CS 143A - Principles of Operating Systems Programming Assignment: CPU Scheduling

1. Introduction

In this programming assignment, you will implement a few CPU scheduling algorithms.
Programming language: The assignment will be implemented in Java. As we will minimize the use of obscure Java language features, it should not be too difficult for anyone with experience in a dynamic language like Python or familiar with the principles of Object Oriented Programming (like C++) to get a handle on Java syntax by reading through some of the code in the provided code skeleton.
Cooperation and third party code: This is an **individual** programming assignment, and you should implement the code by your own. You may not share final solutions, and you must write up your own work, expressing it in your own words/notation. Third party codes are not allowed unless with professor's permission.

2. Overview

You will go through several steps to complete this programming assignment and submit your work. Here is a checklist for your reference:
1. Prepare a Linux environment
2. Implement the algorithms
3. Pack up your code for submission following our instruction
4. Test your submission archive with public cases, on ICS OpenLab.
5. Submit the archive file to Canvas dropbox.

3. Environment setup

For ICS OpenLab users, everything is ready for use.

For Mac users, as Mac OSX itself is based on a BSD code base, most of the tools we need should already have been included. The only tool missed is Java 8, so please install Java 8 following the instruction: https://www.java.com/en/download/help/mac_install.xml

For Ubuntu or other Linux distribution users, run the commands below in case any tool is missed.

```
sudo apt-get update
sudo apt-get install -y unzip
sudo apt-get install -y zip
sudo apt-get install -y ruby
```

And then follow https://docs.datastax.com/en/cassandra/3.0/cassandra/install/installOpenJdkDeb.html to install Java 8.

For Windows users,
You may edit the source code in any editor in Windows, and run your code in "Bash on Ubuntu on Windows".

(1) follow the instruction from https://docs.microsoft.com/en-us/windows/wsl/install-win10 to install Windows subsystem for Ubuntu.
(2) Open “Bash on Ubuntu on Windows”, and run the following commands.

```
sudo apt-get update
sudo apt-get install -y unzip
sudo apt-get install -y zip
sudo apt-get install -y ruby
```

And then follow https://docs.datastax.com/en/cassandra/3.0/cassandra/install/installOpenJdkDeb.html to install java 8.

3. Getting familiar with the code base

3.1 Code overview

The Java code base for the CPU scheduling simulator is available on our course website. To start with, RandomSchedulingAlgorithm.java contains an example of implemented scheduling algorithm that will not be used for grading. This algorithm chooses a random job in the queue to execute for each cycle. You may refer to it when you write your own algorithms. If you find an issue and you believe it is a bug in other files, please try your best to find out where it is likely to be and post a bug report on Piazza.

3.2 Compiling

Follow these steps to compile the simulator:
1. Open your terminal. (If you are using Windows and Windows subsystem for Ubuntu, make sure the terminal you open is “Bash on Ubuntu on Windows”)
2. Navigate to the “scheduler” folder. (If you are using Windows, and if the folder is at “e://my_hw/143_a”, you may type “`cd /mnt/e/my_hw/143_a`”)
3. Run `make` (or `make all`) to compile the source code.

If successful, this will create several compiled classes under bin and generate a .jar file named CPUSchedulingSimulator.jar, which contains all the classes and dependencies.

3.3 Running a single case simulation

To run the simulator, change the working directory to “scheduler” and run `./simu.sh` or `bash simu.sh`. Running such a command will print out the following usage information:

```
Usage: java -jar CPUSchedulingSimulator.jar JOBS ALGORITHM
       java -jar CPUSchedulingSimulator.jar JOBS ALGORITHM [PREEMPT]
       java -jar CPUSchedulingSimulator.jar JOBS RR [QUANTUM]

Run the list of jobs using the specified algorithm.
        Supported algorithms are:
        RR Round Robin
        Random Random Job
        Priority Single-Queue Priority (Optionally Preemptive)
```

```
    FCFS First-Come First-Served
    SJF Shortest Job First (Optionally Preemptive)

QUANTUM is an option if ALGORITHM is RR (Round Robin).
The default QUANTUM is 10.
```

