ACTIVITY NO. 15

SCHEDULING ALGORITHMS

Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed:**
Section:	**Date Submitted:**
Name:	**Instructor:** Engr. Roman M. Richard

1. Objective(s)

Create C++ to implement CPU Scheduling Algorithms.

2. Intended Learning Outcomes (ILOs)

After this activity, the student should be able to:

- Create C++ code to implement non-preemptive scheduling algorithms.
- Create C++ code to implement preemptive scheduling algorithms.
- Analyze use-cases and compare preemptive and non-preemptive scheduling algorithms.

3. Discussion

CPU Scheduling algorithm is an algorithm which is used to assign system resources to processes in a computing system. Consider the case where you are using two apps namely a game like Fortnite and a desktop application like Evernote. Both with require the use of a graphics processor and but only one can use it at a time. It is the CPU scheduling algorithms which manages which process will use a given resource at a time. The focus of such algorithms is to maximize CPU resources usage and minimize waiting time for each process.

The different CPU algorithms are:

- First Come First Serve
- Shortest Job First
- Shortest Remaining Time First
- Round Robin Scheduling
- Priority Scheduling
- Multilevel Queue Scheduling
- Multilevel Feedback Queue Scheduling

When the CPU is free, and its resources are available, then the CPU must select a process from the ready queue and allocate resources for its execution. Selection is performed with the help of CPU schedulers where the CPU scheduler selects a process from the list of available processes (processes which are present in the ready queue). We need algorithms for this task to ensure the computing device is able to use all its resources to the fullest.

Objectives of CPU Scheduling:

- Maximum CPU utilization
- Fair allocation of CPU
- Maximum throughput (number of processes executing per second)
- Minimum turn around time (time taken to finish execution)
- Minimum waiting time (time for which process waits in ready queue)
- Minimum Response Time (time when process produces first response)

Key terms to understand different algorithms:

- **Arrival Time:** Time at which any process arrives in ready queue.
- **Burst Time:** Time required by CPU for execution of a process. It is also called as Running Time or Execution Time.
- **Completion Time:** Time at which process completes execution.
- **Turn Around Time:** Time difference between Completion Time and Arrival Time (Completion Time - Arrival Time)
- **Waiting Time:** Time difference between Turn Around Time and Burst Time (Turn Around Time - Burst Time)
- **Response Time:** Time after which any process gets CPU after entering the ready queue.

<u>Scheduling Techniques:</u>

- **Preemptive Scheduling:** It is used if there is process switching from running state to ready state or from ready state to waiting state.
  - Resources allocated to a process are for a limited time.
  - Process can be interrupted in between.
  - In case, high priority process arrives, low priority processes may starve.
  - It is flexible in nature.

- **Non-Preemptive Scheduling:** It is used if any process terminates or there is process switching from running to waiting state.
  - Resources allocated to a process are hold until it terminates, or it switches to waiting state.
  - Process can't be interrupted in between.
  - In case, process with high burst time is running, other processes may starve.
  - It is rigid in nature.

4. Materials and Equipment

Personal Computer with C++ IDE
Recommended IDE:

- CLion (must use TIP email to download)
- DevC++ (use the embarcadero fork or configure to C++17)

5. Procedure

<u>**Note:** Include all the output from the different sub-sections of each ILO in section 6 as separate tables. Include your screenshots and analyses.</u>

The different CPU algorithms to be implemented are:

- First Come First Serve
- Shortest Job First
- Shortest Remaining Time First
- Round Robin Scheduling

ILO A: Create C++ code to implement non-preemptive scheduling algorithms.

