Lab3 Report - Producer Consumer Problem

Michael Rokas - 20668160

Luke Seewald - 20663616

Collected Data

N	B	P	C	Average Time (ms) Processes	Average Time (ms) Threads	Standard Deviation (ms) Processes	Standard Deviation (ms) Threads
100	4	1	1	1.279266	0.361248	0.193502639	0.079077977
100	4	1	2	1.190868	0.313704	0.141506348	0.030893177
100	4	1	3	1.154732	0.322782	0.114221715	0.030149336
100	4	2	1	1.360598	0.33539	0.157620863	0.06943041
100	4	3	1	1.394128	0.32067	0.228294949	0.03131602
100	4	2	2	1.32003	0.285852	0.204402508	0.030775934
100	4	3	3	1.421572	0.30785	0.186634126	0.032814745
100	8	1	1	1.238836	0.39289	0.165648281	0.0627774
100	8	1	2	1.182594	0.304656	0.129783748	0.027396526
100	8	1	3	1.160688	0.310462	0.118142349	0.029130612
100	8	2	1	1.3373	0.32059	0.149887925	0.069769921
100	8	3	1	1.41634	0.299316	0.224585468	0.027020439
100	8	2	2	1.303162	0.281706	0.198526804	0.032719712
100	8	3	3	1.427558	0.306704	0.199726259	0.034181112
398	8	1	1	1.33999	0.479456	0.150203668	0.064677354
398	8	1	2	1.357598	0.528478	0.12764338	0.031445914
398	8	1	3	1.400166	0.586798	0.107525952	0.028424518
398	8	2	1	1.536028	0.565546	0.142776578	0.059465855
398	8	3	1	1.582538	0.571944	0.187441448	0.032773356
398	8	2	2	1.48852	0.396992	0.21690673	0.032159663
398	8	3	3	1.62687	0.417252	0.198251974	0.129735826

Importance of Averages and Standard Deviation

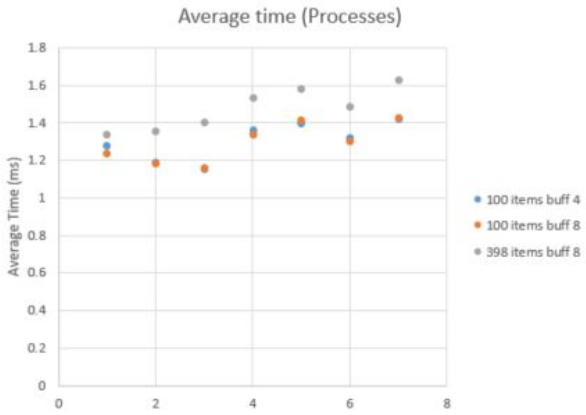

Figure 1.1

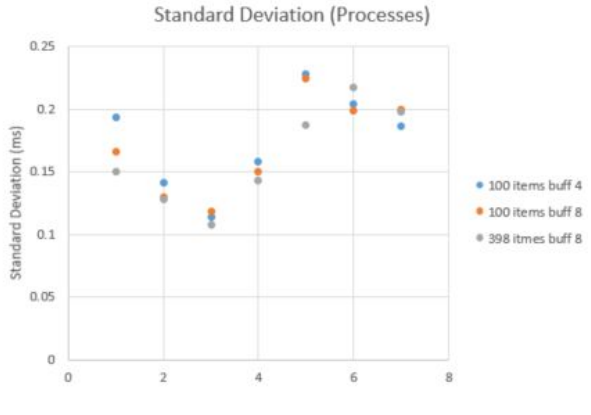

Figure 1.2

It is very important when running concurrency tests with both threads and processes that an average time with many samples is used. This is because both threads and processes are at the mercy of the operating system. It is difficult to determine when the operating system will give them CPU time to execute and how much time they will have before being switched out again (meaning they must wait for the CPU once more). This will have an affect on an individual run times especially if some tasks are waiting on other tasks to proceed. Taking an average shows the most likely result of a test case. It is also important to note that as the overall time increases due to the number of tasks or items to produce, the operating system will have more time to perform task switches which will add more variation to the results. This can be seen by comparing figures 1.1 and 1.2 and seeing as the average time increases or decreases the standard deviation closely follows. The values of standard deviation are also valuable because they show the range of values above and below the average that can be expected to be seen.

