**CS1760: Multiprocessor Synchronization** **Due: 4 October 2018 12:59 PM**
*Professor: Maurice Herlihy* *HTA: Jonathan Lister*

# <u>Programming Assignment 2 - TreeLock</u>

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 stencil code for this assignment is located at the course's pub directory at `/course/cs1760/pub/treeLock/` and can be installed with

```
cs1760_install treelock
```

Recall that Peterson's algorithm, as described in the book, 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 into a *tree of locks* called a `TreeLock`. To acquire a `TreeLock`, a thread must acquire all of the Peterson locks on the path from the `TreeLock`'s leaf 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 copy the stencil code into your directory, you'll be ready to go! The files you need to modify are:

```
TreeLock.java
TestTreeLock.java
```

Note that a portion of this assignment's total score is reserved for testing. Passing basic functionality tests, invoked in `BasicTestTreeLock.java`, will reward you with some points; however, we also expect you 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, the course staff will be unable to conduct a rigorous inspection of non-functional code to award partial credit.

To hand in your code for this assignment, run

```
cs1760_handin treelock
```

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

1. We don't have any gotchas in this or any subsequent assignment, but we still implore you 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 Eclipse debugger, `jdb`, or any other debugger you want!

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`? What can we say about shape of the simplest functional tree that these

nodes form?

3. In the stencil code given to you, we've provided a `ThreadID.java` file. Use the class defined here to give your threads unique `ThreadID`s starting at 0. Look at the comments in this file for an explanation.

4. When implementing your `TreeLock`, feel free to modify `PetersonLock.java` 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.

5. When debugging your program, you may find Java's built-in `assert` statements to be of use. To run your program with assert statements on, run the following from the command line after you've compiled your program:

```
java -ea TestTreeLock
```

6. If you'd like your assert statements to print values of certain state values such as `ThreadIDs` after they've failed, you can do so by modifying your assert statements into the following form. When the assertion is triggered, the variable information will be printed to standard output during runtime. You can write an assert statement like this:

```
assert (true) : "Value of var1: " + var1 + " Value of var2: " + var2;
```