Programming Assignment 4 Report

Submitted Files:

- buffer.h: Provides the Buffer class, containing method signatures and private/public instance variables relating to the invoking object of the new Buffer object
- buffer.cpp: Includes buffer.h, and provides the definitions of the Buffer class methods
- main.cpp: The main program for the producer/consumer problem.
- **Makefile**: Allows for automation for having to compile and link object files of the above files using the "make" command and creates executable ELF file/s: prog4. Also creates object files (.o) for all the source files

How to compile and Run the Program:

To compile the program, use command: make

To run the program, use command: ./prog 10 3 2

Results/Test cases:

Our program successfully completed the effort we put forward on the various test cases and our functions are functioning to the extent of what we know. The test cases are displayed in the Gradescope autograder.

Features Implemented (by class):

Buffer: A class that contains private instance variables: size, counter and buffer_item[BUFFER_SIZE], also public instance variables: mutex and attr.

Buffer(int), ~Buffer() – Constructor that assigns the size to the invoking object's size. Destructor destroys the Buffer object.

insert_item – inserts new item to the buffer, depending on if it is full or not

remove_item – removes an item from the buffer, depending on if it is empty or not

get_size – returns the size of the invoking object *get_count – returns the count (number of items in the buffer) of the invoking object*

is_empty – checks if the buffer is empty

is_full – checks if the buffer is full

print_buffer – prints the buffer

Additional Features:

- *Utilizes <pthread.h> header, which includes thread creation, termination, synchronization (joins, blocks), etc...*

Design and Implementation choices:

Our design and implementation choices were derived from what we learned in class, which includes the slides performing the various features and operations of the many examples regarding buffers and thread management like the event of creation, deletion and building our understanding of functions like *usleep* to pause the execution of a thread. We also noticed the usages of making threads sleep, especially the main thread in case if other portions of the program are utilizing events, and not to immediately terminate the program. We decided to work on the buffer.cpp file first to get the firm grasp of the implementation of the Buffer class and ensuring that it would provide all necessary usage of instance variables and methods when we work with the producer and consumer threads. Once we acknowledged that the Buffer class was complete, we successfully were given various results. We then stumbled on the problem of the segmentation fault that most of our class was having trouble with trying to solve it, but eventually we managed to break out of this problem near the end. The testing was done in the main.cpp.

Overall understanding and learning outcomes:

We learned various operations for the usages and tests regarding the producer/consumer assignment. This helped us understand and manage the producer/consumer problem. It provided us with a fundamental deal of knowledge into how we could abstract these ideas and we may even go about implementing these concepts into our own personal projects.

References:

We heavily relied on the ZyBook's to provide us with the information to being able to understand our tasks. We also wrote out different implementations into going about the producer/consumer problem. We also looked at various Linux man-pages regarding the usage of the *pthread.h* header to ensure that we had a clear understanding of its unique set of given functions and types.

Future improvements:

Future improvements would be trying to utilize this understanding to create a larger project relating to the usage of the concepts relating to the producer/consumer problem.