NOTICE:

For this assignment, I used an implementation of a doubly linked list to implement my queue. Found here:

<http://c.learncodethehardway.org/book/ex32.html>

1. The queue I implemented is a **non-concurrent** queue; so the enqueue and dequeue functions are **not inherently thread safe**. However, I made the structure thread safe in this implementation by using pthread\_mutex\_lock() before every enqueue and dequeue operation (and pthread\_mutex\_unlock() thereafter). I was thinking of implementing the queue using CMPXHCH instructions as mutexes are quite heavy, however I could not be assured that the testing server/terminal would have a CPU with such an instruction set. I was hoping to be able to read/write from the queue in an event driven manner (akin to node.js), however as enqueue and dequeue both have write components to them, a write back/modification of the queue is always necessary and is made necessary to lock the queue on every enqueue and dequeue operation.

2. Because of the usage of mutexes to lockout all but 1 thread from accessing the queue at any given time, the number of threads/processes does not have any effect on performance whatsoever. Whether 1 or 100 threads are being used at any given time, they are forced to be executed in sequential order as the mutex lock forces such behavior. The only way you could see any sort of performance increase from having multiple threads would be if the thread had to do some sort of complex operation before or after the en/de-queue functions where accessing the queue was not immediately needed.