CPU Scheduling Assignment

Overview

One of the main tasks of an operating system is scheduling processes to run on the CPU. In this assignment, you will build a program which schedules simulated CPU processes. Your simulator program will implement three of the CPU scheduling algorithms discussed in this course. The simulator selects a process to run from the ready queue based on the scheduling algorithm chosen at runtime. Since the assignment intends to simulate a CPU scheduler, it does not require any actual process creation or execution. When the CPU scheduler chooses the next process, the simulator will simply print out which process was selected to run at that time. The simulator output is similar to the Gantt chart style.

Design

1. Write a class called **Process** which stores the ID, arrival time, and CPU burst length of a process, all are integers. You can also add data members to keep track of information in order to compute the statistics about the process such as its wait time, response time, and turnaround time. The methods of the Process class are the get and set methods for each data member, the constructor for the class, and any method needed to compute the statistics for that process.

2. Write a class called **Scheduler** to simulate a CPU scheduler for an operating system. The scheduler contains the ready queue and the ready queue is a circular linked list. You'll use the circular linked list you implemented for assignment 1. The only operations the scheduler performs is add and remove. The add operation adds a process into the ready queue into its appropriate spot within the ready queue according to the CPU scheduling algorithm implemented. The remove operation removes a process from the ready queue according to the CPU scheduling algorithm implemented. You'll implement the CPU scheduling algorithms: First Come First Serve, Shortest Remaining Time First which is the preemptive version of Shortest Job First, and Round Robin. Here is a perfect use of inheritance. If needed, you can add an additional method to the circular linked list class to help in the use of the ready queue for a particular CPU scheduling algorithm.

3. Create a driver class and make the name of the driver class **Assignment2** and it should only contain only one method:
   ```
   public static void main(String args[]).
   ```
   The main method receives, via the command line arguments, the name of the CPU scheduler that your simulator program will execute. If Round Robin is the CPU scheduler chosen then the time quantum value is also received via a command line argument. The main method opens the file **assignment2.txt** reading in the entire set of processes and initiates execution of the simulator program. Assume there is only a single processor with only one core. The main method itself should be fairly short.

   The command to launch your program using First Come First Serve scheduling:
   ```
   java Assignment2 FCFS
   ```

   The command to launch your program using Shortest Remaining Time First scheduling:
   ```
   java Assignment2 SRTF
   ```

   The command to launch your program using Round Robin scheduling with a time quantum of 10:
   ```
   java Assignment2 RR 10
   ```

4. The input to your program will be read from a plain text file called **assignment2.txt**. This is the statement you'll use to open the file:

   ```
   FileInputStream fstream = new FileInputStream("assignment2.txt");
   ```

   Assuming you're using Eclipse to create your project, you will store the input file assignment2.txt in the parent directory of your source code (**.java** files) which happens to be the main directory of your project in Eclipse. If you're using some other development environment, you will have to figure out where to store the input file.

Each line in the file represents a process, 3 integers separated by spaces. The process information includes the process ID, arrival time, and CPU burst length. Arrival time is the time at which the scheduler receives the process and places it in the ready queue. You can assume arrival times of the processes in the input file are in non-decreasing order. Process IDs are unique. Arrival times may be duplicated, which means multiple processes may arrive at the same time. The following table is an example of a three process input file. The text in the top row of the table is just to label the value in each column and would not appear in the input file. Remember, the integers on each line are separated by a space or spaces.

Process ID	Arrival Time	CPU Burst Length
1	0	10
2	0	20
3	3	5

5. For the output, the example below best describes what your program should produce. Your program will not be tested on this sample input but a different sample input.

   Here is the example input:

```
1  0 10
2  0  9
3  3  5
4  7  4
5 10  6
6 10  7
```

