CPU Scheduling

CPU Scheduling is a technique in OS which handles the removal or selection of processes.

Types

Pre-Emptive

- If a Low priority task is currently in the CPU and another task with high priority demand CPU access. First priority will be removed until high priority task terminates.
  - → SRTF (Shortest Remaining Time First)
  - → LRTF (Longest Remaining Time First)
  - → Round Robin
  - → Priority based

Non-Preemptive

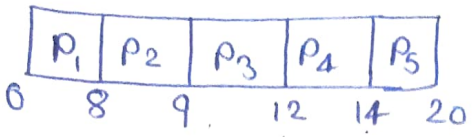
- Once a process enters CPU, until it terminates, other processes have to wait
- Priority is not considered
  - → FCFS (First Come First Serve)
  - → SJF (Shortest Job First)
  - → LJF (Longest Job First)
  - → HRRN (Highest Response Ratio Next)
  - → Multilevel Queue

- **FCFS Scheduling Algorithm**
  - Works on First Come First Serve basis.
  - It is Non-Preemptive

© Consider following table and calculate Average Turn Around Time (TAT) and Average Waiting Time using FCFS Algo.

Process	Arrival Time	Burst Time	Priority
P_1	0	8	3
P_2	1	1	1
P_3	2	3	2
P_4	3	2	3
P_5	4	6	4

GANTT CHART:



Criteria: Arrival Time

Mode: Non Preemptive

Process No.	Arrival Time (A.T)	Burst Time (B.T)	Completion Time (C.T)	(TAT) (CT-AT)	(WT) (TAT-BT)
P_1	0	8	8	8	0
P_2	1	1	9	8	7
P_3	2	3	12	10	7
P_4	3	2	14	11	9
P_5	4	6	20	16	10

$$\text{Avg. WT} = \frac{0+7+7+9+10}{5} = 6.6$$

$$\text{Avg. TAT} = \frac{8+8+10+11+16}{5} = 10.6$$

- SJF (Shortest Job First) Non-Preemptive.

Process with smallest burst Time will get CPU first. If 2 process has same burst Time then FCFS is used to solve the question.

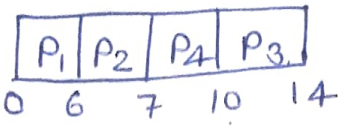
→ Same Arrival, small burst

Criteria: Burst Time Mode: Non-Preemptive

- Calc ATAT & AWT using SJF Algo.

Process	Arrival	Burst
P_1	0	6
P_2	2	1
P_3	4	4
P_4	5	3

GANTT Chart



Process No.	Arrival Time	Burst Time	Completion Time	(TAT)	(WT)
P_1	0	6	6	6	0
P_2	2	1	7	5	4
P_3	4	4	14	10	6
P_4	5	3	10	5	2

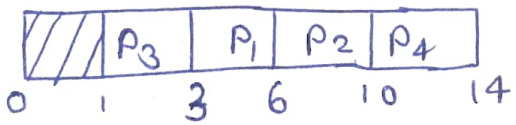
$ATAT = \frac{26}{4} = 6.5$

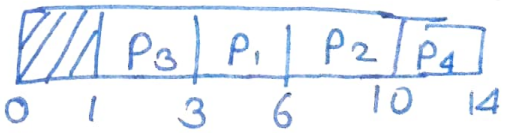
$AWT = \frac{12}{4} = 3$

2nd)

Process No.	Arrival Time	Burst Time	Completion Time	TAT	WT
P_1	1	3	6	5	2
P_2	2	4	16	8	4
P_3	1	2	3	2	0
P_4	4	4	14	10	6

GANTT Chart





SRTF (Shortest Remaining Time First)

It is Preemptive Approach of SJF.

