CS1760: Multiprocessor Synchronization Due 10/18/2022

Project 1: TreeLock + Bakery Algorithm

Professor: Maurice Herlihy *TA: James Scherick*

1 Introduction

For this assignment you will be implementing two types of n-threaded locks: a tree lock and Lamport's Bakery Algorithm.

Lamport's Bakery Algorithm is described in the book, and TreeLock uses Peterson's algorithm to create an n-threaded lock.

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 get around this limitation, 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 its critical section.

2 Setup

The stencil code for this assignment is located here.

The files that need to be modified are TreeLock.java, BakeryAlg.java, and TestDriver.java. However, feel free to modify PetersonLock.java with any additional fields, methods, or parameters for methods. This is not strictly necessary. If you do change this file, it still must remain Peterson's Lock in essence.

3 Assignment Tips

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

- We don't have any "gotchas" in this or any subsequent assignment, but we still implore that 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. Log statements can be useful, but note that `System.out` is not thread safe (and inserting calls to it can make your programs unexpectedly work, due to its internal calls to `synchronize`)!
- 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 the shape of the simplest functional tree that these nodes form?

4 Compiling and Running Your Code

To compile and run your tests, simply run:

```
1 make
```

in your command line.

5 Testing

Note that a portion of this assignment's total score is reserved for testing. Passing basic functionality tests, like the default one in TestDriver.java, will reward you with some points; however, we also expect you to write any additional tests you deem necessary.

To run your own tests, you should create them as functions in TestDriver.java and run them from the `main` method. Each test should create some number of threads and perform operations on a shared object being guarded by either your TreeLock or a BakeryAlg lock. To run these tests, just `make`.

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.

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:

```
1 java -ea TestDriver
```

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:

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

6 Handin

Please submit all files to Gradescope.