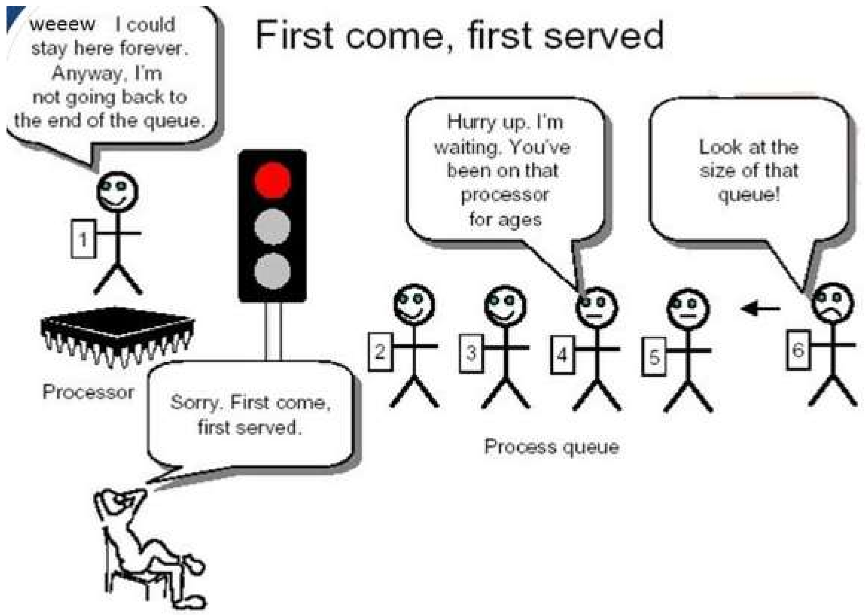
A.1. First Come First Serve

First Come, First Served (FCFS) is a type of scheduling algorithm used by operating systems and networks to efficiently and automatically execute queued tasks, processes and requests by the order of their arrival. An FCFS scheduling algorithm may also be referred to as a first-in, first-out (FIFO) algorithm or a first-come, first-choice (FCFC) algorithm.

Due to its simplistic nature, an FCFS algorithm is predictable, regardless of the type of tasks or requests it has to process. Like a grocery store checkout scheme, FCFS algorithms mimic real-life customer service situations where patrons who arrive first get served first regardless of the size and complexity of their interaction.

First Come, First Served is one of the most efficient and autonomous types of scheduling algorithm because it requires little-to-no human or artificial intelligence (AI) intervention and does not waste time prioritizing tasks and requests by their urgency or level of complexity. Additionally, the party responsible for the scheduling is the CPU itself instead of software or an alternate, more complex job scheduling algorithm.

How a FCFS Scheduling Algorithm Works


weeew I could stay here forever. Anyway, I'm not going back to the end of the queue.
First come, first served
Hurry up. I'm waiting. You've been on that processor for ages
Look at the size of that queue!
1
2
3
4
5
6
Processor
Sorry. First come, first served.
Process queue

Source: ktustudents.in

Here is how an FCFS scheduling algorithm works. To begin, suppose there are three requests to process in the CPU's queue: P1, P2, and P3. Assume P1 is a complex process that requires approximately 25 seconds, P2 a much simpler request that requires only 10 seconds of processing, and P3 a moderately simple request that requires 15 seconds.

When P1 is first put in the queue, the wait time is zero and the CPU starts the processing immediately. P2, on the other hand, would have a wait time of 25 seconds. And P3, having arrived last, would have to wait 35 seconds. As a total, an FCFS scheduling algorithm would need 50 seconds to complete all three requests and empty the queue, which would be the same as other sequential processing, mono-CPU systems.

Because FCFS does not evaluate requests before starting, it has fewer complete tasks per set period of time when compared to an intelligent scheduling algorithm. In this scenario, the FCFS scheduling algorithm would complete a single task in the first half of its run time of 25 seconds. Other algorithms—that start from the simplest of requests, for example—would have finished two requests.

General Algorithm:

```
1 -  Input the processes along with their burst time (bt).
2 -  Find waiting time (wt) for all processes.
3 -  As first process that comes need not to wait so waiting time for process 1
will be 0 i.e. wt[0] = 0.
4 -  Find waiting time for all other processes.
5 -  Find turnaround time = waiting_time + burst_time for all processes.
6 -  Find average waiting time = total_waiting_time / no_of_processes.
7  -  Similarly,  find  average  turnaround  time  =  total_turn_around_time  /
no_of_processes.
```