Process No	Arrival Time	Burst Time
P_1	0	8
P_2	1	6
P_3	2	3
P_4	3	1
P_5	4	2
P_6	5	1

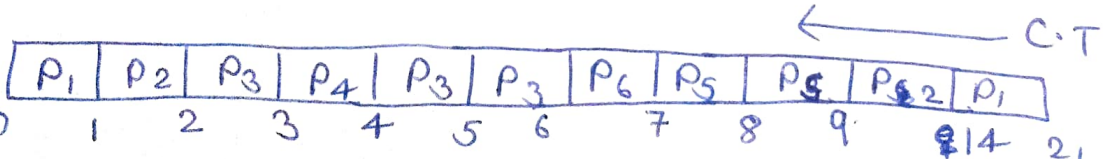
Criteria: Burst Time / Remaining Time

Mode: Preemptive

Process No.	Arrival Time	Burst Time
P_1	0	~~8~~ 7
P_2	1	6 5
P_3	2	~~3~~ 2 1 0
P_4	3	1 0
P_5	4	2 1 0
P_6	5	1 0

- Sabse pehle Arrival Time k hisab se process lo or ek ek process roko or check kro kiska sabse remaining Time bcha hai or usko pehle lo lo.
- Agar do case same toh jo pehle aaya usko Priority do or execute kro for 1 clock cycle

GANTT CHART



P_1 P_2

P_1 P_2 P_3

P_1 P_2 P_3 P_4

P_1 P_2 P_3 P_5

P_1 P_2 P_3 P_5 P_6

P_1 P_2 P_5 P_6

P_1 P_2 P_5

P_1 P_2

Process No.	Arrival Time	Burst Time	CT	TAT	WT
P_1	0	8	21	21	13
P_2	1	6	14	13	7
P_3	2	3	6	4	1
P_4	3	1	4	1	0
P_5	4	2	9	5	3
P_6	5	1	7	2	1

ATAT: 7.5 AWT: 4.16

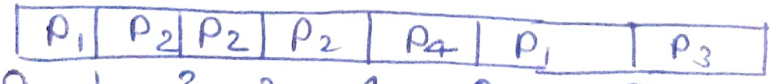
Q-2

Process NO	Arrival Time	Burst Time
P_1	0	5
P_2	1	3
P_3	2	4
P_4	4	1

Ans

Process No.	Arrival Time	Burst Time
P_1	0	~~8~~ 3
P_2	1	~~3~~ ~~2~~ ~~1~~ 0
P_3	2	4
P_4	4	~~1~~ 0

GANTT CHART



0 1 2 3 4 5 9 13

P_1 P_2

P_1 P_3

P_1 P_2 P_3

P_1 P_2 P_3

P_1 P_3 P_4

Process No.	Arrival Time	Burst Time	CT	TAT	WT
P_1	0	5	9	9	4
P_2	1	3	4	3	6
P_3	2	4	13	11	7
P_4	4	1	5	1	0
				6	2.75

- Priority Scheduling Algorithm (Pre-emptive)

CPU is allocated First to highest priority process.

Criteria: Priority

Mode : Pre-emptive

Priority
Lowest Highest
1 ⟶ 10...

Q

Process No.	Arrival Time	Burst Time	Priority
P_1	0	43	2
P_2	1	21	4
P_3	2	65	6
P_4	3	53	10
P_5	4	4	8
P_6	5	40	12
P_7	6	6	9

Arrival time k hisab se sbko load krna hai. Tb tk process execute hoga jb tk running process se bda priority wala nhi aata.

