Concurrent Systems — Exam

June 2018

Name: ____

Duration: 120 minutes — No document authorized

1.

a) You have a choice between buying one uniprocessor that executes five zillion instructions per second, or a ten-processor multiprocessor where each processor executes one zillion instructions per second. Using Amdahl's Law, explain how you would decide which to buy *for a particular application*. As reminder, Amdahl's Law can be expressed as (with *n* the number of processors and *p* the fraction of parallel time):

$$Speedup = \frac{1}{1 - p + \frac{p}{n}}$$

____

____

____

____

____

____

____

____

____

____

b) Explain the principle of exponential back-off in the context of spin locks.

____

____

____

____

____

____

____

____

____

____

c) Consider the simple **TASLock** mutex below. Add a **tryLock()** method that attempts to acquire the lock and indicates to the caller whether it is successful or not.

```
class TTASLock implements Lock {
  AtomicBoolean state = new AtomicBoolean(false);
  void lock() {
    while (state.getAndSet(true)) {}
  }
  void unlock() {
    state.set(false);
  }
}
```

d) In the linked list algorithms seen in the course, would the **contains()** method of the lazy and lock-free algorithms still be correct if logically removed entries were not guaranteed to be sorted? Justify your answer.

2.
Consider the following class:

```
class FastPath implements Lock {
  private static ThreadLocal<Integer> myIndex;
  private Lock lock = new ReentrantLock();
  private volatile int x, y = -1;

  public void lock() {
    int i = myIndex.get(); // Each thread has its own index
    x = i;                 // I'm here
    while (y != -1) {}     // Is the lock free?
    y = i;                 // Me again?
    if (x != i)            // Am I still here?
      lock.lock();         // Slow path
  }

  public void unlock() {
    y = -1;
    lock.unlock();
  }
}
```

Does this class provide mutual exclusion? It so, sketch an argument why this is correct. Otherwise give a counterexample.

3.
Consider the following code:

```
class Example {
  int volatile x = 0;
  int volatile y = -4;
  int volatile z = 0;

  public void writer() {
    if (z == 0) {
      x = 4;
      y = 4;
      z = 1;
    }
  }

  public void cleaner(){
    if (z == 1) {
      System.out.println("The total is " + (x + y));
      x = 0;
      y = 0;
      z = 0;
    }
  }
}
```

Multiple threads can access both methods. What are the possible values printed by the program? Justify your answer.

4.

For each of the histories shown below, draw the corresponding timeline and indicate if they are sequentially consistent and/or linearizable. Justify your answer.

(a) Threads A, B, C; register r.

```
A: r.write(1)
C: r.read()
A: r:void
A: r.write(2)
C: r:2
C: r.read()
B: r.read()
A: r:void
C: r:1
A: r.write(1)
B: r:1
A: r:void
```

(b) Threads A, B; stack s.

```
A: s.push(10)
B: s.push(10)
A: s:void
A: s.pop()
B: s:void
B: s.empty()
A: s:10
B: s:true
A: s.pop()
A: s:10
```

(c) Threads A, B, C; queue q.

```
A: q.enq(x)
B: q.enq(y)
A: q:void
B: q:void
A: q.deq()
C: q.deq()
A: q:y
C: q:y
```

(a)

(b)

(c)

5.
Design a simple bounded, lock-based concurrent **`Stack<T>`** using an array.

6. (Bonus point)

Implement a simple **`ReadWriteLock`** class using Java **`synchronized`**, **`wait()`**, **`notify()`**, and **`notifyAll()`** constructs. Remember there are four methods to implement, **`readLock()`**, **`readUnlock()`**, **`writeLock()`** and **`writeUnlock()`**.