**CSC 460 Program #1**

**Second Reader's Writer's Solution**

Your first program will require you to code the given solution to the Reader's Writer's problem. This particular solution is also known as the "Writer's Preference" solution as it ensures that, once a writer arrives, it will wait no longer than necessary before being allowed access to its critical section (i.e. it is allowed to perform its write operation).

Please be sure that you understand the nature of the Reader's Writer's Problem before you even attempt to implement this solution.

Your ***Java program*** will implement 4 classes – **Reader**, **Writer**, **Shared** and a class containing your main method named **Prog1**.

**Shared Class**

The Shared class is a simple class which has two private int data members named ***readCount*** and ***writeCount***. The constructor for this class should take two integer arguments and assign the first into ***readCount*** and the second into ***writeCount***.

The class will also contain the following 6 methods.

**int getWriteCount**() – returns the current value of writeCount.

**int getReadCount**() – returns the current value of readCount.

**void** **incReadCount**() – increments the readCount value.

**void incWriteCount**() – increments the writeCount value.

**void decReadCount()** – decrements the readCount value.

**void decWriteCount()** – decrements the writeCount value.

**Prog1 Class**

The Prog1 class will be responsible for declaring the following Semaphore variables.

Semaphore rmutex, wmutex, M1, M2, M3; //(all initialized to 1 in main( ))

You should use appropriate Java syntax to create these 5 FAIR (FIFO) Semaphore objects all of which are initialized to 1, within the main() method.

The main method will then enter a loop which will iterate 100 times. Each time through the loop the main thread will choose to create either a Reader object or a Writer object ***with Readers being twice as likely to be randomly selected.*** I would suggest that your program declare a Random object (Be sure to import java.util.Random) and simply execute ***.nextInt(3)*** on the Random object. If the result is 0 then you will instantiate a new Writer object/thread otherwise you will generate a new Reader object/thread. Be sure to pass all of the appropriate arguments to the constructor for the new threaded object (see constructors for Reader and Writer defined later in this document). Be sure to create each object/thread correctly (i.e. ***the Writer class extends Thread while the Reader class implements Runnable*** so the precise way to create each is slightly different – see the slides if you are not clear on this!).

The new thread should then be started. Once the new Reader or Writer has been started, the main thread should Sleep() for a Random period of time between 0 and 30 milliseconds. Simply use the Random object to generate a nextInt(31) value and pass it to Thread.sleep( ). Once awakened from this sleep, the main thread should iterate again through the loop body.

Once the loop is finished, the main should execute Thread.sleep(2000) to delay two seconds before ending execution. Print a message prior to ending the main (see output examples).

**Reader Class**

The Reader class is a threaded class and will **implement the Runnable interface** as discussed in the slides. The class should contain a constructor which expects to receive 5 arguments. The arguments passed to the constructor will be a **Shared** object and 4 **Semaphores**. The Shared will be the static Shared object created by the main() method in the Prog1 class. The 4 Semaphores received will be the **rmutex**, M1 M2, and M3 Semaphores also defined in the Prog1 class. Make sure that your parameters match in type and order to the actual arguments passed by the main( ) method. Store them into static variables defined in the class.

The Reader class will contain a **void run( )** method required of any threaded class. This run( ) method should implement the algorithm shown below. PLEASE NOTE that this is NOT Java code but is rather a detailed pseudocode algorithm. You are expected to review the materials/slides for the course and convert it into equivalent Java source code.

***//READER***

Reader pseudo code() {

**<ENTRY Section>**

M3.wait(); //Indicate a reader is trying to enter

M1.wait()

rmutex.wait(); //lock entry section to avoid

//race condition with other readers

*Shared\_Obj.incReadCount()* //report yourself as a reader

if (shared\_Obj.getReadCount() == 1) //checks if you are first reader

M2.wait(); //if you are first reader, lock the resource

rmutex.signal(); //release entry section for other readers

M1.signal(); //indicate you are done trying to access the resource

M3.signal();

**<CRITICAL Section>**

//reading is performed HERE

Sleep( random milliseconds from 0 to ***20***) // simulate the operation

**<EXIT Section>**

rmutex.wait(); //reserve exit section - avoids race condition with readers

Shared\_Obj.decReadCount() //indicate you're leaving

if (shared\_obj.getReadCount() == 0) //checks if you are last reader leaving

M2.signal(); //if last, you must release the locked resource

rmutex.signal(); //release exit section for other readers

}

Obviously, you will need to place appropriate output messages to match my output in all of your classes.

