TRIBHUVAN UNIVERSITY
INSTITUTE OF SCIENCE AND TECHNOLOGY
Central Department of Computer Science and Information Technology
Kirtipur, Kathmandu

Programming Assignment Report on CPU Scheduling Algorithms

Submitted By
Name: Brihat Ratna Bajracharya
Roll No.: 19/075

Submitted To
Mr. Tej Bahadur Shahi
CDCSIT, Kirtipur

Date of Submission: May 2019

CONTENTS

INTRODUCTION

CPU scheduling is a process which allows one process to use the CPU while the execution of another process is on hold (waiting state) due to unavailability of any resource like I/O, thereby making full use of CPU. The aim of CPU scheduling is to make the system efficient, fast and fair. CPU scheduling decisions may take place under the following four circumstances:

1. When a process switches from the running state to the waiting state (for I/O request or invocation of wait for the termination of one of the child processes).
2. When a process switches from the running state to the ready state (for example, when an interrupt occurs).
3. When a process switches from the waiting state to the ready state (for example, completion of I/O).
4. When a process terminates.

In circumstances 1 and 4, there is no choice in terms of scheduling. A new process (if one exists in the ready queue) must be selected for execution. There is a choice, however in circumstances 2 and 3. When Scheduling takes place only under circumstances 1 and 4, we say the scheduling scheme is non-preemptive; otherwise the scheduling scheme is preemptive.

Non-Preemptive Scheduling

Under non-preemptive scheduling, once the CPU has been allocated to a process, the process keeps the CPU until it releases the CPU either by terminating or by switching to the waiting state. This scheduling method is used by the Microsoft Windows 3.1 and by the Apple Macintosh operating systems. It is the only method that can be used on certain hardware platforms, because it does not require the special hardware (for example: a timer) needed for preemptive scheduling.

Preemptive Scheduling

In this type of Scheduling, the tasks are usually assigned with priorities. At times, it is necessary to run a certain task that has a higher priority before another task although it is running.

Therefore, the running task is interrupted for some time and resumed later when the priority task has finished its execution.

CPU Scheduling: Scheduling Criteria

There are many different criteria to check when considering the "best" scheduling algorithm. They are:

1. CPU Utilization

To make out the best use of CPU and not to waste any CPU cycle, CPU would be working most of the time (ideally 100% of the time). Considering a real system, CPU usage should range from 40% (lightly loaded) to 90% (heavily loaded.)

2. Throughput

It is the total number of processes completed per unit time or rather say total amount of work done in a unit of time. This may range from 10/second to 1/hour depending on the specific processes.

3. Turnaround Time

It is the amount of time taken to execute a particular process, i.e. The interval from time of submission of the process to the time of completion of the process.

4. Waiting Time

The sum of the periods spent waiting in the ready queue amount of time a process has been waiting in the ready queue to acquire get control on the CPU.

5. Load Average

It is the average number of processes residing in the ready queue waiting for their turn to get into the CPU.

6. Response Time

Amount of time it takes from when a request was submitted until the first response is produced. It is the time till the first response and not the completion of process execution (final response).

In general CPU utilization and throughput are maximized and other factors are reduced for proper optimization. In this report, we are making the analysis of average waiting time and average turnaround time for different scheduling algorithms.

SCHEDULING ALGORITHMS

To decide which process to execute first and which process to execute last to achieve maximum CPU utilization, computer scientists have defined some algorithms, they are:

1. First Come First Serve (FCFS) Scheduling
2. Shortest Job First (SJF) Scheduling
3. Priority Scheduling
4. Round Robin (RR) Scheduling
5. Multilevel Queue Scheduling
6. Multilevel Feedback Queue Scheduling

In this report, we are discussing about first four scheduling algorithms.

1. First Come First Serve (FCFS) Scheduling

First come first serve (FCFS) scheduling algorithm simply schedules the jobs according to their arrival time. The job which comes first in the ready queue will get the CPU first. The lesser the arrival time of the job, the sooner will the job get the CPU. FCFS scheduling may cause the problem of starvation if the burst time of the first process is the longest among all the jobs.

Implementation of FCFS Scheduling:

1. Input the processes along with their burst time (burst_time).
2. Find waiting time (wait_time) for all processes.
3. First process that comes need not to wait so waiting time for process 1 will be 0 i.e. wait_time[0] = 0.
4. Find waiting time for all other processes i.e. for process i -> wait_time[i] = burst_time[i-1] + wait_time[i-1].
5. Find turn_around_time = wait_time + burst_time for all processes.
6. Find average waiting time = total_waiting_time / no_of_processes(n).
7. Similarly, find average turn-around time = total_turn_around_time / no_of_processes(n).

