CMPSC 472
Fall 2017
Homework #4

Due: Thursday, October 26, 2017
80 Points

Use SEMAPHORES not monitors for this assignment!

Homework Information

- This homework is due at the beginning of the class period. Late papers are not accepted.
- This assignment is to be done on your own.
- Please be sure to email copies of your code AND hand in hard copies, one problem per file, printing on one side only. If a problem requires more than one page to print out, please staple the pages together.
- All code should have comments that include your name and the problem number.
- Put all solutions in a pocket folder, in order by problem number. (The folder should be a pocket folder, not an index folder or a folder with clamps or rings.)
- Your name and the course info (CMPSC 472) should be on the outside of the folder in the upper right hand corner.
- Files should be emailed using the `mail472` command. Place all files you want to email in a subdirectory and type `mail472`. You will be prompted on what to do. This command will email ALL files in the subdirectory, so be sure to delete any you do not wish to email (such as .lst and .pco files).
- Files must be named as per the instructions below, with no spaces using all lower case.
- Spaces in the subject line cause problems; you can use double quotes around the subject line if you need spaces.
- **Failure to follow these rules may result in lost points.**

Important Things About BACI and Turning in Your Assignment

- Some versions of BACI don't like long (multiline) comments using the /* */ style. Therefore, you may wish to use the // comment style for long comments.
- In addition, some versions do not like semaphores to be initialized with the when they are declared, so you may wish to use INITIALSEM.
- However you set up comments and initialization, please note: ALL PROGRAMS MUST COMPILE ON THE LINUX MACHINES IN THE SUN LAB AND WILL BE GRADED ON THESE MACHINES. So please test your programs before submitting them.
- Each of the following is an independent BACI program. Unless specified, you can only add code; the code already in the program (if a program is given) must not be changed. Your solution should not unnecessarily restrict concurrency -- your solution will be graded not only on its correct output but also correct placement of the synchronization.
- Before you email your files, name the files using your first initial and last name as specified in the problems. Place all files in a subdirectory and use the `mail472` command to email only the .cm (or .pm) files in the subdirectory (the command will email ALL files so be sure to delete ones that should not be mailed). (This command is located in /usr/local/bin.)
- Spaces in file names will cause problems with the `mail472` command, so do no use spaces in file names. If spaces in the subject header cause an error, simply remove them.
- NOTE: When you run mail472, you should get back an acknowledgement email almost immediately. It will be sent to your Sun lab account (unless you have your Sun lab email forwarded). If you do not get this email, it means the files were not sent.

The Problems

1. **[20] Birds, Birds, Birds**

   Your professor has a farm with lots and lots of different types of birds: chickens, geese, ducks, guineas, and peafowl. She has built a feeding shed for her birds. This shed has one door. This door is wired to a button outside and a button inside. Since poultry fight, she limits the feeding house to no more than 5 birds at a time. All of her poultry are trained to press a button when they wish to enter the feeding shed or when they wish to leave the feeding shed. They are also trained so that only one bird enters or leaves (the bird that pushed the button) when the door opens. If a bird is attempting to enter the feeding shed but it is full, the button won't open the door but the bird knows to wait until the door opens when another bird leaves. Write code to synchronize the birds.

   Write the program in BACI and test it thoroughly.

   You should also print messages when a bird is ready to enter the feeding shed, when it is eating, and when it has left. These messages should include the bird's number. Be careful placing these print statements. Be sure all printing is done atomically (either using semaphores around cout statements or using the keyword atomic for functions that print).

   Start 15 bird processes (do NOT do this in a FOR loop to iterate 15 times -- start 15 actual processes in the cobegin block), where each one runs 10 times (i.e, each bird should loop to eat 10 times). Give each process a unique number so output can be traced. Use a random FOR loop in the processes to simulate the passage of time in the appropriate places (for example, to indicate the passage of time when a bird is not hungry so does not want to get into the feeding shed and when the bird is actually eating). This can be easily done by using a `Delay` function:

   ```
   void Delay (void)
   {
     int i;
     int DelayTime;
     DelayTime = random (DELAY);
     for (i = 0; i < DelayTime; i++):
   }
   ```

   where the constant integer DELAY equals some number from 10 to 100. Then you can simply call Delay to simulate the passage of time.

