Spring, 2017 ECEN 4313, CSCI 4830-010

Concurrent Programming

Homework 3

Assigned: 03/07/2017; Due: 03/16/2017, before class.

Please submit your solutions through Moodle. Submit well-commented Java file for Problems 1 and 2, and a PDF with solutions to Problems 3 and 4.

Problem 1 (25 points)

Write a concurrent implementation of a doubly-linked list. Use the Lazy synchronization mechanism described in the textbook. Implement the add, remove and contains methods. Consider carefully the number of nodes that need to be locked in these methods, and explain your choice (in a comment in the Java file).

Problem 2 (25 points)

Implement a concurrent hash table in Java. The hash table should be represented as an array of lists. Implement the `add`, `remove`, and `contains` methods. You do not need to implement the resize function for the hash table. Each list should be represented using your favorite linearizable implementation (fine-grained, optimistic, lazy, lock-free).

Argue briefly (in a comment in the Java file) that your hash table implementation is linearizable.

Question 3 (25 points)

We saw safe, regular, and atomic registers. Define a wraparound register as an atomic register that has the property that there is a value v such that adding 1 to v yields 0, not $v + 1$. If we replace the Bakery algorithms shared variables with either (a) safe, (b) regular, (c) or wraparound registers, then does it still satisfy mutual exclusion?

Question 4 (25 points)

Consider the following implementation of a register in a distributed, message-passing system. There are n processors $P_0, \ldots, P_{n-1}$ arranged in a ring, where P_i can send messages only to $P_{i+1 \bmod n}$. Messages are delivered in FIFO order along each link. Each processor keeps a copy of the shared register.

- To read a register, the processor reads the copy in its local memory.
- A processor P_i starts a `write()` call of value v by sending the message "P_i: write v" to $P_{i+1 \bmod n}$.

- If P_i receives a message "P_j : write v" for $i \neq j$, then it writes v to its local copy of the register, and forwards the message to $P_{i+1 \bmod n}$.
- If P_i receives a message "P_i: write v", then it writes v to its local copy of the register, and discards the message. The `write()` call is now complete.

If `write()` calls never overlap,

- Is this register implementation regular?
- Is it atomic?

Give a justification or a counterexample.