**CS 4310 Operating Systems**

**Project #1 Simulating Job Scheduler and Performance Analysis**

**Due: 4/8**

(Total: 100 points)

**Student Name: Anthony Tran-Phan**

**Date: 4/8/2021**

***Important:   
-*** *Please read this document completely before you start coding.   
- Also, please read the submission instructions (provided at the end of this document) carefully before submitting the project.*

***Project #1 Description:***

Simulating Job Scheduler of the Operating Systems by programming the following four scheduling algorithms that we covered in the class:

1. First-Come-First-Serve (FCFS)
2. Shortest-Job-First (SJF)
3. Round-Robin with Time Slice = 2 (RR-2)
4. Round-Robin with Time Slice = 5 (PR-5)

You can use either Java or your choice of programming language for the implementation. The objective of this project is to help student understand how above four job scheduling algorithms operates by implementing the algorithms, and conducting a performance analysis of them based on the performance measure of their average turnaround times (of all jobs) for each scheduling algorithm using multiple inputs. Output the details of each algorithm’s execution. You need to show which jobs are selected at what times as well as their starting and stopping burst values. You can choose your display format, for examples, you can display the results of each in *Schedule Table* or *Gantt Chart* format (as shown in the class notes). The project will be divided into three parts (phases) to help you to accomplish above tasks in in a systematic and scientific fashion: Design and Testing, Implementation, and Performance Analysis.

The program will read process burst times from a file (job.txt) – this file will be generated by you. Note that you need to generate multiple testing cases (with inputs of 5 jobs, 10 jobs and 15 jobs). A sample input file of five jobs is given as follows (burst time in ms):

[Begin of job.txt]

Job1

7

Job2

18

Job3

10

Job4

4

Job5

12

[End of job.txt]

Note: you can assume that

1. There are no more than 30 jobs in the input file (job.txt).
2. Processes arrive in the order they are read from the file for FCFS, RR-2 and RR-5.
3. All jobs arrive at time 0.
4. FCFS uses the order of the jobs, Job1, Job2, Job3, …

You can implement the algorithms in your choice of data structures based on the program language of your choice. Note that you always try your best to give the most efficient program for each problem. The size of the input will be limited to be within 30 jobs.

***Submission Instructions:***

* ***turn in the following @blackboard.cpp.edu after the completion of all three parts, part 1, part 2 and part 3***
  + - * 1. ***(1) four program files (your choice of programming language with proper documentation)***

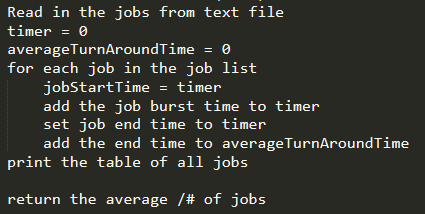
***(2) this document (complete all the answers)***

**Part1**

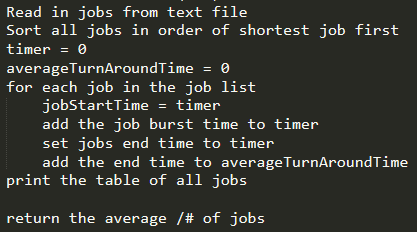
**Design & Testing (30 points)**

* 1. Design the program by providing pseudocode or flowchart for each CPU scheduling algorithm.

