**CS 4310 Operating Systems**

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

**Due: 10/24**

(Total: 100 points)

**Student Name: Bryce Callender**

**Date: 9/19/19**

***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, 15 jobs, 20 jobs, 25 jobs and 30 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 for SJF.
4. FCFS use 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 (your choice of programming language with proper documentation) files***

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

**Part1**

**Design & Testing (30 points)**

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

FCFS

Read in the jobs from text file

int timer = 0;

double averageTurnAroundTime = 0;

for each job in the job list

jobs start time = timer

add the jobs burst time to timer

set the burst time = 0

set jobs end time to timer

add the end time of the job to averageTurnAroundTime

print the table

return the average / (jobs list size)

SJF

Read in the jobs from the text file

Sort the list

int timer = 0;

double averageTurnAroundTime = 0;

for each job in the job list

jobs start time = timer

add the jobs burst time to timer

set the burst time = 0

set jobs end time to timer

add the end time of the job to averageTurnAroundTime

print the table

return the average / (jobs list size)

RR

Read in the jobs from the text file

Make another list to hold the same jobs as the file

int timer = 0;

double averageTurnAroundTime = 0;

int tasksRemoved = 0;

int index = 0;

while( tasksRemoved != jobs size)

get the current job at index

set its start time = timer

if(burst time of job – roundRobinSlice is less than 0)

add the remaining burst time to timer

set end time = timer

set burst time = 0

add to average time

tasksRemoved++

else if(burst time of job – roundRobinSlice is 0)

add slice time to timer

set burst time to 0

set end time to timer

add to average time

tasksRemoved++

else

add slice time to timer

take away slice amount from burst timer

set end time = timer

push job back onto the list

increment index

print the table

return the average / (jobs list size)

* 1. Design the program correctness testing cases. Give at least 3 testing cases to test your program and give the expected correct output of the program for each case in order to test the correctness of each algorithm.

<complete the following table>

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 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 time:  31.4 | √  Job4 finishes at 4  Job1 finishes at 11  Job3 finishes at 21  Job5 finishes at 33  Job2 finishes at 51  Average time:  24 | √  Job4 finishes at 18  Job1 finishes at 29  Job3 finishes at 39  Job5 finishes at 45  Job2 finishes at 51  Average time:  36.4 | √  Job4 finishes at 19  Job1 finishes at 26  Job3 finishes at 36  Job5 finishes at 48  Job2 finishes at 51  Average time:  36 |
| 2 (10 jobs) | Job1  26  Job2  16  Job3  30  Job4  9  Job5  3  Job6  27  Job7  12  Job8  2  Job9  5  Job10  6 | √  Job1 finishes at 26  Job2 finishes at 42  Job3 finishes at 72  Job4 finishes at 81  Job5 finishes at 84  Job6 finishes at 111  Job7 finishes at 123  Job8 finishes at 125  Job9 finishes at 130  Job10 finishes at 136  Average time:  93 | √  Job8 finishes at 2  Job5 finishes at 5  Job9 finishes at 10  Job`0 finishes at 16  Job4 finishes at 25  Job7 finishes at 37  Job2 finishes at 53  Job1 finishes at 79  Job6 finishes at 106  Job3 finishes at 136  Average time:  46.9 | √  Job8 finishes at 16  Job5 finishes at 29  Job9 finishes at 50  Job10 finishes at 52  Job4 finishes at 71  Job7 finishes at 85  Job2 finishes at 97  Job1 finishes at 127  Job6 finishes at 134  Job3 finishes at 136  Average time:  79.7 | √  Job5 finishes at 23  Job8 finishes at 35  Job9 finishes at 40  Job4 finishes at 64  Job10 finishes at 75  Job7 finishes at 97  Job2 finishes at 103  Job1 finishes at 129  Job3 finishes at 134  Job6 finishes at 136  Average time:  83.6 |
| 3  (15 jobs) | Job1  30  Job2  28  Job3  15  Job4  4  Job5  20  Job6  30  Job7  8  Job8  15  Job9  27  Job10  18  Job11  19  Job12  28  Job13  30  Job14  29  Job15  30 | √  Job1 finishes at 30  Job2 finishes at 58  Job3 finishes at 73  Job4 finishes at 77  Job5 finishes at 97  Job6 finishes at 127  Job7 finishes at 135  Job8 finishes at 150  Job9 finishes at 177  Job10 finishes at 195  Job11 finishes at 214  Job12 finishes at 242  Job13 finishes at 272  Job14 finishes at 301  Job15 finishes at 331  Average time:  165.267 | √  Job4 finishes at 4  Job7 finishes at 12  Job3 finishes at 27  Job8 finishes at 42  Job10 finishes at 60  Job11 finishes at 79  Job5 finishes at 99  Job9 finishes at 126  Job2 finishes at 154  Job12 finishes at 182  Job14 finishes at 211  Job1 finishes at 241  Job6 finishes at 271  Job13 finishes at 301  Job15 finishes at 331  Average time:  142.667 | √  Job4 finishes at 38  Job7 finishes at 100  Job3 finishes at 199  Job8 finishes at 204  Job10 finishes at 230  Job5 finishes at 246  Job11 finishes at 251  Job2 finishes at 311  Job9 finishes at 314  Job12 finishes at 316  Job1 finishes at 324  Job6 finishes at 326  Job13 finishes at 328  Job14 finishes at 329  Job15 finishes at 331  Average time:  256.467 | √  Job4 finishes at 19  Job7 finishes at 102  Job3 finishes at 157  Job8 finishes at 172  Job5 finishes at 222  Job10 finishes at 235  Job11 finishes at 239  Job1 finishes at 304  Job2 finishes at 307  Job6 finishes at 312  Job9 finishes at 314  Job12 finishes at 317  Job13 finishes at 322  Job14 finishes at 326  Job15 finishes at 331  Average time:  245.267 |

