Spring, 2014 ECEN 5033, CSCI 7000-007

Concurrent Programming

Homework 2

Assigned: 03/18/2014; Due: 04/08/2014, before class.

Homework submissions: by email to the TA (gowtham.ramkumar@colorado.edu).

Problem 1

You learn that your competitor, the Acme Atomic Register Company, has developed a way to use Boolean (single-bit) atomic registers to construct an efficient write-once single-reader single-writer atomic register. Through your spies, you acquire the code fragment shown, which is unfortunately missing the code for `read()`.

Your job is to devise a `read()` method that works for this class, and to justify (informally) why it works. (Remember that the register is write-once, meaning that your read will overlap at most one write.)

```
1 class AcmeRegister implements Register{
2       // N is the total number of threads
3       // Atomic multi-reader single-writer registers
4       private BoolRegister[] b = new BoolMRSWRegister[3 * N];
5       public void write(int x) {
6             boolean[] v = intToBooleanArray(x);
7             // copy v[i] to b[i] in ascending order of i
8             for (int i = 0; i < N; i++)
9                   b[i].write(v[i]);
10                  // copy v[i] to b[N+i] in ascending order of i
11            for (int i = 0; i < N; i++)
12                  b[N+i].write(v[i]);
13                  // copy v[i] to b[2N+i] in ascending order of i
14            for (int i = 0; i < N; i++)
15                  b[(2*N)+i].write(v[i]);
16      }
17      public int read() {
18            // missing code
19      }
20 }
```

Problem 2
We have n threads, each of which executes method `foo()` followed by `bar()`. We want to add synchronization to ensure that no thread starts executing `bar()` until all threads have finished executing `foo()`. To achieve this, we will insert some barrier code between the two methods. Implement the following two schemes for barrier code:

- Use a shared counter protected by test-and-test-and-set lock. Each thread locks the counter, increments it, releases the lock, and repeatedly reads the counter until it reaches `n`.
- Use an n-element Boolean array `A`. Initially all entries are 0. When thread 0 executes its barrier, it sets `b[0]` to 1, and repeatedly reads `b[n - 1]` until it becomes 1. Every other thread i, repeatedly reads `b[i - 1]` until it becomes 1, then it sets `b[i]` to 1, and repeatedly reads `b[n - 1]` until it becomes 1.

Implement both these schemes in Java. Each of the methods `foo()` and `bar()` just sleeps for 20 milliseconds. Test the two schemes for $n = 16$. Run each scheme at least ten times, measure the total runtime in each test run, discard the highest and lowest values, take the average, and use it to compare the two schemes. Which performs better? Can you explain the reason? You can run the experiments on `ecen5033.colorado.edu`. Submit the code as well as experimental results.