# CSCE 3600: Systems Programming

## Minor Assignment 3 – Threads, Mutexes, and Condition Variables

## Due: 11:59 PM on Wednesday, October 16, 2024

**PROGRAM DESCRIPTION:**

In this assignment, you will be given a functioning program, called `minor3.c`, that simply a) reads user input keys and b) echoes them back to the screen using the producer-consumer paradigm. The single **producer thread** reads user input keys and adds them to the **shared buffer** while two **consumer threads** read the added keys from the buffer and echo them back to the screen. To complicate matters, each key is read and echoed by exactly one consumer thread. A shared variable, called `shared_count`, keeps track of the number of items in the shared buffer.

While this program does work (thanks to the mutex locks and unlocks already provided), it is unfortunately very inefficient. To see just how inefficient this program is, compile the original `minor3.c` program (using the `pthread` library) and execute the program. You should type in some keys and see them echoed back on the screen in their correct order. To see the inefficiency, though, run the `top` command from another shell (don't just run the `minor3.c` program in the background and then run `top`, but actually open up another shell/window). Then check out the `%CPU` column in the `top` command and you should see your original `minor3.c` program using up a significant percentage of the CPU, which is not good.

Your goal for this assignment is to **modify this program to use condition variables** that will drastically reduce its CPU percentage usage. Here are the details:

- You will modify the original `minor3.c` file to **add two condition variables**, but not change the "spirit" of the program other than necessary changes that are needed for how conditional variables work, including handling of "spurious wakeup" discussed in class.
- You will **add two global `pthread` condition variables** – one to handle when the shared buffer is full (and therefore, nothing else can be added to the buffer until a key is removed from the buffer) and one to handle when the shared buffer is empty (and therefore, nothing can be read/echoed back to the screen until a key is added to the buffer).
- In the `main` function, you will **initialize and destroy both of the condition variables**.
- You will modify the code in the `producer` function to **wait** on and **signal** the appropriate condition variable(s) based on what is happening with the shared variables (i.e., the shared buffer and shared counter). Note that this will require some small changes in logic to accomplish, but you should not change the lines that work with the `prod_index` variable.

- You will modify the code in the `consumer` function to **wait** on and **signal** the appropriate condition variable(s) based on what is happening with the shared variables (i.e., the shared buffer and shared counter). Note that this will require some small changes in logic to accomplish, but you should not change the lines that work with the `cons_index` variable.

Be sure to run your solution along with the `top` command to verify that the program is more efficient (i.e., does not use nearly as much percentage of the CPU). It is required that you implement and utilize the condition variables effectively in a manner that significantly reduces the CPU utilization of this program and not gain the reduction in another way. Note that the grading rubric requires that the condition variables be implemented "logically correct", which includes accounting for "spurious wakeup", not just reducing the CPU utilization.

**REQUIREMENTS:**

- Your code should be well documented in terms of comments. For example, good comments in general consist of a header (with your name, course section, date, and brief description), comments for each variable, and commented blocks of code.
- Your program should be named "`minor3.c`", without the quotes.
- Your program will be graded based largely on whether it works correctly on the CSE machines (e.g., `cell01`, `cell02`, …, `cell06`), so you should make sure that your program compiles and runs on a CELL machine.
- No **SAMPLE OUTPUT** is provided since your solution should run (from an input/output perspective) the same as the original, provided `minor3.c` program.
- This is an individual programming assignment that must be the sole work of the individual student. Any instance of academic dishonesty will result in a grade of "F" for the course, along with a report filed into the Academic Integrity Database.

**SUBMISSION:**

- You will electronically submit your program to the **Minor 3** dropbox in Canvas by the due date.