Computer Science 483/580
Concurrent Programming
Midterm Exam 2 - Sample

Your name _____

There are 6 pages to this exam printed front and back. Please make sure that you have all the pages now.

The exam is designed to take about 50 minutes.

This exam is **open book and open notes.** You may use a laptop computer to refer to an electronic copy of the textbooks and to your notes. You may also use the Java and Erlang compilers but this should not be necessary. You ***must not*** use your computer to ***access the network***; if possible turn the network interface. Other electronic devices – e.g. cellphones – may not be used.

Honor statement: I have followed the above instructions regarding use of electronic devices while taking this exam.

Signature _____ Date _____

1. (Executor flavors) Listing 8.1 in the Java textbook is an example of code that suffers a thread-starvation deadlock when run using a SingleThreadExecutor. In general, SingleThreadExecutors have a heightened danger of deadlock. Why then did the library designers include them as a standard executor – that is, under what circumstances is a SingleThreadExecutor a better choice than any of the other standard executors?

2. (Executor sizing) Suppose you're designing software for a 16-processor multiprocessor. The each task processed by your software uses one processor 25% of the time (assume that the rest of the time is spent waiting for messages from a remote host – the network itself is not a bottleneck). How many threads should be available in a thread pool for processing these tasks so that the overall system is 50% busy on average? (Hint: what is the ratio W/C in the formulas on p. 171?)

3. (GUIs) GUI libraries usually use a single-threaded design in which *the* GUI thread processes *events* by sequentially calling *listeners* attached to GUI objects such as text boxes, buttons, etc.

a) What is the advantage of the single-threaded design over a multi-threaded design? That is, why has the world settled on single-threaded GUIs as the design of choice?

b) A big drawback to a single-threaded GUI design is that long-running listeners can make the whole GUI unresponsive to further user input and unable to provide feedback about the long-running operation. Describe how a long-running listener can be designed to run in a separate thread so that the GUI does not lock up during its execution. What are the constraints on this additional thread with respect to updating the GUI, for example to provide a progress indicator or to indicate completion of its processing?

4. A wait-for graph is a directed graph that can be used to determine whether a program execution is currently in deadlock. The two kinds of nodes in a wait-for graph correspond to (a)_____ and (b)_____. An edge from an (a) node to a (b) node means that _____. An edge from a (b) node to an (a) node means that _____. A deadlock is indicated by _____ in the wait-for graph.

5. Early in the class we discussed Dekker's algorithm for mutual exclusion between two processes and we noted its many deficiencies. In Java we usually rely on locks (in any of several different kinds) to provide mutual exclusion but locks, themselves, are abstractions that are not typically provided by computer processor hardware. To implement locks on a modern processor usually involves some hardware instruction that implements an atomic check-then-act sequence, for example compare-and-swap or test-and-set. Describe the operation of such an instruction.

6. In studying Java concurrency we have stressed issues of synchronization and visibility of updates. In talking about Erlang we have stressed the benefits of single-update variables and absence of shared state for avoiding many of these problems. Give an example where these synchronization and visibility issues are inherently present in the Erlang language but they have been hidden from Erlang programmers by the language implementation. Explain.

7. Give an example of an Erlang **`receive`** expression that will always process the first (oldest) message in the mailbox, regardless of the message's value.

8. Give an example of an Erlang **`receive`** expression that will always leave the message `{one, 1}` in the mailbox (assuming, of course, that the message is already in the mailbox when the receive expression is evaluated).

9a. Describe the *use* and *function* of the `link` operation in Erlang.

9b. What is the role of *system processes* with respect to `link`?

10. On the next page you will find skeleton code implementing semaphore servers in Erlang. Complete the code as indicated in the comments. Note: this code does not use the "object-oriented" style that we discussed in class last Wednesday.

```
-module(semaphore).
-export([newServer/1, release/2, acquire/2]).
% Implementation of a classic semaphore.

% Create a new semaphore with R initial resources
newServer(R) ->
   spawn(_____________________________________________)
   .

% release N resources to the semaphore
release(Server, N) ->
   rpc(_____________________________________________)
   .

% acquire N resources from sem; wait if not available
acquire(Server, N) ->
   rpc(_____________________________________________)
   .

server(__________________) ->
   receive
      {Client, {_______________, N}}
         when _____________________ ->
            Client ! {self(), {}},
            ___________________________________________;
      {Client, {release, N}} ->
         Client ! {self(), {}},
         ____________________________________________;
   end
   .

% the usual rpc function
rpc(Server, Msg) ->
   Server ! {self(), Msg},
   receive
      {Server, Response} -> Response
   end.
```