ICS 143 Spring 2015

Programming Assignment (100 points)

Electronic Copy Due: 6/10/15 by 11:55pm

* Code individually.
* The project needs to be implemented in Java.
* Please read the complete assignment **really, really** carefully, at least twice.
* Start early.
* Submit via EEE Dropbox.

Project Description

The goal of this project is to simulate 3 different scheduler algorithms you learned in class: First-Come First-Serve (FCFS), Shortest Remaining Time First (SRTF), and (for bonus credit) Proportional Share (PS). A simulation environment is provided for you (in Java), so that you only need to fill in the actual job scheduling logic. Each of your algorithms will need to read a file containing the arrival time, burst time, and share (required only for PS) of a set of processes. The goal of your code is then to schedule these jobs according to FCFS, SRTF, or PS by writing to an output file the finish time, waiting time, and turnaround time of all processes.
Please refer to the CPU Scheduling chapter in the course book on the definition of anything in this assignment.

Algorithms

1. First-Come, First-Serve Scheduling (FCFS Scheduler) – 30 points

- FCFS is a non-preemptive scheduling algorithm that schedules processes in the order they arrive. For a detailed description of this algorithm please refer to the course book, page 266.
- There may be multiple processes with the same arrival time. See below on how to break a tie.
- There will not be any entries for process share in the input file.

2. Shortest Remaining Time First Scheduling (aka Preemptive SJF) – 50 points

- SRTF is a preemptive version of the shortest job first (SJF) scheduling algorithm. In this scheduling algorithm, the process with the smallest amount of time remaining until completion is selected to execute. For a detailed description of this algorithm please refer to the course book, starting on page 269 with last paragraph.
- There may be multiple processes with the same arrival time OR with the same burst time, but not both. See below on how to break a tie.
- There will not be any processes with same remaining time at the same CPU time. Example is say if we have a process A arriving at time 0 with a burst time of 5, then we will not have a process B arriving at time 2 with a burst time of 3 since at time=2, both A and B will have the same remaining time.

- There will not be any entries for process share in the input file.

3. Proportional Share Scheduling (PS Scheduler) - 25 extra points

- This will be a **bonus credit** to the assignment.
- PS is a type of **scheduling** which pre-allocates certain amount of CPU time to each of the processes. In a proportional share algorithm every job has a weight, and jobs receive a share of the available resources proportional to the weight of every job. For a detailed description of this algorithm please refer to the course book, page 289.
- Take total shares to be 100.
- A process will not be run unless it is completely given the requested share. Example is say we have process D which requests 30 shares, however system only has 25 shares available. Then, an admission controller would deny D entry into the system. D will be run when there are at least 30 shares available.

Please note the following when implementing your algorithms:

- It is possible for multiple processes to arrive at the same time.
- Break any ties by scheduling the process with the lowest process ID first. For example:
    - In FCFS, PID 12 and PID 13 might arrive at the same time. Schedule PID 12 first.
    - In SRTF, PID 3 and PID 14 might have the same remaining time. Schedule PID 3 first.
- For simplicity, neglect context switch time. That is, if a process arrives at time t, it can be scheduled and run right away at time t.

Each of your algorithms will need to read a file containing the simulation data and need to write an output file, which includes the simulation results. The format of these files is explained next.

Input File Format

Input file is a text file that has all the necessary details of the schedulable processes. Each line will represent a unique process in the following format:
<process-id> <arrival-time> <burst-time> <share>

- <process-id>: Unique, unsigned long, 2^{64}-1 maximum, that represents a process.
- <arrival-time>: Time when a process arrives, an unsigned integer.
- <burst-time>: CPU execution time, an integer number (0, 100]. This is the total time the process would spend running on CPU.
- <share>: An integer number (0, 100]. This will only be present in input files for the PS scheduler.

A single whitespace character separates items in each line.

