CS332 Midterm Take-Home Winter 2016

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 (in office hours or via email) any questions you have, worst case I will say it's something I can't answer.

Deliverables: Submit a zipped folder named with your Carleton username that contains the files *faculty.c* and *readwrite.c*.

1. Note: due to my delayed flight I have not yet had time to create an actual code version of the given pseudo-code, however I will do so on Sunday evening, so you should feel free to wait until I post that to do this problem, or you could pretty easily write the code yourself if you want to do this before then!

   The readers/writers problem is a classic thread synchronization problem where the constraints are that only one thread may write to a given file at a time, but many threads may read from the file at the same time. This is a common case in real applications involving any file that is "read-only". It is safe for many users to access it for reading at once, as the readers cannot modify the file, but only 1 user may have write access to actually modify the file, and then only if there are no readers. So in general at any given time a file may be being accessed by exactly 1 writer (and no readers), 1 or more readers (and no writers), or no one. Following is the pseudo code for one solution to the readers/writers problem.

```
// Initialize Variables
  lock                    // a mutex lock
  okToRead                // condition var
  okToWrite               // condition var
  activeReaders = 0       // integer
  activeWriters = 0       // integer
  waitingReaders = 0      // integer
  waitingWriters = 0      // integer
```

```
 // Algorithm for a thread reading from the file
   Reader()
       acquire lock
       while (activeWriters + waitingWriters > 0)
           waitingReaders++
           wait on okToRead
           waitingReaders--
       activeReaders++
       release lock

       // reading stuff here

       acquire lock
       activeReaders--
```

```
        if (activeReaders==0 && waitingWriters>0)
            signal okToWrite
        release lock
```

```
  // Algorithm for a thread writing to the file
    Writer()
        acquire lock
        while (activeWriters + activeReaders > 0) {
            waitingWriters++
            wait on okToWrite
            waitingWriters--
        }
        activeWriters++
        release lock

        // writing stuff here

        acquire lock
        activeWriters--;
        if (waitingWriters > 0) signal okToWrite
        else if (waitingReaders > 0) broadcast to okToRead
        release lock
```

As the code is given, starvation of readers is possible if writes occur frequently enough. Modify the code so that if a reader waits for 3 writers the reader then gets to go next. Submit your solution in *readwrite.c*.

2. Submit your solution program in the file *faculty.c*

   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. Students of both departments are of course are above this silliness, and therefore do not add any constraints or impact the synchronization algorithm you are about to design.

   This policy is implemented using a sign on the door with a magnet that can be placed on one of 3 options, “Math”, “CS”, or “Avail”. In order to ensure at least some fairness, if members of one group are waiting when the last member of the other group leaves, the waiting group members should immediately be allowed to enter, and in general 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).

   Each individual prof 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.

   Your task is to implement a program in faculty.c with at least the following 4 functions:
   mathProfArrive()
   csProfArrive()
   mathProfLeave()
   csProfLeave()

   - The arrive functions can at first only see the sign on the door and act accordingly. If an arriver is able to enter the room, they can then see only the others inside the room. An arriver not able to enter the room should wait by going to sleep, not busy-waiting (profs can always use a nap).
   - The leave functions are aware of the others in the room at the time they leave, and can see anyone sleeping outside the room as soon as they walk out, so they are allowed to use that information and act accordingly. Even during a feud a prof leaving is willing to wake sleeping profs of the other department if necessary.

   Your final submission should include include ONLY the following print statements:
   - X prof arrives (X = math or cs as appropriate)
   - X prof enters lounge
   - X prof leaves lounge

   This is a relatively big class and I am a slow grader, so I really appreciate your help by following the print instructions =)

You may use any of the synchronization constructs we have learned to solve this problem.

I will of course test your solution with a main function I make where I may create any number of prof threads who arrive (call the appropriate arrive()) in any order and remain in the lounge for any amount of time before leaving (call the appropriate leave()). You should still include a main() function of your own that correctly runs your program with some number of prof threads, so if I have any problems I can at least see a case where the program worked for you…