**CSC 460/560 Operating Systems**

**Program #3**

**Semaphores and Reader’s/Writers Problem**

Your task is to implement the **First** Readers Writers Problem using Java’s Semaphore Class. This class is described in the latest set of notes and will be/was discussed in class.

# Reader’s / Writer’s Solution – ***NON-Java Version***

### SHARED DATA (you should be able to code all of this in Java!!!!)

Int readcount = writecount = 0; // two static vars shared among all threads/classes.

SEM wmutex = rmutex = 1; // semaphores to gain mutual exclusion on counters

SEM wantin = 1; // writers waiting to get in

SEM writerswaiting = 0; // readers block here if writers are waiting

**READER PROCESS:**

Do {

Wait( rmutex); //AKA rmutex.acquire( );

readcount++;

Signal( rmutex); // AKA rmutex.release( );

Wait( wmutex ); // You know the drill…. wmutex.acquire( );

If (writecount > 0 ) // writer active or waiting

{

signal(wmutex) ; // release mutual exclusion with .release( ) so that we do not

// block any future arriving writers

wait( writerswaiting ); // writers have preference so we wait with .acquire()

}

else // no writer in or waiting

{

signal( wmutex) ; // release mutual exclusion with .release( );

}

## PERFORM READ here

Wait( rmutex ); // get mutual exclusion to update readcount as we are leaving

Wait( wmutex); // need to access BOTH counters so get mutual exclusion on both

readcount--; // one less reader now

if ( (readcount == 0 ) && (writecount > 0 ) ) // this is last reader to leave

{

signal( wantin ); // let waiting writer in and release mutual exclusion

signal( rmutex);

signal ( wmutex);

}

else { // either we’re not last active reader or no writers want in….

signal(rmutex);

signal(wmutex);

}

} Forever;

**WRITER PROCESS**

Do {

Wait( wmutex); // get mutual exclusion for write count

writercount ++;

Signal( wmutex);

If ( writercount > 1 || readcount > 0 )

{

wait( wantin );

}

## PERFORM WRITE here

Wait ( wmutex);

writercount--; // writer is done

If (writecount > 0 )

{

signal( wantin ); // let in the next waiting writer

}

else {

wait(rmutex);

while ( readcount > 0 )

{

signal( writerswaiting); // let reader(s) in

readcount--;

} // end while

} // end else

signal( wmutex); // readers are in now (if there were any)

signal( rmutex); // we are done clean up and leave

} forever;

I want you to simulate the reader’s writer’s problem in java ***using the java semaphore class***. You **MUST** use the Semaphore class. No other solutions will be accepted. If you are not familiar with it then try **READING** – it is an amazing solution to ignorance.

As an additional aspect of the program I also want you to code your own “driver”. That is, create Reader threads, and Writer threads, and have them approach the shared resource (which is a hypothetical resource!!!) at random times. YOU will need to experiment with the timing to see how it impacts the problem. I do not wish to see an execution where all readers or all writers execute followed by the other. I wish to see the solution play out as intended where control does periodically shift between them. Please keep the simulation to no more than 3 minutes of execution though a much shorter simulation will be appreciated.

**CSC 560 additional requirements**

Alter your program so that each thread is exactly the same and chooses its behavior (reader or writer) at random with readers having a 75% probability and each writer 25% and have each process execute through 10 iterations. Create 50 threads in total.