Example

Process	Burst time	Arrival time
P1	6	2
P2	2	5
P3	8	1
P4	3	0
P5	4	4

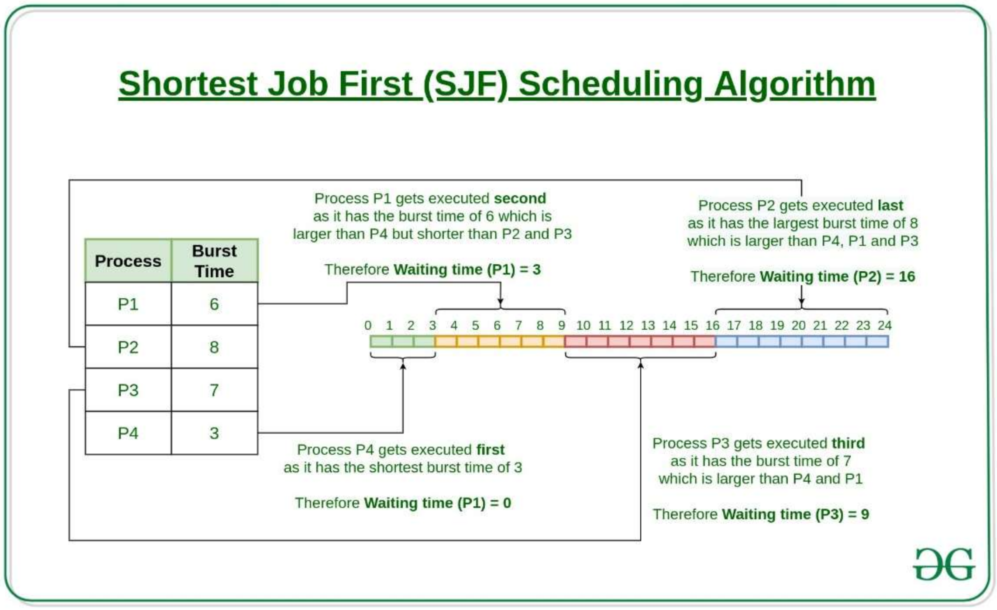
Using the FCFS scheduling algorithm, these processes are handled as follows.

Step 0) The process begins with P4 which has arrival time 0
Step 1) At time=1, P3 arrives. P4 is still executing. Hence, P3 is kept in a queue.
Step 2) At time= 2, P1 arrives which is kept in the queue.
Step 3) At time=3, P4 process completes its execution.
Step 4) At time=4, P3, which is first in the queue, starts execution.
Step 5) At time =5, P2 arrives, and it is kept in a queue.
Step 6) At time 11, P3 completes its execution.
Step 7) At time=11, P1 starts execution. It has a burst time of 6. It completes execution at time interval 17
Step 8) At time=17, P5 starts execution. It has a burst time of 4. It completes execution at time=21
Step 9) At time=21, P2 starts execution. It has a burst time of 2. It completes execution at time interval 23
Step 10) Let’s calculate the average waiting time for above example.

$$AWT = 40/8 \ = \ 5$$

A.2. Shortest Job First

Shortest job first is a scheduling algorithm in which the process with the smallest execution time is selected for execution next. Shortest job first can be either preemptive or non-preemptive. Owing to its simple nature, shortest job first is considered optimal. It also reduces the average waiting time for other processes awaiting execution. Shortest job first is also known as shortest job next (SJN) and shortest process next (SPN).


Shortest Job First (SJF) Scheduling Algorithm
Process
Burst Time
P1
6
P2
8
P3
7
P4
3
Process P1 gets executed second as it has the burst time of 6 which is larger than P4 but shorter than P2 and P3
Therefore Waiting time (P1) = 3
Process P2 gets executed last as it has the largest burst time of 8 which is larger than P4, P1 and P3
Therefore Waiting time (P2) = 16
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Process P4 gets executed first as it has the shortest burst time of 3
Therefore Waiting time (P1) = 0
Process P3 gets executed third as it has the burst time of 7 which is larger than P4 and P1
Therefore Waiting time (P3) = 9

Source: GeeksForGeeks

Shortest job first depends on the average running time of the processes. The accurate estimates of these measures help in the implementation of the shortest job first in an environment, which otherwise makes the same nearly impossible to implement. This is because often the execution burst of processes does not happen beforehand. It can be used in interactive environments where past patterns are available to determine the average time between the waiting time and the commands. Although it is disadvantageous to use the shortest-job-first concept in short-term CPU scheduling, it is considered highly advantageous in long-term CPU scheduling. Moreover, the throughput is high in the case of shortest job first.

