

Homework 2

CS176 Fall 2008
Due 16 October

October 8, 2008

1 Peterson's Algorithm and Regular Registers

Does Peterson's two-thread mutual exclusion algorithm work if we replace shared atomic registers with regular registers?

2 Implementing Registers with Messages

Consider the following implementation of a `Register <>` in a distributed, message-passing system. There are n processors P_0, \dots, 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 to register x , by sending the message " P_i : write v to x " to $P_{i+1 \bmod n}$.
- If P_i receives a message " P_j : write v to x ," for $i \neq j$, then it writes v to its local copy of x , and forwards the message to $P_{i+1 \bmod n}$.
- If P_i receives a message " P_i : write v to x ," then it writes v to its local copy of x , and discards the message. The `write()` call is now complete.

Give a short justification or counterexample.

Suppose no `write()` call starts until all activity on behalf of any previous `write()` call by any process has finished.

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

3 Power of Binary Consensus

Show that with sufficiently many n -thread *binary* consensus objects and atomic registers one can implement n -thread consensus over n values.

```

1 class Queue {
2   AtomicInteger head = new AtomicInteger(0);
3   AtomicReference items[] =
4     new AtomicReference[Integer.MAX_VALUE];
5   void enq(Object x){
6     int slot = head.getAndIncrement();
7     items[slot] = x;
8   }
9   Object deq() {
10    while (true) {
11      int limit = head.get();
12      for (int i = 0; i < limit; i++) {
13        Object y = items[i].getAndSet(); // swap
14        if (y != null)
15          return y;
16      }
17    }
18  }
19 }

```

Figure 1: Queue Implementation

4 Multiple CAS

Consider three threads, A , B , and C , each of which has a MRSW register, X_A , X_B , and X_C , that it alone can write and the others can read.

In addition, each pair shares a read-modify-write RMWRegister register that provides only a `compareAndSet()` method: A and B share R_{AB} , B and C share R_{BC} , and A and C share R_{AC} . Only the threads that share a register can call that register's `compareAndSet()` method or read its value.

Your mission: either give a consensus protocol and explain why it works, or sketch an impossibility proof.

5 Programming Assignment Warmup: Locking and Counting

You will create three programs in Java that explore a simple problem: concurrently incrementing a shared counter, with and without synchronization.

5.1 Assignment Detail

Create three programs in Java, each with 16 threads. Have each thread increment a counter in shared memory $2^{24}/16 = 2^{20}$ times. One program should use

normal ints (with no synchronization!). The second program should use Java's `AtomicInteger` class. The last program should use any method of your choosing to lock the critical section of the first program. Print the running time and final value of the counter when the program ends its execution.

In addition to your programs, hand in a README file that discusses the differences between the three methods.

5.2 Handing In

Run `~/mcfeldma/handin.sh <names of files you want to hand in>` on Plover.

6 Graduate Credit Paradox

Fig. 1 shows a FIFO queue implemented with `read`, `write`, `getAndSet()` (that is, `swap`) and `getAndIncrement()` methods. You may assume this queue is linearizable, and wait-free as long as `deq()` is never applied to an empty queue. Consider the following sequence of statements.

- Both `getAndSet()` and `getAndIncrement()` methods have consensus number 2.
- We can add a `peek()` simply by taking a snapshot of the queue (using the methods studied earlier in the course) and returning the item at the head of the queue.
- Using the protocol devised for Exercise 54 (in the book), we can use the resulting queue to solve n -consensus for any n .

We have just constructed an n -thread consensus protocol using only objects with consensus number 2. Identify the faulty step in this chain of reasoning, and explain what went wrong.