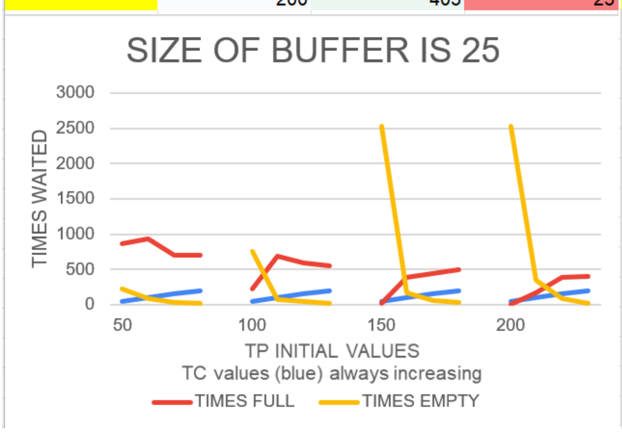
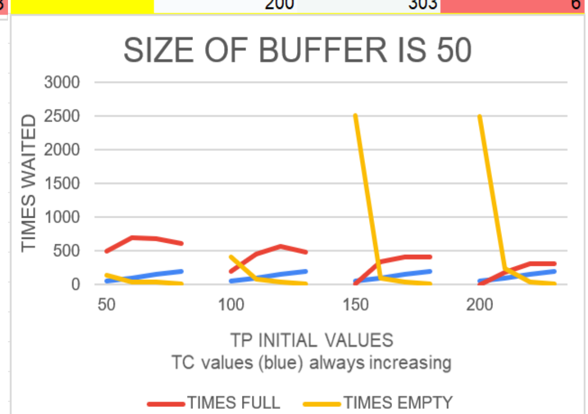
Eric Smrkovsky
CSCI 114, Fall 21
BBQ Project

In this project I implemented a bounded buffer queue using the C++ programming language. This problem is also known as the "Producer and Consumer" problem and is a common example of a multithreading program. In this assignment I use locks and conditional variables within threads to synchronize access of the queue. The program creates twenty threads, ten of which are producer threads that add an item to the queue per iteration and the other ten are consumer threads that remove an item from the queue per iteration.

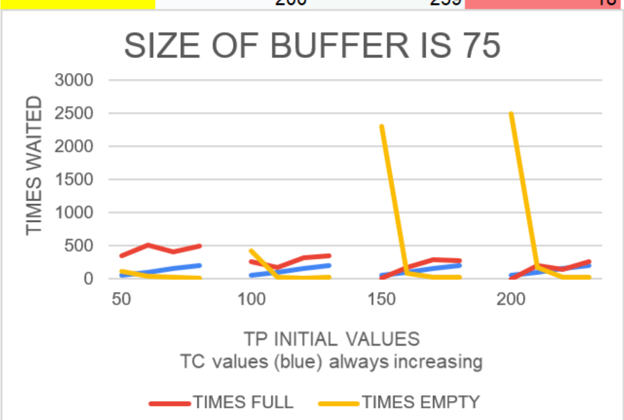
Producers control their speed of producing by changing the sleeping time range between two consecutive produce operations. Consumers sleeping time range remains constant throughout program execution, but consumers stop consuming when the queue is empty. The sleep time between calls to producers and consumers is a random value within a range (0, T). Producers dynamically change speed of producing by gradually slowing down producing when the queue is over 75% occupancy and stops producing when queue is 100% full. Producers also gradually accelerate the producing when the queue is below 25% occupancy and reaches twice the initial sleeping time range when buffer is empty. I was able to verify that the program is performing correctly by doing some tests as shown below in charts and tables.

TP	TC	TIMES FULL	TIMES EMPTY	TP	TC	TIMES FULL	TIMES EMPTY
50	50	867	222	50	50	493	133
	100	928	80		100	701	38
	150	704	32		150	675	34
	200	706	15		200	604	10
100	50	226	749	100	50	198	416
	100	680	75		100	457	84
	150	593	42		150	566	42
	200	544	19		200	475	15
150	50	11	2525	150	50	11	2504
	100	391	172		100	344	102
	150	446	62		150	408	32
	200	496	24		200	413	16
200	50	0	2526	200	50	0	2493
	100	169	338		100	183	238
	150	383	88		150	308	40
	200	405	23		200	303	6





TP	TC	TIMES FULL	TIMES EMPTY
50	50	347	108
	100	504	40
	150	408	20
	200	499	13
100	50	265	427
	100	170	30
	150	311	6
	200	347	17
150	50	10	2299
	100	170	90
	150	291	24
	200	277	20
200	50	0	2501
	100	195	166
	150	136	17
	200	259	18

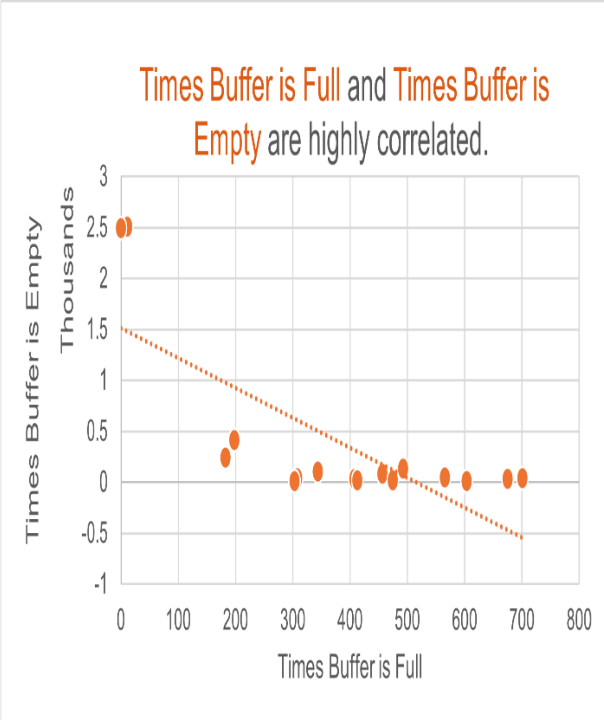


I tested the program by running with all combinations of the following datasets:

- TP = 50, 100, 150, 200
- TC = 50, 100, 150, 200
- Queue size = 25, 50, and 75 item max
- Timing random value of 1 to T Seconds for total 15 seconds per test

The results I came up with are consistent with the results that Dr. Li said we should get, and they make perfect sense logically. You can see the three photos of the table show the following:

When the starting TP value is lowest, and the starting TC value is highest we see that the queue gets filled to capacity the most out of all the different possible combinations of TP/TC values. This is because we have consumer threads waiting rand (1-200) milliseconds while producer threads always will wait less depending on the queue size. We also see the opposite happen when the values are reversed. This produces the expected results of a steady decline of both full and empty times as the maximum size of the queue is increased.



As you can see in this final graph the full and empty events all happen in pairs because they are dependent on the same combinations of producer and consumer starting wait times. During program execution if you extend the times to be exponentially larger you find that the randomness of the results will go away. This is interesting and I would be interested in diving more into this type of theory with further testing and changing the amount of increase and decrease that occurs during the events that are effected by the constraints.