Program 4 is due at the beginning of class on **March 31st**. I reserve the right to have you “demo” your program for me and explain various parts of your code. Such a demo will take about 10 minutes. Late submissions will be penalized.

One of the most commonly used thread libraries on Linux is called “POSIX Threads” or pthreads. First, go through the pthread tutorial at [https://computing.llnl.gov/tutorials/pthreads/](https://computing.llnl.gov/tutorials/pthreads/).

In this assignment, you will use pthread’s synchronization mechanisms to solve a mutual exclusion problem and a deadlock problem.

Download the following files into a directory: Makefile, mutex.c and deadlock.c. Run make to compile the programs, then execute the two programs and observe what they do:

```
$ make
...

$ ./mutex
Mutex program starting.
Hello world, I'm thread 1
Hello world, I'm thread 2
Hello world, I'm thread 3
Hello world, I'm thread 4
Hello world, I'm thread 5
Hello world, I'm thread 6
Hello world, I'm thread 7
Hello world, I'm thread 8
Hello world, I'm thread 9
Hello world, I'm thread 10
thread 3 counting 1
thread 1 counting 2
thread 6 counting 3
thread 10 counting 4
thread 4 counting 5
thread 7 counting 6
thread 9 counting 7
thread 2 counting 8
thread 8 counting 9
thread 1 counting 10
thread 5 counting 11
thread 2 counting 12
thread 4 counting 13
...

$ ./deadlock
deadlock program starting.
Hello world, I'm thread 1deadlock program starting.

```
Hello world, I’m thread 1
thread 1 about to get resource A
thread 1: I got resource A
Hello world, I’m thread 2
thread 2 about to get resource B
thread 2: I got resource B
thread 2 about to get resource A
thread 1 about to get resource B
thread 1 about to get resource A
thread 1: I got resource A

thread 1 about to get resource A
thread 1: I got resource A
Hello world, I’m thread 2
thread 2 about to get resource B
thread 2: I got resource B
thread 2 about to get resource A
thread 1 about to get resource B
thread 1 about to get resource A

Use pthread’s synchronization primitives (mutexes and condition variables) to solve the two problems.

For the mutex program you need to force the threads to execute in order so that thread 1 counts then thread 2 then thread 3 and so on. You may not remove the sleep() call or alter the section that makes them sleep for a random number of seconds.

In the end the output should look like the following:

$ ./mutex
Mutex program starting.
Hello world, I’m thread 1
Hello world, I’m thread 2
Hello world, I’m thread 3
Hello world, I’m thread 4
Hello world, I’m thread 5
Hello world, I’m thread 6
Hello world, I’m thread 7
Hello world, I’m thread 8
Hello world, I’m thread 9
Hello world, I’m thread 10
thread 1 counting 1
thread 2 counting 2
thread 3 counting 3
thread 4 counting 4
thread 5 counting 5
thread 6 counting 6
thread 7 counting 7
thread 8 counting 8
thread 9 counting 9
thread 10 counting 10
The deadlock program demonstrates a classical deadlock example of two threads that need two resources A and B. The two threads attempt to acquire the resources in some order. You need to resolve the deadlock.

You must submit (using submit on stono) the following files:

1. Makefile
2. Modified mutex.c
3. Modified deadlock.c