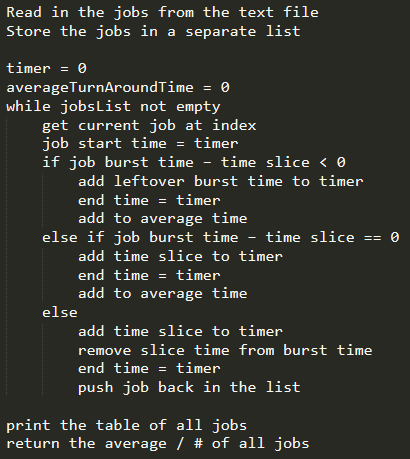
**First-Come First-Serve (FCFS)**



**Shortest-Job-First (SJF)**



**Round-Robin with Time Slice**



* 1. Design the program correctness testing cases. Give at least 3 testing cases to test your program, and give the expected correct **average turnaround time** (for each testing case) in order to test the correctness of each algorithm.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Testing case # | Input  (table of jobs with its job# and length | Expected output for FCFS  (√ if Correct after testing in Part 3) | Expected output for  SJF (√ if Correct after testing in Part 3) | Expected output for  RR-2 (√ if Correct after testing in Part 3) | Expected output for  RR-5 (√ if Correct after testing in Part 3) |
| 1  (5 jobs) | Job1: 7  Job2: 18  Job3: 10  Job4: 4  Job5: 12 | Job1 finishes at 7  Job2 finishes at 25  Job3 finishes at 35  Job4 finishes at 39  Job5 finishes at 51  Average Turn Around Time: 31.4  √ | Job4 finishes at 4  Job1 finishes at 11  Job3 finishes at 21  Job5 finishes at 33  Job2 finishes at 51  Average Turn Around Time: 24  √ | Job4 finishes at 18  Job1 finishes at 29  Job3 finishes at 39  Job5 finishes at 45  Job2 finishes at 51  Average Turn Around Time: 36.4  √ | Job4 finishes at 19  Job1 finishes at 26  Job3 finishes at 36  Job5 finishes at 48  Job2 finishes at 51  Average Turn Around Time: 36  √ |
| 2 (10 jobs) | Job1: 7  Job2: 18  Job3: 10  Job4: 4  Job5: 12  Job6: 3  Job7: 11  Job8: 16  Job9: 1  Job10: 8 | Job1 finishes at 7  Job2 finishes at 25  Job3 finishes at 35  Job4 finishes at 39  Job5 finishes at 51  Job6 finishes at 54  Job7 finishes at 65  Job8 finishes at 81  Job9 finishes at 82  Job10 finishes at 90  Average Turn Around Time: 52.9  √ | Job9 finishes at 1  Job6 finishes at 4  Job4 finishes at 8  Job1 finishes at 15  Job10 finishes at 23  Job3 finishes at 33  Job7 finishes at 44  Job5 finishes at 56  Job8 finishes at 72  Job2 finishes at 90  Average Turn Around Time: 34.6  √ | Job9 finishes at 17  Job4 finishes at 27  Job6 finishes at 30  Job1 finishes at 51  Job10 finishes at 63  Job3 finishes at 67  Job5 finishes at 77  Job7 finishes at 78  Job finishes at 88  Job2 finishes at 90  Average Turn Around Time: 58.8  √ | Job4 finishes at 19  Job6 finishes at 27  Job9 finishes at 38  Job1 finishes at 45  Job3 finishes at 55  Job10 finishes at 73  Job5 finishes at 80  Job7 finishes at 81  Job2 finishes at 89  Job8 finishes at 90  Average Turn Around Time: 59.7  √ |
| 3  (15 jobs) | Job1: 7  Job2: 18  Job3: 10  Job4: 4  Job5: 12  Job6: 3  Job7: 11  Job8: 16  Job9: 1  Job10: 8  Job11: 2  Job12: 21  Job13: 19  Job14: 20  Job15: 13 | Job1 finishes at 7  Job2 finishes at 25  Job3 finishes at 35  Job4 finishes at 39  Job5 finishes at 51  Job6 finishes at 54  Job7 finishes at 65  Job8 finishes at 81  Job9 finishes at 82  Job10 finishes at 90  Job11 finishes at 92  Job12 finishes at 113  Job13 finishes at 132  Job14 finishes at 152  Job15 finishes at 165  Average Turn Around Time: 78.86  √ | Job9 finishes at 1  Job11 finishes at 3  Job6 finishes at 6  Job4 finishes at 10  Job1 finishes at 17  Job10 finishes at 25  Job3 finishes at 35  Job7 finishes at 46  Job5 finishes at 58  Job15 finishes at 71  Job8 finishes at 87  Job2 finishes at 105  Job13 finishes at 124  Job14 finishes at 144  Job12 finishes at 165  Average Turn Around Time: 59.8  √ | Job9 finishes at 17  Job11 finishes at 21 Job4 finishes at 37  Job6 finishes at 40  Job1 finishes at 77  Job10 finishes at 89  Job3 finishes at 101  Job5 finishes at 119  Job7 finishes at 120  Job15 finishes at 141  Job8 finishes at 145  Job2 finishes at 153  Job13 finishes at 161  Job14 finishes at 163  Job12 finishes at 164  Average Turn Around Time: 103.2  √ | Job4 finishes at 19  Job6 finishes at 27  Job9 finishes at 38  Job11 finishes at 45  Job1 finishes at 67  Job3 finishes at 77  Job10 finishes at 95  Job5 finishes at 122  Job7 finishes at 123  Job15 finishes at 146  Job2 finishes at 149  Job8 finishes at 150  Job13 finishes at 159  Job14 finishes at 164  Job12 finishes at 165  Average Turn Around Time: 103.06  √ |