Ans GANTT CHART

P_1^2 P_2^4 P_3^6 P_4^{10} P_5^8 P_7^9

P_1	P_2	P_3	P_4	P_6	P_4	P_7	P_5	P_3	P_2	P_1

0 1 2 3 5 9 12 18 22 29 30 33

P_1 P_2 P_3 P_5

Process No.	Arrival Time	Burst Time	Completion Time	TAT	WT
P_1	0	4	33	33	29
P_2	1	2	30	29	27
P_3	2	8	29	27	19
P_4	3	5	12	9	4
P_5	4	4	22	18	14
P_6	5	4	9	4	0
P_7	6	6	18	12	6
				18.85	14.14

- Round Robin Algorithm (Preemptive)

Q Time quantum: 2 unit

Process No.	Arrival Time	Burst Time
P_1	0	~~8~~ ~~6~~ 4 2 0
P_2	1	~~6~~ ~~4~~ ~~2~~ 0
P_3	2	~~2~~ 0
P_4	3	~~2~~ 0
P_5	4	5 ~~3~~ 1
P_6	6	~~4~~ ~~2~~ 0

Ans

Queue : ~~P_1 P_2 P_3 P_1 P_4 P_5 P_2 P_6 P_5~~ P_6 P_5

GANTT CHART :

P_1

P_1 P_2 P_3

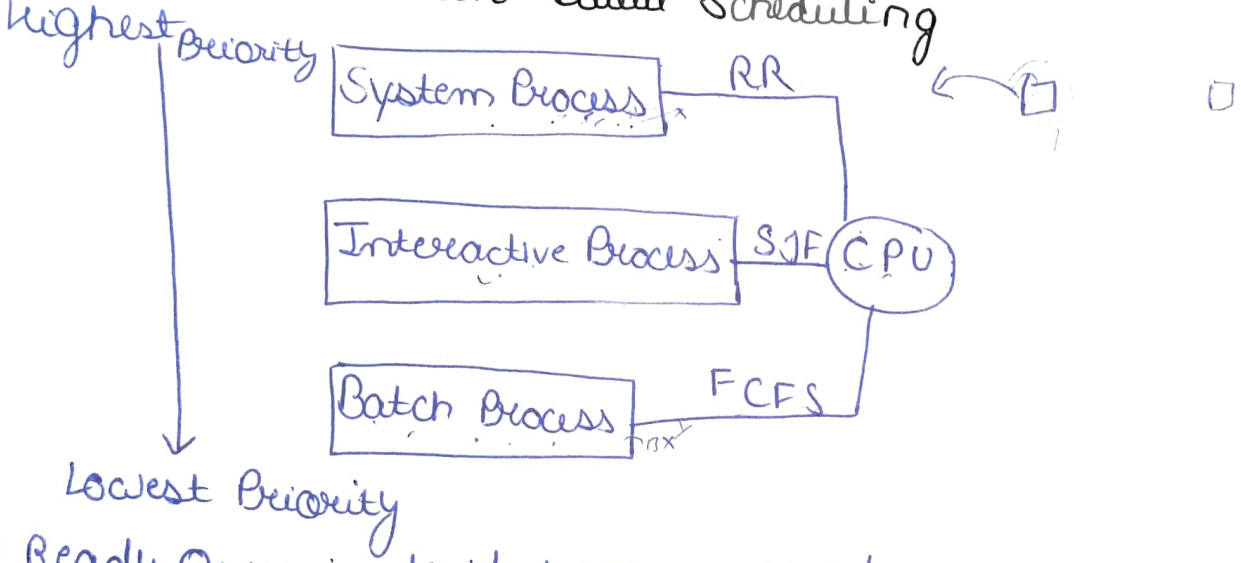
19

P_1	P_2	P_3	P_1	P_4	P_5	P_2	P_6	P_5	P_6	P_5

0 2 4 6 7 9 11 13 15 17 21 22

Process No.	Arrival Time	Burst Time	Completion Time	TAT	WT
1	6	3	7	7	4
2	1	6	19	18	12
3	2	2	6	4	2
4	3	2	9	6	4
5	4	4	22	18	14
6	5	4	21	15	11
				ATAT 11.33	AWT 7.83

~~Deadlock~~ • Multi-level Queue Scheduling



→ Ready Queue is divided into multiple queue depending upon priority.

→ A process is permanently assigned to one of the queues based on some property of process, memory size, process priority or process type.

→ Each queue has its own scheduling algorithm.

