Scheduling Algorithms (Overview & Problems)

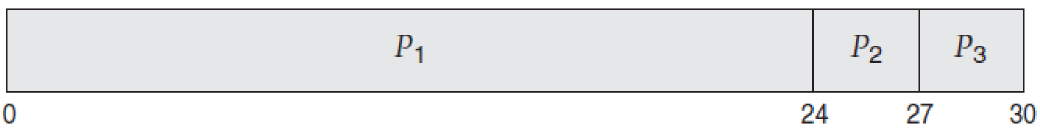
Algorithms decide which of the processes in the ready queue is to be allocated the CPU.

1. First-Come, First-Served Scheduling (FCFS)

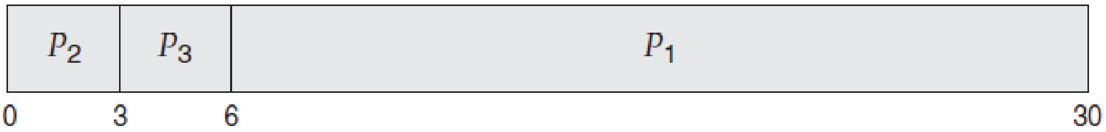
- The process that requests the CPU first is allocated the CPU first.
- The implementation of the FCFS policy is easily managed with a **FIFO** queue.
- **Always non-preemptive**
- Consider the following set of processes that arrive at time 0, with the length of the CPU burst given in milliseconds:

Process	Burst Time
P_1	24
P_2	3
P_3	3

- If the processes arrive in the order *P*1, *P*2, *P*3, and are served in FCFS order, we get the result shown in the following '**Gantt chart**', ***which is a bar chart that illustrates a particular schedule, including the start and finish times of each of the participating processes:***



- The waiting time is 0 milliseconds for process *P*1, 24 milliseconds for process *P*2, and 27 milliseconds for process *P*3.
- Thus, the **average waiting time** is (0+ 24 + 27)/3 = **<u>17 milliseconds</u>**.
- If the processes arrive in the order *P*2, *P*3, *P*1,



  The waiting time is 6 milliseconds for process *P*1,
  0 milliseconds for process *P*2, and
  3 milliseconds for process *P*3.
- The average waiting time is now (6 + 0 + 3)/3 = **<u>3 milliseconds</u>**.
  This reduction is substantial.
  Thus, the average waiting time under an FCFS policy is generally not minimal and may vary substantially if the processes' CPU burst times vary greatly.

<u>Advantages</u>:

- Simplest
- Easy to write and implement

<u>Disadvantages</u>:

- **The average waiting time under the FCFS policy is often quite long**.
- It may vary substantially if the processes' CPU burst times vary greatly.

- There is a ‘**convoy effect’,** as all the other processes wait for the one big process to get off the CPU. This effect results in lower CPU and device utilization. It can be solved if the shorter processes were allowed to go first. Convoy effect occurs if one big CPU bound process comes along with some I/O bound processes
- FCFS scheduling algorithm is **nonpreemptive**. Once the CPU has been allocated to a process, that process keeps the CPU until it releases the CPU, either by terminating or by requesting I/O. So FCFS algorithm is **not suitable for time-sharing systems**
- It would be **disastrous** to allow one process to keep the CPU for an extended period.

2. Shortest-Job-First Scheduling (SJF)

- 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.**
- 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.
- Consider the following set of processes that arrive at time 0, with the length of the CPU burst given in milliseconds

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

The scheduling is done as follows:

P_4	P_1	P_3	P_2

0 3 9 16 24

The waiting time is 3 milliseconds for process *P*1,
16 milliseconds for process *P*2,
9 milliseconds for process *P*3, and
0 milliseconds for process *P*4.

- Thus, the average waiting time is (3 + 16 + 9 + 0)/4 = **7 milliseconds**.
- By comparison, if we were using the FCFS scheduling scheme for this example, the average waiting time would be 10.25 milliseconds.

