YSC3248: Parallel, Concurrent and Distributed Programming

Programming Assignment 2

For this assignment, you will be using Peterson's algorithm to create a new kind of lock that will provide n-thread mutual exclusion. The template repository for this assignment is available on Canvas.

Recall that Peterson's algorithm, as we studied it, works only when (at most) two threads are competing for the lock at any time. To solve this, we can arrange individual instances of the Peterson lock (aka Peterson Nodes) into a *tree of locks* called a TreeLock. To acquire a TreeLock, a thread must acquire *all* of the Peterson nodes on the path from the TreeLock's leaf (determined from the thread's `ThreadID`) to its root. Once a thread acquires the root node, it is free to then move on to the critical section to do its work.

Once you clone the template repository, you'll be ready to go! The files you need to modify are:

- `TreeLock.scala`
- `TreeLockTests.scala`

Note that a portion of this assignment's total score is reserved for better testing. Passing basic functionality tests, invoked in `TreeLockTests.scala`, will reward you with some points; however you are also expected to write any additional tests you deem necessary. Failing basic functionality tests does not necessarily imply that you will receive no credit for the assignment. At the same time, I will be unable to conduct a rigorous inspection of non-functional code to award partial credit.

To hand in your code for this assignment, submit the link to the tagged release in your repository on GitHub. Feel free to add additional comments about your implementation and discovery to the `README.md` file in the root of the project.

Here are some tips to get you started on the assignment:

1. You are encouraged to look through the requirements and your code carefully. Murphy's Law applies here: any subtle bugs you might have will eventually pop up, so be very careful about how you write your code. Feel free to use the IntelliJ Debugger (which is really good) to catch them!

2. A thread that acquires the root node in the TreeLock would ideally lock as few nodes as possible in the process. What is the minimum required number of nodes in the TreeLock for n threads? What can we say about shape of the simplest functional tree that these nodes are arranged to?

3. Use the `ThreadID` to get your threads unique ThreadIDs starting at 0. Look at the comments in `ThreadID.scala` for an explanation.

4. When implementing your TreeLock, feel free to modify `PetersonNode.scala` with any additional fields/methods you see fit, or modify the existing method calls with additional arguments. Be careful, however: submissions that modify Peterson's algorithm such that it is no longer essentially Peterson's algorithm will receive no credit.