**Writer Class**

The Writer class is also a threaded class except that ***this class is to be implemented by extending Java's Thread class***. Again, be sure that you have reviewed the course materials to ensure that you understand the two ways to create a threaded class (e.g. extending the Thread class or implementing the Runnable interface).

The Writer class will implement a constructor. The constructor should expect to receive 5 arguments. The arguments will be the Shared object passed from the main() method and 4 Semaphores. The 4 Semaphores will be the **wmutex** semaphore, the M1, M2 and M3 semaphores, all of which are defined in and passed from the Prog1 class.

Once again, I cannot stress enough that the following code is a detailed form of pseudocode and is not complete or correct Java code. It is your responsibility to translate this algorithm into appropriate, working Java source code.

***//WRITER***

writer() {

**<ENTRY Section>**

wmutex.wait(); //reserve entry section for writers –

// avoids race conditions

Shared\_obj.incWriteCount() //report yourself as a writer entering

if (Shared\_obj.getWriteCount() == 1) //checks if you're first writer

M1.wait(); //if you're first, then you must lock the

// readers out. Prevent them from trying to

// enter CS

wmutex.signal(); //release entry section

M2.wait(); //reserve the resource for yourself –

//prevents other writers from simultaneously

// editing the shared resource

**<CRITICAL Section>**

//writing is performed

Sleep(Random milliseconds from 0-***30***) // simulates write operation

M2.signal(); //release resource

**<EXIT Section>**

wmutex.wait(); //reserve exit section

Shared\_obj.decWriteCount() //indicate you're leaving

if (Shared\_obj.getWriteCount() == 0) //checks if you're the last writer

M1.signal(); //if you're last writer, you must unlock

// the readers. Allows them to try enter CS

// for reading

wmutex.signal(); //release exit section

}

**Submitting your Assignment**

You should only submit your source code files (i.e. .java files). Please do not submit any other files just the Prog1.java, Writer.java, Reader.java, Shared.java source files. Follow these instructions exactly please.

1. Create a new folder on your machine.

2. Name the folder lastname\_firstname\_prog1 (e.g. newell\_gary\_prog1).

3. Place the files Prog1.java, Reader.java, Writer.java, Shared.java into the folder.

4. Optionally, place a README file (word doc or textfile etc.) into the folder. This file should describe any known problems or issues with your code .

5. Zip/compress the folder.

6. Upload the resulting zipped file/folder using the link on Canvas.

7. Confirm that your file was uploaded successfully.

**IMPORTANT INFORMATION**

a) There are slight differences on how you should access certain Thread methods depending upon whether you are implementing Runnable or extending the thread class!

When extending the Thread class you can access a thread's name by simply using

***this.getName()***

and can call the sleep() method simply by executing

***sleep( time\_to\_sleep)***

where time\_to\_sleep is a long value indicating the number of milliseconds the thread should sleep.

If your class implements the Runnable interface, then to access the thread's name you will need to use

***Thread.currentThread( ).getName()***

and to sleep the thread you will need to use syntax such as

***Thread.sleep(time\_to\_sleep)***

b) You will need to make sure that the code inside the void run( ) methods of both the Reader and Writer classes is within a try-catch clause. This is required in case of interrupts. Any call to sleep() must also be protected within a try-catch.

The following pages contain a sample execution of the program. NOTE that due to the random generation of sleep times and of Writer and Reader threads your execution will be different from mine. You should look through your output, however, to confirm if the program is working properly.

**SAMPLE EXECUTION – Do NOT expect to ever match this exact output!!!**

Creating new READER Thread-0

READER Thread-0 is ENTERING READER CODE

READER Thread-0 is in CRITICAL SECTION performing READ

READER Thread-0 is reading for 9 milliseconds.