P1	P2	P3	P4

0 6 14 21
24

The waiting time is 0 milliseconds for process *P*1,
6 milliseconds for process *P*2,
14 milliseconds for process *P*3, and
21 milliseconds for process *P*4.

- Thus, the average waiting time is (0 + 6 + 14 + 21)/4 = **10.25 milliseconds**.

Advantage:

- The SJF scheduling algorithm is **optimal**, in that it gives the **minimum average waiting time** for a given set of processes.

Disadvantages:

- 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**. With short-term scheduling, there is no way to know the length of the next CPU burst. One solution to this problem is to **try to approximate** SJF scheduling. We may be able to **predict** its value. We expect that the next CPU burst will be similar in length to the previous ones. The next CPU burst is generally predicted as an exponential average of the measured lengths of previous CPU bursts. It is **calculated using exponential average formula**

- The SJF algorithm can be either **preemptive or nonpreemptive**.
- **Preemptive SJF scheduling is sometimes called shortest-remaining-time-first scheduling (SRTF).**
- Example, consider the following four processes, with the length of the CPU burst given in milliseconds:

Process	Arrival Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

- **Nonpreemptive SJF** scheduling would result in an average waiting time of 7.75 milliseconds.

P1	P2	P4	P3

0 8 12 17 26

The waiting time for process *P*1: P1 started at 0, P1 arrived at 0. So waiting time for P1 in first slot is (0-0) = 0.

For process *P*2, started at 8, but arrived at 1, So waiting time is (8-1) = 7 milliseconds

For process *P*3, started at 17, but arrived at 2, So waiting time is (17-2) = 15 milliseconds

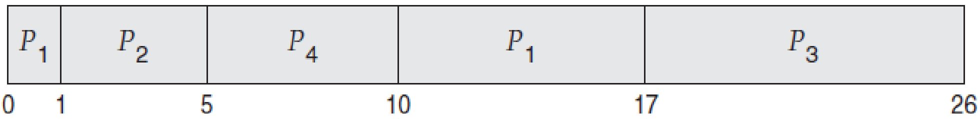
For process *P*4, started at 12, but arrived at 3, So waiting time is (12-3) = 9 milliseconds

- Average waiting time is (0+7+15+9)/4 = **7.75 milliseconds.**

From Gantt chart, we can directly calculate the average waiting time as

[(0-0) + (8-1) + (17-2) + (12-3)] / 4 = **7.75 milliseconds**

- **Preemptive SJF (SRTF)** schedule is as depicted in the following Gantt chart:



Process $P1$ is started at time 0, since it is the only process in the queue. Process $P2$ arrives at time 1. The remaining time for process $P1$ (7 milliseconds) is larger than the time required by process $P2$ (4 milliseconds), so process $P1$ is preempted, and process $P2$ is scheduled.

The waiting time for process $P1$: P1 is executed in 2 slots. In first slot, P1 started at 0, P1 arrived at 0. So waiting time for P1 in first slot is (0-0) = 0. In second slot, it started at time 10, but first slot ended at time 1. So waiting time in second slot is (10-1) = 9. So total waiting time for P1 is 0+9 = 9 milliseconds.

For process $P2$, only one slot. Starts at time 1, but arrived at time 1 only. So waiting time is (1-1) = 0 milliseconds

For P3, (17 – 2) = 15 and for P4, (5-3) = 2 milliseconds.

Average waiting time is (9+0+15+2)/4 = **6.5 milliseconds.**

From Gantt chart, we can directly calculate the average waiting time

[((0-0)+(10 − 1)) + (1 − 1) + (17 − 2) + (5 − 3)]/4 = 26/4 = **6.5 milliseconds**.