To actually run the simulator with a list of jobs and an algorithm, you will need to provide at least two parameters: JOBS - The path to a configuration file that contains a list of jobs, and ALGORITHM - The identifier of a supported algorithm as is listed above.
The optional PREEMPT parameter is applicable to algorithms listed as “optionally preemptive". It can be set to true or false. When set to true, the algorithm should run in the preemptive way. When set to false or not provided, the algorithm should run in the non-preemptive way. The optional QUANTUM parameter is applicable to the Round Robin (RR) scheduling algorithm only. It can be set to a positive integer, e.g. 20. If not provided, the default value 10 will be used. Here are a few examples of single-case simulation commands:

```
./simu.sh samples/sample_1.jobs.txt Random
./simu.sh samples/sample_2.jobs.txt FCFS
./simu.sh samples/sample_4.jobs.txt SJF
./simu.sh samples/sample_4.jobs.txt SJF true
./simu.sh samples/sample_3.jobs.txt Priority
./simu.sh samples/sample_3.jobs.txt Priority true
./simu.sh samples/sample_5.jobs.txt RR
```

The simulator prints out the metrics in a CSV structure like below:

```
Single-Queue Priority,,Non-preemptive
"PID","Burst","Priority","Arrival","Start","Finish","Wait","Response",
"Turnaround"
1001,300,2,0,0,299,0,0,300
1002,200,1,10,400,599,390,390,590
1003,100,0,20,300,399,280,280,380
,,,,,,,,
,,,,,Min,0,0,300
,,,,,Mean,223.33,223.33,423.33
,,,,,Max,390,390,590
,,,,,StdDev,201.08,201.08,149.78
```

You can use this single-case simulation to test your program with sample cases provided in the samples folder to make sure nothing crashes during runtime.

3.4 Understanding Job Lists

Job lists are stored in the *.jobs.txt files. You can find a bunch of such files in the samples and cases-public directories. These files have the following format for each line:

    BURST DELAY PRIORITY

where each of these is an integer text value and is separated by a space, with no leading or trailing spaces. BURST should be positive, while DELAY and PRIORITY should be non-negative. You can assume these conditions always hold for test cases we use for grading. For example, the content of

samples/sample 4.jobs.txt is as follows:

```
300 0 2
200 10 1
100 10 0
```

Note that the DELAY of these jobs are respectively 0, 10, 10, which means the arrival time of these jobs are actually 0, 10, 20. For randomly generated job lists, the BURST of each job is between 1-500, DELAY is between 0-199, and PRIORITY is between 0-9. You are not required to handle invalid input files (overflowing integers, etc.) At the same time, you are welcomed to make your own job lists and share them with the class via Piazza.

4. Algorithm implementation

This section describes the scheduling algorithms that you will implement. We have included the skeleton code for each algorithm in the corresponding .java file, so you only need to fill in the blanks in a few specified functions, adding additional members and auxiliary functions as necessary.

The most important source code file is SchedulingAlgorithm.java. This file is the interface that defines all of the methods that your scheduling algorithm classes must implement. This allows you to write any number of scheduling algorithm classes implementing this interface.

The second most important source code file is BaseSchedulingAlgorithm.java. Extend this class in each of yours (already done in the provided skeleton code classes) in order to inherit some of the methods defined in this class and avoid rewriting them. In particular, note the isJobFinished() function and the activeJob member, which are useful for keeping track of the currently running job and determining whether to preempt it or not.

The third most important source code file is Process.java. Feel free to use any of the public functions in the latter half of it (from getResponseTime() onwards). In particular, note the getBurstTime() function, which returns the remaining CPU burst time of the specified Process. Do not access the members directly from your code.
Only one of the algorithms will be active in each simulation, from the beginning to the end. Therefore, you could just leave transferJobsTo() unimplemented.

4.1 First-Come First Served (FCFSSchedulingAlgorithm.java) (15 points)

A simple FCFS algorithm. If there is a tie, use PID as the tie-breaker (smaller-PID first).

4.2 Shortest Job First (SJFSchedulingAlgorithm.java) (25 points)

This algorithm should run the job with the shortest remaining time first. If there is a tie, use PID as the tie-breaker (smaller-PID first).
It should implement the OptionallyPreemptiveSchedulingAlgorithm interface and preempt the currently running processes only if it is set to be preemptive. The non-preemptive operation is worth 15 points, and the preemptive operation is worth 10 points.

4.3 Single-Queue Priority (PrioritySchedulingAlgorithm.java) (25 points)

This algorithm should run the job with the highest priority (lowest priority number). If there is a tie, use PID as the tie-breaker (smaller-PID first). It should implement the OptionallyPreemptiveSchedulingAlgorithm interface and preempt currently running processes only if it is set to be preemptive. The non-preemptive operation is worth 15 points, and the preemptive operation is worth 10 points.