* 1. Design testing strategy for the programs. Discuss about how to generate and structure the randomly generated inputs for experimental study later in Part 3.

*Hint 1: To study the performance evaluation of the four job scheduling algorithms, this project will use three different input sizes, 5 jobs, 10 jobs and 15 jobs. It is the easiest to use a random number generator for generating the inputs. Note that you need to decide the maximum value of job length (use at least 20). However, student should store each data set in various sizes and use the same data set for each job scheduling algorithm.*

*The performance of average Turnaround Time of each input data size (5 jobs, 10 jobs and 15 jobs) can be calculated after an experiment is conducted in 20 trail (with 20 input sets of jobs). We can denote the results as the set X which contains the 20 computed Turnaround Times of 20 trails, where X = {x1, x2, x3 … x20}, from the simulator.  
  
For each data size (5 jobs, 10 jobs and 15 jobs):*

*Average Turnaround Time =*

*The student should decide the maximum value of the job length (at least 20).*

For all three scheduling algorithms, I plan to use an average of 15 trials for determining the average turnaround time. The tests will have job lengths of 5, 10, 15, 20, 25, and 30. The length of each job will have a length of 1 to 30 (inclusive) as stated in the guidelines. The numbers will be generated using Java’s Random class to generate random job length sizes. Each scheduling algorithm will receive the same jobs from a text file to make sure all tests are consistent and fair.

**Part 2**

**Implementation (30 points)**

1. Code each program based on the design (pseudocode or flow chart) in Part 1(a).  
   Done
2. Document the program appropriately.

Done

1. Test you program using the designed testing input data given in the table in Part 1(b), Make sure each program generates the correct answer by marking a “√” if it is correct for each testing case for each program column in the table. Repeat the process of debugging if necessary.

Done

1. For each program, capture a screen shot of the execution (Compile&Run) using the testing case in Part 1(b) to show how this program works properly

Test Case:

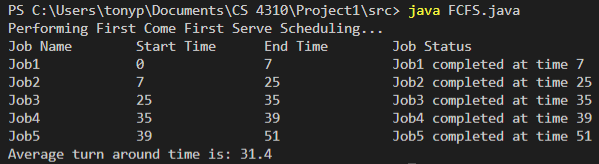
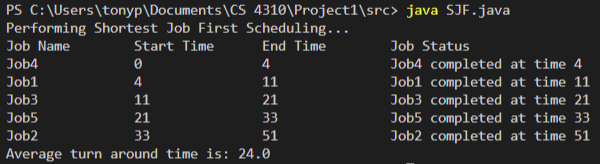
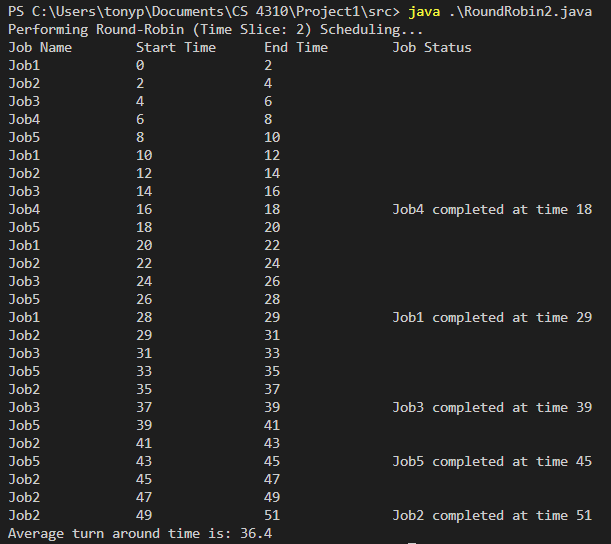
Job1: 7