Creating new WRITER Thread-1

WRITER Thread-1 is ENTERING WRITER CODE

WRITER Thread-1 is now in Critical Section performing WRITE

READER Thread-0 is now EXITING

WRITER Thread-1 is writing for 19 milliseconds.

Creating new READER Thread-2

READER Thread-2 is ENTERING READER CODE

Creating new READER Thread-3

READER Thread-2 is in CRITICAL SECTION performing READ

READER Thread-2 is reading for 11 milliseconds.

READER Thread-3 is ENTERING READER CODE

WRITER Thread-1 is now exiting

READER Thread-3 is in CRITICAL SECTION performing READ

READER Thread-3 is reading for 13 milliseconds.

READER Thread-2 is now EXITING

READER Thread-3 is now EXITING

Creating new READER Thread-4

READER Thread-4 is ENTERING READER CODE

READER Thread-4 is in CRITICAL SECTION performing READ

READER Thread-4 is reading for 17 milliseconds.

Creating new WRITER Thread-5

WRITER Thread-5 is ENTERING WRITER CODE

READER Thread-4 is now EXITING

WRITER Thread-5 is now in Critical Section performing WRITE

WRITER Thread-5 is writing for 16 milliseconds.

WRITER Thread-5 is now exiting

Creating new WRITER Thread-6

WRITER Thread-6 is ENTERING WRITER CODE

WRITER Thread-6 is now in Critical Section performing WRITE

WRITER Thread-6 is writing for 12 milliseconds.

WRITER Thread-6 is now exiting

Creating new WRITER Thread-7

WRITER Thread-7 is ENTERING WRITER CODE

WRITER Thread-7 is now in Critical Section performing WRITE

WRITER Thread-7 is writing for 12 milliseconds.

Creating new READER Thread-8

READER Thread-8 is ENTERING READER CODE

WRITER Thread-7 is now exiting

READER Thread-8 is in CRITICAL SECTION performing READ

READER Thread-8 is reading for 17 milliseconds.

Creating new WRITER Thread-9

WRITER Thread-9 is ENTERING WRITER CODE

READER Thread-8 is now EXITING

WRITER Thread-9 is now in Critical Section performing WRITE

WRITER Thread-9 is writing for 16 milliseconds.

Creating new WRITER Thread-10

WRITER Thread-10 is ENTERING WRITER CODE

Creating new WRITER Thread-11

WRITER Thread-11 is ENTERING WRITER CODE

WRITER Thread-9 is now exiting

WRITER Thread-10 is now in Critical Section performing WRITE

WRITER Thread-10 is writing for 3 milliseconds.

Creating new READER Thread-12

READER Thread-12 is ENTERING READER CODE

WRITER Thread-10 is now exiting

WRITER Thread-11 is now in Critical Section performing WRITE

WRITER Thread-11 is writing for 4 milliseconds.

WRITER Thread-11 is now exiting

READER Thread-12 is in CRITICAL SECTION performing READ

READER Thread-12 is reading for 17 milliseconds.

READER Thread-12 is now EXITING

Creating new READER Thread-13

READER Thread-13 is ENTERING READER CODE

READER Thread-13 is in CRITICAL SECTION performing READ

READER Thread-13 is reading for 19 milliseconds.

READER Thread-13 is now EXITING

Creating new WRITER Thread-14

WRITER Thread-14 is ENTERING WRITER CODE

WRITER Thread-14 is now in Critical Section performing WRITE

WRITER Thread-14 is writing for 30 milliseconds.

Creating new WRITER Thread-15

WRITER Thread-15 is ENTERING WRITER CODE

Creating new READER Thread-16

READER Thread-16 is ENTERING READER CODE

Creating new READER Thread-17

READER Thread-17 is ENTERING READER CODE

Creating new READER Thread-18

READER Thread-18 is ENTERING READER CODE

WRITER Thread-14 is now exiting

WRITER Thread-15 is now in Critical Section performing WRITE

WRITER Thread-15 is writing for 3 milliseconds.