2. Shortest Job First (SJF) Scheduling

In FCFS Scheduling, we scheduled the processes according to their arrival time. However, Shortest Job First (SJF) or Shortest Job Next (SJN) scheduling algorithm schedules the processes according to their burst time. The process with the lowest burst time, among the list of available processes in the ready queue, is going to be scheduled next. However, it is very difficult to predict the burst time needed for a process hence this algorithm is very difficult to implement in the system.

Implementation:

1. Sort all the processes in increasing order according to burst time.
2. Then simply apply FCFS.

3. SJF (Pre-emptive) Scheduling

In this approach, the process with the smallest amount of time remaining until completion is selected to execute. Since the currently executing process is the one with the shortest amount of time remaining by definition, and since that time should only reduce as execution progresses, processes will always run until they complete or a new process is added that requires a smaller amount of time.

Implementation:

```
1.  Traverse until all process gets completely executed.
    1.1  Find process with minimum remaining time at every single time lap.
    1.2  Reduce its time by 1.
    1.3  Check if its remaining time becomes 0.
    1.4  Increment the counter of process completion.
    1.5  Completion time of current process = current_time +1;
    1.6  Calculate waiting time for each completed process.
         1.6.1  wait_time[i] = service_time - arrival_time - burst_time
    1.7  Increment time lap by one.
2   Find turnaround time (wait_time + burst_time).
```

4. Priority Scheduling

Priority scheduling is a non-preemptive algorithm and one of the most common scheduling algorithms in batch systems. Each process is assigned a priority. Process with the highest priority is to be executed first and so on. Processes with the same priority are executed on first come first served basis. Priority can be decided based on memory requirements, time requirements or any other resource requirement.

Implementation:

1. First input the processes with their burst time and priority.
2. Sort the processes, burst time and priority according to the priority.
3. Now simply apply FCFS algorithm.

5. Round Robin (RR) Scheduling

Round Robin scheduling algorithm is one of the most popular scheduling algorithm which can actually be implemented in most of the operating systems. It is the preemptive version of First Come First Serve Scheduling. The Algorithm focuses on time-sharing. In this algorithm, every process gets executed in a cyclic way. A certain time slice is defined in the system which is called time quantum. Each process present in the ready queue is assigned the CPU for that time quantum, if the execution of the process is completed during that time then the process will terminate else the process will go back to the ready queue and waits for the next turn to complete the execution.

Implementation:

```
1. Create an array remaining_burst_time[] to keep track of remaining burst time of processes.
   This array is initially a copy of burst_time[] (burst times array)
2. Create another array wait_time[] to store waiting times of processes. Initialize this array as
   0.
3. Initialize time: current_time = 0
4. Keep traversing the all processes while all processes are not done. Do following for i'th
   process if it is not done yet.
   4.1. If remaining_burst_time[i] > quantum,
        4.1.1. current_time = current_time + quantum
        4.1.2. remaining_burst_time[i] -= quantum
   4.2. Else // Last cycle for this process
        4.2.1. current_time = current_time + remaining_burst_time[i]
        4.2.2. wait_time[i] = current_time - burst_time[i]
        4.2.3. remaining_burst_time[i] = 0; // This process is over
```

FINDINGS

1. First Come First Serve (FCFS) Scheduling

Input Data 1

Process	Arrival Time	Burst Time
1	0	3
2	2	5
3	1	4
4	4	2

Output 1

Total Waiting Time = 16

Average Waiting Time = 4

Total Turn Around Time = 30

Average Turn Around Time = 7.5

Input Data 2 (for arrival time, t = 0)

Process	Arrival Time	Burst Time
1	0	6
2	0	8
3	0	7
4	0	3

Outputs 2

Total Waiting Time = 41

Average Waiting Time = 10.25

Total Turn Around Time = 65

Average Turn Around Time = 16.25

2. Shortest Job First (SJF) Scheduling

Input Data

(Assumes all process arrives at same time)

Process	Arrival Time	Burst Time
1	0	6
2	0	8
3	0	7
4	0	3

Outputs

Total Waiting Time = 28

Average Waiting Time = 7

Total Turn Around Time = 52

Average Turn Around Time = 13

3. Priority Scheduling

Input Data

Process	Arrival Time	Burst Time	Priority
1	0	3	2
2	2	5	6
3	1	4	3
4	4	2	5

Outputs

Total Waiting Time = 15

Average Waiting Time = 3.75

Total Turn Around Time = 29

Average Turn Around Time = 7.25

4. Round Robin (RR) Scheduling

Input Data

(Assumes all process arrives at same time)

