Programming Assignment 4 Report

Andres Hinojosa

Submitted Files:

- **buffer.h:** The buffer class header file, including a vector and multiple integer variables
- **buffer.cpp:** The buffer implementation file, including the initialization of the vector size and its functions.
- **main.cpp:** The main program used to create the number of producer and consumer threads while using synchronization in the producer/consumer functions.

How to compile and run the program:

- To compile the program, use command "make."
- To run the program, use command: ./prog4 followed by three integer parameters which are the main program's sleep time in seconds, the number of producer threads, and the number of consumer threads.

Results for programs:

- After testing different numbers, my program was able to run correctly by having only one producer/consumer thread run for the given time.

Features implemented:

- **Buffer**: I started by working on the implementation of the buffer file. The constructor was simple, I used the resize vector function to change its size to whatever was passed in the function even though the skeleton hard codes it to be 5. I used the clear vector function in the destructor to delete all the items. To insert, I make sure the vector is not full, and add the item to wherever the in variable is located. I then update the counter and in using mod and the function size to make it like a loop. The remove function work in the same way, the only difference is the item is passed and returned. The out variable is used as opposed to the in and is updated in the same way. Get size, count, and is empty are all simple and return size, count, and if count == 0 respectively. Finally, print buffer goes from out, to in, printing all the items in between and uses the mod to loop around.
- **Threads**: I implemented the thread functions by creating an array for both producers and consumers using the size passed in the parameters. A loop is used to create them and add them to the array. I initialize them using the pthread_create function and pass the appropriate parameters including its number.
- **Mutex lock**: In the main file, I implemented the mutex lock by creating an object as a global variable. I initialized it in the main function and use it at the start of the producer and consumer functions so that they run independently and one at a time. I then unlock it once the producer/consumer finishes. The mutex is destroyed after the program completes.

- **Semaphores**: To help with the synchronization, I also decided to implement two semaphores. I created to global objects. One for full and one for empty. I also initialize it in the main function and use it in the same way I use the mutex lock, except I use the wait and signal functions. Both semaphores are destroyed once the program is finished.

Design and implementation choices:

This assignment was a bit difficult to start, especially since I am not 100% confident in how the C functions work, but I was able to use the lecture slides to know which functions I should use and where they should go in the producer and consumer functions so that the synchronization is implemented correctly.

Lessons Learned:

In this assignment, a lesson that I learned was to review so that I know what I should be doing as opposed to going in blind. I was able to understand mutex locks better, But the semaphores are a little bit hard for me to fully understand. Another lesson I learned was to use resources to understand how some functions work. For example, I spent a long time figuring out why usleep was not working, but I didn't know that it took in milliseconds as a parameter, which I learned through a quick search.

References:

The reference I used were the lecture slides for the functions of mutex locks, semaphores, threads, and how they're initialized and placed. I also used a YouTube channel called CodeVault where a person explains how the mutex, semaphore, and thread functions work.

Future improvements:

To improve my program, I would remove any redundant code that I have to make it more compact. I would also find ways to do some things in a more efficient manner to further clean up my code and reduce its size.