   Here is the output produced for the above example input given the command `java Assignment2 FCFS` to execute the program:

```
Scheduling algorithm: First Come First Serve
============================================================
<system time    0> process    1 is running
<system time    1> process    1 is running
<system time    2> process    1 is running
<system time    3> process    1 is running
<system time    4> process    1 is running
<system time    5> process    1 is running
<system time    6> process    1 is running
<system time    7> process    1 is running
<system time    8> process    1 is running
<system time    9> process    1 is running
<system time   10> process    1 is finished....
<system time   10> process    2 is running
<system time   11> process    2 is running
<system time   12> process    2 is running
<system time   13> process    2 is running
<system time   14> process    2 is running
<system time   15> process    2 is running
<system time   16> process    2 is running
<system time   17> process    2 is running
```

```
<system time   18> process    2 is running
<system time   19> process    2 is finished....
<system time   19> process    3 is running
<system time   20> process    3 is running
<system time   21> process    3 is running
<system time   22> process    3 is running
<system time   23> process    3 is running
<system time   24> process    3 is finished....
<system time   24> process    4 is running
<system time   25> process    4 is running
<system time   26> process    4 is running
<system time   27> process    4 is running
<system time   28> process    4 is finished....
<system time   28> process    5 is running
<system time   29> process    5 is running
<system time   30> process    5 is running
<system time   31> process    5 is running
<system time   32> process    5 is running
<system time   33> process    5 is running
<system time   34> process    5 is finished....
<system time   34> process    6 is running
<system time   35> process    6 is running
<system time   36> process    6 is running
<system time   37> process    6 is running
<system time   38> process    6 is running
<system time   39> process    6 is running
<system time   40> process    6 is running
<system time   41> process    6 is finished....
<system time   41> All processes finished......
============================================================
Average CPU usage:       100.00%
Average waiting time:     14.17
Average response time:    14.17
Average turnaround time:  21.00
============================================================
```

6. **Tip:** Make your program as modular as possible, not placing all your code in one .java file. You can create as many classes as you need in addition to the classes described above. Methods should be reasonably small following the guidance that "A function should do one thing, and do it well."

7. Do **NOT** use your own packages in your program. If you see the keyword **package** on the top line of any of your .java files then you created a package. Create every .java file in the **src** folder of your Eclipse project.

8. Do **NOT** use any graphical user interface code in your program!

Grading Criteria

The total project is worth 20 points, broken down as follows:

If the program does not compile successfully then the grade for the assignment is zero.

If the program compiles successfully then the grade computes as follows:
Proper submission instructions:

1. Was the file submitted a zip file, 1 point.
2. The zip file has the correct filename, 1 point.
3. The contents of the zip file are in the correct format, 1 point.
4. The keyword **package** should not appear at the top of any of the .java files, 1 point.

Program execution:

5. The program uses the correct input file name, 2 points.
6. The program properly opens and reads from the input file, 1 point.
7. The program properly reads and processes the input, 2 points.
8. The program produces the correct result for the input, 2 points.

Code implementation:

9. The driver file has the correct filename, **Assignment2.java**, 1 point.
10. The driver file contains only the method **main** performing the exact tasks as described in the assignment description, 1 point.
11. The code performs all the tasks as described in the assignment description, 3 points.
12. The code is free from logical errors, 2 points.

Code readability (2 points):

13. Essential guidelines:
    a. Good variable, method, and class names.
    b. Variables, classes, and methods that have a single purpose.
    c. Consistent indentation and formatting style.
    d. Reduction of the nesting level in code.

Late submission penalty: assignments submitted after the due date are subjected to a 2 point deduction for each day late.

Submission instructions

Go to the folder containing the .java files of your assignment and select all (and **ONLY**) the .java files which you created for the assignment in order to place them in a Zip file. The file should **NOT** be a **7z** or **rar** file! Then, follow the directions below for creating a zip file depending on the operating system running on the computer containing your assignment's .java files.

Creating a Zip file in Microsoft Windows (any version):

1. Right-click any of the selected .java files to display a pop-up menu.
2. Click on **Send to**.
3. Click on **Compressed (zipped) Folder**.
4. Rename your Zip file as described below.
5. Follow the directions below to submit your assignment.

Creating a Zip file in Mac OS X:

1. Click **File** on the menu bar.
2. Click on **Compress ? Items** where ? is the number of .java files you selected.
3. Mac OS X creates the file **Archive.zip**.
4. Rename **Archive** as described below.
5. Follow the directions below to submit your assignment.

Save the Zip file with the filename having the following format:

your last name,
followed by an underscore _,
followed by your first name,
followed by an underscore _,
followed by the word **Assignment2**.

For example, if your name is John Doe then the filename would be: **Doe_John_Assignment2**

Once you submit your assignment you will not be able to resubmit it!
Make absolutely sure the assignment you want to submit is the assignment you want graded.
There will be **NO** exceptions to this rule!

You will submit your Zip file via your CUNY Blackboard account.
Follow these instructions:

Log onto your CUNY BlackBoard account.
Click on the CSCI 340 course link in the list of courses you're taking this semester.
Click on **Content** in the green area on the left side of the webpage.
You will see the **Assignment 2 – CPU Scheduling Assignment**.
Click on the assignment.
Upload your Zip file and then click the submit button to submit your assignment.

Due Date: Submit this assignment by Monday, December 11, 2017.