Extra Credit Programming Project

This is an extra credit project (not mandatory). If you successfully implement the program, you will get 7% ON TOP OF YOUR FINAL GRADE.

1. Introduction

When you write a program in which several concurrent threads are competing for resources, you must take precautions to avoid starvation and deadlock. Starvation occurs when one or more threads in your program are blocked indefinitely from gaining access to a resource and, as a result, cannot make progress. Deadlock occurs when two or more threads are waiting on a condition that cannot be satisfied. Deadlock most often occurs when two (or more) threads are waiting for the other(s) resulting in a cyclic waiting that no process can continue.

2. Problem Details

Your program must take 'number of threads' as command line parameter to the program. (We assume that number of threads can be only power of 2 numbers: 2, 4, 8, …) We will assume that there is a top level shared resource which all threads will be competing to access. The mutual exclusion will be achieved using a number of semaphores which will be assumed to be logically arranged in a full binary tree hierarchy. These semaphores will be logically treated as internal nodes of the binary tree and the threads will be treated as leaf nodes of this logical binary tree. For example suppose your application is to start 8 threads, then your program must calculate that it would need 7 semaphores to construct the logical semaphore binary tree. An illustrative diagram is given below for further explanation:

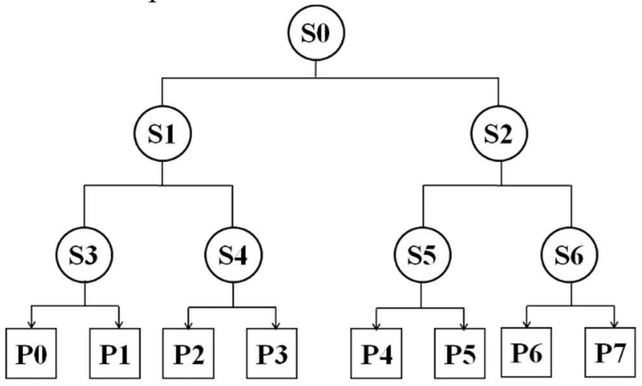

Figure 1. Semaphore Tree

As shown in the Figure 1, process P0 (in our program, it is a thread.) must invoke wait operations (in order to get access to the shared resource) and signal operations with proper semaphores in the order as follows:

```
wait(s3)
wait(s1)
wait(s0)
access the critical section / shared resource
signal(s0)
signal(s1)
signal(s3)
```

Your program must know which semaphores the processes must invoke wait and signal operations and in which order. Remember that the 8 semaphores in the Figure 1 are in global scope (all processes can access all semaphores) and the tree representation is just a logical arrangement of semaphores and processes (threads) to represent mutual exclusion.

3. Additional Details

Once the threads are created, they sleep for a random duration (between 100 - 1000 millisecond) before becoming active. This is to create randomness in the program execution. Once the wait operation on the top semaphore (S0) is successful by a process (thread), it must wait for a random duration (500 - 1500 millisecond) before releasing all locks. Also after releasing all the locks the thread must again sleep for random amount of time before becoming active again to access the shared resource (250 - 500 millisecond). Your code must follow the sample output format below very closely. The order of messages (of course) will not be the same and will vary between different executions (because of the randomness) but the output statement format and style must be preserved. After 20 seconds of running, the program (with all threads) should be terminated gracefully without any error.

Sample Partial Output:

```
Starting thread id: p0
Starting thread id: p1
Starting thread id: p2
Starting thread id: p3
Starting thread id: p4
Starting thread id: p5
Starting thread id: p6
Starting thread id: p7
Thread p7 trying to lock semaphore s6 : success
Thread p0 trying to lock semaphore s3 : success
Thread p6 trying to lock semaphore s6 : failure [waiting]
Thread p7 trying to lock semaphore s2 : success
Thread p2 trying to lock semaphore s4 : success
Thread p7 trying to lock semaphore s0 : success
Thread p7 in critical section now. Will be in critical section for 847 milliseconds ..
Thread p3 trying to lock semaphore s4 : failure [waiting]
Thread p2 trying to lock semaphore s1 : success
Thread p5 trying to lock semaphore s5 : success
Thread p2 trying to lock semaphore s0 : failure [waiting]
Thread p7 releasing lock on semaphore s0
Thread p7 releasing lock on semaphore s2
Thread p7 releasing lock on semaphore s6
....
```

*** Your program MUST MAKE a log file (say, log.txt or report.txt) in the format above. Don't just display it on the screen with standard output.

4. **JAVA Doc. for Semaphore Class**
   You are allowed to use Semaphore Class in Java API. For the reference, please visit at https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/Semaphore.html.

5. **Report**

You are to provide a report named "report.txt" that should include:

   a. Problem definition and proposed solution (how your program is implemented).
   b. What are the results you got (in plain English, discuss the output of your system and **do not copy and paste the program output**)
   c. All the bugs or problems known, any missing items and limitations of your implementation IF ANY. (honesty deserves additional points)
   d. Any additional sections you see necessary

Please note that your reports (your program as well) MUST consist of your own sentences, if you have to copy anything from anywhere you have to quote it and provide reference point. Also a perfect program does not necessarily deserve full points if it is not complemented with a good report. A good report is a brief one that helps its reader understand the system thoroughly from the problem definition through the limitations.

6. **Submission Guidelines**
   a. Only submit source code files. (NO object or project files)
   b. Include your 1~3 page(s) project report.
   c. Please compress your files to ".zip" format.
   d. You must submit the zip file after logging into your blackboard account.

*** If plagiarism is detected (any small part of the code), the entire project portion of your final grade (25% of your entire grade) will be 0.