CptS 370
FAQ on Program 4: Synchronization

Instructor: Jeremy E. Thompson

Q1: I saw QueueNode.class in the ThreadOS directory. What's that?

A1: This class is used to instantiate synchronized objects that are maintained by SyncQueue's private array. This is a part of the solution.

SyncQueue's private array maintains objects instantiated from *QueueNode.java*. Each *QueueNode* object corresponds to a different condition number and provides two synchronized methods, one to put the calling thread to sleep on this condition and the other to wake up a thread sleeping on this condition. More specifically, *SyncQueue.java*'s *enqueueAndSleep(int condition)* searches its private array for the *QueueNode* object corresponding to this condition, and calls this *QueueNode*'s method that puts the current thread to sleep inside this object. On the other hand, *SyncQueue.java*'s *dequeueAndWakeup(int condition, int tid)* searches its private array for the *QueueNode* object corresponding to this condition, and calls another method of *QueueNode* that wakes up a thread sleeping in this object.

Q2: May I use QueueNode.class?

A2: No. You cannot use a part of the answer key.

Q3: Are we supposed to build upon previous assignments?

For example, are we supposed to include the Shell.java and Scheduler.java programs we did for previous assignments? If so which Scheduler are we supposed to use?

A3: No, we aren't

You should use the original Shell.class and Scheduler.class coded in ThreadOS.

Q4: How should I test SyncQueue.java? How should I run my Test2/Test3 on my Shell.java?

In the previous assignments, we have typed in "java Boot" and then at the prompt "-->" we have typed in the name of the file to run. In this assignment however you ask us to run our program from Shell. Does this mean that we type "java Shell SomeProgram"?

A4: Whenever you start ThreadOS, you msut type "java Boot". When you run a user thread on *Shell*, you need to type "l Shell" and thereafter type the name of a user thread.

```
% java Boot
threadOS ver 1.0:
Type ? for help
threadOS: a new thread (thread=Thread[Thread-3,2,main] tid=0 pid=-1)
-->l Shell
l Shell
threadOS: a new thread (thread=Thread[Thread-6,2,main] tid=1 pid=0)
Shell[1]% Test3
threadOS: a new thread (thread=Thread[Thread-8,2,main] tid=2 pid=1)
....
```

Q5: When I test my SyncQueue.java on ThreadOS, I got the following result. Is it correct?

```
lisu@seawolf:~/430/ThreadOS> java Boot
threadOS ver 1.0:
Type ? for help
threadOS: a new thread (thread=Thread[Thread-3,2,main] tid=0 pid=-1)
-->l Test2
l Test2
threadOS: a new thread (thread=Thread[Thread-6,2,main] tid=1 pid=0)
-->threadOS: a new thread (thread=Thread[Thread-8,2,main] tid=2 pid=1)
threadOS: a new thread (thread=Thread[Thread-10,2,main] tid=3 pid=1)
```

A5: No.

As soon as ThreadOS Loader invokes Test2, it displayed a new prompt, (i.e., "-->"). This means that ThreadOS Loader did not wait for the termination of Test2.

Q6: In the specification of SyncQueue.java, you said we should use wait and notify. How can we know which thread will wake up if we only use notify?

A6: SyncQueue.java itself should not call wait nor notify, otherwise you cannot distinguish different conditions.

See the answer to Q1 for details.

Q7: Can I use notifyAll instead?

A7: No. You should wake up only one thread.

Q8: How can we make sure to wake up the thread in the first-come-first service order?

A8: You don't have to enforce the FCFS order

Q9: I got a message that is "illegal monitor state exception." What's wrong with my code?

```
public void sleep( ) {
          try{
            currentThread.wait( );
          }catch (InterruptedException e){}
    }
```

A9: wait( ) and notify( ) must be used inside a synchronized method. Also, don't add any keyword or object name to wait( ) and notify( ).

```
public synchronized void sleep( ) {
          try{
            wait( );
          }catch (InterruptedException e){}
    }
```

Q10: I thought I implemented SyncQueue.java, but I found that Test2.java had not been always woken up even after the termination of all its five child threads. I have no clue.

A10: For each condition, you must maintain a Vector queue of child thread IDs in QueueNode.java.

When a child thread calls SyncQueue.dequeueAndWakeup( ) that calls QueueNode.wakeup( ) or something, it enqueues its TID in this Vector queue of thread IDs. When a parent thread calls SyncQueue.enqueueAndSlee( ) that calls QueueNode.sleep( ) or something, it checks if the Vector queue of child thread IDs is empty. If so, it puts itself to sleep. Upon a wake-up, it checks this TID queue and picks up the first ID from this queue as a return value.