Process	Arrival Time	Burst Time
1	0	6
2	0	8
3	0	7
4	0	3

Output 1

Enter time quantum: 2

Total Waiting Time = 55

Average Waiting Time = 13.75

Total Turn Around Time = 79

Average Turn Around Time = 19.75

Output 2

Enter time quantum: 3

Total Waiting Time = 50

Average Waiting Time = 12.5

Total Turn Around Time = 74

Average Turn Around Time = 18.5

Output 3

Enter time quantum: 4

Total Waiting Time = 53

Average Waiting Time = 13.25

Total Turn Around Time = 77

Average Turn Around Time = 19.25

5. Shortest Job First – Preemptive Scheduling

Input Data

Process	Arrival Time	Burst Time
1	0	3
2	2	5
3	1	4
4	4	2

Outputs

Total Waiting Time = 11

Average Waiting Time = 2.75

Total Turn Around Time = 25

Average Turn Around Time = 6.25

SOURCE CODE

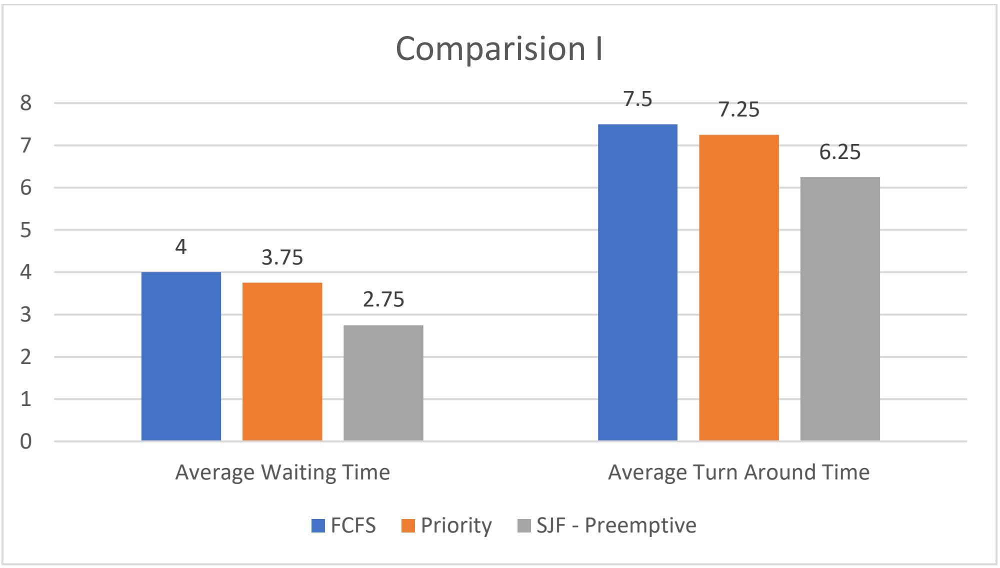
Complete code of implementation of different scheduling algorithms is available in GitHub at https://github.com/Brihat9/AOS/blob/master/AOS_Programming_Assignment_1/aos_programming_assignment_1.cpp

RESULT ANALYSIS

Different arrival time for different process		
Scheduling Algorithms	**Average Waiting Time**	**Average Turn Around Time**
FCFS	4	7.5
Priority	3.75	7.25
SJF - Preemptive	2.75	6.25
With all process arriving at t = 0		
Scheduling Algorithms	**Average Waiting Time**	**Average Turn Around Time**
FCFS	10.25	16.25
SJF	7	13
RR (quanta = 2)	13.75	19.75
RR (quanta = 3,6)	12.5	18.5
RR (quanta = 4)	13.25	19.25
RR (quanta = 5)	14.75	20.75
RR (quanta >= 8)	10.25	16.25

