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 structure maintains its own mutex lock and locks/unlocks it during the enqueue and dequeue operations. 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 also thinking of executing 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 in atomic actions such as en/de-queue. Leading to the conclusion that mutexes where in fact the best solution to implement the queue.

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 there be 1 or 100 threads being used at any given time, they are forced to be executed in sequential order as the mutex lock forces such behavior and we it to do maintain transaction tolerance. 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 access to the queue was not immediately needed. In order **to maintain ACIDity for the queue, transaction safety must be assured**; therefore, it is necessary to **lock the entire queue from outside interference during atomic transactions such as en/de-queue.**