Introduction

In this assignment you will be solving the classic bounded buffer Producer-Consumer problem with one producer and multiple consumer threads. You have been provided with the following files:

- **student.c** The C file where you will be writing code
- **Makefile** A makefile to compile your program
- **shortlist** The shorter of the two input files
- **longlist** The longer input file
- **solution.txt** A sample output for shortlist with 4 threads

The program takes the number of consumers as argument (defaulting to 1) and a sequence of numbers from stdin. More information on the workings can be found in the form of comments on top of the *student.c* file.

The producer thread reads the sequence of numbers and feeds that to the consumer. Consumers pick up the number, do some ”work” with the number, and then go back for another number.

Your Task

As you may have seen, the given *student.c* file has some incomplete functions. You are required to do the following in order to get a working solution.

- You are responsible for making a synchronized buffer that can hold **TEN** integers. This buffer should follow a first in first out order. We recommend using a queue backed by an array.
- Fill out the following functions in a thread safe manner so that multiple threads can access the buffer simultaneously. Below is a list of functions you are going to implement:
  - `void buffer_init(void)` This function is called by main() before the producer and consumer threads are started. Use this function initialize your buffer and mutex / condition variables, etc.
  - `void buffer_insert(int number)` This function, called by the producer thread inserts a number into the next available slot in the buffer. If no slots are empty, the thread should wait (not spin-wait!) for an empty slot to become available.
  - `int buffer_extract(void)` This function, called by the consumer threads removes and returns the number from the next available slot. If no numbers are available, the thread should wait (not spin-wait!) for a number to become available. Note: Multiple consumer threads may call `buffer_extract` simultaneously.
- Your implementation must not spin-wait. There are several possible strategies. An excellent strategy with pthreads is to use mutex and condition variables, i.e. use `pthread_cond_wait()` to wait when the buffer is empty or full.

NOTE: If you are running Linux in a virtual machine, make sure to enable multiple cores. If you do not enable multiple cores, running your code on our machine may produce deadlocks that won’t show up on your computer.

Testing

Compile your code by typing `make`. You can run it by doing the following

```
$:  ./m3 <Number of consumers> < <Input Filename>
```
Note: The '<' is used to pipe the contents of the file as an input to the program

- You should be able to reproduce the output in solution.txt when running your code with *shortlist* and *four* threads. Note: Your printed console output may not identically match what is shown below due to randomness of thread scheduling. However your output should show all entries being produced in the correct order and consumed in the correct order.
- Try measuring the execution time using the `/bin/time` command in Linux. When running with 'longlist', doubling the number of consumers should roughly halve the execution time. What is the minimum possible execution time?

Deliverables

Use the `make submit` command to generate a deliverable tarball. Make sure the generated file contains:

- student.c
- Makefile
- shortlist
- longlist

NOTE: Keep in mind that code that does not compile will not get any credit at all