→ System Process : Created by OS (Highest Priority)
Interactive Process : Needs user input
Batch Process : Runs silently, no user input required

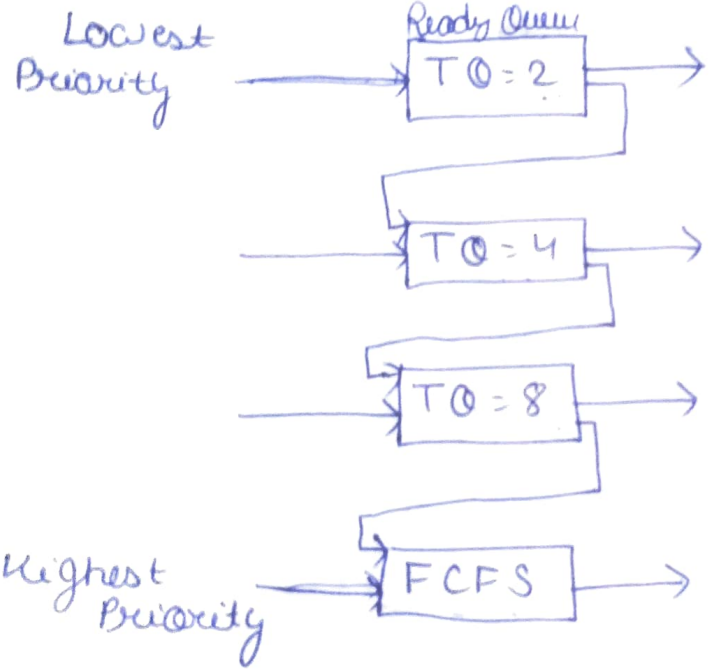
→ Scheduling among different sub-queues is implemented as fixed priority preemptive scheduling

→ If an interactive process comes & batch process is currently executing. Then batch process will be preempted

→ Problem: Only after completion of all the processes from the top-level ready queue the further level ready queue will be scheduled.

This came starvation for lower priority process

- **Multilevel Feedback Queue Scheduling**



→ Multiple Sub-queues are present

→ Allows the process to move between queues. This idea is to separate processes acc. to the characteristic of their Burst Time.

→ Less Starvation

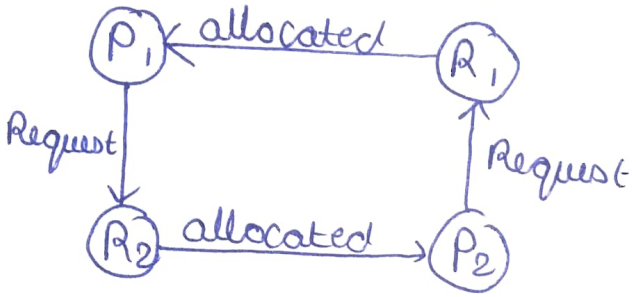
Deadlock

Deadlock is a situation where a set of processes are blocked because each process ~~are~~ ~~blocked~~ is holding a resource and waiting for another resource acquired by some other process.

OR

Two or more processes are waiting on some resource's availability which will never be available as it is also busy with some other process. The processes are said to be in Deadlock.

- Deadlock is a bug present in the process / thread synchronization method.
- Example of resource : Memory space, CPU cycles, files, I/O devices.
- Single resource can have multiple instances of that.
  eg- CPU is a resource and a system can have 2 CPU's.



How a process utilize a resource ?

a) Request : Request the R_1 if R is free lock it, else wait till it is available.

b) Use

c) Release : Release resource instance and make it available for other resource processes.

Deadlock Neassary Conditions:

4 condition should hold simultaneously

a) Mutual Exclusive

- Only 1 process at a time can use the resource, if another process requests that resource, the requesting process must wait until the resource has been released.

b) Hold & Wait

- A process must be holding at least one resource & waiting to acquire additional resources that are currently being held by other processes.

c) No preemption

- The process which once scheduled will be executed till the completion. (No resource preemption)

d) Circular wait

- All the prauses must be waiting for the resource in a cyclic manner.