WRITER Thread-15 is now exiting

READER Thread-16 is in CRITICAL SECTION performing READ

READER Thread-18 is in CRITICAL SECTION performing READ

READER Thread-18 is reading for 1 milliseconds.

READER Thread-17 is in CRITICAL SECTION performing READ

READER Thread-17 is reading for 0 milliseconds.

READER Thread-16 is reading for 9 milliseconds.

READER Thread-17 is now EXITING

READER Thread-18 is now EXITING

READER Thread-16 is now EXITING

Creating new READER Thread-19

READER Thread-19 is ENTERING READER CODE

READER Thread-19 is in CRITICAL SECTION performing READ

READER Thread-19 is reading for 4 milliseconds.

READER Thread-19 is now EXITING

Creating new READER Thread-20

READER Thread-20 is ENTERING READER CODE

READER Thread-20 is in CRITICAL SECTION performing READ

READER Thread-20 is reading for 19 milliseconds.

Creating new READER Thread-21

READER Thread-21 is ENTERING READER CODE

READER Thread-21 is in CRITICAL SECTION performing READ

READER Thread-21 is reading for 6 milliseconds.

READER Thread-21 is now EXITING

READER Thread-20 is now EXITING

Creating new READER Thread-22

READER Thread-22 is ENTERING READER CODE

READER Thread-22 is in CRITICAL SECTION performing READ

READER Thread-22 is reading for 4 milliseconds.

READER Thread-22 is now EXITING

Creating new WRITER Thread-23

WRITER Thread-23 is ENTERING WRITER CODE

WRITER Thread-23 is now in Critical Section performing WRITE

WRITER Thread-23 is writing for 30 milliseconds.

Creating new READER Thread-24

READER Thread-24 is ENTERING READER CODE

WRITER Thread-23 is now exiting

READER Thread-24 is in CRITICAL SECTION performing READ

READER Thread-24 is reading for 11 milliseconds.

READER Thread-24 is now EXITING

Creating new READER Thread-25

READER Thread-25 is ENTERING READER CODE

READER Thread-25 is in CRITICAL SECTION performing READ

READER Thread-25 is reading for 7 milliseconds.

READER Thread-25 is now EXITING

Creating new READER Thread-26

READER Thread-26 is ENTERING READER CODE

READER Thread-26 is in CRITICAL SECTION performing READ

READER Thread-26 is reading for 5 milliseconds.

Creating new WRITER Thread-27

WRITER Thread-27 is ENTERING WRITER CODE

READER Thread-26 is now EXITING

WRITER Thread-27 is now in Critical Section performing WRITE

WRITER Thread-27 is writing for 25 milliseconds.

Creating new WRITER Thread-28

WRITER Thread-28 is ENTERING WRITER CODE

WRITER Thread-27 is now exiting

WRITER Thread-28 is now in Critical Section performing WRITE

WRITER Thread-28 is writing for 18 milliseconds.

Creating new READER Thread-29

READER Thread-29 is ENTERING READER CODE

Creating new READER Thread-30

WRITER Thread-28 is now exiting

READER Thread-29 is in CRITICAL SECTION performing READ

READER Thread-29 is reading for 7 milliseconds.

READER Thread-30 is ENTERING READER CODE

READER Thread-30 is in CRITICAL SECTION performing READ

READER Thread-30 is reading for 4 milliseconds.

READER Thread-30 is now EXITING

READER Thread-29 is now EXITING

Creating new READER Thread-31

READER Thread-31 is ENTERING READER CODE

READER Thread-31 is in CRITICAL SECTION performing READ

READER Thread-31 is reading for 14 milliseconds.

READER Thread-31 is now EXITING

Creating new READER Thread-32

READER Thread-32 is ENTERING READER CODE

READER Thread-32 is in CRITICAL SECTION performing READ

READER Thread-32 is reading for 4 milliseconds.

READER Thread-32 is now EXITING

Creating new READER Thread-33

READER Thread-33 is ENTERING READER CODE

READER Thread-33 is in CRITICAL SECTION performing READ

READER Thread-33 is reading for 4 milliseconds.

