Concurrent Programming

COMP 409, Fall 2010

Assignment 1

Due date: Thursday, October 7, 2010
6pm

Assignment Questions

1. Suppose two threads are executing the following code:

```
volatile int x=1, y=1, stop=0;
```

Thread 1	Thread 2
`while (stop==0) {`	`while (x==y) {`
`  y=2*y;`	`  y=x;`
`  x=2*x;`	`  x=y;`
`}`	`  stop=x-y;`
	`}`

In all cases assume Java-style atomicity guarantees.

(a) Are there any race conditions in this program? Why or why not? **1**

(b) What control flow (interleaving), if any, would let this program terminate? **3**

(c) Assume every interleaving is equally likely. What are the odds of this program terminating after each thread has executed 2 iterations of its loop? **6**

Programming Questions

You may use Java, or PThreads (C/C++) for the following implementations. In all cases your code must be in a professional style: **well-commented,** properly structured, and appropriate symbol names. Marks will be very generously deducted if not! All programs should include demonstrative, but not excessive, output. Your programs should not have race conditions and should maximize the ability of threads to execute concurrently.

2. You and 4 of your friends are cleaning out a zombie invasion. You've secured a warehouse consisting of just one very large room and 4 doors to the street. Zombies move slowly, so it's easy to control in general. You only have one weapon though, so you've set up the following protocol. Each of your 4 friends controls each of the 4 doors; they let in individual zombies, keeping count of how many have entered. You stand in the center, and eliminate the zombies that have entered as fast as you can, keeping track of how many you have removed.

   You don't want too many zombies in the room for obvious reasons, and so must periodically check on how many zombies are in the building in total. If there are too many you need to ensure no new zombies enter until you've had opportunity to reduce their numbers. For this you can only radio each of your friends individually to find out how many they have let in and/or ask them to close the door. Only once the total number is below a reasonable threshold should you allow zombies back in, again only by radio-ing each friend individually.

   Simulate this as a multi-threaded program. You should have 5 threads, one representing you and one for each friend/door. Each friend thread should let in a zombie with a 10% chance every 10ms, keeping track of the number she admitted. The thread representing you has a 40% probability of removing a zombie once every 10ms. You should check the total every 2s, and if it is below n then everything is ok, otherwise no new zombies should be admitted until you've reduced the number to below $n/2$. n is a command-line parameter.

You may user either Java or PThreads and appropriate synchronization (ie `synchronized` or mutexes). Your solution should allow for maximal concurrency—operations should not be serialized unless necessary. Run your program for a few minutes with $n = 5$, $n = 10$, $n = 100$. What is your throughout (zombies eliminated/second)? **18**

3. A circular, singly-linked list of letters is maintained by thread 1, which traverses the list and sometimes chooses to remove a letter or insert a new (random) letter. Thread 2 accesses the same data structure, also constantly traversing the list but just printing out the contents.

   Develop a simulation of the above. In this case you must find a solution that does not use blocking synchronization (`synchronized` in Java or mutexes in PThreads). You must still ensure your program has no race conditions of course. Note that items being traversed by thread 2 may also be in the process of being accessed as part of a deletion or insertion by thread 1—nevertheless, it should always be the case that thread 2 (eventually) rejoins the main, circular linked list.

   Both threads should perform an action and then sleep for 100ms, repeat. Thread 1 should randomly add (1/3) or remove (1/3) a node (letter) in the linked list, or move to the next item in the linked list (1/3). It never removes the last item in the linked list. Thread 2 should print the current letter and move to the next node. The simulation should end after 1000 characters have been printed. **12**

What to hand in

For assignment submission you will use `handin`. See the SOCS docs on the SOCS website for how to do this. Note that clock accuracy varies, and late assignments will not be accepted without a medical note: you have ample time, **do not wait until the last minute** to do the assignment nor to sort out how handin works. Assignments must be submitted on the due date **before 6pm**.

Where possible hand in only **source code** files containing code you write. Do not submit compiled binaries or .class files. For the written answer questions submit either an ASCII text document or a .pdf file *with all fonts embedded*. Do not submit .doc or .docx files.

Note that for written answers you must show all intermediate work to receive full marks.

This assignment is worth 10% of your final grade. **40**