7CSCE-313 Quiz 3 [20 points]

Solutions must be posted by 11:59pm Tue Mar 29 on eCampus to receive credit. Late submissions are not allowed. You can print, write, scan and upload to eCampus. Or you can type answers and upload. In either case, please upload your answers in PDF format.

Q1. [4 points] Rebecca has written a multithreaded program which produces an incorrect answer some of the time, but always completes. She suspects a race condition. Which of the following are strategies that can reduce, and with luck eliminate, race conditions in her program? **Circle the R, E, or U below to indicate that the given approach can Reduce race conditions, Eliminate them, or is Useless when it comes to race conditions.**

[✓ **R**] **E** **U** Separate the multithread program into multiple single-threaded programs, and run each thread in its own process. Share data between the programs via named pipes and read and write calls. No other changes to the program.

R **E** [✓ **U**] Same as above, but share data between the programs via shared memory segments.

[✓ **R**] **E** **U** Apply the "one-writer" rule. Here you modify the program so that each variable has only one writer, i.e. only one thread can write to a shared variable. Multiple threads can read from it.

R [✓ **E**] **U** Ensure that each shared variable is protected by some lock.

Q2. [6 points] A binary semaphore is initialized to TRUE. Then 5 P() operations are performed on it in a row followed by 8 V() operations. Now 5 more P() operations are performed on it. What is the number of processes waiting on this semaphore? **Show your work.**

The P() operations have only 2 spots to load into, but the semaphore is unable to clear. There are 2 P() operations in the queue, and 3 waiting. The first of the V() operations will clear the 5 P() operations, and there will be 3 V() operations left. Then the semaphore loads the 5 P()'s, and the remaining V() operations will clear and leave only 2 remaining processes.

Q3. [10 points] Consider the following two threads of a process, to be run concurrently in a shared memory (all variables are shared between the two threads):

Thread A	Thread B
`for (i=0; i<5; i++) {`	`for (j=0; j<5; j++) {`
`    x = x + 1;`	`    x = x + 2;`
`}`	`}`

Assume the following:

1. a single-core system
2. memory load and store are **atomic** (i.e. they start and finish without interruption)
3. x is initialized to 0 before either thread starts, and
4. x must be loaded into a register before being incremented (and stored back to memory afterwards).

The following questions consider the <u>final</u> value of x <u>after</u> both threads have completed.

3a. [2 points] State the upper and lower bounds of value of x when both threads have completed.

Lower bound is 5 and upper bound is 15. This is because the first process upon completion (since we assume atomic) finishes with a value of x = 5, and the second running simultaneously can finish with a value of 3 * (whatever Thread A returns), or 15 in this case.

3b. [3 pts]: Give a concise proof why x≠1 when both threads have completed.

Since the threads are atomic and must complete all the way through, the second must run if the first runs. Regardless of Thread A or Thread B, once the value of x is set to 0 it stays there. The only way that x = 1 is for Thread A to start with a 0 and return a 1. Since there are at least 4 iterations before this can happen, there is no way that Thread A can load a 0.

3c. [5 pts]: Suppose we replace 'x = x+2' in Thread B with an **atomic double increment** (i.e. equivalent to adding the number 2) operation atomicIncr2(x) that **cannot be preempted** while being executed. What are **ALL possible final values** of x? Explain.

Values are 5, 7, 9, 11, 13 or 15. The x=x+2 statements can be interrupted by the x=x+1 statement, since the x=x+2 statement is atomic. However, the x=x+1 can never be erased, since the load and store of the x=x+2 cannot be separated from one another. For this reason, our final value of Thread A must be at least 5, with anywhere from 0 to 5 successful iterations of Thread B.