   The output should be formatted in such a manner that it is easy to read as well. Sample output might be as follows:

   ```
   Bird 1 wishes to enter the feeding shed.
     Bird 2 wishes to enter the feeding shed.
   Bird 1 enters the shed and eats.
       Bird 3 wishes to enter the feeding shed.
         Bird 4 wishes to enter the feeding shed.
     Bird 2 enters the shed and eats.
       Bird 3 enters the shed and eats.
   Bird 1 is done and leaves the shed.
         Bird 4 enters the shed and eats.
   (etc.)
   ```

   Name this file using your first intial, last name, and "bird" (for example jdoebird.cm).

2. **[60] The Grumpy Professor**

Your professor needs your help again. She needs code to synchronize her students and herself during office hours. The professor will sleep (hey, it's a tough job!) if no students are in the office to ask questions. If there are students who want to ask questions, they must synchronize with each other and with the professor according to the rules below.

You are to write four processes: a CMPSC 312 student process, a CMPSC 472 student process, a COMP 519 student process, and a professor process. Your professor can be a little grumpy, so students in undergraduate classes (those in CMPSC 312 and CMPSC 472) don't like to go into the office unless they are in pairs. Since they are asking class questions, they need to be in pairs from the same class. Note that each student is in only one of the 312, 472, or 519 classes. However graduate students in COMP 519 are more comfortable with the grumpy professor so they don't need to ask questions in pairs. When a 519 student arrives, this student might have to wake up the professor or wait on other students, but does not have to wait for a "companion" 519 student to enter the office. In fact, the professor will work with only one 519 student at a time. In addition, the professor will allow only two 312 or 472 students in the office concurrently.

The professor process should sleep until two students from the same undergraduate class (312 or 472) or one student from COMP 519 are waiting to ask questions. The students wake up the professor and ask the question (you may assume if two students go in that they both have the same question). The professor process answers the question, and then either goes back to sleep or answers another question if more students are waiting. Each student will go the professor's office, wait for another student from the same class if necessary, wake up the professor or wait for previous students, ask a question, wait for the answer (it is rude to leave before the professor answers your question!), and then leave. Make sure students follow the rules regarding entering the office, that each question is answered by the professor, and that no student enters the office and asks another question before the professor is done answering the previous one.

Since BACI allows only 18 concurrent processes, you should start one professor, six 312 students, six 472 students, and four 519 students. YOU DO NOT NEED TO LOOP THE STUDENT PROCESSES. However, be warned that the professor process must loop enough times to answer all questions. Be sure to put the loop in the professor process, NOT in the main program.

Regarding priority: there is no priority among students. Try to make your solution as fair as possible.**You do NOT have to worry about the fact that BACI implements weak semaphores!** So, if some students jump in front of others that are waiting in the same queue, that's ok. For example, if four 312 students are waiting, it's ok if the last two that arrived go first. However, aside from weak semaphores, the solution should be as fair as possible. For example, if a 312 student arrives, followed by a 472 student, and then a 519 student, the 519 student can go ask a question (no sense having the professor sleep when there is a "viable" student waiting). If a 472 student then arrives, the two 472 students would ask a question. While they are asking a question, if another 519 student arrives, followed by a 312 student, it would be the most fair to allow the 312 pair of students to see the professor (as one of them has been waiting for some time) when the 519 student is done. This is a difficult synchronization issue, but let's see what you an do. Regardless of how you deal with fairness, be sure your code does not starve out any particular group of students. We are only running a few students; however, if your code were to run an infinite number of student processes, there should be no possibility of starvation for a particular group of students.

Use random FOR loops (as in Problem 1) in appropriate places (for example, when the professor is answering questions and when the students are heading to the office). Provide print statements so it is obvious from looking at the output what is going on. For example, when the professor is answering a question, please print which students are listening. You do not need to make up specific questions and answers. Please print at all important synchronization points. You should NOT simply print "Student is asking" followed by "Student is listening" followed by "Student is leaving". The student cannot ask until he/she enters the office; a student cannot listen until the professor speaks; the student cannot leave until the professor is done. These are all important synchronization points that must be addressed. Therefore, after a student prints "Student is asking", the professor should print "Professor is answering", then the

student can print "Student is listening", etc. Please identify your students in the print statements so tracing your code will be easier. The easiest way to id students is using 312Student1, 519Student1, etc.

Name your file using your first initial and last name and "prof" (for example, jdoeprof.cm).