* 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 six different input sizes, 5 jobs, 10 jobs, 15 jobs, 20 jobs, 25 jobs and 30 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.*

*Hint 2: The average time of each input data size can be calculated after an experiment is conducted in m (use at least 10) trails. We can denote the run time results (performance of the average turnaround time) as the set X which contains the m number of trails, where X = {x1, x2, x3 … xm} and each xi is one trial of the documented run time.  
  
The Average time =*

*The student should think about and decide how many trials (the value of m) are needed for this experiment, and the maximum of the job length values.*

For testing each algorithm, I will be using the average of 15 trials to determine its final average turnaround time. There will be tests for Job lengths of 5, 10, 15, 20, 25, and 30. For each Job I allow it to have a length between 1 and 30 inclusive. These numbers will be generated by a uniform integer distribution algorithm in C++. This will ensure there is a uniform distribution of numbers across each trial. Each algorithm will be receiving the same input of jobs from a text file to ensure the testing is uniform and fair.

**Part 2**

**Implementation (30 points)**

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

<generate four programs and stored them in four files, needed to be submitted>

<DONE>

1. Document the program appropriately.

<generate documentation inside the four program files>

<DONE>

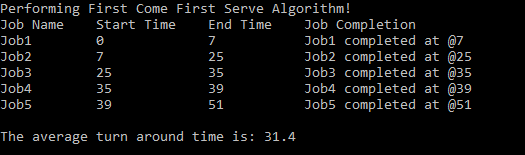
1. Test you program using the designed testing input data given in the table in Part 1(b), Make sure each program generates the correct answer by marking a “√” if it is correct for each testing case for each program column in the table. Repeat the process of debugging if necessary.

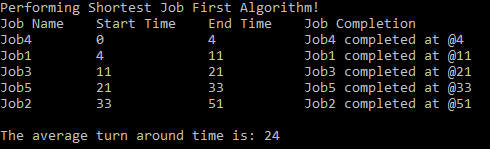
<Complete the four columns of the four algorithms in the table @Part 1(b)>

