CS332 Midterm Take-Home Fall 2017

You may use your book, notes, previous assignments and any materials from Moodle as resources for this exam. You may not discuss it at any level with classmates or anyone else except for myself, nor use the internet beyond looking at man pages or official C documentation including mac-specific documentation for libraries like pthreads. You should definitely never be consulting Stack Overflow or Wikipedia =) You SHOULD feel free to ask me any questions you have, worst case I will say it's something I can't answer.

Deliverables: Submit a zipped folder containing the files *lounge.c* and *bridge.xxx* (where xxx is some extension easily opened on my mac, e.g. pdf). Please name your folder with your carleton username BEFORE compressing/zipping it.

1. Write a solution to the bridge problem from the exam using **ONLY semaphores** (problem description is given below). Do this with pseudocode in some sort of text document named *bridge.xxx*. I do NOT want you to turn in a program, or use C syntax. This solution should be relatively simple and will be much easier for me to grade by reading the pseudocode than by running tests and looking through actual C code. You may use as many semaphores and other global variables as you wish, though again if you're using many you're making the problem too complicated. Make sure it's clear what the starting value is of any semaphores you create.

   An old bridge has only one lane and can only hold at most 3 cars at a time without risking collapse. At any given time, there may be at most 3 cars on the bridge, and all of them must be going the same direction. Write pseudocode for the two methods `ArriveBridge(int direction)` and `ExitBridge()` to control traffic. Each thread representing a car calls `ArriveBridge` when it arrives at the bridge and wants to go in the specified direction (0 or 1); `ArriveBridge` should not return until the car is allowed to get on the bridge. A car thread then calls `ExitBridge` when it gets off the bridge, potentially allowing other cars to get on. Don't worry about starving cars trying to go in one direction; just make sure cars are always on the bridge when they can be.

   For example, here is a bad solution: Create a mutex lock. In ArriveBridge acquire the lock, and in ExitBridge release the lock. This solution ensures the bridge won't collapse but by only allowing one car on the bridge at a time, ever, when 3 should be able to go as long as they are traveling in the same direction.

2. Submit your solution program in the file *lounge.c.* Any working solution will receive partial credit. **For full credit use only one mutex and one condition variable** for individuals to wait on before entering the lounge.

   A major feud has developed between math and cs faculty in the CMC! No one knows how it started, but rumor indicates it somehow involves the appropriate size of chalk pieces. The result is that fights are breaking out frequently in the shared faculty lounge on the 3rd floor of the CMC, and have escalated past ignoring each other to actual dirty looks. Something must be done! Through the arbitration of Sue, an agreement has been reached that will allow only faculty of one or the other departments to use the lounge at any given time. However, given the faculty are all somewhat out of their minds over the conflict, they need specific instructions on what to do when they want to use the lounge in order to ensure this separation. Your job is to come up with an algorithm using synchronization constructs that each prof of either department can follow. Specifically, the agreement states that once a prof from either department has entered the lounge, only others of that department may enter until everyone leaves and the lounge is empty again.

   As an extra twist, Laird is undergoing construction, so the deans sometimes use the CMC faculty lounge as a meeting place as well. It's awkward for either cs or math faculty to share the lounge with deans, so they also will only be able to enter when the lounge is empty or occupied by other deans. However, the deans have priority, such that when a dean arrives and the lounge is already occupied by profs, no one else may enter the lounge until the current occupants have left at which point the dean gets in next. Some example behavior for different arrivals is given below.

   This policy is implemented using a sign on the door with a magnet that can be placed on one of 4 options, “Open”, “Math”, “CS”, or "Dean". the algorithm must ensure that the sign on the door accurately reflects the current situation (i.e. no group can use the sign to pretend they are in the room when they are not). If the room is open and anyone is waiting, someone must be allowed to enter. You may add one more sign of some sort to the door to handle the deans.

   Each individual will be a separate thread that will simply arrive at the lounge at some point and then leave the lounge after some amount of time. The file loungeTest.c has example test code similar to what I will use to grade your solution. You should include the provided loungeTest.h in your solution file lounge.c, and then compile it together with the given test code, e.g. "gcc loungeTest.c lounge.c" to make sure it works correctly.

   You must implement the init and arrive/leave functions for each type of individual (cs, math, dean). Your final submission should NOT print anything, all output is in the test code. You may add as many global variables as you wish. The sign variable is already created and initialized, you must ensure that it's value is accurate at any given time. The init function should initialize your global variables and synchronization constructs. The arrive functions cannot see into the room, they must act based on the sign on the door (and possibly some second sign for the deans). The arrive function should not return until that individual has entered the lounge (at which point it should return). If an individual cannot yet enter the lounge, they should sleep and not busy-wait (profs & deans can always use a nap)! The leave functions are aware of the others in the room at the time they leave, can see the sign(s) on the door, and can see anyone sleeping outside the room as soon as they walk out, so they are allowed to use any/all of that information and act accordingly. Even during a feud a prof

leaving is willing to wake sleeping profs of the other department, and update the sign to be accurate, if necessary.