Hint: The SJF and single-queue priority algorithms can be implemented in almost the same way. How about writing a helper method and let one algorithm extend the other?

4.4 Round Robin (RoundRobinSchedulingAlgorithm.java) (30 points)

We recommend working on this algorithm last, as it is by far the most complicated. You must maintain state between each call to getNextJob(int currentTime), which can be difficult if you are not careful enough. The default time quantum is 20. The actual time quantum we use for grading might be different. Note that the skeleton code includes some methods for setting/getting the current quantum. Don't modify them as they are required by the simulator.

Hint: Round Robin scheduling doesn't put requirements on the underlying implementation. Therefore, with different implementations the outcome can be different. To make sure your Round Robin outputs are consistent with the reference outputs, please maintain a list of unfinished jobs in increasing order of their PIDs. In the case of job completion or preemption, switch to the next job (if does not exist, the first job) in the list. The simplest way to implement it is to always maintain a sorted list, and when a job finishes (isJobFinished returns true), find smallest PID which is bigger than the current one (if no bigger one, return the smallest one in the list) For example, if you have list [101 103 104 105], and active job is 103, and it just finishes, then you may just iterate the list, and find the smallest one which is bigger than 103, then you get 104, which is the next job you should run.

5. Make a correct submission (5 points)

When you are done with implementation of all the required algorithms and confirmed that your simulator compiles properly with the Makefile we provide in the code base, you are ready to make a submission. You may have noticed that the scores of four scheduling algorithms add up to 95 points. The last 5 points are given to your appropriate submission. Follow these steps to pack up your source code for submission:
1. Open your terminal.
2. Navigate to the student folder.
3. Run `./submit.sh` or `bash submit.sh` to compress your source code into a .zip file.
Please note that you are REQUIRED to prepare your assignment submission with the steps above. Failing to do so will add to our workload and lower your final score. If successful, this will create a .zip file named submission.zip, which contains all the .java source code files under src, the Makefile, the simu.sh script, and all dependencies under lib. Please note that these operations WILL NOT actually submit your assignment to EEE Canvas for you. It does nothing but packing up your source code, as well as some dependencies, into a .zip file. You will need to submit the .zip file manually. Before actual submission to EEE Canvas, you should test

your simulator with the public test cases we prepared for you. The submission.zip file is also the one you will use for testing.

6. Test your submission on ICS OpenLab with public cases

If you have no OpenLab account yet, please follow this instruction to activate one:
http://www.ics.uci.edu/%7elab/students/acct_activate.php

Upon successful configuration of your environment, you should be able to test your simulator with our grading tools. Extract “grader.zip”, and you will get the grading tools in “grader” folder. The grading tools come with public test cases and the reference outputs, so you don't need to prepare them by yourself. You may also do the following in your own linux environment, but as we will grade your submission on ICS OpenLab, please test on ICS OpenLab at least once in case there is any environment difference.

Follow these steps to test your submission:
1. Copy your submission.zip to the grader folder.
2. Rename your submission.zip with your student ID number as appears in your EEE account, which should be eight digits or “X" plus seven digits. For example, if your student ID number is 12345678, then the file should be renamed to 12345678.zip.
3. Open your terminal.
4. Navigate to the grader folder.
5. Run `./run.sh 12345678.zip` or `bash run.sh 12345678.zip` to automatically extract, compile, and run your code.
6. Run `./grade.sh s_12345678` to compare your outputs with reference outputs and see how many test cases your code pass. If your code is all correct, the grader should give you 95 points (5 points left for correct submission).

Note that we are going to use a private test set which contains similar but different cases. Therefore, passing all the public test cases does not necessarily mean you will get full credit.

7. Possible issues

7.1 Fixing File Permissions

If you get a “permission denied" error running any of our scripts, please check if you have execute permissions for the .sh BASH scripts. In a UNIX-like environment, to make these scripts executable, you need to navigate to the folder that contains those scripts, and run `chmod +x *.sh`.

7.2 I cannot get access to ICS OpenLab when I am off-campus.

For off-campus access and on-campus wireless/residential access to OpenLab, you must use:

- VPN and SSH, or
- SSH with SSH keys.

For more information, please check https://www.ics.uci.edu/computing/linux/hosts.php