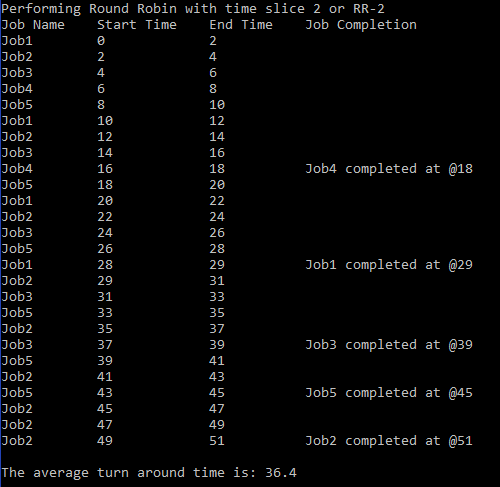
<DONE>

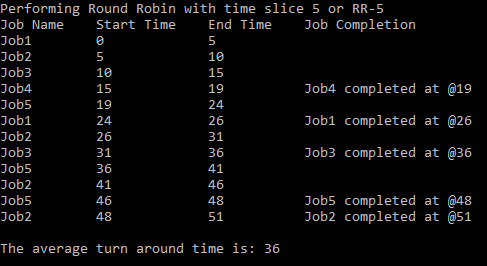
1. For each program, capture a screen shot of the execution (Compile&Run) using one testing case to show how this program works properly

<Insert totally four screen shots, one for each program, here>









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

**Part 3   
Performance Analysis (100 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 | 15 Trails  Average: 42.1333 | 15 Trails  Average: 31.2267 | 15 Trails  Average: 47.5067 | 15 Trails  Average: 46.8933 |
| 10 jobs | 15 Trails  Average: 83.72 | 15 Trails  Average: 61.22 | 15 Trails  Average: 105.2 | 15 Trails  Average: 103.753 |
| 15 jobs | 15 Trails  Average: 113.182 | 15 Trails  Average: 79.9333 | 15 Trails  Average: 141.458 | 15 Trails  Average: 140.116 |
| 20 jobs | 15 Trails  Average: 163.067 | 15 Trails  Average: 116.34 | 15 Trails  Average: 212.577 | 15 Trails  Average: 212.2 |
| 25 jobs | 15 Trails  Average: 194.512 | 15 Trails  Average: 135.347 | 15 Trails  Average: 249.931 | 15 Trails  Average: 247.773 |
| 30 jobs | 15 Trails  Average: 238.458 | 15 Trails  Average: 165.124 | 15 Trails  Average: 307.547 | 15 Trails  Average: 306.907 |

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 first come first serve algorithm turnaround times did decently well looking at this graph. We can see from here that with the gradual increase in the number of jobs the time starts to grow higher and higher based on the previous iteration. Only goes up to around 250 average at the max job count of 30. Its decent but not too good.

The shortest job first algorithm turnaround times did excellent as the number of jobs kept scaling up. This goes up to around 170 which is great for the average time for the max job count of 30.

The Round robin turnaround times for a quantum slice of 2 did alright. A time slice of 2 is not a lot of time so we must constantly context swap, but in this program the swap doesn’t affect performance. The max job count of 30 goes to around 300 average which is a bit high.

The round robin turnaround time for a quantum slice of 5 did only a little bit better than the one with a slice of 2. The max job count of 30 goes to around 300 average which is a bit high.

1. 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 we can see here the shortest job first algorithm, orange in the graph, was the fastest of all the algorithms. The next best was the first come first served algorithm, blue in the graph. The round robin pairs performed almost identically for each iteration of increasing jobs. Round robin 5 was faster though as the iterations got higher, around 20 jobs. The reason why shortest job first was handling performance well was that it could execute all the short jobs first instead of having to wait to eventually do them making the job execution flow faster for the smaller ones, but longer wait for the longer jobs. First come first served shows no bias so it does them as it gets them serving a decent runtime. The round robins must address every job with the quantum it has decided to use which can make each program wait several iterations just to finish. This stacks up the times to what we can see here on the graphs. The final rankings go SJF, FCFS, RR-5, then RR-2 based on the performances.

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.)

This project was interesting as we got to pretend to be the CPU and learn how it handles allocating jobs on itself. The strengths of my project were that my random number generator used a uniform distribution to make the job lengths. This allowed me to get a somewhat better random experience then using the rand function from C++ as it must deal with seeds. I feel like my results for this project are strong in a sense that I was able to perform 15 iterations of the algorithms to find a nice average of each trial set. If I was to do this project again, I would possibly find an even better way of picking the random numbers and using a higher number for the trails so that the distribution is even more secured and that every number has its chance of combinations. This could change the results as the algorithms depend on the lengths of the job and for round robin the quantum length. I would also try redoing my test data from 30 to 20 to get a closer distribution of numbers. If I did it again, I would possibly be able to get better graphs that would show the growth of the algorithm as the job sizes increase to even higher lengths.