CS 470 - Operating Systems
Spring 2021 - Thread Synchronization Project
40 points
Out: February 22, 2021
Due: March 8, 2021 (Monday)

The purpose of this project is to provide experience with Pthreads and (thread) synchronization.

Problem Statement
Complete the producer-consumer project described on pages 262-264 in the textbook **using Pthreads**. In this project, there may be multiple producers and consumers accessing a bounded buffer as a circular queue, each of which is running in a thread. Although similar in concept to the producer-consumer processes with shared memory and semaphores example given in class, differences from the example include:

- Use of threads rather than processes
- Multiple producers and consumers
- No external shared memory, since threads share global data.
- Use of unnamed semaphores instead of external semaphores.
- Use of thread mutex lock instead of a mutex semaphore.

As noted in the project description, the Pthreads API for creating threads is discussed in Section 4.4.1 (pages 172-174) and the API for mutex locks and unnamed semaphores is discussed in Section 5.9.4 (pages 237-238). This is for C/C++, but the libraries for other languages should be similar.

In addition, the following requirements must be met:

- There should be output similar to the process synchronization examples shown in class that indicates when a thread is attempting to gain a semaphore and when it releases a semaphore.
- Both the producer and the consumer threads should be passed the loop index number when it is created so that the output indicates which thread produced or consumed an item. I.e., the output of the project should say something like "Producer <#> produced <value>" and "Consumer <#> consumed <value>". Note that a single integer value may be passed as a thread parameter by simply casting it to the **`void*`** type, and casting it back to an integer in the thread.
- The use of counting semaphore synchronization means that a **`numItems`** counter is not needed and **should not be included** in this project. However, with multiple producers and consumers, the **`in`** and **`out`** index counters for the back and front, respectively, of the buffer will need mutex lock protection.
- Depending on the language and routines used, output statements may not be atomic. For C/C++ programs it is recommended that a single **`printf()`** statement per output line be used. If something like the C++ **`operator<<`** is used, a mutex lock should be used to create a critical section for displaying output to prevent interleaving.

Assignment
(20 points) Implementation. This project may be written in any language as long as it uses Pthreads (not Windows or programming language threads) and runs on csserver, but the instructor will only be able to help with C/C++ programs. Provide a makefile that will make your project, if needed. Recall that when compiling a Pthread program using gcc or g++ that you must include the command-line option: **`-lpthread`**

Note for Mac users: The Mac OS does not support unnamed semaphores. Therefore, this project cannot be completed natively on a Mac.

(10 points) Provide a high-level discussion describing the functionality of the program . In particular, describe where the synchronization statements are used and why they are there. If the program does not meet all of the

project requirements, describe which parts are missing and what you think should be done to implement them.

(10 points) In addition, answer the following questions:

1. What aspect of thread manipulation did you find most difficult to understand?
2. What aspect of thread manipulation did you find least difficult to understand?
3. What aspect of thread synchronization did you find most difficult to understand?
4. What aspect of thread synchronization did you find least difficult to understand?
5. What, if anything, would you change in your current design?
6. What, if anything, did you find interesting or surprising about thread manipulation or thread synchronization that you did not know before doing this project?

What to Submit

Create a **tarfile** or **zipfile** archive containing the following items:

- The code source code for your project. If the project is not written in C/C++, put instructions on how to build and/or run the program in a comment at the beginning of the main program file.
- A makefile to make your project, if needed
- A single document **in PDF format** with the discussion of the functional design of your project and the answers to the questions above.

Submit your archive using the submission system (http://submission.evansville.edu) no later than 11:59pm on the due date. Reminder: your username is your ACENET username with "-cs470" appended to it, and your password is your student ID number including the leading 0 (i.e. 7 digits) unless you have changed it. The grading script only will accept submissions. It will not run anything.