multithreading switches b/n threads at a much finer level, mainly at the boundary of an instruction cycle. This includes logic for thread switching and as a result the cost of switching b/n threads is small.