**CS 4310 Operating Systems**

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

**Due: 10/31**

(Total: 100 points)

**Student Name: Loc Nguyen**

**Date: 10/27/2024**

**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 @Canvas 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.

This is a shared class for all 4 algorithms

#Create Job class with attributes of name, slice time, remaining time, start & end time, and turn around time

#create a function to read in the jobs from the .txt file

FCFS:

For every job needed to be done, calculate the start & end & turn around time and append it to the schedule array. Use turnaround time for each job to calculate average turnaround time.

SJF:

Sort the job based on the time required to complete. While the there’s still job to complete, the start time & end time & turnaround time is calculated and append to the schedule array. Set the remaining time to 0 to indicate the job is completed and +1 to completed jobs.

RR2:

While there’s job to complete, check if a job has any remaining time, and if it does, if it’s the first time the job is running (i.e., start\_time is None), the current time is recorded as the job’s start\_time. If the job can finish within the time\_slice, its remaining\_time is added to time, and the job is marked as complete (remaining\_time = 0). The job's end\_time and turnaround\_time are also updated. If the job cannot finish within the time\_slice, the time\_slice is added to time, and the job's remaining\_time is reduced by time\_slice. Append the job name, start time, end time, and remaining\_time to the schedule array.

RR5:

Same algorithm as RR2 but change the time slice to 5.

* 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 2) | Expected output for  SJF (√ if Correct after testing in Part 2) | Expected output for  RR-2 (√ if Correct after testing in Part 2) | Expected output for  RR-5 (√ if Correct after testing in Part 2) |
| 1  (5 jobs) | Job1  8  Job2  15  Job3  6  Job4  9  Job5  20 | 31.2 (Correct) | 27.8 (Correct) | 41.8 (Correct) | 41.2 (Correct) |
| 2 (10 jobs) | Job1  14  Job2  8  Job3  17  Job4  3  Job5  21  Job6  11  Job7  5  Job8  12  Job9  7  Job10  19 | 63.9 √ | 47.9 √ | 82.6 √ | 81 √ |
| 3  (15 jobs) | Job1  5  Job2  13  Job3  6  Job4  20  Job5  10  Job6  7  Job7  18  Job8  4  Job9  15  Job10  12  Job11  8  Job12  11  Job13  14  Job14  3  Job15  9 | 84.47 √ | 61.8 √ | 110.13 √ | 106.73 √ |

* 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).

Each iteration, generate a new set of jobs and overwrite the file and use a loop to calculate the average time for each iteration. Test for 4 algorithms for turnaround time before moving on to the next iteration.

**Part 2**

**Implementation (30 points)**

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

Done and attached

1. Document the program appropriately.

Documentation are the comments

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 and completed

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

A screenshot of a computer program

Description automatically generated

A computer screen shot of a black screen

Description automatically generated

A screenshot of a computer

Description automatically generated

A screen shot of a computer

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 | 34.33 | 27.58 | 42.87 | 42.77 |
| 10 jobs | 54.875 | 40.38 | 68.19 | 66.835 |
| 15 jobs | 86.77 | 61.56 | 109.05 | 108.41 |

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.

Mostly a linear relationship as jobs increase, the time it takes to complete also increase

Mostly a linear relationship as jobs increase, the time it takes to complete increases but a slower pace

May have a trend of exponential increases due to idle time if jobs take similarly time to complete as number of jobs increase

May have a trend of exponential increases due to idle time if jobs take similarly time to complete as number of jobs increase. Similar completion time versus RR2

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.    

SJF appears to have the lowest turnaround time because short jobs aren’t idle and are completed ASAP, lowering the average turnaround time. On the other hand, RR with time slice perform similarly and have the highest turnaround time because jobs are completed sequentially for a portion or slice at a time which means altogether, longer jobs will cause higher turn around time.

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’m able to express the speed and how efficient each algorithm is at scheduling jobs and completing it. Each algorithm can be utilized in different ways to adapt to different situations and depending on the needs of the programmers, clients, or stakeholders. Sometimes, we might want to get the most amount of jobs done as fast as we can which we can utilize SFJ. Other times, we want to divide up time slots and have jobs incrementally work toward complete at the same rate. Of course, we may also have it done on a first come first serve basis. I would probably increase the length of the job beyond the cap of 20 to get more data at how these algorithm perform at a higher level.