COMSC 322
Operating Systems
Programming Assignment 2: Process Synchronization
Due: November 1st, 9:00 PM

Note: Please do not put this off to the end!

1. Thirsty Persons Problem: Consider a system with three thirsty threads and one server thread. To drink water, each thread needs three ingredients: water, ice, and a cup. One of the threads has water, another has ice, and the third has a cup. The server has an infinite supply of all three. The server places two of the ingredients (chosen randomly) on the table. The thread that has the remaining ingredient then prepares and has a drink, signalling the server on completion. The server then puts out another two of the three ingredients (at random), and the cycle repeats.

A) To make your task easier, I am providing a skeleton of solution for this problem below. Before proceeding to code, complete this pseudocode. (10)
B) Write a **semaphore** based solution in Java to synchronize the server and the thirsty threads. (20)

```
semaphore a[3] = 0; /* a[0] for water, a[1] for ice, a[2] for cup */
semaphore server = 1;

Server(void) {
int i,j;
        While(TRUE){
                i = random(3); /* returns a random integer 0, 1 or 2 for i */
                j = random(3); /* returns a random integer 0, 1 or 2 for j */
                        if (i != j) { /* i and j must be different */
                                Wait(_________);
                                k = 3 - (i+j);  //the drinker with the k-th ingredient is identified
                                Signal(_________);
                        }
        }//end of while
}//end of Server

Drinker(int r) { /* r indicates which ingredients this drinker has */
        While(TRUE){
                        Wait(_________);
                        Drink( );
                        Signal(_________);
        }
}//end of Drinker
```

2. Lazy Dentist Problem: There is a dental clinic with N chairs for waiting patients; the clinic has one doctor. If a patient enters the clinic and there are no free chairs, the patient leaves. If a patient enters the clinic and the dentist is sleeping, the patient wakes up the dentist and consults him. Otherwise, the patient enters the clinic, takes a seat, and waits. If the dentist finishes with one patient and there are waiting patients, the dentist takes the next patient. Otherwise, the dentist goes to sleep in his chair.

A) Below, I am providing a skeleton of solution for this problem. Before proceeding to code, complete this pseudocode. And to make your task (very) simple I have provided the comments corresponding to each of the statements. Use these as hints for filling in the blanks! (10)

B) Write a **semaphore** based solution in Java to control the actions of patients and the dentist. (20)

```
// The first two are semaphores are mutexes (only 0 or 1 possible)
Semaphore dentistReady = 0
Semaphore seatCountWriteAccess = 1     // if 1, the number of seats in the waiting room can be
                                       // incremented or decremented
Semaphore patientReady = 0         // the number of patients  currently in the waiting room, ready to be
                                   //served
int numberOfFreeWRSeats = N     // total number of seats in the waiting room
```

```
def Dentist():
  while true:                // Run in an infinite loop.
        wait(_____________)          // Try to acquire a patient - if none is available, go to sleep.
        wait(_____________)       // Awake - try to get access to modify # of available seats,
                                        //otherwise sleep.
        numberOfFreeWRSeats += 1    // One waiting room chair becomes free.
        signal(_____________)        // I am ready to consult.
        signal(_____________)       // Don't need the lock on the chairs anymore.
       // (Talk to patient here.)
```

```
def Customer():
  while true:                // Run in an infinite loop to simulate multiple patients.
        wait(_____________)        // Try to get access to the waiting room chairs.
        if numberOfFreeWRSeats > 0: // If there are any free seats:
              numberOfFreeWRSeats -= 1  //   sit down in a chair
              signal(_____________)        //   notify the dentist, who's waiting until there is a patient
              signal(_____________)     //   don't need to lock the chairs anymore
              wait(_____________)        //   wait until the dentist is ready
              // (Consult dentist here.)
        else:                    // otherwise, there are no free seats; tough luck --
              signal(_____________)     //   but don't forget to release the lock on the seats!
              // (Leave without consulting the dentist.)
```

Testing your code: Add a main method to test your code. Assume N = 3.

A) Test with one dentist thread and one patient thread.
B) Test with one dentist thread and two patient threads.
C) Test with one dentist thread and three patient threads.
D) Test with one dentist thread and five patient threads.

---

3. PeaceLover and PeaceBuff are two neighboring countries battling for the disputed land of PeaceShire. To win the war, PeaceLover's army has inducted several spies in the army of PeaceBuff. These spies employ the following protocol for sending secret information to their headquarters: A spy writes the message on a piece of paper and drops it at a predetermined location. An agent from PeaceLover then picks up the paper and delivers it to the headquarters. A spy will drop a new message only if the previous message has already been picked up. If not, the spy will wait until the message is collected by an agent. Obviously, an agent cannot pick up the message unless a spy drops one. To avoid interpretation (on discovery) by PeaceBuff's army, the spies write only **one word** messages.

Write an Espionage class to simulate the above communication protocol. This class has two methods: dropSpyMsg(String message) and agentPickMsg(). "Message" is the one word secret message. In your implementation you can assume that dropSpyMsg () waits until agentPickMsg () is called on the same Espionage object, and then transfers the word over to agentPickMsg (). Once the transfer is made, both can return. Similarly, agentPickMsg () waits until dropSpyMsg () is called, at which point the transfer is made, and both can return (agentPickMsg () returns the word).

Write your Espionage class as a **monitor**. Your solution should work even if there are multiple spies and agents for the same Espionage object. As we discussed in the class, you would not need any locks besides what the monitor already provides. (25)

Hint: This is equivalent to a zero-length bounded buffer. Since the buffer has no room, the producer and consumer must interact directly, requiring that they wait for one another.

Testing your code: Add a main method to test your code.

A) Test with a single spy thread and a single agent thread.
B) Test with multiple spies and multiple agent threads.

---

4. Prepare a "README.txt" file for your submission. The file should contain the following: (10)

a) Names of all the group members.
b) Instructions for compiling and executing your program(s). Include an example command line for the same.

c) If your implementation does not work, you should also document the problems in the README file, preferably with your explanation of why it does not work and how you would solve it if you had more time.

5. You should also comment your code well. The best way to go about it is to write comments while coding. (5)

What should you submit?

Email your assignment as a zip file to amishra@mtholyoke.edu; copy all your teammates in this email. The zip file should contain:

1. All your (java) code files and any other files that might be needed for executing your code.
2. README.txt.
3. A pdf file (report) containing answers to the above pseudocode questions.