CS1760: Multiprocessor Synchronization Due 14 October 2022

Homework 3

Professor: Maurice Herlihy *TA: James Scherick*

Problem 1. In practice, almost all lock acquisitions are uncontended, so the most practical measure of a lock's performance is the number of steps needed for a thread to acquire a lock when no other thread is concurrently trying to acquire the lock.

```
1  class FastPath implements Lock {
2    private static ThreadLocal<Integer> myIndex;
3    private Lock lock;
4    private int x, y = -1;
5    public void lock() {
6      int i = myIndex.get();
7      x = i;                    // I'm here
8      while (y != -1) {}        // is the lock free?
9      y = i;                    // me again?
10     if (x != i)               // Am I still here?
11       lock.lock();            // slow path
12   }
13   public void unlock() {
14     y = -1;
15     lock.unlock();
16   }
17 }
```

Figure 1: Fast path mutual exclusion algorithm.

Scientists at Cantaloupe-Melon University have devised the following "wrapper" for an arbitrary lock, shown in Figure 1. They claim that if the base Lock class provides mutual exclusion and is starvation-free, so does the FastPath lock, but it can be acquired in a constant number of steps in the absence of contention. Sketch an argument why they are right, or give a counterexample.

```
1  class Bouncer {
2    public static final int DOWN = 0;
3    public static final int RIGHT = 1;
4    public static final int STOP = 2;
5    private boolean goRight = false;
6    private ThreadLocal<Integer> myIndex; // initialize myIndex
7    private int last = −1;
8    int visit () {
9      int i = myIndex.get();
10     last = i;
11     if (goRight)
12       return RIGHT;
13     goRight = true;
14     if (last == i)
15       return STOP;
16     else
17       return DOWN;
18   }
19 }
```

Figure 2: The Bouncer class implementation.

Problem 2. Suppose n threads call the visit () method of the Bouncer class shown in Figure 2. Prove the following:

- At most one thread gets the value STOP.
- At most $n-1$ threads get the value DOWN.
- At most $n-1$ threads get the value RIGHT.

Note that the last two proofs are *not* symmetric.

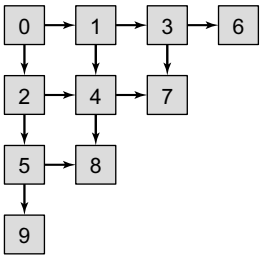

Figure 3: Array layout for Bouncer objects.

Problem 3. So far, we have assumed that all n threads have unique, small indexes. Here is one way to assign unique small indexes to threads. Arrange Bouncer objects in a triangular matrix, where each Bouncer is given an id as shown in Figure 3. Each thread starts by visiting Bouncer zero. If it gets STOP, it stops. If it gets RIGHT, it visits 1, and if it gets DOWN, it visits 2. In general, if a thread gets STOP, it stops. If it gets RIGHT, it visits the next Bouncer on that row, and if it gets DOWN, it visits the next Bouncer in that column. Each thread takes the id of the Bouncer object where it stops.

- Prove that each thread eventually stops at some Bouncer object.
- How many Bouncer objects do you need in the array if you know in advance the total number n of threads?

```
1  public class AtomicSRSWRegister implements Register<int> {
2    private static int RANGE = M;
3    boolean[] r_bit = new boolean[RANGE]; // atomic boolean SRSW
4    public AtomicSRSWRegister(int capacity) {
5      for (int i = 1; i <= RANGE; i++)
6        r_bit [i] = false;
7      r_bit [0] = true;
8    }
9    public void write(int x) {
10     r_bit [x] = true;
11     for (int i = x − 1; i >= 0; i−−)
12       r_bit [i] = false;
13   }
14   public int read() {
15     for (int i = 0; i <= RANGE; i++)
16       if (r_bit [i]) {
17         return i;
18       }
19     return −1; // impossible
20   }
21 }
```

Figure 4: Boolean to M-valued SRSW Atomic Register Algorithm

Problem 4. Imagine running a 64-bit system on a 32-bit system, where we simulate a single 64-bit memory location (register) using two atomic 32-bit memory locations (registers). A write operation is implemented by simply writing the first 32-bits in the first register, then the second 32-bits in the second register. A read, similarly, reads the first half from the first register, then the second half from the second register, and returns the concatenation. What is the strongest property that this 64-bit register satisfies?

- ☐ Regular register
- ☐ Safe Register
- ☐ Atomic register
- ☐ Does not satisfy any of these properties

Problem 5. You are given the algorithm in Figure 4 for constructing a single-reader single-writer (SRSW) M-valued atomic register using single-reader single-writer (SRSW) Boolean atomic registers. Does this proposal work? Either prove the correctness or present a counterexample.