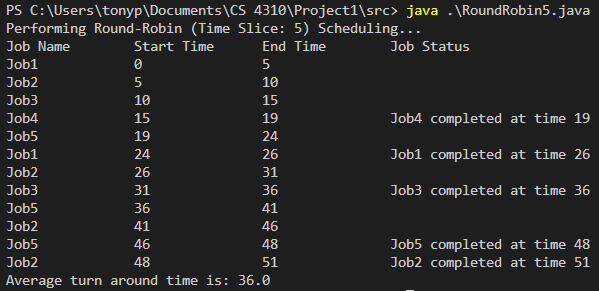
Job2: 18

Job3: 10

Job4: 4

Job5: 12



By now, four working programs are created and ready for experimental study in the next part, Part 3.

**Part 3   
Performance Analysis (40 points)**

1. Run each program with the designed randomly generated input data given in Part 1(c). Generate a table for all the experimental results for performance analysis as follows.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input Size  n jobs | Average of average turnaround times  (FCFS Program) | Average of average turnaround times  (SFJ Program) | Average of average turnaround times  (RR-2) | Average of average turnaround times  (RR-5) |
| 5 jobs | Average Turn Around Time: 40.08  20 Trials | Average Turn Around Time: 38.17  20 Trials | Average Turn Around Time: 57.24  20 Trials | Average Turn Around Time: 60.53  20 Trials |
| 10 jobs | Average Turn Around Time: 86.88  20 Trials | Average Turn Around Time: 60.67  20 Trials | Average Turn Around Time: 111.65  20 Trials | Average Turn Around Time: 106.85  20 Trials |
| 15 jobs | Average Turn Around Time: 125.83  20 Trials | Average Turn Around Time: 89.67  20 Trials | Average Turn Around Time: 167.91  20 Trials | Average Turn Around Time: 149.76  20 Trials |

1. Plot a graph of each algorithm, average turnaround time vs input size (# of jobs), and summarize the performance of each algorithm based on its own graph.

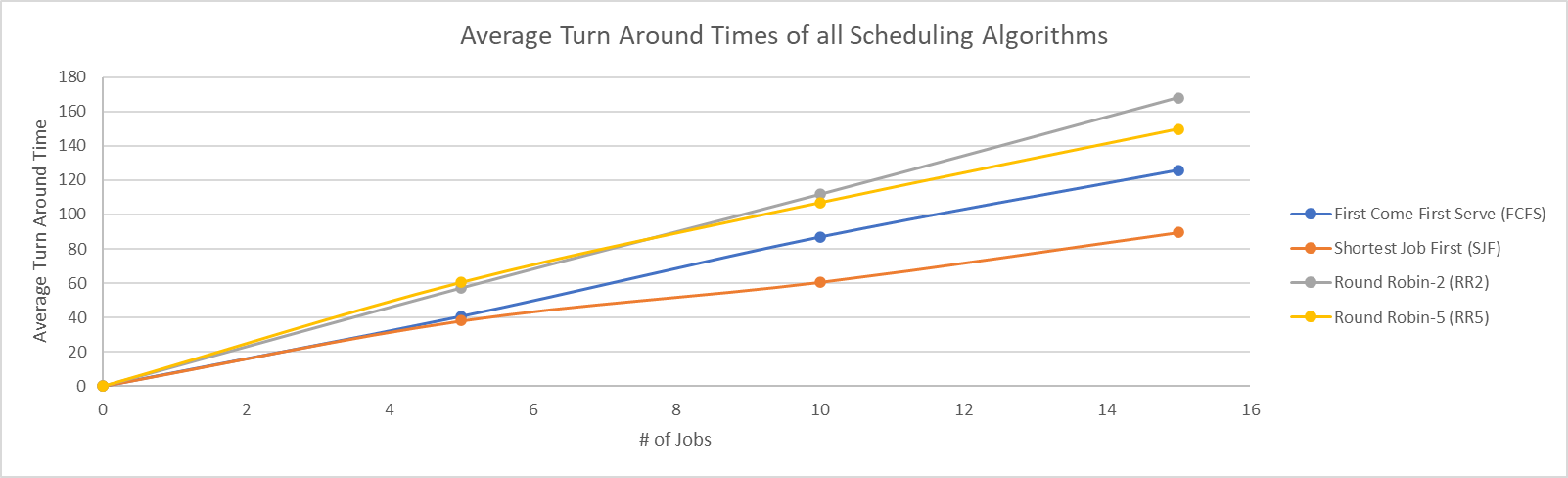
The FCFS scheduling algorithm performed decently well based off the graph. The number of jobs increases, we can see that the average turnaround time gradually increases. The data maxes out at approximately 125 at 15 jobs.

