CSE451
Exam #1 (version A)
Fall 2006

Your Name:

Short Answer (worth 4 points each)

1) What is the difference between a process and a thread?

2) Suppose we have three jobs: A, B, and C. These jobs have runtimes of 5, 15, and 10 seconds, respectively (that is, A=5, B=15, C=10). If the scheduler is first-come, first-serve, what is the worst-case arrival ordering for these processes?

3) What is the average turnaround time for the schedule you listed above? Assume the jobs arrive at approximately the same time, and that there is zero context switch cost.

4) The `String` class in Java is immutable. Also, `String` contains a `substring` method, which creates and returns a new `String` object. How many threads can concurrently execute inside this `substring` method while preserving thread safety?

5) What prevents ordinary users from directly reading and writing to the disk (instead of passing through the file system)?

6) What is an advantage of layered OS structures, such as Dijkstra's THE system?

7) In class, we discussed the concept of “user threads”, which are implemented entirely in user-mode code. One problem with user threads is that a single blocking operation causes the entire process (and hence, all the user threads) to block.

Through a clever implementation, it is possible to implement user threads that can block without blocking the underlying kernel thread. Let’s call this version of user threads NB-Threads. In all other respects, NB-Threads behave like the ordinary user threads that we learned about.

(*Implementation details: the NB-Threads package uses non-blocking I/O operations. The threads package transparently transforms all blocking read and write operations into non-blocking calls of equivalent functionality. During the non-blocking I/O operation, the threads package blocks the user thread (but not the kernel thread!). Once the I/O operation has completed, the user threads package wakes up the appropriate user thread.*)

Would NB-Threads provide any benefit for the multi-threaded π calculator? If yes, would NB-Threads work as well as kernel threads? Explain your answer.

8) Would NB-Threads provide any benefit to the multi-threaded web server? If yes, would NB-Threads work as well as kernel threads? Explain your answer.

Implementation Problems

9) The class shown below implements a Set data structure. A set is an unordered collection of Objects that does not contain duplicates. This implementation uses a hashing strategy (similar to a hash table). Each object is mapped to a bucket using its hashCode method. Each bucket is represented by a LinkedList, which handles the case when multiple objects map to the same bucket.

```
public class SimpleSet {
    private static final int NUM_BUCKETS = 16;

    // The set is implemented with an array of buckets;
    // each bucket contains a linked list of items
    // Note: LinkedList is not internally synchronized, so we
    // must manually synchronize to maintain thread-safety
    private final LinkedList [] buckets;

    public SimpleSet() {
        buckets = new LinkedList [NUM_BUCKETS];
        for (int i = 0;i<NUM_BUCKETS;i++) {
           buckets[i] = new LinkedList();
       }
    }

    // A helper method which gives the hash bucket for the given value
    private LinkedList hashBucket (Object value) {
       return buckets[value.hashCode() % NUM_BUCKETS];
    }

    // does the set contain the given value?
    public boolean contains (Object value) {
       LinkedList bucket = hashBucket(value);
       synchronized(bucket) {
          return bucket.contains(value);
       }
    }

   // add the given value to the set
   public void add (Object value) {
      LinkedList bucket = hashBucket(value);
      synchronized(bucket) {
        // make sure we don't add the value twice!
        if (! bucket.contains(value))
           bucket.add(value);
      }
   }

   // remove the value from the set
   public Object remove(Object value) {
      LinkedList bucket = hashBucket(value);
      synchronized(bucket) {
         return bucket.remove(value);
      }
   }
}
```

9a) [4 points] Notice that the SimpleSet does not use any synchronized methods. Instead, it synchronizes on the individual hash buckets. What is the primary advantage of this approach (as compared to synchronized methods)?

9b) [10 points] The code below shows a broken implementation of an xorAdd method (xor means “exclusive or”). The goal of the method is to ensure that exactly one of the two arguments is contained in the set. However, the given implementation contains safety and/or liveness problems. Re-write this method so that it is free from concurrency errors. Your implementation should be reasonably efficient.

```
// Ensure that exactly one of these arguments is in the set.
// If xor is already true, do nothing.
// Otherwise, give preference to the first argument
// TODO: This is not thread-safe: FIXME!
public void xorAdd(Object val1,Object val2) {
    boolean contains1 = this.contains(val1);
    boolean contains2 = this.contains(val2);

    // if the xor is already true, we do nothing
    if (contains1 ^ contains2) {
        return;
    }
    // otherwise, force XOR to be true
    else {
        this.add(val1);
        this.remove(val2);
    }
  }
}
```

9c) [6 points] Describe (but do not code) an implementation for a getSize method, which returns the total number of elements in the set. Your implementation should be reasonably efficient.

10a) [10 points] In project #2, you implemented a semaphore using a condition variable. This problem asks you to do the opposite.

In class, we looked at a BufferPool, which provides a thread-safe repository for a set of data buffers. When the pool is empty, any attempt to acquire a buffer will block. This blocking is implemented using a condition variable. When a buffer is returned to the pool, a single blocked thread is awoken. The source code for the BufferPool is provided below.

Your task is to re-write the BufferPool to use semaphore(s) instead of a condition variable. Your new version should behave the same as the current version. You can assume the existence of a Semaphore class, with the same interface as project #2 (shown on the next page). The Semaphore is correctly written (i.e., it uses sufficient synchronization to ensure thread-safety). Your new implementation **cannot** directly use a condition variable. In other words, the words "wait" and "notify" should not appear in your solution.

(As you learned in project #2, the Semaphore *might* be implemented with a condition variable. The implementation details of the Semaphore do not matter for this problem).

```
// This class maintains a statically-allocated pool of memory buffers
// TODO: Convert this to use a Semaphore instead of a condition variable
public class BufferPool {

   public static final int BUFFER_POOL_SIZE = 512;

   // Note: the list is not internally thread-safe.
   // So, every read and write must be synchronized
   private final List<Buffer> freeBuffers = new LinkedList<Buffer>();

   public BufferPool() {
      for (int i=0;i<BUFFER_POOL_SIZE; i++) {
         Buffer buf = new Buffer();
         freeBuffers.add(buf);
      }
   }

   public synchronized Buffer getFreeBuffer() throws InterruptedExcepetion {
      while (freeBuffers.isEmpty()) {
         wait();
      }

      assert (!freeBuffers.isEmpty());
      return freeBuffers.remove(0);
   }

   public synchronized void returnFreeBuffer(Buffer buf) {
      freeBuffers.add(buf);
      notify();
   }
}
```

```
// Interface to a Semaphore object; You can assume this is already written!
public abstract class Semaphore {
  // initialize a semaphore with some value
  public Semaphore (int initialValue) { ; }

  // initialize a sempahore with value zero
  public Semaphore () {  }

  public void down ();

  public void up ();
}
```

10b) [4 points] In class, we discussed how the buffer pool can deadlock. One proposed solution was for each thread to request its entire allocation up-front. Can you explain how this would prevent deadlock?

10c) [6 points] Could you implement this deadlock-prevention scheme with the current Semaphore? If yes, describe how you would do it. If no, describe how the Semaphore would need to change. In either case, you do not need to implement your approach.