Programming Assignment 1: CPU Scheduling
Due: Feb 26th 2015

Note: Start working on the assignment as soon as possible; do not put this off to the end!

ASSIGNMENT GUIDELINES

In this assignment you will implement and simulate the performance of some of the CPU scheduling algorithms that we have studied in class. You can either implement the following tasks as a single Java program, or you can split them - the design decision is yours.

To simulate each scheduling algorithm, you will have to implement a data structure to hold all the processes in the queue, and their associated job ID and any other relevant parameters associated with that job (e.g., total wait time in the system). Irrespective of the design decisions you make, your program should take as command line input - the accompanying "jobs.txt" file.

Please note following guidelines to construct your program(s):

- All the jobs arrive at time $t = 0$.
- The order of arrival is same as the order of job IDs i.e., jobs with smaller IDs arrive earlier.
- Use the provided "jobs.txt" file for answering the questions that follow. This file has two columns (comma separated), the first column contains the job ID and the second column indicates the CPU burst (in seconds) of the corresponding job.
- For final evaluation of your code, I may use a different input file, which will be in exactly the same format as "jobs.txt", but will have different job parameters (such as different job lengths and different number of jobs).
- For full credit - your code should work on different job input files.
- Partial credit will be given if the reported/computed values are incorrect (based on the corresponding implementation code).

What should you submit?

Submit your assignment as a `.tgz` file on `aurora`. The submission instruction is `submit proj1 yourProj.tgz`. You can name 'yourProj.tgz' a name of your choice. The zipped file should contain:

(a) All your code files, and any other files that might be needed for executing your code.
(b) README.txt
(c) A PDF file (report) containing answers to the questions from Tasks $1-4$.

I. Task 1: First-Come, First Served Scheduling

In the first part you will implement the **First-Come, First Served** (FCFS) Scheduling Policy. This is the simplest scheduling policy. The process that arrives first is allocated the CPU first. As we have discussed in class, this is a **non-preemptive policy**. For the purposes of this assignment, you can assume jobs executing on CPU are allowed to run to completion.

Your implementation of FCFS should report the following for processes in "jobs.txt": (5+5+5)

1.1 Average **turnaround time** for all the jobs.
1.2 Overall **throughput** of the system (number of processes completed per minute.
1.3 Average waiting time for all the jobs.

II. Task 2: Round Robin Scheduling

Next, you need to simulate the **Round-Robin** (RR) Scheduling Policy. RR scheduler keeps the ready queue as a FIFO (First In, First Out) queue of processes. New processes are added to the tail/end of the ready queue. The scheduler picks the first process from the ready queue, sets a timer to interrupt after 1 time quantum, and dispatches the process. If the process has a CPU burst of less than 1 time quantum, the process itself will release the CPU voluntarily. The scheduler will then proceed to the next process in the ready queue. Otherwise, if the CPU burst of the currently running process is longer than 1 time quantum, the process will be preempted by the scheduler and put to the end of the ready queue. The CPU scheduler will then select the next process in the ready queue. This continues until all jobs finish execution.

Assuming time quantum = 1 second, compute the following for RR scheduling, using your simulator, for processes in "jobs.txt": (5+5+5)

2.1 Averate **turnaround time** for all the jobs

2.2 Overall **throughput** of the system (number of processes completed per minute)

2.3 Average **waiting time** for all the jobs.

For the remaining part of this question, we will vary the length of time quantum.

2.4 Vary the time quantum of RR scheduling from 1 to 10 seconds (in steps of 1 second and plot a graph showing how the average **turnaround time** for processes in "jobs.txt" varies with increase in time quantum. The X-axis of this graph is the length of time quantum in seconds, and Y-axis is the average **turnaround time** in seconds. (10)

2.5 Once again vary the time quantum of RR scheduling from 1 to 10 seconds (in steps of 1 second and plot a graph showing how the average **waiting time** for processes in "jobs.txt" varies with increase in time quantum. The X-axis of this graph is the length of time quantum in seconds, and Y-axis is the average **waiting time** in seconds. (10)

III. Task 3: Shortest-Job-First Scheduling

Finally, you will implement and evaluate the **Shortest-Job-First** (SJF) Scheduling: When the CPU is available, SJF scheduling allocates the CPU to the proess that has the smallest CPU burst. If two processes have the same CPU burst, FCFS scheduling is used to break the tie.

Use your implementation of the SJF policy to compute the following for processes in "jobs.txt": (5+5+5)

3.1 Average **turnaround time** for all the jobs

3.2 Overall **throughput** of the system (number of processes completed per minute).

3.3 Average **waiting time** for all the jobs.

IV. Task 4: Comparison

4.1 How does the average waiting time in 1.3, 2.3 and 3.3 compare; which policy gives the shortest wait time? Why? (5+10)

V. Task 5: Documentation

5.1 Prepare a "README.txt" file for your submission. The file should contain the following (10):

- **Names** of all the group members
- **Instructions** for compiling and executing your program(s). Include an example command line for the same.
- **Summary of your design choices**. Keep it short and to the point.
- If your implementation does not work, you should also **document the problems** in the "README.txt" preferably with your explanation of why it does not work and how you would solve it if you had more time.

5.2 You should **comment your code** well. I won't be able to give you credit for something I don't understand! The best way to go about it is to write comments while coding (10).