Methods for handling Deadlocks:

a) Use a protocal to prevent or avoid deadlocks, ensuring that the system will never enter a deadlocked state.

b) Allow the system to enter a deadlocked state, detect it, and recover it.

c) Ignore the problem altogether and pretend that deadlocks never occur in system (Ostrich algorithm) or deadlock ignorance. (Deadlock kbhi ho hi nhi skta, System assume krta hai sb thik programmer kr OS kuchbhi krega)

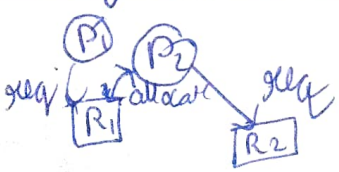
Deadlock Prevention: by ensuring at least one of the necessary condition cannot hold.

a) Mutual Exclusion
- Use locks only for non-shareable resource.
- Sharable resources like Read-~~only~~ files can be accessed by multiple processes/threads.
- However not always possible to prevent deadlock by preventing Mutual exclusion (Making all resources shareable)

b) Hold and wait
- To ensure H & W condⁿ never occur in the system, we must guarantee that whenever a process requests a resource, it doesn't hold any other resource.
- Before start → all required resources acquired.
- reduce throughput → those not needed even get earlier.

c) No Preemption
- Forceful preemption → We allow a process to forcefully preempt the resource holding by other ~~res~~ processe.
- This method may be used by high priority process or system process.

d) Circular Wait
- To ensure that this condition never holds is to impose a proper ordering of resource allocation.



- Banker's Algorithm (Deadlock Avoidance, Deadlock detection)

Process	Allocation			Max Need			Available			Remaining Need (Max − Allocation)		
	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3	R_1	R_2	R_3
P_1	0	1	0	7	5	3	3	3	2	7	4	3 ✓
							+2	0	0			
P_2	2	0	0	3	2	2 ✗	5	3	2	1	2	2 ✓
							+2	1	1			
P_3	3	0	2	9	0	2	7	4	3	6	0	0 ✓
							+0	0	2			
P_4	2	1	1	4	2	2 ✗	7	4	5	2	1	1 ✓
							+0	1	0			
P_5	0	0	2	5	3	3 ✗	7	5	5	5	3	1 ✓
							+3	0	2			
	7	2	5				10	5	7			

Total A=10, B=5, C=7

$P_2 \rightarrow P_4 \rightarrow P_5 \rightarrow P_1 \rightarrow P_3$

- Process Synchronization

Two types of Process categories:

1) Independent Process: When one Process is executing, it does not affect other executing proces

2) Cooperative Process: It affects other executing process.

-Race Condition: It occurs when two or more Processes are executed at the same time, not scheduled in Proper sequence & not executed critical section correctly, which causes 'Data inconsistency and data loss'.

Concurrent Process ⟶ Same Time Shared variable Access

A = 7 Shared variable

P1

```
{
Read(A);
A = A*2;
Write(A)
}
```

Print

P2

```
{
Read(A);
A = A+5;
Write(A);
}
```

A	P_1	P_2	
7	14	19	→ Synchronized way running
7	14	12	→ Asynchronized

Process Synchronization: It helps to maintain shared data consistency & cooperating process execution. When Process execute it should be ensured that concurrent access to shared data does not creat inconsistencies.