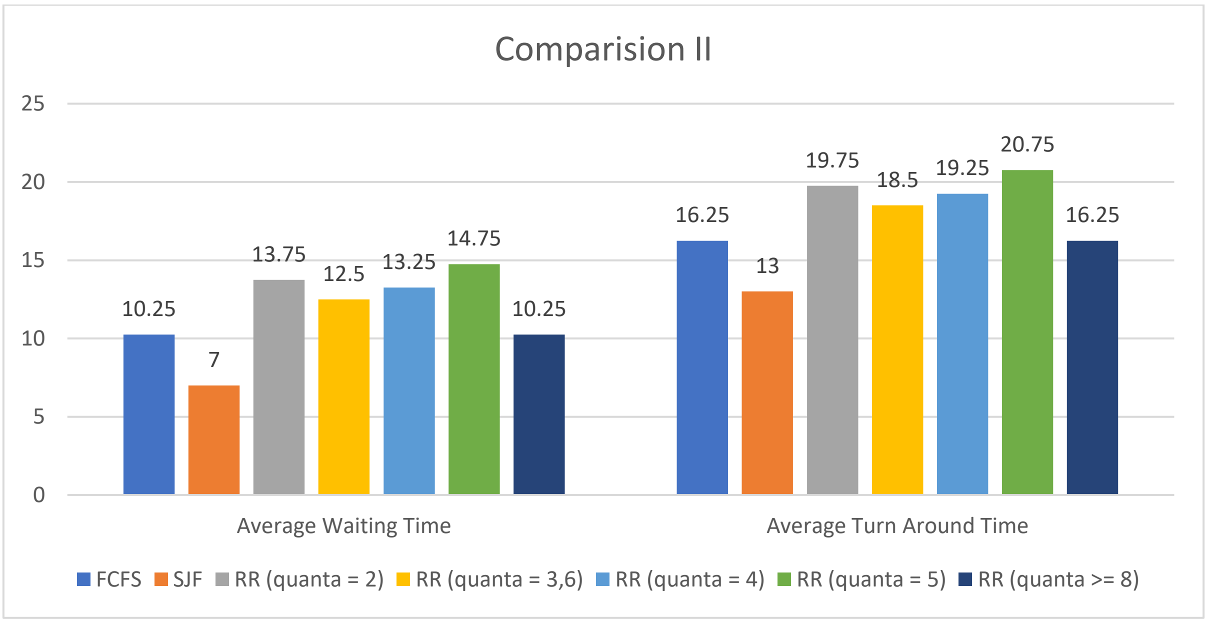
1. Comparison for varying arrival time (FCFS, Priority and SJF – Preemptive)

In this comparison, we have taken different arrival times for different process and computed average waiting time and average turn around time for FCFS, Priority and SJF – Preemptive scheduling algorithms. As seen from above table, SJF – Preemptive the best out of these three scheduling algorithms and FCFS performed worst.



2. Comparison for same arrival time (t=0) (FCFS, SJF, RR and its variants)

In second comparison, we assumed all process arriving at time t = 0 and simulated the scheduling algorithms to calculate average waiting time and average turn around time. We compared FCFS, SJF and RR (with its variants on basis of time quanta) scheduling algorithms and found out that SJF scheduling performed the best and RR with quanta = 5 performed worst. Also among the variants of RR, RR with quanta >= 8 performed the best which is same as FCFS.



CONCLUSION

From the simulation and analysis in this report, we found that Shortest Job First (SJF) or Shortest Job Next (SJN) performed better than other scheduling algorithms considered.

WORKS CITED

1. Chhabra, S. (2019, April 24). *Program for FCFS Scheduling | Set 1.* Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/program-fcfs-scheduling-set-1/

2. Chhabra, S. (2019, April 25). *Program for Priority Scheduling | Set 1.* Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/program-priority-scheduling-set-1/

3. Chhabra, S. (2019, April 26). *Program for Round Robin scheduling | Set 1.* Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/program-round-robin-scheduling-set-1/

4. Chhabra, S. (2019, April 24). *Program for Shortest Job First (or SJF) scheduling | Set 1 (Non- preemptive).* Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/program-shortest-job-first-sjf-scheduling-set-1-non-preemptive/

5. Chhabra, S. (2019, April 26). *Program for Shortest Job First (SJF) scheduling | Set 2 (Preemptive).* Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/program-shortest-job-first-scheduling-set-2srtf-make-changesdoneplease-review/

6. *CPU Scheduling in Operating System.* (2019, April 26). Retrieved from Studytonight: https://www.studytonight.com/operating-system/cpu-scheduling

7. *OS FCFS Scheduling.* (2019, April 26). Retrieved from javatpoint: https://www.javatpoint.com/os-fcfs-scheduling

8. *OS Round Robin Scheduling Algorithm.* (2019, April 26). Retrieved from javatpoint: https://www.javatpoint.com/os-round-robin-scheduling-algorithm