3. Priority Scheduling

- Based on the priority of the process
- Both pre-emptive and non-preemptive

Advantage:

- Best suitable for real-time systems

Disadvantage:

- Starvation (Indefinite blocking) – Solution Aging

Q1. Consider the following set of processes that arrive at time 0, with the length of the CPU burst given in milliseconds:

Process	Burst Time	Priority
P1	6	2
P2	8	3
P3	7	1
P4	3	4

- Answer:

P3	P1	P2	P4

0 7 13 21 24

The waiting time is 7 milliseconds for process *P*1,

13 milliseconds for process *P*2, and

0 milliseconds for process *P*3.

21 for P4

Thus, the **average waiting time** is (7+ 13+0 + 21)/4 = **10.25 milliseconds**.

Finding Turnaround Time

Turnaround time for P1: 13 – 0 = 13

Turnaround time for P2: 21 – 0 = 21

Turnaround time for P3: 7 – 0 = 7

Turnaround time for P4: 24 – 0 = 24

Average Turnaround Time is : (13+21+7+24)/4 = **<u>16.25 mS</u>**

Since arrival times for all processes are 0, both preemptive and non-preemptive scheduling will give the same answers

Q2. Consider the following set of processes with the length of the CPU burst given in milliseconds:

Process	Arrival Time	Burst Time	Priority
P1	0	6	2
P2	1	8	3
P3	2	7	1
P4	3	3	4

- **Non-Preemptive priority scheduling**

P1	P3	P2	P4

0 6 13 21 24

The waiting time for process *P*1 : 0 – 0 = 0

The waiting time for process *P*2 : 13 – 1 = 12

The waiting time for process *P*3 : 6 – 2 = 4

The waiting time for process *P*4 : 21 – 3 = 18

Thus, the **average waiting time** is (0+ 12 + 4 + 18)/4 = **<u>8.5 milliseconds</u>**.

Turnaround time for P1: 6 – 0 = 6

Turnaround time for P2: 21 – 1 = 20

Turnaround time for P3: 13 – 2 = 11

Turnaround time for P4: 24 – 3 = 21

Average Turnaround Time is : (6+20+11+21)/4 = **14.5 mS**

- **Preemptive priority scheduling**

P1	P3	P1	P2	P4

0 2 9 13 21 24

The waiting time for process *P*1 : (0 – 0)+(9 – 2) = 7

The waiting time for process *P*2 : 13 – 1 = 12

The waiting time for process *P*3 : 2 – 2 = 0

The waiting time for process *P*4 : 21 – 3 = 18

Thus, the **average waiting time** is (7+ 12+0 + 18)/4 = **9.25 milliseconds**.

Turnaround time for P1: 13 – 0 = 13

Turnaround time for P2: 21 – 1 = 20

Turnaround time for P3: 9 – 2 = 7

Turnaround time for P4: 24 – 3 = 21

Average Turnaround Time is : (13+20+7+21)/4 = **15.25 mS**

4. Round Robin Scheduling (RR)

- RR scheduling is **always preemptive.**
- **Time slice / Time quantum to be given in the question**
- **A new process is scheduled from FIFO ready queue.**
- It works in a circular fashion

Advantages:

- Best suitable for time sharing systems

- Each process gets equal opportunity.

Disadvantages:

- Average waiting time is long
- Performance depends on the length of time slice. Careful fixing of time slice is required
- Performance depends on the effect of context switching time

Q1. Consider the following set of processes that arrive at time 0, with the length of the CPU burst given in milliseconds: Assume that time slice is 4 mS

Process	Burst Time
P1	6
P2	8
P3	7
P4	3

- **RR Scheduling**:

  Since all arrive at 0, assume the order P1, P2, P3, P4

P1	P2	P3	P4	P1	P2	P3

0 4 8 12 15 17 21 24

The waiting time for process *P*1 : (0-0)+(15-4) = 11

The waiting time for process *P*2 : (4-0)+(17-8) = 13

The waiting time for process *P*3: (8-0)+(21-12) = 17

The waiting time for process *P*4: 12 – 0 = 12

Thus, the **average waiting time** is (11+ 13+ 17 + 12)/4 = **13.25 milliseconds**.