Producer Consumer Problem OR Bounded Buffer Problem

Producer code

```
int count=0;
void Producer(void)
{
  int Itemp;
  while(true)
  {
    producer_item(Itemp);
    while(Count==N);
    // Buffer full - do nothing
    Buffer[in]=Itemp;
    in=(in+1) MOD N;
    Count=Count++;
  }
}
```

- Count is Global (shared variable)
  Tot. items present

N = 8
Count = 3
in = 3

	Buffer[0, N-1]
0	x
1	y
2	z
3	
4	
5	
6	
7	

out = 0

(P)

Count = Count++;

1) LOAD R_p in[Count]
2) INCR R_p
3) STORE m[Count], R_p

Consumer Code

```
void Consumer(void)
{
  int Itemc;
  while (True)
  {
    while (Count == 0);
    // Buffer Empty - do nothing
    Itemc = Buffer(out);
    OUT = OUT+1 MODN;
    Count = Count-1;
    Process_item(Item());
  }
}
```

(c)

Count = Count - 1

4) LOAD R_c m[Count]

5) DECR R_c

6) STORE in[Count], R_c

- Critical Section

The portion of program text, where shared resources will be placed

eg: Let COUNT is a shared variable

```
void main()
{
  ==========
  Count++;
  ==========
}
```

```
void main()
{
  Non Critical Section
  Critical Section
  Non Critical Section
}
```

General structure of a typical Process.

```
P
{
  Entry Section
  Critical Section
  Exit Section
  Remainder Section
}
```

Q.

The important feature of the system is that, when one process is executing in its critical section, no other process is to be allowed to execute in its critical section. That is, no two other processes is executing in their critical section at the same time.

→ Requirement for the solution of Critical Section/ Process Synchronization

i) Mutual Exclusion: Only one process is allowed in Critical Section at any point of time.

ii) Progress: No process running outside the critical section should lock the other process for entering in critical section.

iii) Bounded waiting: No process should wait infinitely for critical section. There should be time bound on getting chance to enter in critical section.

iv) No assumption regarding: No. of CPU / Speed of hardware.

- Solution of Process Synchronization.

1) Software Support

→ a) Lock variable

→ b) Strike Alteration OR Dekker's Algorithm

→ c) Peterson's Algorithm

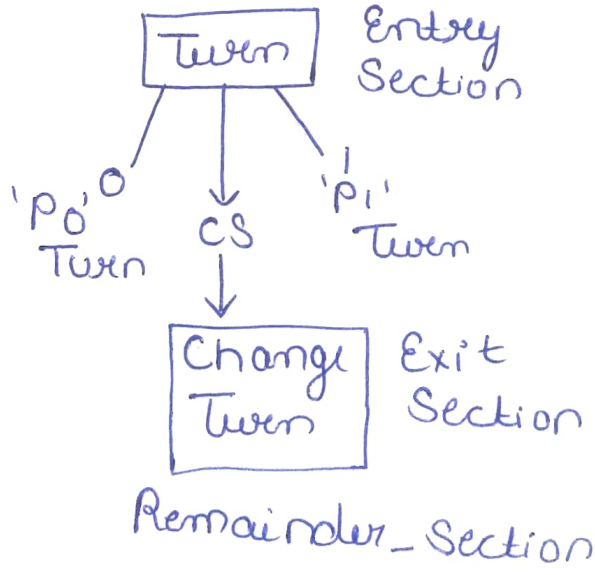
Tot. items present

2) Hardware Support
   - a) TSL instruction set
   - b) Disabling interrupt

3) OS Support
   - Semaphore
     - Counting Semaphor
     - Binary Semaphor

4) Compiler Support
   - Monitors.

→ Strict Alteration OR Turn Variable
- It is S/W Solution
- Busy waiting
- Only 2 Process solution



Criti

P2 Criti

P1

P5 4 6 4

Process 'P0' Code

```
P0()
{
  while(Turn)
  {
    NON-CS();
    while(Turn != 0)
      CS:
    Turn = 1;
  }
}
```

Process 'P1' Code

```
P1()
{
  while(1)
  {
    while(Turn != 1)
      Critical Section;
    Turn = 0;
  }
}
```

- **Semaphores**

Semaphore (S) is an integer variable which can only be accessed through two standard atomic operations Down & up.

Counting Semaphore : It used for giving control access to resources consisting infinity no. of instances.

Binary Semaphore: To deal with critical section problem.