READER Thread-33 is now EXITING

Creating new WRITER Thread-34

WRITER Thread-34 is ENTERING WRITER CODE

WRITER Thread-34 is now in Critical Section performing WRITE

WRITER Thread-34 is writing for 7 milliseconds.

WRITER Thread-34 is now exiting

Creating new WRITER Thread-35

WRITER Thread-35 is ENTERING WRITER CODE

WRITER Thread-35 is now in Critical Section performing WRITE

WRITER Thread-35 is writing for 7 milliseconds.

WRITER Thread-35 is now exiting

Creating new READER Thread-36

READER Thread-36 is ENTERING READER CODE

READER Thread-36 is in CRITICAL SECTION performing READ

READER Thread-36 is reading for 20 milliseconds.

Creating new WRITER Thread-37

WRITER Thread-37 is ENTERING WRITER CODE

READER Thread-36 is now EXITING

WRITER Thread-37 is now in Critical Section performing WRITE

WRITER Thread-37 is writing for 27 milliseconds.

WRITER Thread-37 is now exiting

Creating new WRITER Thread-38

WRITER Thread-38 is ENTERING WRITER CODE

WRITER Thread-38 is now in Critical Section performing WRITE

WRITER Thread-38 is writing for 20 milliseconds.

Creating new WRITER Thread-39

WRITER Thread-39 is ENTERING WRITER CODE

WRITER Thread-38 is now exiting

WRITER Thread-39 is now in Critical Section performing WRITE

WRITER Thread-39 is writing for 4 milliseconds.

WRITER Thread-39 is now exiting

Creating new READER Thread-40

READER Thread-40 is ENTERING READER CODE

READER Thread-40 is in CRITICAL SECTION performing READ

READER Thread-40 is reading for 20 milliseconds.

Creating new WRITER Thread-41

WRITER Thread-41 is ENTERING WRITER CODE

Creating new WRITER Thread-42

WRITER Thread-42 is ENTERING WRITER CODE

READER Thread-40 is now EXITING

WRITER Thread-41 is now in Critical Section performing WRITE

WRITER Thread-41 is writing for 25 milliseconds.

Creating new READER Thread-43

READER Thread-43 is ENTERING READER CODE

WRITER Thread-41 is now exiting

WRITER Thread-42 is now in Critical Section performing WRITE

Creating new READER Thread-44

WRITER Thread-42 is writing for 13 milliseconds.

READER Thread-44 is ENTERING READER CODE

Creating new WRITER Thread-45

WRITER Thread-42 is now exiting

WRITER Thread-45 is ENTERING WRITER CODE

READER Thread-43 is in CRITICAL SECTION performing READ

READER Thread-43 is reading for 0 milliseconds.

READER Thread-44 is in CRITICAL SECTION performing READ

READER Thread-44 is reading for 20 milliseconds.

READER Thread-43 is now EXITING

Creating new READER Thread-46

READER Thread-46 is ENTERING READER CODE

READER Thread-44 is now EXITING

WRITER Thread-45 is now in Critical Section performing WRITE

WRITER Thread-45 is writing for 16 milliseconds.

Creating new WRITER Thread-47

WRITER Thread-47 is ENTERING WRITER CODE

WRITER Thread-45 is now exiting

WRITER Thread-47 is now in Critical Section performing WRITE

WRITER Thread-47 is writing for 23 milliseconds.

Creating new WRITER Thread-48

WRITER Thread-48 is ENTERING WRITER CODE

Creating new READER Thread-49

READER Thread-49 is ENTERING READER CODE

WRITER Thread-47 is now exiting

WRITER Thread-48 is now in Critical Section performing WRITE

WRITER Thread-48 is writing for 4 milliseconds.

WRITER Thread-48 is now exiting

READER Thread-46 is in CRITICAL SECTION performing READ

READER Thread-46 is reading for 0 milliseconds.

READER Thread-49 is in CRITICAL SECTION performing READ

READER Thread-49 is reading for 4 milliseconds.

READER Thread-46 is now EXITING

READER Thread-49 is now EXITING

####### MAIN THREAD ENDING NOW!