CS6868 Concurrent Programming

Programming Assignment 4

Due Wednesday, April 3, 2013 @ 11:55 am (NOT pm)

In this assignment, you will use Pthread to simulate three synchronization problems. The main purpose of this assignment is to become familiar with Pthread programming interface and further understand the process (thread) synchronization.

1. (35%) **The one-lane bridge problem.** Cars coming from the north and the south arrive at a one-lane bridge. Cars heading in the same direction can cross the bridge at the same time, but cars heading in opposite directions cannot. And, if there are ever more than 3 cars on the bridge at one time, the bridge will collapse under their weight. In this system, each car is represented by one thread, which executes the procedure OneVehicle when it arrives at the bridge:

   ```
   OneVehicle(int dir) {
       ArriveBridge(dir);
       CrossBridge();
       ExitBridge(dir);
   }
   ```

   In the code above, dir is either 0 (from the north) or 1 (from the south); it gives the direction in which the vehicle will cross the bridge.

   Write the procedures ArriveBridge and ExitBridge (the CrossBridge procedure should just print out a debug message) using mutex and condition variables. ArrivingBridge must not return until it is safe for the car to cross the bridge in the given direction (it must guarantee that there will be no head-on collision). ExitBridge is called to indicate that the caller has finished crossing the bridge; ExitBridge should take steps to let additional cars cross the bridge.

   In the main thread, you need to create 50 child threads, one for each car. The driving direction for each car should be randomly assigned. Make sure that your simulation outputs information that clearly shows that your solution works. The message should indicate car number and **driving direction**. In particular, messages should be printed at the following times:

   - whenever a car arrives at a bridge.
   - whenever a car is crossing the bridge.
   - whenever a car exits from the bridge.

2. (35%) You have been hired by the CSE Department to write code to help synchronize a professor and his/her students during office hours. The professor, of course, wants to take a nap if no students are around to ask questions; if there are students who want to ask questions,

they must synchronize with each other and with the professor so that (i) only one person is speaking at any one time, (ii) each student question is answered by the professor, and (iii) no student asks another question before the professor is done answering the previous one. You are to write four procedures: *AnswerStart(), AnswerDone(), QuestionStart(), and QuestionDone()*. The professor loops through the following sequence of actions: AnswerStart(); give answer; AnswerDone(). AnswerStart doesn't return until a question has been asked. Each student loops through the following: QuestionStart(); ask question; QuestionDone(). QuestionStart() does not return until it is the student's turn to ask a question. Since professors consider it rude for a student not to wait for an answer, QuestionDone() should not return until the professor has finished answering the question. Each student has a unique id.

```
 TheProfessor () {
     while(1) {
          AnswerStart();
          Cout << "The professor is answering the question" <<endl;
          AnswerDone();
     }
}

OneStudent() {
     QuestionStart();
     Cout << StudentNo <<" The student is asking a question" << endl;
     QuestionDone();
}
```

Make sure that your simulation outputs information that clearly shows that your solution works. In particular, messages should be printed at the following times:

- whenever a student is ready to ask a question
- whenever a student starts asking a question
- whenever a student is done asking a question
- whenever the professor wants to be asked a question
- whenever the professor starts an answer
- whenever the professor is finished with an answer

Each student is given a unique id. Any student can ask more than one question: the questions are numbered from 1 for each student. The student id and the question number must be printed clearly for each question that is being asked. Also, to show correctness, the professor should say whose question he/she is answering and the question number of that student. There should be at least 25 students and each one can ask at most 10 questions. The questions could be of different difficulties and therefore the professor may take anywhere from 2 time units to 15 time units to answer all the questions. Always, questions take 1 time unit to ask.

Also, insert a `thr->sched_yield()` as the last line in each of the four functions; this will increase the chance of interleaving and hence test your program more thoroughly. Make your system as fair to students as possible. No student should starve.

3. (35%) 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 and 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 radioing 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 should use pthreads to solve this problem.

   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)?