CS1760: Multiprocessor Synchronization Due 27 September 2018

Homework 2

Professor: Maurice Herlihy *TA: Jonathan Lister*

Problem 1. Programmers at the Flaky Computer Corporation designed the protocol shown in Figure 1 to achieve n-thread mutual exclusion.

```
1  class Flaky implements Lock {
2    private int turn;
3    private boolean busy = false;
4    public void lock() {
5      int me = ThreadID.get();
6      do {
7        do {
8          turn = me;
9        } while (busy);
10       busy = true;
11     } while (turn != me);
12   }
13   public void unlock() {
14     busy = false;
15   }
16 }
```

Figure 1: The Flaky lock used in Exercise 1.

For each question, either sketch a proof, or display an execution where it fails.

- Does this protocol satisfy mutual exclusion?
- Is this protocol starvation-free?
- Is this protocol deadlock-free?
- Is this protocol livelock-free?

Hint: Some of these properties imply other properties.

Problem 2. Consider a variant of Peterson's algorithm, where we change the unlock method to be as shown in Figure 2. Does the modified algorithm satisfy deadlock-freedom? What about starvation-freedom? Sketch a proof showing why it satisfies both properties, or display an execution where it fails.

Problem 3. Suppose n threads call the Visit() method of the Bouncer class shown in Figure 3.

```
1  public void unlock() {
2    int i = ThreadID.get();
3    flag[i] = false;
4    int j = 1 - i;
5    while (flag[j] == true) {}
6  }
```

Figure 2: The revised unlock method for Peterson's algorithm used in Assignment 3

```
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 3: The Bouncer class implementation.

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.

Problem 4. We saw earlier the following theorem on the bounds of shared memory for mutual exclusion: any deadlock-free mutual exclusion algorithm for n processes must use at least n shared registers. For this exercise, we examine a new algorithm that shows that the space lower bound of the above theorem is tight. Specifically, we will show that:

> **Theorem**: There is a deadlock-free mutual exclusion algorithm for n processes that uses exactly n shared bits.

To prove this new theorem, we study the One-Bit algorithm, which uses exactly n bits to achieve mutual exclusion. This algorithm was developed independently by J.E. Burns and L. Lamport and is interesting in that it uses the minimum possible shared space. The pseudocode is shown in Figure 4. The way the One-Bit algorithm works as follows: First, a thread must indicate that it is contending for the critical section by setting its bit to true. Then, it loops and reads the bits of all threads

```
1
2   Initially : all entries in Flag are false
3   repeat
4       Flag[i] := true; j := 1
5       while (Flag[i] = true and j < i) do
6           if Flag[j] = true then
7               Flag[i] := false; await Flag[j] = false;
8           j := j+1;
9   until Flag[i] = true;
10  for j:=i+1 to n do await Flag[j] = false;
11
12  {enter critical section}
13
14  Flag[i] := false;
```

Figure 4: Pseudocode for the One-bit algorithm

with smaller identifiers than its own. If all of these bits are false (while its own bit is true), then the thread exits the loop. Otherwise, the thread sets its bit to false, waits until the bit it is waiting on becomes false, and starts all over again. Afterwards, the process reads the bits of all threads that have identifiers greater than its own, and waits until they are all false. Once this check passes, the thread can safely enter the critical section.

- Prove that the One-Bit algorithm satisfies mutual exclusion.
- Prove that the One-Bit algorithm is deadlock-free.