Introduction

This problem has you solve the classic bounded buffer problem with one producer and multiple consumer threads.

The program takes the number of consumers as an argument (defaulting to 1) and a sequence of numbers from stdin. We give you a couple of test sequences: "shortlist" and "longlist". For more explanation of how this works, see the comment at the top of "hw3.c".

The producer thread reads the sequence of numbers and feeds that to the consumers. Consumers pick up a number, do some work with the number, then go back for another number. The program as provided includes output from the producer and consumers. For reference, a working version of the code with a bounded buffer of size 10 running on "shortlist" with four consumers produces this output (the comments on the right are added): (NOTE: Your printed console output may not match what is shown identically due to the randomness of thread scheduling. However, your output should show all entries being produced in the correct order and consumed in the correct order).

```
>>> ./hw3 4 < shortlist

main: nconsumers = 4

  consumer 0: starting

  consumer 1: starting

  consumer 2: starting

  consumer 3: starting

  producer: starting

producer: 1

producer: 2

producer: 3

producer: 4

producer: 5

producer: 6

producer: 7

producer: 8

producer: 9

producer: 10

producer: 9

producer: 8

producer: 7
```

```
producer: 6
  consumer 0: 1
producer: 5
  consumer 1: 2
producer: 4
  consumer 2: 3
producer: 3
  consumer 3: 4
producer: 2
  consumer 0: 5
producer: 1
  consumer 1: 6
producer: read EOF, sending 4 '-1' numbers
  consumer 2: 7
  consumer 3: 8
  consumer 0: 9
  consumer 1: 10
producer: exiting
  consumer 2: 9
  consumer 3: 8
  consumer 0: 7
  consumer 1: 6
  consumer 2: 5
  consumer 3: 4
  consumer 3: exiting
  consumer 0: 3
  consumer 0: exiting
```

```
consumer 2: 1

consumer 2: exiting

consumer 1: 2

consumer 1: exiting
```

A. Finish the bounded-buffer code in hw3.c, adding synchronization so that the multiple threads can access the buffer simultaneously.

NOTE: If you are running your linux environment in a virtual machine, make sure to enable multiple cores. If you do not enable multiple cores, running your code on our machines may produce deadlocks that are not reproducable on your machine.

- There are really two problems here: managing the bounded buffer and synchronizing it. We suggest to write and test your bounded buffer implementation first before implementing synchronization.
- Your implementation must not spin-wait. There are several possible strategies. An excellent strategy with pthreads is to use a mutex and condition variables, i.e. using pthread_cond_wait() to wait when the buffer is empty or full.

B. Testing suggestions:

- You should be able to reproduce the output above.
- Try measuring the execution time with /bin/time. When running with "longlist", doubling the number of consumers should roughly halve the execution time. What is the minimum possible execution time?

1 Additional Resources

Help for using pthreads:

- `http://www.llnl.gov/computing/tutorials/pthreads/`
- `http://www.humanfactor.com/pthreads/pthread-tutorials.html`
- `http://en.wikipedia.org/wiki/Pthreads`
- `>>> man -k pthread`
- `>>> man pthread_mutex_init`
- `>>> man pthread_cond_init`

Help for debugging using gdb:

- GDB Tutorial: `http://www.cs.cmu.edu/~gilpin/tutorial/`
- GDB Cheatsheet: `http://darkdust.net/files/GDB%20Cheat%20Sheet.pdf`
- `>>> gdb --args ./hw3.c 4 < shortlist # to start debugging`