Evan Nichols
EECS 678
Lab 6 Report
3-30-16

1. I am solving the Producer-Consumer problem, a common implementation pattern for threads or processes working with shared data. The producer and consumer threads share a common buffer. In this case, the buffer is bounded, which means the consumer process must wait if the buffer is empty, and the producer process must wait if the buffer is full. Problems arise when there are multiple producers and consumers. Interleaving can cause issues with checking the current capacity of the buffer, and CPU cycles are wasted if threads are forced into a “busy waiting” situation while they wait for the right conditions to execute.
2. The busy-wait solution is inefficient; in the sample execution from the slides, there were 43,000 “EMPTY” messages printed in one run of the program. While these while loops enforce the correct behavior, they do so in a CPU-intensive way that is undesirable.
3. I believe my solution to be correct. For one, the max run length of the four test cases was 196 lines, orders of magnitude smaller than the “busy waiting” implementation.
    a. Each producer and consumer are operating on a unique item. Each item is both produced and consumed. If you look at the narrative1.raw, you can see that each item is consumed before exiting.
    b. Looking at narrative1.sorted:

```
    P-1 C-1
************
    Sorted
************
con 0:   EMPTY.
  con 0:   0.
  con 0:   1.
  con 0:   2.
  con 0:   3.
  con 0:   4.
  con 0:   5.
  con 0:   6.
  con 0:   7.
  con 0:   8.
  con 0:   9.
 con 0:   10.
 con 0:   11.
 con 0:   12.
 con 0:   13.
 con 0:   14.
 con 0:   15.
 con 0:   16.
```

```
  con 0:   17.
  con 0:   18.
  con 0:   19.
  con 0:   20.
  con 0:   21.
  con 0:   22.
  con 0:   23.
  con 0:   24.
  con 0:   25.
  con 0:   26.
  con 0:   27.
  con 0:   28.
  con 0:   29.
con 0:   exited
```

It is clear that both the producer and consumer iterated over numbers 0-29.

Looking at narrative4.sorted (in my submission), you can see that each thread, both producer and consumer, iterates over 6 items total before exiting. The solution passes all of the tests, as each one still produces and consumes 30 numbers total, and all threads exit appropriately.