Alex Berliner

The approach taken in the locking and non-locking queue for this program are identical aside from the locking queue using mutexes to block other threads from accessing the add & remove functions.

The approach to the overall queue was to bind the size of the array using a head and tail atomic integers to make sure that the thread does not add too many elements or go out of bounds by using a size atomic integer.

The approach to the add function was to do a check to see if the array is full. If it is not, the thread will add a value at the next available element and increment the tail pointer. Add works with pointers instead of by-value integers to maintain data integrity with multiple threads working. The linearization point is defined by incrementing the size variable last.

Similarly, the remove function will check if the array is empty. If it isn’t, then it will return the pointer to the removed element. Like add, size is decremented to alert threads of the updated information.

Difficulties:

It was troublesome originally to figure out how to create a remove function that should not return a value if there are no elements. I decided to implement a remove that returns a pointer that is null if 0 elements are in the queue and a reference to the pointer otherwise.

I originally could not figure out how to use threads until I discovered the -pthread g++ command.

I was unable to install the Intel TBB library on Windows and had to move my code over to Linux where I was able to use a package manager to install the library.