CEG 7370 Distributed Computing

2014 Spring • Final Exam • 100 points

Given to you: April 29, 9:30 PM;
Due: April 30, 11:59 PM.

This take-home exam permits the use of a Linux/Windows laptop/PC running javac, g++, scala, Eclipse, Idea, and other software development and drawing tools. *You are welcome to refer to your notes in a way equivalent to using a simple cheatsheet to solely help you only to remember the syntax of the various notations and langauges.* It is otherwise a traditional closed book, closed notes exam. In particular, you are honor bound **not** to Internet surf or access any content already existing (other than as indicated) once you access the final until you turnin the answers. The primary reasons in making this a take-home are to relieve time pressure and give you a comfortable (computing/home) environment. Do not give or take help from others.

Submit your answers on `thor.cs.wright.edu` using `~pmateti/ceg7370/turnin Final answers.pdf bounded-buffers-with-actors.scala`

1. (5 * 8 points) The following statements may or may not be (fully or partially) valid. Explain the underlined technical terms occurring in each statement. Explain/ discuss/ dispute the statement. It is *possible* to write no more than, say, five, lines each, and yet receive full score.
    i. A "safety" property is defined thus: Let **bad** be a predicate characterizing a "bad" state of program code segment S. Assume that {P} S {Q} holds. Assume that I is a global invariant. If I implies the negation of **bad**, we say that **not bad** is a safety property for S. Thus, it all depends on whether *we consider a certain property good or bad.*
    ii. In a collection of semaphores that constitutes a split binary semaphore, at least one of them must be 1.
    iii. The happened before relation, as discussed, requires a logical clock for every process. But a typical machine (node) in a distributed system runs several processes. So, it is sufficient to maintain a logical clock per node that is shared by all processes running on that node.
    iv. In the context of our distributed semaphore implementation, not all fully acknowledged messages are necessary.
    v. Detection of termination is difficult only in peer-to-peer distributed computing not in client-server computing.
    vi. RPC/RMI based solutions are distributed, but not parallel.
    vii. We wish to maximize (a) concurrency, we prefer (b) symmetric solutions, we rate (c) correctness much higher than efficiency, and cannot tolerate (d) deadlocks or (e) livelocks. Defend *why* for each of these five.
    viii. "A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable." Is this a good definition?
2. (20 * 3 points)
    i. Using the technique of weakened assertions, prove that `{x = 0} S {x = 9}` is a theorem. S is `var x := 0; co ‹x += 2› || ‹x += 3› || ‹x += 4› oc`

ii.

```
int x := 0; int y := 0;
do true → x := x + 1 ;
[] true → y := y + 1 ;
od
```

Consider the code block at left. Do any of the temporal assertions

(a) (5 points) always x = 0

(b) (5 points) eventually x > 5

(c) (10 points) eventually always y = 3

hold? Where? When? How? Under weakly fair scheduling?

iii. Implement the standard Producer/ Conumers example in Scala using Akka Actors. Use either become or FSM or neither. You may want (but not required) to see an implementation using threads and synchronize: bounded-buffers-with-threads.scala. The solution to this question is to replace the threads with actors. Feel free to use (or ignore) portions of this source code. Submit your file of Scala code as a separate file named `bounded-buffers-with-actors.scala`

3. (0 points) [For survey purposes only.] Please record your effort in minutes for each of the above items. Other feedback you wish to give is also welcome.
