(Shortest-Job-First Scheduling)

- This algorithm associates with each process the length of the process's next CPU burst.
- When the CPU is available, 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.

The SJF algorithm can be either preemptive or nonpreemptive

A more appropriate term for this scheduling method would be the

Shortest-Next-CPU-Burst Algorithm

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

Example of SJF Scheduling (Non-Premptive)

Consider the following set of processes,with the length of the CPU burst given in milliseconds:

Process ID	Burst Time
P1	6
P2	8
P3	7
P4	3

Waiting Time for P1 = 3 ms

Waiting Time for P2 = 16 ms

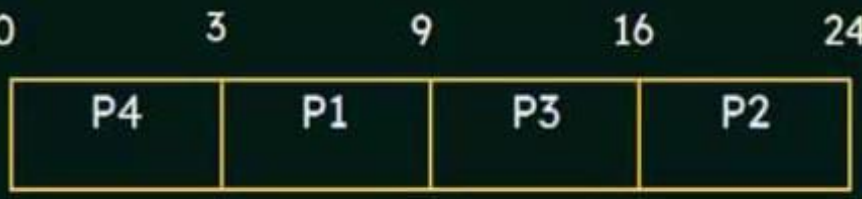
Waiting Time for P3 = 9 ms

Waiting Time for P4 = 0 ms

Average Waiting Time

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

Gantt Chart:



By comparison, if we were using the FCFS scheduling scheme, the average waiting time would be 10.25 milliseconds

Example of SJF Scheduling (Premptive)

Consider the following four processes, with the length of the CPU burst given in milliseconds and the processes arrive at the ready queue at the times shown:

Process ID	Arrival Time	Burst Time
P1	0	~~8~~ 7
P2	1	4
P3	2	9
P4	3	5

Waiting Time for P1 = (10 – 1 - 0) = 9 ms

Waiting Time for P2 = (1 – 0 - 1) = 0 ms

Waiting Time for P3 = (17 – 0 - 2) = 15 ms

Waiting Time for P4 = (5 – 0 - 3) = 2 ms

Gantt Chart:

Average Waiting Time

= (9 + 0 + 15 + 2)/4 = 6.5 ms

Waiting Time =Total waiting Time – No. of milliseconds Process executed – Arrival Time

Example of SJF Scheduling (Premptive)

Consider the following four processes, with the length of the CPU burst given in milliseconds and the processes arrive at the ready queue at the times shown:

Process ID	Arrival Time	Burst Time
P1	0	~~8~~ 7
P2	1	4
P3	2	9
P4	3	5

Waiting Time for P1 = (10 – 1 - 0) = 9 ms

Waiting Time for P2 = (1 – 0 - 1) = 0 ms

Waiting Time for P3 = (17 – 0 - 2) = 15 ms

Waiting Time for P4 = (5 – 0 - 3) = 2 ms

Gantt Chart:

Average Waiting Time

= (9 + 0 + 15 + 2)/4 = 6.5 ms

Waiting Time =Total waiting Time – No. of milliseconds Process executed – Arrival Time

Problems with SJF Scheduling:

- The real difficulty with the SJF algorithm is knowing the length of the next CPU request.
- Although the SJF algorithm is optimal, it cannot be implemented at the level of short-term CPU scheduling.
- There is no way to know the length of the next CPU burst.

One approach is:

- To try to approximate SJF scheduling.
- We may not know the length of the next CPU burst, but we may be able to predict its value.
- We expect that the next CPU burst will be similar in length to the previous ones.
- Thus, by computing an approximation of the length of the next CPU burst, we can pick the process with the shortest predicted CPU burst.

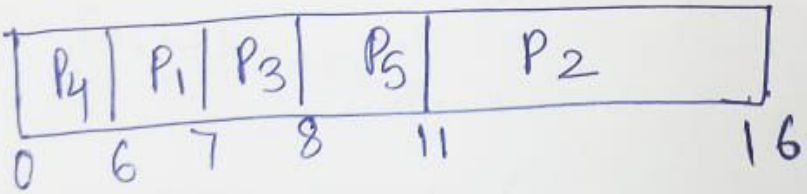
1.

SJF
→ Pre-emptive (SRTF)
→ Non-preemptive

	Arrival time	Burst time	C.T	TAT	W.T	R.T
P_1	2	1	7	5	4	4
P_2	1	5	16	15	10	10
P_3	4	1	8	4	3	3
P_4	0	6	6	6	0	0
P_5	2	3	11	9	6	6

Ready
P_1 P_2 P_3 P_5

Gantt chart



DISADvantage:

→ can not be implemented
→ starvation problem with process having large burst time.

Pre-emptive

Whenever new process arrives, there may be pre-emption of the running process.

	AT	BT	CT	TAT	WT	Response time
P_1	2	1	3	1	$P_1 \rightarrow 0$	0
P_2	1	5	16	15	$P_2 \rightarrow 10$	10
P_3	4	1	5	1	$P_3 \rightarrow 0$	0
P_4	0	6 $\cancel{5}_4$	11	11	$P_4 \rightarrow 5$	0
P_5	2	3	7	5	$P_5 \rightarrow 2$	1

P_4, P_2, P_1, P_5, P_3

Gantt-chart

P_4	P_4	P_1	P_5	P_3	P_5	P_4	P_2

0 1 2 3 4 5 7 11 16

Advantage → Min. average WT & min average Tat.

→ Better response time than FCFS

→ Max. throughput

→ Provide a standard for other Algo in case of avg. WT.

Scheduling Algorithms
(Priority Scheduling)

- A priority is associated with each process, and the CPU is allocated to the process with the highest priority.

- Equal-priority processes are scheduled in FCFS order.

- An SJF algorithm is simply a priority algorithm where the priority is the inverse of the (predicted) next CPU burst.
  The larger the CPU burst, the lower the priority, and vice versa.

Priority scheduling can be either preemptive or nonpreemptive.

A preemptive priority scheduling algorithm will preempt the CPU if the priority of the newly arrived process is higher than the priority of the currently running process.

A nonpreemptive priority scheduling algorithm will simply put the new process at the head of the ready queue.

Consider the following set of processes, assumed to have arrived at time 0, in the order P1, P2, P3, P4, P5, with the length of the CPU burst given in milliseconds:

Process ID	Burst Time	Priority
P1	10	3
P2	1	1
P3	2	4
P4	1	5
P5	5	2

Using Priority Scheduling, we would schedule these processes according to the following Gantt Chart:

Waiting Time for P1 = 6 ms

Waiting Time for P2 = 0 ms

Waiting Time for P3 = 16 ms

Waiting Time for P4 = 18 ms

Waiting Time for P5 = 1 ms

Average Waiting Time

= (6 + 0 + 16 + 18 + 1) / 5

= 41 / 5 ms

= 8.2 ms

Problem with Priority Scheduling

A major problem with priority scheduling algorithms is indefinite blocking, or starvation.

A process that is ready to run but waiting for the CPU can be considered blocked.

A priority scheduling algorithm can leave some low priority processes waiting indefinitely.

In a heavily loaded computer system, a steady stream of higher-priority processes can prevent a low-priority process from ever getting the CPU.

Solution to the Problem

A solution to the problem of indefinite blockage of low-priority processes is aging.

Aging is a technique of gradually increasing the priority of processes that wait in the system for a long time.

For example,

If priorities range from 127 (low) to 0 (high), we could increase the priority of a waiting process by 1 every 15 minutes.

Eventually, even a process with an initial priority of 127 would have the highest priority in the system and would be executed.