Computer Science 483
Concurrent Programming
Sample Final Exam

There are 6 pages in this exam printed front and back. Please make sure that you have all the pages now. This exam is **open book and open notes,** but cell phones, PDAs, computers, and other electronic devices are **not allowed.**

For this exam you should provide as much *relevant and correct* information as you can in your answers. I'm not looking for brain dumps but neither can many of the questions be answered in a sentence or two. Partial answers will receive partial credit reflecting the degree to which they approach a complete solution.

The exam is 120 minutes so take your time. It is very important that you take the time to understand what the questions are asking for.

Your name _____

1. a. (15) Using pseudocode, give an example of a message passing program that would deadlock under the assumption of synchronous message passing but would not deadlock using asynchronous (buffered) message passing.

b. (15) Again using pseudocode, sketch an implementation of asynchronous message passing channels built using synchronous message passing primitives. Assume that you have all of the primitives of the synchronous message passing project (synchronous send, receive, select, synch, etc.) but you only need to show implementations for asynchronous send and receive.

2. (20) For each of the four ACID properties of transactions name and define the property, then describe an implementation mechanism that could be used to achieve it.

3. We looked at several different server behaviors that are implemented by the OTP framework. One was called a server with "transactional semantics". Given our later discussions about transactions, which of the ACID transactional properties are implemented in the generic server code below. Point out specific features of the code that implement each of the ACID properties or describe why the property is not implemented.

```
%% loop for transactional semantics
loop(Name, Mod, OldState) ->
    receive
      {From, Request} ->
          try Mod:handle(Request, OldState) of
            {Response, NewState} ->
                From ! {Name, ok, Response},
                loop(Name, Mod, NewState)
          catch
            _:Why ->
                log_the_error(Name, Request, Why),
                %% send a message to cause the client to crash
                From ! {Name, crash},
                %% loop with the *original* state
                loop(Name, Mod, OldState)
          end
    end.
```

4. The following is a wait-free implementation of a counter using compare-and-swap. Assume that CASwapInt is a primitive datatype holding an int value and providing a CASwap method and a get method as its setter and getter, respectively.

```
private CASint val;
public int getValue() { return value.get(); }
public int increment() {
    int v;
    do {
        v = value.get();
    } while (v != value.CASwap(v, v+1));
    return v+1;
}
```

a. (15) Explain how the code works.

b. (15) Give an alternate implementation using the primitives of the STM paper and explain how your code works.

5. a. (15) The following code is not safe in the Java memory model. Describe how it violates the Java memory model's requirements and consequently how it can fail.

```
public class S {
    private static T t = null;
    public static T getT() {
        if (t==null) {
            synchronized (S.class) {
                if (t==null) {t = new T();}
            }
        }
        return t;
    }
}
```

b) (15) suggest at least one way to fix the problem

6. Describe the purpose of the MapReduce primitive and briefly describe how it works.