1.Briefly describe the problem you are trying to solve and describe the design of your solution.

The problem we are trying to solve is fixing multiple race conditions that occur because of the multiple producer/consumer problem. There are occurrences where, even with mutex locks, consumers are consuming items that do not exist and producers are producing when the max produced limit has been reached. The design of the solution is to use a combination of mutex locks and condition variables, in the form of pthread_wait() and pthread_signal(), to better manage arbitrary numbers of producers and consumers.

2.Discuss why the busy-wait solution is inefficient for threads running on the same CPU, even though it produces the "desired behavior".

The busy-wait solution is a spinlock which consumes a horrendous amount of cpu time just checking the state of the full and empty flags. While it does produce the desired behavior, that behavior can eb better achieved through the use of signals and condition variables which remove the thread from the cpu scheduler while the thread would normally be spinlocked.

3.You will need to argue from your narrated output as to why your solution is correct. Note, your output will likely not match the output listed here exactly. Two successive runs of your application will probably not match even vaguely, due to random variations in how threads are scheduled. However, your report should discuss each of the following points and discuss how your output supports your discussion of each:

- •Are each producer and consumer operating on a unique item? Is each item both produced and consumed?
- •Do the producers produce the correct number of items and the consumers consume the correct number of items?
- •Does each thread exit appropriately?
- •Does your solution pass the above tests with the different numbers of producer and consumer threads in the Makfile tests?

Observing the output included below, it's obvious that the five consumers begin checking for produce and print “EMPTY” before entering their respective pthread_wait() functions. This behavior is followed by the five producers spinning up and filling the queue before alerting the consumers to the notEmpty queue. Then the consumers consume and the cycle begins again. It can been seen in the output below that as signals are sent between producers and consumers, the total number of items produced is equal to 30 (a predetermined max produced value we set) and the number of items consumed is also limited to 30. Once the max produced and max consumed limits are reached, the producers and consumers emit broadcast signals to all producers and consumers waiting for signals to allow each thread to exit one at a time. My code works for all numbers of consumers >0 and producers > 0.

Console Output

```
dan@dan-Satellite-L775D:~/Desktop/Lab_07(Really Lab_06)/pthreads_pc$ ./producer_consumer 5 5
con 0:  EMPTY.
con 1:  EMPTY.
con 2:  EMPTY.
con 3:  EMPTY.
con 4:  EMPTY.
prod 1:  0.
prod 2:  2.
prod 4:  3.
prod 0:  1.
prod 3:  4.
prod 2:  5.
prod 1:  6.
prod 4:  7.
prod 0:  8.
prod 3:  9.
prod 1:  FULL.
prod 2:  FULL.
prod 4:  FULL.
prod 0:  FULL.
prod 3:  FULL.
con 2:  1.
prod 1:  10.
con 0:  2.
```

prod 2: 11.

con 3: 3.

prod 4: 12.

con 1: 0.

prod 0: 13.

con 4: 4.

prod 3: 14.

prod 1: FULL.

prod 2: FULL.

prod 4: FULL.

prod 0: FULL.

prod 3: FULL.

con 2: 5.

prod 1: 15.

con 0: 6.

prod 2: 16.

con 3: 7.

prod 4: 17.

con 4: 9.

con 1: 8.

prod 3: 18.

prod 0: 19.

prod 1: FULL.

prod 2: FULL.

prod 4: FULL.

prod 3: FULL.

prod 0: FULL.

con 2: 10.

prod 1: 20.

con 3: 12.

prod 2: 21.

con 4: 13.

con 0: 11.

prod 3: 22.

con 1: 14.

prod 0: 23.

prod 4: 24.

prod 1: FULL.

prod 2: FULL.

prod 3: FULL.

prod 0: FULL.

prod 4: FULL.

con 2: 15.

prod 1: 25.

con 3: 16.

prod 2: 26.

con 4: 17.

con 0: 18.

prod 0: 27.

prod 3: 28.

con 1: 19.

prod 4: 29.

prod 1: exited

prod 2: exited

prod 0: exited

prod 3: exited

prod 4: exited

con 2: 20.

con 3: 21.

con 4: 22.

con 0: 23.

con 1: 24.

con 2: 25.

con 2: exited

con 4: 27.

con 4: exited

con 3: 26.

con 3: exited

con 0: 28.

con 0: exited

con 1: 29.

con 1: exited