Turnaround time for P1: 17 – 0 = 17
Turnaround time for P2: 21 – 0 = 21
Turnaround time for P3: 24 – 0 = 24
Turnaround time for P4: 15 – 0 = 15
Average Turnaround Time is : (17+21+24+15)/4 = **<u>19.25 mS</u>**

Q2. Consider the following set of processes with the length of the CPU burst given in milliseconds: Assume that time slice is 4 mS

Process	Arrival Time	Burst Time
P1	0	6
P2	1	8
P3	2	7
P4	3	3

- **RR scheduling**
  (Diagram is same since P1 comes first and starts execution)

P1	P2	P3	P4	P1	P2	P3

0 4 8 12 15 17 21 24

The waiting time for process *P*1 : (0-0)+(15-4) = 11
The waiting time for process *P*2 : (4-1)+(17-8) = 12
The waiting time for process *P*3: (8-2)+(21-12) = 15
The waiting time for process *P*4: 12 – 3 = 9
Thus, the **average waiting time** is (11+ 12+ 15 + 9)/4 = **<u>11.75 milliseconds</u>**.

Turnaround time for P1: 17 – 0 = 17
Turnaround time for P2: 21 – 1 = 20
Turnaround time for P3: 24 – 2 = 22
Turnaround time for P4: 15 – 3 = 12

Average Turnaround Time is : (17+20+22+12)/4 = **17.75 mS**

Note: When P1 completes the first slot, (at time 4), all other processes have arrived in the ready queue in the order P2, P3, P4. So there is no confusion. Please look into next problem.

Q3. Consider the following set of processes with the length of the CPU burst given in milliseconds: Assume that time slice is 4 mS (**University Question KTU APRIL 2018**)

Process	Burst Time	Arrival Time
0	11	0
1	13	5
2	6	9
3	9	13
4	12	17

Answer: RR

Here Arrival time gaps are more. So focus on the order of arrival in ready queue.

At time 0, only one process P0, it can start.

P0

0 4

At time 4, no other process, So P0 can take the next time slice of 4 mS

P0	P0

0 4 8

At time 5, P1 has arrived in the ready queue (Q). At time 8, P0 is removed from CPU and P1 starts execution. At that time P0 is added to the tail of Q (since it is not completed), where no other processes has arrived. So P0 will be in the front next.

P0	P1

0 8 12

At time 9, P2 is added to Q behind P0. At time 12, P1 is removed and added to Q behind P2.

At time 12, Q is: P0, P2, P1. So next term is for P0. But P0 requires only 3 mS and terminates.

P0	P1	P0

0 8 12 15

The scheduling continues like this, based on the ready Q order and finally Gantt chart becomes:

P0	P1	P0	P2	P1	P3	P4

0 8 12 15 19 23 27 31

P2	P1	P3	P4	P1	P3	P4

31 33 37 41 45 46 47 51

Entry to ready Q is shown as below:

Time	Process	Remarks	Full Q
0	P0	Arrival	P0
5	P1	Arrival	P1
8	P0	Removed from CPU	P1,P0
9	P2	Arrival	P0,P2
12	P1	Removed from CPU	P0,P2,P1
13	P3	Arrival	P2,P1,P3
15	-	P0 terminates	P1,P3
17	P4	Arrival	P1,P3,P4
19	P2	Removed from CPU	P1,P3,P4,P2
23	P1	Removed from CPU	P3,P4,P2,P1
27	P3	Removed from CPU	P4,P2,P1,P3
31	P4	Removed from CPU	P2,P1,P3,P4
33	-	P2 terminates	P1,P3,P4
37	P1	Removed from CPU	P3,P4,P1
41	P3	Removed from CPU	P4,P1,P3
45	P4	Removed from CPU	P1,P3,P4
46	-	P1 terminates	P3,P4
47	-	P3 terminates	P4
51	-	P4 terminates	-

(End of Module 2)

*****