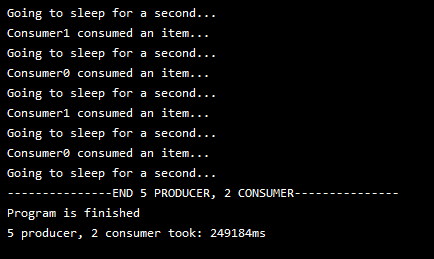
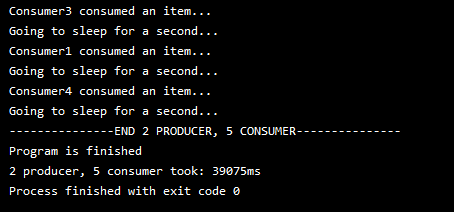
Bryce Callender

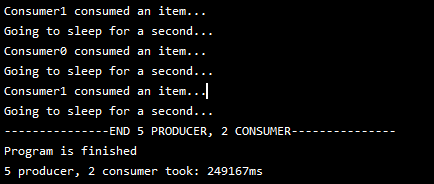
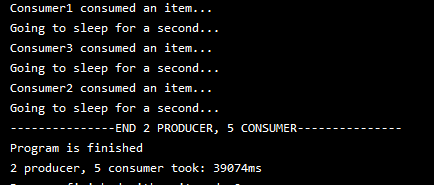
Producer/Consumer (everything but the actors that is included below): <https://github.com/BryceCallender/CS-3700/tree/master/HW5/src>

Actors: <https://github.com/BryceCallender/CS-3700/tree/master/ActorsHw5/src/main/java>

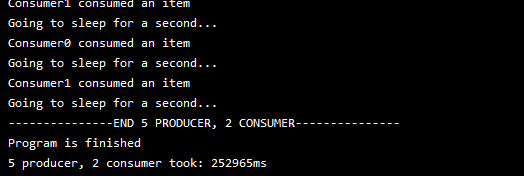
Locks:

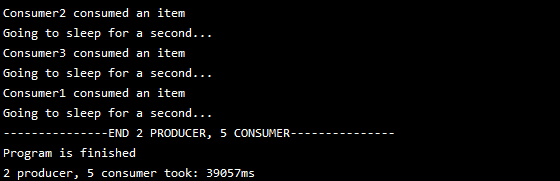
1. 
2. 

Isolated Sections:

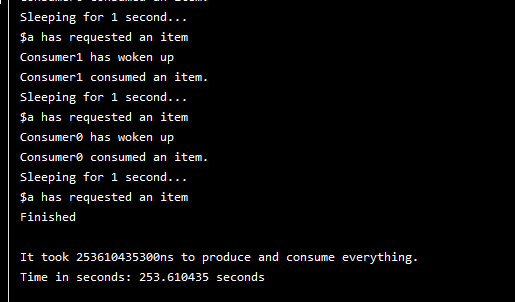
1. 
2. 

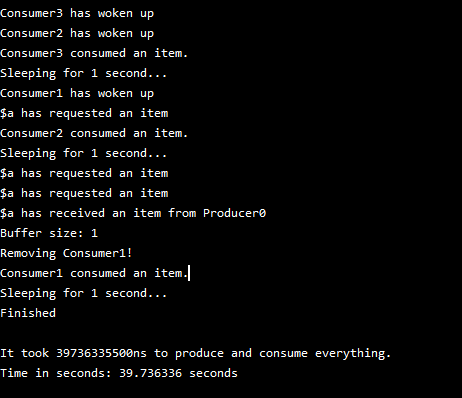
Atomics:

A) 

B) 

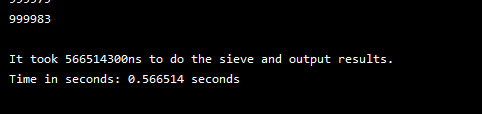
Actors:

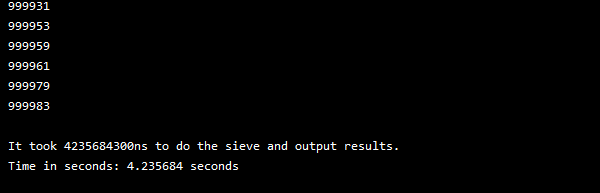
A) 

B) 

1. As we can see here the implementations of locks and isolated sections are almost identical in nature except the two outliers are Atomics and Actors. Isolated sections and locks should be close however since locks are the most low-level, they should be the faster of the two since isolated sections abstracts that. The time for Atomics theoretically should have been the fastest, however it didn’t seem to be the case. I think this was because there was a decent chunk in the critical section which could cause some performance issues since we want it as small as possible. Another cause would be the consumers being a bottleneck. The producers would have to try to fill in as much as they could before a consumer would check and then go to sleep. This can cause issues of not getting the most out of the time which in turn slows the program down a bit. Actors was tough to program. It was just like the atomics in term of performance, however that was due to the fact of all the messages piling into the buffer actor, so the speedup was not there for this to help.

2. Sieve

Single Thread: 

Using Actors: 

c) I got no speedup for my implementation because of the nature of having to create threads and the messaging system for the actor. The actor system is a heavyweight thing since it sets up the threadpool and all the resources. The single thread was able to just execute the numbers really fast while the other was having to create and send/receive messages. Single threaded algorithm was also very optimized so it could be hard to beat out as well.