**CSE 4300**
**Assignment 4**
**Released: 4/18/2019**
**Due: 5/5/2019, 11:59 pm**

This project has two parts. Part A and Part B. Details are below.

This assignment requires you to implement synchronization primitives for OS/161, and use them to solve a synchronization problem. For this assignment, you need to be familiar with OS/161 thread code. The thread subsystem provides interrupts, control functions, and semaphores. You will be extending the thread subsystem to provide additional functions that are found in similar thread libraries.

**Part A. Implement locks for OS/161 (100 points)**
The first part is to implement the locking primitive for OS161. The interface for the lock structure is defined in file **kern/include/synch.h**. Stubbed code is in kern/threads/synch.c. These should be "sleep locks" and not "spin locks," in the sense that a waiting thread will sleep, rather than spin, until the lock is available.

**Part B. Implement condition variables for OS/161 (100 points)**
In this part, you need to implement condition variables for OS/161. The interface for the cv structure is defined in **synch.h** and stubbed code is provided in synch.c.

The original idea behind a condition variable is to allow a thread to suspend itself while executing in a monitor until some Boolean condition is satisfied. "Executing in a monitor" implies that the thread is holding some lock, which must be passed as an argument to the cv_wait function. Threads that call cv_wait **always** block, and they must release the lock before doing so. Before returning from cv_wait, a thread must re-acquire the lock. Thus, the lock is held when cv_wait is called, and is held when cv_wait returns, but must be released while the thread sleeps.

The pseudo code for cv_wait is given below:

```
cv_wait (struct lock *lock, struct cv *cv)
{
// Atomically, do the following:
// 1. release the lock
// 2. put the thread to sleep on cv
// 3. re-acquire the lock
}
```

The "signal" operation on a condition variable allows a thread executing in a monitor (which implies it must be holding the same lock that the waiting thread was holding when it called cv_wait) to notify a waiting thread that the thing it was waiting for has happened. The cv_signal function should wake up exactly one thread waiting on the address "cv". If no threads are waiting, then a cv_signal has no effect. A read of the thread_wakeup function shows that it does not have the desired behavior for condition variable signals. You will need to implement a new function that does what you want. The cv_broadcast operation is identical to signal, except that it should wake up all threads waiting on the cv.

**Testing**

For testing your implementations, please look at the following built-in synch tests:

- "sy2" tests the lock implementation
- "sy3" tests the condition variable implementation.

These tests are implemented in ../kern/test/synchtest.c. You should look at this code to understand how they are testing the various synchronization objects.

Try running "sy2" and "sy3" without any implementations for locks and condition variables to see what happens when these fail.

When you have a correct implementation of locks, the "sy2" test will take about 2 seconds (or much less) to complete if everything goes well. If it takes much longer and you get no output, then you probably have a deadlock and will need to re-examine your lock implementation.

For cv, run the "sy3" test from the OS/161 menu. If cv is implemented correctly, the thread numbers will print out in reverse order, several times, without any "noisy" output, if everything is correct.

For this programming assignment, you need to compile as follows -

cd os161-1.11
cd kern/conf

./config ASST1

cd ../compile/ASST1

make depend
make
make install

From root directory, you can load the new kernel by typing the following -

sys161 kernel-ASST1

From OS161 menu, just type sy2 or sy3 to execute the test programs.

**What to Hand In**

1. Submit your modified synch.c and synch.h files.
2. Submit a report explaining your changes. Take screenshots of the test program output and include that in the report.