The SJF scheduling algorithm performed very well and had the best average turnaround times out of all scheduling algorithms. The algorithm scaled very well as the number of jobs increase and the turn around time maxes at around 90 at 15 jobs, which is great for an average time.

The Round Robin scheduling algorithm at a time slice of 2 did not scale well as the number of jobs increased. A time slice of 2 is not long enough to make a positive impact on the turn around time because the CPU is making too many swaps between jobs. The turn around time maxes out at approximately 160 at 15 jobs, which is significantly higher than FCFS and SJF algorithms.

The Round Robin scheduling algorithm at a time slice of 5 performed slightly better than Round Robin at time slice of 2, but still performed below average overall.

Plot all four graphs on the same graph and compare the performance of all four algorithms. Rank four scheduling algorithms. Try giving the reasons for the findings.    



As shown in the graph, the ranking for all scheduling algorithms based on best turnaround time is of follows: SJF, FCFS, RR-5, and RR-2. For all job sizes the SJF algorithm had the fastest turnaround times; I think the SJF algorithm performed the best because it could execute all the short jobs first rather than having to wait to eventually process them. This made the execution flow faster overall as the small jobs could be dealt with first and then the longer jobs afterwards. The next algorithm that performed the best was FCFS. This algorithm performed slightly worse than the SJF algorithm, but still good enough to be a reliable option. I think the FCFS algorithm did not perform the best because it has no bias as to which job is processed first. In some situations, large jobs will be processed before smaller ones, so the turn around time is slightly worse compared to the SJF algorithm. Finally, the Round-Robin algorithms at time slices of 2 and 5 both performed worse than SJF and FCFS. A reason for this is because the algorithm must process every job at a specified time slice, which can make each job wait several iterations to finish. Another issue is that switch between jobs incurs overhead and having a time slice that is too long leaves less time for other jobs to be processed.

1. Conclude your report with the strength and constraints of your work. At least 100 words.

(Note: It is reflection of this project. If you have a change to re-do this project again, what you like to keep and what you like to do differently in order get a better quality of results.)

I would say the strengths of my projects were that my algorithm implementations were accurate, and my random number generator generated a random set of numbers to be tested on. These strengths allowed me to get good data and that I was able to perform tests on all algorithms at 5, 10, and 15 jobs at 20 iterations to get plenty of data for the average turn around times. Something I would change if I were to do this project again would be to test the jobs at very large job burst times as well as for hundreds to jobs. Although the data I gathered for this project will suffice, I do think it would further help analyze the performance of each algorithm if the job burst times and amount were scaled to very large numbers. Overall, I am happy with the results, and I think this project was very interesting as it gave me the opportunity to understand how the CPU executes jobs under limited resources and manages bottlenecks.