CSCE 311
Spring 2020
Project # 3
Assigned: February 27, 2020
Due: March 17, 2020 (timestamp must be earlier than lecture time)

Objective: To implement Thread scheduling on the OSP2 simulator using a modified version of the Linux Completely Fair Scheduler strategy. 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. You must submit an electronic copy of your `Linux Completely Fair Scheduler` solution (or your best try) via dropbox. Late assignments will be assessed a 10% penalty per day. You must document your code and provide a one-page explanation of how you accomplished the assignment (or what you have currently and why you could not complete). You should describe your use, creation, and manipulation of data structures to accomplish the assignment.

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 `Linux Completely Fair Scheduler` solution.

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

1) You will need to add three instance variables and three class variables to the ThreadCB class:
   `Private long lastDispatch`; an instance variable that keeps track of when a thread is dispatched.
   `private int lastCpuBurst`; an instance variable that you use to update `vRunTime`.
   `private long vRunTime`; an instance variable that keeps track of the virtual run time of each thread, and
   `private static long numThreads`; a static class variable that keeps track of how many threads there are.
   `private static ThreadCB[] heap;` a static class variable that is used as a heap structure.
   `private static int heapSize;` a static class variable that is used keep track of how many threads are in the heap. Be sure to initialize it to 0.

2) Each time you change the status of a thread from ThreadRunning to any other state, you will update its `vRunTime` value. You will start by determining its current burst time. 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()`. Next you update `vRunTime` using the following expression:

```
thread.vRunTime = ((thread.vRunTime+thread.lastCpuBurst);
```

3) Instead of using a `GenericList readyQueue`, you will use a priority queue implemented as a heap. You will code the heap yourself. This is much simpler than the red black tree that Linux actually uses. While you could create a separate heap class, I suggest that you simply declare the heap and the two methods indicated below. My suggestion is that you look at the implementation of the heap from Mark Weiss' Data Structures and Problem Solving with Java, Addison Wesley for inspiration.
http://www.cs.duke.edu/csed/ap/subset/annotated/ap/HeapPriorityQueue.java

The declaration of the heap is listed in step 1) on the previous page. The actual initialization of the heap is:

```
heap = new ThreadCB[200];
heapSize = 0;
```

The methods that your will have to code are:

```
private static void insert(ThreadCB t)
private static ThreadCB removeTopOfHeap()
```

The method `insert()` inserts the `ThreadCB t` into the appropriate position in the heap. The thread with the smallest `vRunTime` should be at the top of the heap. In other words, your insert method should be comparing the `vRunTime` values of threads in order to place a newly inserted thread into the appropriate position in the heap.

The method `removeTopOfHeap()` removes and returns the thread at the top of the heap and repairs the heap so that of the remaining threads, the thread with the smallest `vRuntTime` is at the top of the heap.
BE SURE TO REMOVE ALL REFERENCES TO THE GENERICLIST READYQUEUE FROM YOUR CODE SO YOU DON'T GET TRIPPED UP.

4) In `do_create()`, the newly created thread will be assigned the default ***vRunTime*** of 0 and be placed in the heap. In the context of the heap, `vRunTime` is the value that is used to sort the heap. Don't forget to update the value of `numThreads`.

5) In `do_kill()`, you will update the value of `numThreads`. You also make sure that the thread you are killing is no longer in the heap.

6) In `do_dispatch()` if the current thread has not used up its quantum, do NOT preempt it (same policy as in previous project).

7) In `do_dispatch()` when the current thread is preempted, you need to update it's values for `lastDispatch` and `vRunTime` and insert it into the heap.

8) In `do_dispatch()`, you should dispatch the thread with the smallest `vRunTime`. If the heap is empty, then should you idle the CPU by setting PTBR to null and returning FAILURE. Otherwise, use the heap method `removeTopOfHeap()` to retrieve the thread with the smallest `vRunTime` value. The quantum for the thread will be calculated by the following expression:

   ```
   1000/numThreads;
   ```

   Use this quantum value to set the timer interrupt.

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.

Turning in Your Assignment via dropbox

When you are satisfied that your CPU scheduling algorithm in ThreadCB works, use dropbox to submit your project. **Use dropbox to submit the following files to Project 3. Do not submit to Project 2, that will overwrite your RR solution from the week before**:

1. `ThreadCB.java` (the `Linux Completely Fair Scheduler` version that you have implemented)
2. One-page explanation of how you accomplished the assignment (or what you have currently and why you could not complete). You should describe your use, creation, and manipulation of data structures to accomplish the assignment.

Remember to turn your assignment in on time. Late assignments will be assessed a 10% penalty per day.