Lab 7: Semaphore Solutions to the Bounded Buffer and Readers-and-Writers Problems

Purpose:

In this lab you will be provided with two Semaphore classes: BSemaphore defines a binary semaphore and CSemaphore defines a counting (general) semaphore. In the first part of the lab you will use these semaphore classes to provide the synchronization necessary for the Bounded Buffer Problem. Then you will code the semaphore solution to the Readers and Writers Problem.

Hand in:

The execution results and discussion from questions 1 - 5. A listing of `BoundedBuffer.java` requested in Step 2. A listing of the four files requested at the end of Step 4. E-mail copies of these four files to your instructor as well. A listing of the file `RWDriver.java` requested in Step 5.

1. Start with the BoundedBuffer class from Lab 6 along with the Producer and Consumer classes and the main class PCTest.

   Modify the BoundedBuffer class by removing the synchronization, viz., *synchronized*, *wait()*, *notifyAll()*, and the boolean variables *full* and *empty*.

   Modify the program PCTest to create a BoundedBuffer of size 4, create one Producer thread that deposits 10 random values into the buffer, and create one Consumer thread that fetches 10 values from the buffer.

   After making these modifications run the program and record your results. You should be in the same situation you found yourself at the end of Step 1 of Lab 6 (prior to adding any synchronization).

   Now you will use semaphores to enforce the correct interaction. Add the following files to your project: `BSemaphore.java` and `CSemaphore.java`.

   The Semaphore class in `BSemaphore.java` defines a *binary semaphore* since the value is restricted to 0 and 1. The Semaphore class in `CSemaphore.java` defines a *counting semaphore* that can take any value between 0 and N-1 for some value N.

Now recall the semaphore solution to the Producer/Consumer problem that was given in class. (It can also be seen as Figure 5.16 on p. 228 of our textbook.) This solution uses a binary semaphore (*mutex*) for the critical sections and two counting semaphores: one for blocking the Producer when the buffer is full (which I called *emptySlots*) and one for blocking the Consumer when the buffer is empty (which I called *fullSlots*). Implement this solution to the Producer/Consumer problem by adding the binary semaphore and the two counting semaphores to the BoundedBuffer class and using them as described in the solution in class. Don't forget to properly initialize the semaphores.

Run the program and record your results. Are the results correct? Explain.

Print a copy of `BoundedBuffer.java` and turn it in with your lab.

2. Modify the program to create a buffer of size 3, create two Producers that each deposit 6 random values into the buffer, and create three Consumers that each fetch 4 items from the buffer.

   Run the program and record your results. Are the results correct? Explain.

Background for the Readers-and-Writers Problem

The *Readers-and-Writers Problem* was recently introduced in class along with a solution that uses semaphores.

Recall that in the Readers-and-Writers Problem there are two classes of processes (i.e., threads): Readers and Writers that require access to a shared database. The problem is to control the access in such a manner as to allow any number of Readers to access the database at the same time but whenever a Writer accesses the database to guarantee that the Writer has exclusive access to the database.

I have provided a Reader class and a Writer class along with a driver program RWDriver. A Reader thread maintains an id, a start time, and a duration which is the length of time it reads the database. Similarly a Writer thread maintains an id, a start time, and a duration which is the length of time it writes to the database. Both Reader and Writer threads use the sleep function to delay starting and delay during read/write access. The threads also display messages at strategic locations within the code that indicate where they are during execution. This enables one to trace the execution.

To start this lab you will need the following files:

RWDriver.java - the driver program for the simulation of the Readers and Writers
Reader.java - defines the behavior of a Reader thread
Writer.java - defines the behavior of a Writer thread

Instructions:

3. Create a new project with the files RWDriver.java, Reader.java, and ~~Writer.java.~~

   Execute the program and print the results of the execution.

   In what order do the threads access the database? Is there any point at which the Writer does not have exclusive access to the database? Explain, referring to the results of the execution.

4. Design and code a Database class that controls access to the database. In addition to the constructor, the class should provide four methods: *startRead()*, *endRead()*, *startWrite()*, and *endWrite()*. The methods are to implement the semaphore solution to the Readers-andWriters problem that was presented in class and which can be found in Figure 5.25 on p. 243 of our textbook. The Database will need to manage the semaphores needed as well as the number of readers.

   Modify the behaviors of the Reader and Writer threads to call these methods at the appropriate places. To make all of this work, have the driver program create the Database object then pass it to the Reader and Writer threads when they are created. (Which means that you will need to modify their constructors.) Print an execution of the modified program.

   In what order do the threads access the database? Is there any point at which the Writer does not have exclusive access to the database? Explain, referring to the results of the execution.

   Hand in appropriately documented copies of the files `RWDriver.java`, `Reader.java`, `Writer.java`, and `Database.java`. E-mail these files to your instructor as well.

5. Modify the driver program `RWDriver.java` to generate five Readers and three Writers as follows:

Thread	Starts	Duration
R_1	0	40
R_2	10	10
W_1	20	50
R_3	30	60
W_2	50	20
R_4	60	50

R_5	80	50

Produce two executions, one without any access control (without calls to startRead, endRead, startWrite, endWrite), and one with the access control. Discuss the correctness of the two executions referring to the results.

Hand in a final copy of the file `RWDriver.java`.