Massachusetts Institute of Technology Handout 16
6.852: Distributed Algorithms
Prof. Nancy Lynch October 30, 2003

Problem Set 5, Part a

Due: Thursday, November 13, 2003

Reading:

Distributed Algorithms, Chapters 19, 9 (skim), 10 (skim).
Maurice Herlihy and Nir Shavit. *Introduction to Multiprocessor Synchronization and Concurrent Data Structures*. Chapter 2. (In the course cabinet.) *Note: this is copyrighted material. It must not be distributed and can only be used for the purposes of this class.*

Reading for next week:

Chapter 13.

Problems:

1. Exercise 19.1
2. Exercise 19.13.
   Don't give formal code, or a formal correctness/complexity analysis proof. Rather, describe the ideas informally, in words. No more than 1 page.
3. Programmers at the Flaky Computer Corporation designed the following protocol for n-process mutual exclusion with deadlock-freedom.

```
class Flaky implements Lock{

  private int turn;
  private bool busy = false;  // initially to false

  void acquire(int i){        // code for thread i
    do {
      do {                    // loop until this.busy is false
        this.turn = i;
      } while (this.busy);
      this.busy = true;
    } while (this.turn != i);
  }

  void release(int i){
    this.busy = false;
  }
}
```

(a) Does this protocol satisfy mutual exclusion? Either sketch a proof, or display an execution where it fails. Is this a safety, liveness or fairness property?

(b) Does this protocol satisfy no-lockout? Either sketch a proof, or display an execution where it fails. Is this a safety, liveness or fairness property?

(c) Does this protocol satisfy no-deadlock? Either sketch a proof, or display an execution where it fails. Is this a safety, liveness or fairness property?

(d) Does this protocol satisfy no-starvation (i.e., every process that wants to get in the critical section, gets in eventually)? Either sketch a proof, or display an execution where it fails. Is this a safety, liveness or fairness property?

(e) What are the differences between satisfying no-lockout, satisfying no-deadlock and satisfying no-starvation? Is any of this concepts stronger than another?

4. Another way to generalize the two-thread Peterson mutual exclusion algorithm is to arrange a number of two-thread Peterson locks in a binary tree. Suppose n is a power of two. Each thread is assigned a leaf lock which it shares with one other thread.
In the tree-lock's acquire method, the thread acquires every two-thread Peterson lock from that thread's leaf to the root.
The tree-lock's release method for the tree-lock unlocks each of the two-thread Peterson locks that thread has acquired, from the root back to its leaf.

   (a) Either sketch a proof that this tree-lock satisfies mutual exclusion, or given an execution where it does not.

   (b) Either sketch a proof that this tree-lock satisfies no-lockout, or given an execution where it does not.

   (c) Is there an upper bound on the number of times the tree-lock can be acquired and released while a particular thread is trying to acquire the tree-lock?

5. The ℓ-exclusion problem is a variant of the lockout-free mutual exclusion problem. We make two changes: as many as ℓ threads may be in the critical section at the same time, and fewer than ℓ threads might fail (by halting) anywhere in their code, including in the critical section.

   Your implementation must satisfy the following conditions:

   ℓ-Exclusion: At any time, at most ℓ threads are in the critical section.

   ℓ-Lockout-Freedom: As long as fewer than ℓ threads are in the critical section, then any thread that wants to enter the critical section will eventually succeed (even if some threads in the critical section have halted).

   Modify the Peterson n-process mutual exclusion algorithm to make it into an ℓ-exclusion algorithm, and prove its correctness.