Lab 2: Threads and Synchronization

In this assignment we will use threads and synchronization to implement the producer/consumer and the bounded buffer problem. This lab needs to be completed in C/C++ using the pthreads library with the overall goal of exposing you to threading and synchronization concepts.

Request Scheduling Using Threads and Synchronization

The goal of the lab is to first implement a bounded buffer class and then use it to implement the producer consumer problem. The first step is to implement a bounded buffer using synchronization. Implement a bounded buffer class using pthreads mutex (i.e., locks) and condition variables. The constructor for your bounded buffer class should take *N* as an input and instantiate an integer array of size N. Assume that the bounded buffer contains integer data items.

The bounded buffer is a *circular buffer*. Upon reaching the end of the buffer array, your code should cycle back to the start of the array to add new data items to the circular buffer. Items are always added to the tail of the buffer and removed from the head of the buffer

Keep in mind that pthreads condition variables use Mesa-style implementation, not a Hoare implementation.

Next implement a `ProducerConsumer` class that uses this Bounded Buffer to produce and consumer data. Your class should have two main methods: a **producer()** method where a producer thread sits in a loop and in each iteration, it first sleeps for a specified duration, then produces a data item (the data item in this case is an integer with a random value), and calls the add method of the bounded buffer to add the data item to the buffer.

In the **consumer()** method, a consumer thread sits in a loop, and first sleeps for a specified duration and then calls the remove method of the bounded buffer to remove an item from the circular buffer.

The constructor for the `ProducerConsumer` class takes several inputs:

- **p**: the number of producer thread to create, p should be at least 1
- **c**: the number of consuemer threads to create, c should be at least 1
- **psleep**: the time duration in milliseconds that a producer threads sleeps in each iteration
- **csleep**: the time duration in milliseconds that a consumer threads sleeps in each iteration
- **items**: the total number of data items that should be produced by all the producer threads collectively before they are done.

Given these inputs, the constructor method should create *p* producer threads, each of which start in the **producer()** method, and *c* consumer threads, each of which start in the **consumer()** method. After producer a data item, a producer thread should increment a shared integer variable. When this variable reaches **item**, all producer and consumer threads should quit (of course the consumer threads should consume all items in the buffer as well before quitting).

Note that the bulk of the synchronization is done in the bounded buffer class, while the bulk of thread management is done in the producer consumer class (if you use any shared variables in producers and consumers, be sure to use synchronization to protect those variables as well).

Your producer and consumer methods should print the following messages to a file called "output.txt":

```
Producer #i, time = current time, producing data item #j, item value=foo
Consumer #k, time = current time, consuming data item with value=foo
```

Helpful References

If you are not familiar with pthreads, be sure to review this reference on Pthreads programming. A brief tutorial is also available.

Hints

Do not forget to init your pthread mutexes and condition variables.

Understanding Your Code

Once you have working code, you should run your code with the following inputs to understand its behavior:

- Run your code with 1 producer and 1 consumer thread. Choose psleep=1000ms, csleep=1ms and have the producer generate 10 items. What behavior do you observe?
- Run your code with 1 producer and 1 consumer thread. Choose psleep=1ms, csleep=1000ms and have the producer generate 10 items. What behavior do you observe now?
- Run your code with 5 producer and 5 consumer threads. Choose psleep=1ms, csleep=1ms and have the producer generate 20 items. What do you observe? Does the behavior change if you run your code multiple times?