Concurrent Systems — Exam

June 2013

Name: _____

Duration: 120 minutes — No document authorized

1.

a) Explain informally the notion of linearizability. How does it differ from sequential consistency?

_____

_____

_____

_____

_____

_____

b) Explain the principle of the MESI protocol. As a reminder, MESI stands for Modified, Exclusive, Shared, Invalid.

_____

_____

_____

_____

_____

_____

c) Why is TTAS (test-and-test-and-set) more efficient on many types of architectures than TAS (test-and-set) for implementing a spin lock?

_____

_____

_____

_____

_____

_____

d) What does the `volatile` keyword guarantee in Java?

e) Rank the three following progress conditions from the strongest to the weakest: *lock-freedom, obstruction-freedom, wait-freedom*.

f) What is the main disadvantage of work shedding as compared to work stealing?

2.
What are the “getAndSet” and “getAndDecrement” operations? Describe their semantics informally or using pseudo-code.

Can we implement “getAndDecrement” using “getAndSet”?
If the answer is “no”, explain why.
If the answer is “yes”, write the corresponding code and indicate if the implementation is lock-free, and/or wait-free.

3.
Consider the following 2-thread mutual exclusion algorithm seen in the course:

```
public class MyLock implements Lock {
  private volatile int victim;

  public void lock() {
    // Convention: i is local thread
    victim = i;
    while (victim == i) {}
  }

  public void unlock() {}
}
```

Does this algorithm ensure mutual exclusion?
What problem does it suffer from?
How can this problem be avoided?

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

4.

Considering a shared queue `Q` and two threads `T1` and `T2`, write four sample histories that respectively have the following properties:

a) Linearizable and sequentially consistent
b) Not linearizable but sequentially consistent
c) Linearizable but not sequentially consistent
d) Neither linearizable nor sequentially consistent

If there are no valid histories with some of the requested properties, explain why.

a)

_____

_____

_____

_____

b)

_____

_____

_____

_____

c)

_____

_____

_____

_____

d)

_____

_____

_____

_____

5.

Consider the bounded lock-free queue below. We assume that integer overflows are not an issue.

What is the problem with this implementation?

How can we fix it while preserving the lock freedom property of the implementation?

```
class LockFreeQueue {
  static final int MAX = 256;
  AtomicInteger head, tail;
  int queue[] = new int[MAX];

  public void enq(int v) {
    while (true) {
      int t = tail.get();
      if (t == head.get() + MAX)
        throw new FullQueueException();
      if (tail.compareAndSet(t, t + 1)) {
        queue[t % MAX] = v;
      }
    }
  }

  public int deq() {
    while (true) {
      int h = head.get();
      if (h == tail.get())
        throw new EmptyQueueException();
      if (head.compareAndSet(h, h + 1)) {
        return queue[h % MAX];
      }
    }
  }
}
```

6.

A barrier is a synchronization aid that allows a set of threads to all wait for each other to reach a common point. A barrier is initialized with a thread count (`n`). All threads that call the `cross()` method block until `n` threads have called the method. At that point, all threads are allowed to proceed.

A barrier can be easily implemented in Java using a lock and a condition variable, whose interfaces are summarized below as a reminder:

```
interface Lock {
  void lock();
  void unlock();
  Condition newCondition();
}
```

```
interface Condition {
  void await();
  void signal();
  void signalAll();
}
```

Complete the code below to implement a barrier in Java.

```
class Barrier {

_____

_____

_____

  public Barrier(int n) {

_____

_____

_____

  }
  public void cross() {

_____

_____

_____

_____

_____

_____

_____

_____

  }
}
```