Spring, 2017 ECEN 4313, CSCI 4830-010

Concurrent Programming

Homework 4

Assigned: 04/04/2017; Due: 04/13/2017, before class.
Please submit your homework through Moodle.

Problem 1 (30 points)
A trinary register holds values $\perp$, 0, 1, and provides `compareAndSet()` and `get()` methods with the usual meaning. Each such register is initially $\perp$.

1. Give a wait-free protocol that uses one such register to solve n-thread consensus if the inputs of the threads are binary, that is, either 0 or 1.
2. Can you use multiple such registers to solve n-thread consensus even if the inputs are in the range $0 \ldots k-1$, for k greater than 2? Devise a wait-free protocol that uses at most n trinary registers.

Problem 2 (30 points)
A savings account object holds a nonnegative balance, and provides `deposit(k)` and `withdraw(k)` methods. The method `deposit(k)` adds k to the balance, and `withdraw(k)` subtracts k, if the balance is at least k, and otherwise blocks until the balance becomes k or greater. `getbalance()` gives the current balance. Your implementation should be linearizable.

1. Implement this savings account using locks and conditions (use `java.util.concurrent.locks.ReentrantLock`). Test your implementation by using the three functions.
2. Now suppose there are two kinds of withdrawals: ordinary and preferred. Devise an implementation that ensures that no ordinary withdrawal occurs if there is a preferred withdrawal waiting to occur.

Problem 3 (40 points)
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:

1. 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.
2. 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. The method `foo()` prints `foo` and sleeps for 20 milliseconds. Similarly, the method `bar()` prints `bar` and sleeps for 20 milliseconds. Test the two schemes for n threads, for $n = 4, 16, 64$. Run each scheme at least five 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? Submit the code as well as experimental results.