General Algorithm:

```
1 - Sort all the processes according to the arrival time.
2 - Then select that process that has minimum arrival time and minimum Burst time.
3 - After completion of the process make a pool of processes that arrives afterward
till the completion of the previous process and select that process among the pool
which is having minimum Burst time.
```

Example:

SJF Process	**Burst Time (in ms)**	**Arrival Time**
P1	2	0
P2	8	0
P3	1	0

P4	4	0

The order in which the CPU processes the process are(Gantt Chart) –

P3	P1	P4	P2	
0	1	3	7	15

Steps –

1. Since Burst time for P3 (Burst = 1 sec) is lowest it is executed first.
2. Then Burst time for P1 is lowest in order thus it gets executed the 2nd time.
3. Similarly P4 and then P2

Computation for Waiting time:

1. P1 waiting time = 1
2. P2 waiting time = 7
3. P3 waiting time = 0
4. P4 waiting time = 3

$$AWT = (1 + 7 + 0 + 3)/4 = 2.75\ time\ units$$

<u>ILO B: Create C++ code to implement preemptive scheduling algorithms.</u>

B.1. Shortest Remaining Time First (SRTF)

This Algorithm is the **preemptive version of SJF scheduling**. In SRTF, the execution of the process can be stopped after certain amount of time. At the arrival of every process, the short-term scheduler schedules the process with the least remaining burst time among the list of available processes and the running process. Once all the processes are available in the ready queue, no preemption will be done and the algorithm will work as SJF scheduling.

Example

In this Example, there are five jobs P1, P2, P3, P4, P5 and P6. Their arrival time and burst time are given below in the table.

Process ID	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting Time	Response Time
1	0	8	20	20	12	0
2	1	4	10	9	5	1
3	2	2	4	2	0	2
4	3	1	5	2	1	4
5	4	3	13	9	6	10
6	5	2	7	2	0	5

Process Gantt Chart:

P1	P2	P3	P3	P4	P6	P2	P5	P1	
0	1	2	3	4	5	7	10	13	20

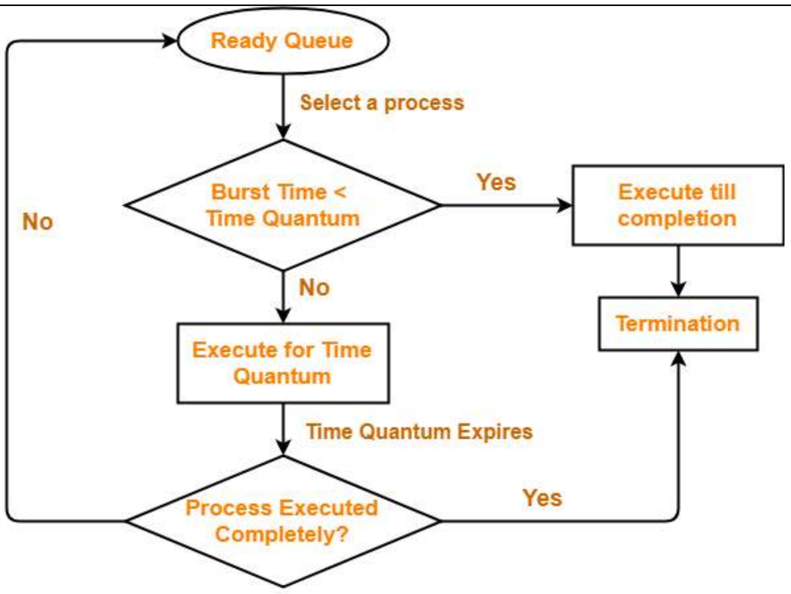
$$AWT = 24/6 = 4\ time\ units$$

The Gantt chart is prepared according to the arrival and burst time given in the table.

