Jay Offerdahl
EECS 678
Lab 6 Report

The Producer Consumer Problem

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

- The problem we're trying to solve here is twofold. First, we have a race condition that occurs when a consumer consumes an item, and before it can report that the queue is empty, a producer produces an item. Afterwards, the consumer would spin infinitely if the producer exited. Next, there's an interleaving TOCCTOU problem (time to check to time of use). This occurs because some producer might check that the queue is not full, and before it acquires the lock, another producer wins the race condition, grabs the lock, and fills the last spot. Afterwards, the second producer would try to add an item to the queue even though it's full.
- The design of my solution revolves around the use of a mutex and condition variables to first only allow one producer/consumer to access the queue at any single time, and second to ensure there's room on the queue for producers or items in the queue for consumers. First, each producer/consumer acquires the mutex lock. Next, they check if the queue is full/empty, respectively, if so, they sleep, giving up the lock momentarily. These threads are awoken when another thread broadcasts a signal that either the queue was not full or not empty, and the status is checked again. Only when this condition is met does the producer/consumer continue, do their job, broadcast a not full/not empty signal, and give up the lock.

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

- This is inefficient because each thread will spin while waiting instead of sleeping. This causes unnecessary processing time to be consumed, which is especially bad for threads running on the same CPU because only one thread will ever be executing at one time.

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?
    - Yes, each producer and consumer operate on a unique item, and, of course, the item is produced and consumed. We can see this by looking at any of the test outputs. For example, any of the tests show that producers produce integers from 0 to 29, and consumers consume integers from 0 to 29. Even though these are in order, they are unique items. Since the producers and

consumers are not double reporting any values, it's clear that they're operating on unique items.

- Do the producers produce the correct number of items and the consumers consume the correct number of items?
  - Yes, as mentioned above, the producers only produce items with values from 0 to 29, which can all be accounted for in any of the tests conducted. Also, consumers only consume values that are on the queue, which are the items produced by the producers (vals 0 to 29). This means that because we only see 30 entries describing items produced, and 30 entries describing items consumed, no item is being consumed twice/produced twice.

- Does each thread exit appropriately?
  - Yes, we can clearly see at the end of the .raw files that with larger numbers of producers/consumers there's a mass exodus when the total produced/consumed is reached. However, this is also visible in the sorted file for each producer and consumer. I ensured this worked properly by unlocking the mutex in the if statement that checked if the total produced/consumed was greater than the work max variable we had defined.
    - Before adding this unlock, threads would break out and deadlock would occur because the lock was never given up.

- Does your solution pass the above tests with the different numbers of producer and consumer threads in the Makefile tests?
  - Yes, I implemented tests for 2 producers and 2 consumers, 10 producers and 1 consumer, 1 producer and 10 consumers, as well as 10 producers and 10 consumers. Even though the output became less and less manageable due to its size with the larger numbers, the producers only produced 30 items, and the consumers only consumed 30 items. I can tell this was working because some producers and several consumers never did any work.
    - This was a different case when I increased the work max variable, as there was more work to be done. However, in the 10 - 10 case, there were still some producers/consumers that did no work.