Output File Format

Output file will record the results of your simulation. Each line will representall necessary scheduler information for a unique process in the following format:
<process-id> <finish-time> <wait-time> <turnaround-time>

- <process-id>: Unique id read from input file. File should be sorted in increasing order of <process-id>s.
- <finish-time>: The time when the process finishes.
- <wait-time>: The total amount of time the process spends waiting in the ready queue.
- <turnaround-time>: The period from when the process enters the system to when the process completes execution.

Lines should be ordered from least to greatest process identifier.

Last line of the output file should contain:
<average-wait-time> <average-turnaround-time>

where you average the wait-time and turnaround-time of all processes. Note that this value should be rounded using Math.round() function in Java.

A single whitespace character should separate items in each line.

Source Code Format

You can find all the source code in https://github.com/ekinoguz/ProcessSchedulers. Please check this repository regularly for updates. The package contains the following files:

- Scheduler.java: We are giving you the interface each of your schedulers need to implement. Specifically, this means that all of the schedulers need to provide at least a schedule() method. You are not allowed to change this interface since doing so will cause our black box testing environment to fail and not give you any credit.
- FCFSScheduler.java: A skeleton class to implement the FCFS scheduling algorithm using the above Scheduler interface. You must implement the schedule() method, but are free to add more methods as you see fit. You must not change the class name.
- SRTFScheduler.java: A skeleton class to implement the SRTF scheduling algorithm using the above Scheduler interface. You must implement the schedule() method, but are free to add more methods as you see fit. You must not change the class name.
- PSScheduler.java: A skeleton class to implement the PS scheduling algorithm using the above Scheduler interface. You must implement the schedule() method, but are free to add more methods as you see fit. You must not change the class name.
- Makefile, which will be used to compile and run your project.
- Tester.java which will be used to test your project.

- tests/ directory which includes public tests and expected outputs.
- grade.sh which is the script we will use for unzipping, compiling, running and grading your project.

Submission

After you are done, you should submit all your source code in a single .zip file via EEE Dropbox. The .zip file should contain the following files:

1. Please name your zip file as <your-uci-student-id-number>.zip, e.g. if your UCI student id number is 4521254, then your submission file should be **4521254.zip**
2. Makefile: please make sure that you edit this file to accommodate changes you made to project. You are responsible to make sure your code compiles and runs.
3. All the source code you wrote, including Scheduler.java, FCFSScheduler.java, SRTFScheduler.java, and PSScheduler.java.
4. An additional file named "code.java" that includes all your (duplicated, copied) source code. We will use this file in our plagiarism detector tool. So simply copy your code from all your source files and paste it into "code.java". Order is not important.
5. Do **not** include Tester.java and grade.sh.
6. Do **not** include tests/ folder and none of your own tests. We will be using our own private tests to grade your project.

Before you submit your code package, make sure the grade.sh script runs without a problem for your zipped submission.

In addition to your source code, you need to write an up to one page report about your project. Report will worth 20 points (plus 5 bonus points for the PS scheduler). Please discuss how you implemented the three algorithms and comment about run time efficiency of your implementation. Please attach your report to your zip file and name your report as <your-uci-student-id-number>.pdf format, e.g. **4521254.pdf.**

Grading

We will be doing Black box testing in a virtual machine. Our grading will be performed with the script grade.sh on a Linux machine. Please note that it is your responsibility to make sure that your code compiles and runs in Linux, otherwise you might unnecessarily complicate the process and lose points.
To be absolutely sure that your code will compile and run the public tests without any problems, follow the below instructions (which is exactly what we will do when grading your code):

1. Download and install VirtualBox 4.3.26 for Linux.
2. Download and install Ubuntu 12.04.5 LTS inside VirtualBox.
    a. If you have resolution problems, follow the instructions here
3. Download and install openjdk-7-jdk
    a. Type "sudo apt-get install openjdk-7-jdk" in a Terminal
    b. Check your javac version by typing "javac -version". You should see "javac 1.7.0_79"

4. Open up a terminal.
5. Go to directory where 4521254.zip is.
6. Type “sh grade.sh 4521254” and press enter.
7. You should see “Grade is: X” where X is some number between 0 and 90.