**CS 4310 Operating Systems**

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

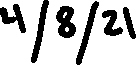
**Due: 4/8**

(Total: 100 points)

**Student Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**



**Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**



***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.

public void FCFS (string[] jobQueue, int[] jobTime)

counter = 0

int [] turnAround

for i in jobQueue

print “start time for “ + jobQueue(i) + “is ” + counter

counter += jobTime(i)

print jobQueue(i) + “ takes ” + jobTime(i)

turnaround(i) = counter

print “end time for “ + jobQueue(i) + “is ” + counter

print “total time is ” + counter

int turnaroundtime = sum(turnAround)/ jobQueue.length

print “average turn around time is ” + turnaroundtime

public void SJF(ArrayList jobs)

sortShortestFirst(jobs)

for i in jobs

int[] turnaround

print “start time for “ + jobs(i).name + “is ” + counter

counter += jobs(i).time

print jobs(i).name + “ takes ” + job(i).time

job(i).turnaround = counter

turnaround(i) = counter

print “end time for “ + job(i).name + “is ” + counter

print “total time is ” + counter

int turnaroundtime = sum(jobs.turnAround)/ jobs.length

print “average turn around time is ” + turnaroundtime

public void RoundRobin (ArrayList jobs)

counter =0

while jobsNotDone

for i in jobs

if job.notDone

print “start time” + counter

jobTime – timeslot

counter + timeslot

print “end time ” + counter

if not job.notDone

job.turnaround time = counter

print “total time is ” + counter

int turnaround = sum(jobs.turnaroundtime)/jobs.length

* 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.

<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 | 31.4 | 24 | 36.4 | 36 |
| 2 (10 jobs) | Job1 = 4  Job 2 = 18  Job 3 = 8  Job 4 = 12  Job 5 = 15  Job 6 = 7  Job 7 = 20  Job 8 = 5  Job 9 = 13  Job 10 = 21 | 61.7 | 50.9 | 85.8 | 81.2 |
| 3  (15 jobs) | Job 1 = 12  Job 2 = 7  Job 3 = 6  Job 4 = 19  Job 5 = 14  Job 6 = 23  Job 7 = 11  Job 8 = 4  Job 9 = 16  Job 10 = 5  Job 11 = 16  Job 12 = 9  Job 13 = 22  Job 14 = 18  Job 15 = 13 | 98.13 | 78.86667 | 140 | 138.467 |

* 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).*

I will design a class that includes all the different scheduling algorithms and use them with a singular randomly generated list of jobs. There will be randomly generated lists for lists of 5 jobs, 10 jobs, and 15 jobs. Each lists’ jobs will have a maximum job length of 20 and minimum of 1. Then once the list is created, it will be tested by each algorithm and save the average turn around time until it is ran 20 times. In those 20 times, a new list will be randomly generated.

**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>

1. Document the program appropriately.

<Generate documentation inside the four program files>

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)>

1. Graphical user interface, text, application

   Description automatically generatedFor 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

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

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 | *29.16* | *23.56* | *34.02* | *31.86* |
| 10 jobs | *56.03* | *42.695* | *68.365* | *62.27* |
| 15 jobs | *86.01* | *62.06* | *103.31* | *95.15* |

1. Chart, line chart, scatter chart

   Description automatically generatedChart, line chart

   Description automatically generatedPlot 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.

Chart, line chart, scatter chart

Description automatically generatedChart, line chart, scatter chart

Description automatically generated

In the first come first serve graph it seems to have a normal upward trend as more jobs are added. The shortest job first graph also seems to have an upward trend as more jobs are added to the algorithm. The round robin algorithms also have an upward trend with more jobs but the round robin algorithm with time slice of 5 seems to have steeper graph.

Chart, line chart

Description automatically generatedPlot 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.    

With all the graphs put into one graph we can see which algorithm is the most effective/ efficient. The best seems to be the shortest job first algorithm. The worst algorithm seems to be round robin algorithm with time slices of 2. Both round robin algorithms were the worst algorithms in the graph. The shortest job first separated itself far from the algorithms as being the best performing algorithm.

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 felt like this project was straight forward and self-explanatory. Coding the first come first serve and shortest job first were the easier algorithms to code. The round robin ones were a bit tougher but was not too difficult. The most tedious part of this project was the documentation. The part of having to hand calculate all the algorithms and creating the graphs definitely took up a lot of the time. The most fun part of this project was the random generated job lists part. I think if I could go back and redo the project, I would not do anything different other than maybe structuring the code for each program better. Have each algorithm structured so that it would be possible to just copy and paste the code into the randomly generated job lists file. I had to do a little adjusting to make the algorithms work with the randomly generated job lists. Also maybe having job list files with a more varied job lengths would have made the graphs look a little better. Overall, the project was a great learning experience.