If all your five child threads have been terminated first, Test2.java will not sleep at all even when it has called SysLib.join( ) five times.

Q11: When you refer to "running time" do you mean execution time or turnaround time?

A11: Test3.java should measure the time elapsed from the spawn to the termination of its child threads.

You may precisely measure the execution time and turnaround time for each of TestThread3a.java and TestThread3b.java or (a computation-intensive and an I/O intensive thread,) too.

Q12: How can I get the current thread's TCB?

A12: Use Scheduler.getMyTcb( ).

Scheduler.getMyTcb( ) returns the current thread's TCB. Assuming that the *curTcb* refers to the current thread's TCB obtained from Scheduler.getMyTcb( ), you can get the current thread's id and parent id by calling curTcb.getTid( ) and curTcb.getPid( ) respectively.

Q13: I'm not sure how to use the rawread, rawwrite, and sync method calls.

I tried to write a little thread to test rawwrite, but I got an ArrayOutOfBounds Error from Disk.run() when it tries to do an array copy.

```
public class newTest extends Thread {
        public void run () {
                byte[] bytes = new byte[512];
```

```
                String str = "message";
                bytes = str.getBytes();
                SysLib.rawwrite( 1, bytes );
                SysLib.exit( );
        }
}
```

A13: Be reminded that the assignment in Java is regarded as substituting the left-hand-side variable with a reference to the right-hand-side variable.

In the above example, bytes originally got an array of 512 bytes, however it was substituted with a reference to "message" whose size is 7 * 2 bytes, (i.e., 14 bytes). (Recall that a character needs two bytes in Java.) So, at the time when you called rawwrite, the bytes array had only 14bytes instead of 512 bytes. That's why rawwrite went awry.

Q14: I got an illegal monitor state exception.

```
Java.lang.IllegalMonitorStateException: current thread not owner
        at java.lang.Object.wait(Native Method)
        at java.lang.Object.wait(Object.java:426)
        at SyncQueue.enqueueAndSleep(SyncQueue.java:56)
        at Kernel.interrupt(Kernel.java:98)
        at SysLib.join(SysLib.java:10)
        at Loader.run(Loader.java:48)
        at java.lang.Thread.run(Thread.java:536)

I have done some debugging and this code is causing the error:

        public void enqueueAndSleep( int condition ) {
                Thread t = Thread.currentThread( );
                try {
                        this.wait( );
                } catch ( InterruptedException e ) {
                        SysLib.cerr( e.toString( ) + "\n" );
                }
                QueueNode node = new QueueNode(t, condition);
                addNode(node);

        } // End endqueueAndSleep
```

A14: You should simply call wait( ). Don't use "this.wait( )".

See also the answer to Q9.

Q15: SysLib.join() and SysLib.exit().

The assignment description talks about the wait() and notify() methods available in Java, then goes on to say that the methods do not account for different conditions to be signaled. So then it goes on to talk about the SysLib.join() and SysLib.exit() methods, which should be used in our SyncQueue.java.

Where I'm confused is in the program 4 FAQ.pdf, there are snippets of code from inside SyncQueue.enqueueAndSleep(), where you tell us to use the wait() method, and there's no mention of

SysLib.join() or SysLib.exit(). So I guess I'm lost on where we should be using these different methods? I was under the impression that we would be coding SyncQueue.enqueueAndSleep() and SyncQueue.dequeueAndWakeup() with SysLib.join() and SysLib.exit().

A15: You will note the following snippets in the provided Kernel.java code:

```
public final static int WAIT    =  2;  // SysLib.join( )

public final static int EXIT    =  3;  // SysLib.exit( )
```

This should lead you to look at the implementation of "WAIT" and "EXIT", which you'll note have no actual implementation:

```
case WAIT:
  // get the current thread id
  // let the current thread sleep in waitQueue under the
  // condition = this thread id
  return OK; // return a child thread id who woke me up
case EXIT:
  // get the current thread's parent id
  // search waitQueue for and wakes up the thread under the
  // condition = the current thread's parent id
  // tell the Scheduler to delete the current thread (since it is exiting)
  return OK;
```

Q16: I can't find the SyncQueue.java file.

A16: You are expected to code it from scratch.

Per the assignment document, "Design and code SyncQueue.java following the above specification."