1. Since, at time 0, the only available process is P1 with CPU burst time 8. This is the only available process in the list therefore it is scheduled.
2. The next process arrives at time unit 1. Since the algorithm we are using is SRTF which is a preemptive one, the current execution is stopped and the scheduler checks for the process with the least burst time. Till now, there are two processes available in the ready queue. The OS has executed P1 for one unit of time till now; the remaining burst time of P1 is 7 units. The burst time of Process P2 is 4 units. Hence Process P2 is scheduled on the CPU according to the algorithm.
3. The next process P3 arrives at time unit 2. At this time, the execution of process P3 is stopped and the process with the least remaining burst time is searched. Since the process P3 has 2 unit of burst time hence it will be given priority over others.
4. The Next Process P4 arrives at time unit 3. At this arrival, the scheduler will stop the execution of P4 and check which process is having least burst time among the available processes (P1, P2, P3 and P4). P1 and P2 are having the remaining burst time 7 units and 3 units respectively. P3 and P4 are having the remaining burst time 1 unit each. Since, both are equal hence the scheduling will be done according to their arrival time. P3 arrives earlier than P4 and therefore it will be scheduled again.
5. The Next Process P5 arrives at time unit 4. Till this time, the Process P3 has completed its execution and it is no more in the list. The scheduler will compare the remaining burst time of all the available processes. Since the burst time of process P4 is 1 which is least among all hence this will be scheduled.
6. The Next Process P6 arrives at time unit 5, till this time, the Process P4 has completed its execution. We have 4 available processes till now, that are P1 (7), P2 (3), P5 (3) and P6 (2). The Burst time of P6 is the least among all hence P6 is scheduled. Since, now, all the processes are available hence the algorithm will now work same as SJF. P6 will be executed till its completion and then the process with the least remaining time will be scheduled.

Once all the processes arrive, no preemption is done, and the algorithm will work as SJF.

B.2. Round Robin



Source: gatevidyalay.com

A round-robin scheduling algorithm is used to schedule the process fairly for each job a time slot or quantum and the interrupting the job if it is not completed by then the job come after the other job which is arrived in the quantum time that makes these scheduling fairly.

Additionally:

- Round-robin is cyclic in nature, so starvation doesn't occur
- Round-robin is a variant of first come, first served scheduling
- No priority, special importance is given to any process or task
- RR scheduling is also known as Time slicing scheduling

Example:

Process	**Burst Time**
P1	21
P2	3
P3	6
P4	2

Gantt Chart for the RR scheduling:

P1	**P2**	**P3**	**P4**	**P1**	**P3**	**P1**	**P1**	**P1**	
0	5	8	13	15	20	21	26	31	32

$$AWT = 11\ time\ units$$

In the above diagram, arrival time is not mentioned so it is taken as 0 for all processes. The value of time quantum in the above example is 5. Let us now calculate the Turnaround time and waiting time for the above example:

Processes	Burst Time	Turn Around Time	Waiting Time
P1	21	32-0=32	32-21=11
P2	3	8-0=8	8-3=5
P3	6	21-0=21	21-6=15
P4	2	15-0=15	15-2=13

Average waiting time is calculated by adding the waiting time of all processes and then dividing them by no.of processes.

$$average\ waiting\ time = 11 + 5 + 15 + \frac{13}{4} = \frac{44}{4} = \ 11\ time\ units$$

6. Output

7. Supplementary Activity

ILO C: Analyze use-cases and compare preemptive and non-preemptive scheduling algorithms.

Question 1: For the implemented non-preemptive algorithms, do they all run in the same time complexity? Justify your answer by showing both theoretical and empirical analysis of the given algorithms.

____
____
____
____
____
____
____
____

Question 2: For the implemented preemptive algorithms, do they all run in the same time complexity? Justify your answer by showing both theoretical and empirical analysis of the given algorithms.

____
____
____
____

8. Conclusion

Provide the following:

- Summary of lessons learned
- Analysis of the procedure
- Analysis of the supplementary activity
- Concluding statement / Feedback: How well did you think you did in this activity? What are your areas for improvement?

9. Assessment Rubric

References

https://prepinsta.com/operating-systems/shortest-job-first-scheduling-non-preemptive/
https://www.javatpoint.com/os-srtf-scheduling-algorithm
https://www.studytonight.com/operating-system/shortest-remaining-time-first-scheduling-algorithm
https://www.studytonight.com/operating-system/round-robin-scheduling
https://www.gatevidyalay.com/round-robin-round-robin-scheduling-examples/
https://www.geeksforgeeks.org/program-for-shortest-job-first-or-sjf-cpu-scheduling-set-1-non-preemptive/