

Homework 3

CS176 Fall 2008

Due 6 November

October 26, 2008

1 Programming Assignment

Imagine n threads, each of which executes method `foo()` followed by method `bar()`. Suppose we want to make sure that no thread starts `bar()` until all threads have finished `foo()`.

For this kind of synchronization, we place a *barrier* between `foo()` and `bar()`. You will implement two alternative barriers.

- Use a counter protected by a test-and-test-and-set lock. Each thread locks the counter, increments it, releases the lock, and spins, rereading the counter until it reaches n .
- Use an n -element Boolean array `b`, initially all *false*. When thread 0 arrives, it sets `b[0]` to *true*, and waits until `b[n - 1]` becomes *true*. Every other thread i , for $0 < i < n$, spins until `b[i - 1]` is *true*, sets `b[i]` to *true*, and waits until `b[n - 1]` becomes *true*.

Implement these algorithms. Test them for 16 threads on Plover. Have `foo()` and `bar()` each pause for 10 milliseconds. Run each algorithm at least ten times, measure the elapsed times, discard the highest and lowest values, and report the averages.

Which performs better? Why?

2 Starvation

For each of the following lock implementation, define doorway and waiting sections. Using your definitions, which of these locks are first-come-first served? Which permit starvation?

- Any `testAndSet()` spin lock
- The CLH queue lock, and
- The MCS queue lock.

```

1 public class BadCLHLock implements Lock {
2     // most recent lock holder
3     AtomicReference<Qnode> tail;
4     // thread-local variable
5     ThreadLocal<Qnode> myNode;
6     public void lock() {
7         Qnode qnode = myNode.get();
8         qnode.locked = true;           // I'm not done
9         // Make me the new tail, and find my predecessor
10        Qnode pred = tail.getAndSet(qnode);
11        // spin while predecessor holds lock
12        while (pred.locked) {}
13    }
14    public void unlock() {
15        // reuse my node next time
16        myNode.get().locked = false;
17    }
18    static class Qnode { // Queue node inner class
19        public boolean locked = false;
20    }
21 }

```

Figure 1: An incorrect attempt to implement a CLHLock.

3 Programming Assignment: Testing Lock State

For each of the following lock implementations:

- the testAndSet() spin lock,
- the CLH queue lock, and
- the MCS queue lock,

write a Java implementation of an isLocked() method that tests whether a thread is holding a lock (but does not acquire that lock). Your implementation should compile and run correctly. Include a short but thorough test to convince us that your code is correct.

You can find the source for these locks at:

<http://books.elsevier.com/companions/defaultindividual.asp?isbn=9780123705914>

Find the “Example Programs” section, download and unzip the folder for Chapter 07, and look in the ch07/Spin/src/spin folder.

4 CLH Lock

Fig. 1 shows an alternative implementation of CLHLock in which a thread reuses its own node instead of its predecessor node. Explain how this implementation can go wrong.

5 Graduate Credit Question

- Give an example showing how the universal construction can fail for objects with nondeterministic sequential specifications.
- Propose a way to fix the universal construction to work for objects with nondeterministic sequential specifications.