Student name: ....

Multicore Programming: Exam

JMCS

19/06/2019

Duration: 120 minutes — No document authorized

Exercise 1

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

b) What do we mean by “linear scalability”?

c) What is false sharing?

d) You have an application that can use up to 4 processors for 40% of its execution (parallel part). You developed a second version of the application that can use only up to 2 processors, but for 80% of its execution.

Using Amdahl's law, determine if it is better to buy a machine with 4 processors and use the first version of the application, or a machine with 2 processors and use the second version. Justify your answer by computing and comparing the speedups.

As reminder, Amdahl's Law can be expressed as (with *n* the number of processors and *p* the fraction of parallel time):

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

....

....

....

....

....

....

....

....

Student name: ....

Exercise 2

Consider a shared stack *S* and two threads T_1 and T_2. Are the following histories linearizable and/or sequentially consistent? If so, write the equivalent sequential history.

Note: It could be helpful to draw a graphical representation of the histories.

a) **Linearizable (Y/N)?**..

Sequentially consistent (Y/N)? ....

T1 S.push(a) ....

T1 S:void ....

T2 S.push(b) ....

T1 S.pop() ....

T2 S:void ....

T1 S:b ....

T2 S.pop() ....

T2 S:a ....

b) **Linearizable (Y/N)?**..

Sequentially consistent (Y/N)? ....

T1 S.push(a) ....

T1 S:void ....

T2 S.push(b) ....

T1 S.pop() ....

T2 S:void ....

T1 S:a ....

T2 S.pop() ....

T2 S:b ....

c) **Linearizable (Y/N)?**..

Sequentially consistent (Y/N)? ....

T1 S.push(a) ....

T1 S:void ....

T2 S.pop() ....

T1 S.push(b) ....

T2 S:b ....

T2 S.pop() ....

T2 S:a ....

Exercise 3

Complete the code below to implement a read/write lock using Java monitors (`synchronized` blocks and `wait`/`notify`/`notifyAll` methods). The lock must be *starvation-free* for writers (readers cannot prevent writers from acquiring the lock infinitely).

```
public class ReadWriteLock {
.........................................................................................................
.........................................................................................................
.........................................................................................................
.........................................................................................................
.........................................................................................................
.........................................................................................................
.........................................................................................................
.........................................................................................................

  // Lock for reading (multiple readers can lock for reading if
  // RWLock not locked for writing, and not writers waiting to
  // lock for writing)
  public synchronized void lockRead() {
.........................................................................................................
.........................................................................................................
.........................................................................................................
.........................................................................................................
.........................................................................................................
.........................................................................................................
.........................................................................................................
.........................................................................................................
  }

  // Unlock for reading
  public synchronized void unlockRead() {
.........................................................................................................
.........................................................................................................
.........................................................................................................
.........................................................................................................
.........................................................................................................
.........................................................................................................
.........................................................................................................
.........................................................................................................
  }
```

Student name: ....

```
  // Lock for writing (only one writer can lock for writing,
  // and only if no threads are currently reading)
  public lockWrite() {
.................................................................................................................
.................................................................................................................
.................................................................................................................
.................................................................................................................
.................................................................................................................
.................................................................................................................
.................................................................................................................
.................................................................................................................
.................................................................................................................
.................................................................................................................
.................................................................................................................
  }
  public unlockWrite() {
.................................................................................................................
.................................................................................................................
.................................................................................................................
.................................................................................................................
.................................................................................................................
.................................................................................................................
.................................................................................................................
.................................................................................................................
.................................................................................................................
.................................................................................................................
.................................................................................................................
  }
}
```

Exercise 4

Consider an unbounded lock-based queue with the following deq() method:

```
public T deq() throws Exception {
  T result;
  deqLock.lock();
  try {
    if (head.next == null) {
      System.out.println("Queue empty");
      throw new Exception();
    }
    result = head.next.value;
    head = head.next;
  } finally {
    deqLock().unlock();
  }
  return result;
}
```

Is it necessary to protect the check for a non-empty queue in the deq() method with a lock or the check can be done outside the locked part? Explain.

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

....

Student name: ....

Exercise 5

A cyclic counter is a counter that can be incremented until it reaches an upper limit (specified when creating the counter), after which it rolls over to 0. Implement a lock-free cyclic counter in Java (without using monitors).

Hint: use an atomic variable to store the value of the counter.

```
public class CyclicCounter {
..........................................................................................................
..........................................................................................................
..........................................................................................................
..........................................................................................................
  public CyclicCounter(int limit) {
..........................................................................................................
..........................................................................................................
..........................................................................................................
..........................................................................................................
  }
  public int cyclicallyIncrementAndGet() {
..........................................................................................................
..........................................................................................................
..........................................................................................................
..........................................................................................................
..........................................................................................................
..........................................................................................................
..........................................................................................................
..........................................................................................................
..........................................................................................................
..........................................................................................................
..........................................................................................................
..........................................................................................................
  }
}
```

Exercise 6

A semaphore[1] is an abstract data type used for controlling access, by multiple processes, to a common resource in a parallel programming environment. A useful way to think of a semaphore is as a record of how many units of a particular resource are available, coupled with operations to safely (i.e., without race conditions) adjust that record as units are required or become free, and, if necessary, wait until a unit of the resource becomes available. Semaphores that allow an arbitrary resource count are called counting semaphores.

Consider the following implementation of a counting semaphore:

```
public class CountingSemaphore {
  private int available;
  public CountingSemaphore(int n) {
    available = n;
  }
  public synchronized void acquire() {
    while(available == 0)
      try { wait(); } catch(InterruptedException e) { }
    available--;
  }
  public synchronized void release() {
    available++;
    notify();
  }
}
```

Propose an equivalent implementation of a counting semaphore that is lock-free.

Hint: you can use an atomic integer for keeping track of available resources.

[1] https://en.wikipedia.org/wiki/Semaphore_(programming)

Student name: ....

```
public class LockFreeCountingSemaphore {
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
  public LockFreeCountingSemaphore(int n) {
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
  }
  public void acquire() {
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
  }
  public void release() {
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
....................................................................................................
  }
}
```

You can use this additional space upon need (indicate the question number):