CSCE 311

Programming Assignment #3

Logistics

This is the third Programming assignment for the course. This will be due on the department dropbox (https://dropbox.cse.sc.edu/login/index.php).

Objective

To implement Thread scheduling on the OSP2 simulator using a version of the Shortest Job Remaining strategy with ageing. You will implement the module `ThreadCB.java` to further your understanding of CPU scheduling.

Required to Turn In

Follow all directions regarding hand-in procedures. **Points will be deducted if you do not follow directions precisely.**

1. You must also submit an electronic copy of your SJR solution i.e. ThreadCB.java and TimerInterrupHandler.java (or your best try) via dropbox.
2. You must document your code. Each java file should include your name and email address at the top of the file.
3. You must provide a one-page explanation of how you accomplished the assignment (or what you have currently and why you could not complete). In this one page write-up you should describe your use, creation, and manipulation of data structures to accomplish the assignment. Also submit this via dropbox.
4. Submit a tarball to dropbox. Assuming the folder with your project is "Project3"
    a. Linux
    tar cvf Project3.tar Project3
    gzip Project3.tar
    b. Windows
    http://gettingeek.com/how-to-create-tarball-compress-to-gzip-under-windows-tar-gz-379.html

Building and Executing the Simulation

As in the previous assignment, I suggest that you create a new directory. Copy your files from the previous cpu scheduling project and starting with that version of `ThreadCB.java` modify it to create the `SJR` solution. The only file you should have to modify is `ThreadCB.java`. Modifying the other files will probably "break" OSP2.

The only changes you need to make to your code from the previous project are:

1) Each time you change the status of a thread from ThreadRunning to any other state, you will estimate the cpu burst time using exponential averaging ($\tau_n = \alpha t_{n-1} + (1-\alpha)\tau_{n-1}$). Use an **alpha value of 0.75**, i.e., $\tau_n = 0.75t_{n-1} + 0.25\tau_{n-1}$. If $\tau_n < 5$ then set $\tau_n = 5$. To make this work you will need to keep track of the last cpu burst for the thread. You will also need to keep track of the last estimate of the burst time. Do this by adding additional variables to the ThreadCB class:

a. int lastCpuBurst (this is t_{n-1} in the equation)
b. int estimatedBurstTime (this is τ in the equation)
c. long lastDispatch (you need this to compute lastCpuBurst)

When you first create a thread you should initialize the values of lastCpuBurst and estimatedBurstTime to 10. The reason for initializing these values is that there are no previous values of t_{n-1} and τ_{n-1} that you can use to compute estimatedBurstTime.

2) In order to estimate the burst time, you need the value of lastCpuBurst. Each time you change the status of a thread from ThreadRunnig to some other state you should calculate the value of lastCpuBurst and save it in this variable. You calculate the value using the following equation:

   thread.lastCpuBurst = HClock.get() – thread.lastDispatch;

   As you can see, you will need the value of lastDispatch in order to do this. Be sure to set this variable each time you dispatch a new thread, i.e., thread.lastDispatch = HClock.get() just before returning from do_dispatch.
3) Be sure to save the estimated burst time into the class variable estimatedBurstTime.
4) When your `do_dispatch()` is invoked, after checking whether there was a thread running (and calculating how much time is remaining for that thread by taking the difference of the current time, HClock.get() and the lastDispatch time of that thread), you will check the ready queue to see if there is a thread that has a shorter remaining burst time. If the remaining burst time for the current thread is as short or shorter than the estimated burst time of the threads in the ready queue then continue with the current thread and return SUCCESS. Or if the current burst of the current thread is less than 2 milliseconds, then continue with the current thread. In this case, do not change the value of lastDispatch since you are continuing to run the current thread. Otherwise, preempt the current thread and dispatch the thread in the ready queue with the shortest estimated burst time and return SUCCESS. In this case be sure to update lastDispatch for this new thread. If there is no current thread running and no threads are available in the ready queue then set `PTBR=NULL` and return FAILURE.
5) Since we are not doing round robin, DO NOT SET THE TIMER.
6) Ageing: every time you check the ready queue for a thread to dispatch, decrement the estimated burst time for each of the threads in the ready queue by one. If $\tau < 5$ then set $\tau = 5$. The reason for ageing is that we want to ensure that even threads with large estimated burst time eventually get scheduled.

Some more hints:

- The outlined algorithm for the solution to CPU scheduling is derived from a discussion in your OSP2 book regarding the thread scheduling module ThreadCB. Your OSP2 book (and the Silberschatz book) are your best references for conceptual questions on implementation of ThreadCB.java.
- Be very careful that you do not leave a thread in the ready queue when it is dispatched. If you do, and then re-insert it after a block, the queue may become disconnected, making some ready threads inaccessible. These stranded threads can never be re-scheduled, so threads may starve in the ready queue.

Write up

You must include with your code a one page write up of how you accomplished your assignment (or what you have currently and why you couldn't get it working). In this write-up, you should describe your use, creation and manipulation of data structures used in the assignment as well as your through process as you sent about solving the problem. Include at the bottom a list of any websites or books used (besides the text) while coming to your solution.

Submission

Submit all of your files in a tarball to dropbox. You can create a tarball using 7-zip on Windows (instructions at https://linhost.info/2012/08/gzip-files-in-windows/) or by running the following commands in Linux:

```
tar cvf Project2.tar Project2
gzip Project2.tar
```

This will take all of the files in the Project2 directory and put them in a tar file called Project2.tar. The second command will gzip that tar file. Ensure that you include the following:

1. ThreadsCB.java
2. TimerInterruptHandler.java
3. Your 1-page write-up.

Late submission

You are encouraged to turn in your assignment on time. Late submission will be accepted for 5 days at a penalty of 10% per day. Make sure you have included all of the required files. "Forgotten" files will receive a late penalty.