Effect of Number of Items and Buffer Size

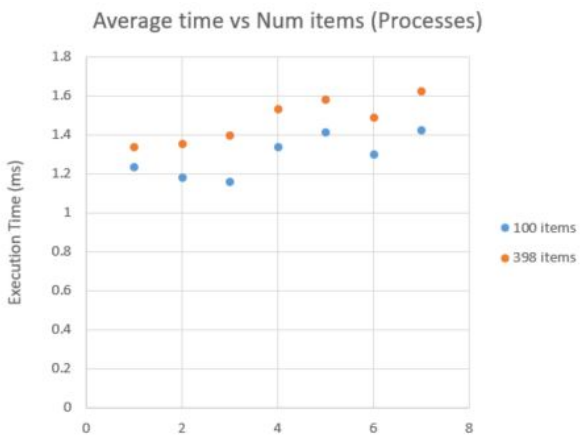

Figure 2.1

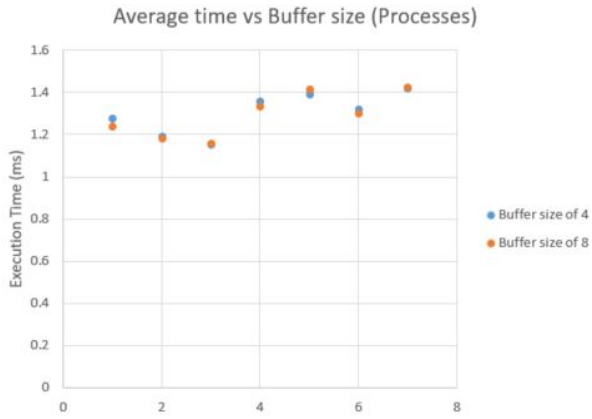

Figure 2.2

It is clear that as the number of items increases, the time of both the thread and process implementation increases, as there are more items which take more time to generate/consume. This is shown clearly for processes in figure 2.1, it can be seen for every duplicate test case, the test with 398 consistently took longer than the the test cases with 100 items. Alternatively, it can be seen in figure 2.2, the buffer size had little effect on the total execution time for the test cases run. The example provided is for processes but the results are similar for threads. The lack of affect from the buffer size is due to the fact that the slowest part of the system is the consumers, so when they go to access items there is always items there because the producers do not struggle to keep up. A larger buffer size could prove advantageous if we wanted to run a lot of producer tasks before spawning consumers, especially if the consumers are fast. This would mean that the producers could create a lot more data before being blocked and the fast consumers would more likely have data available. But in the test cases code implementation used, which was to spawn all consumers and producers simultaneously, the larger buffer size does not get taken advantage of.

Effect of Number of Consumers and Producers

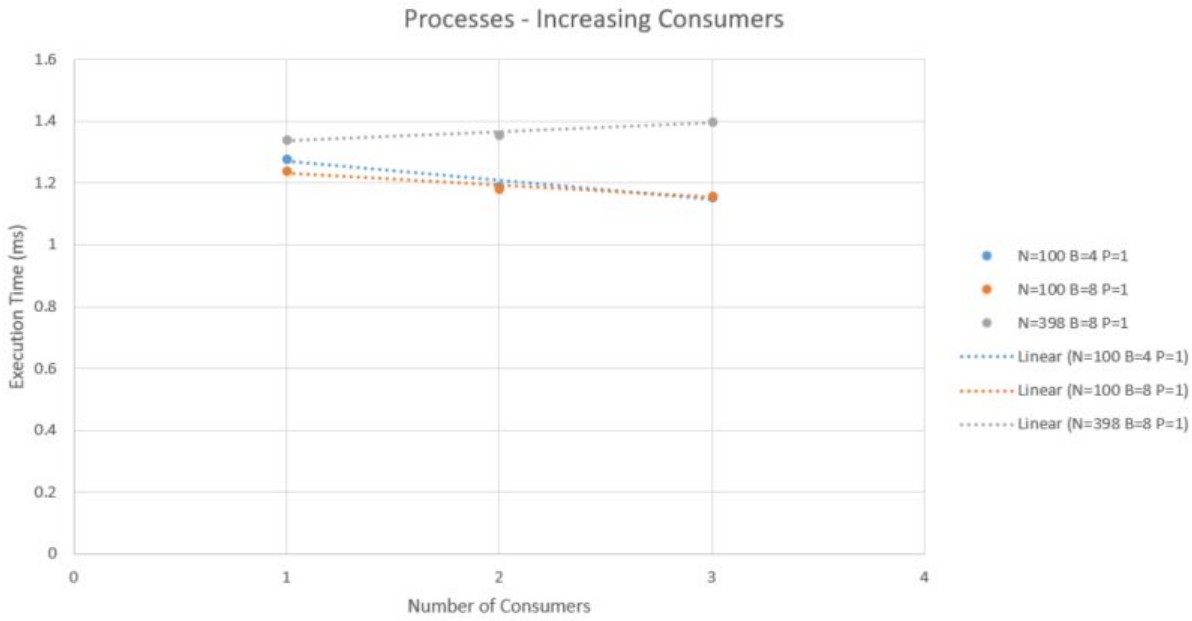

Figure 3.1

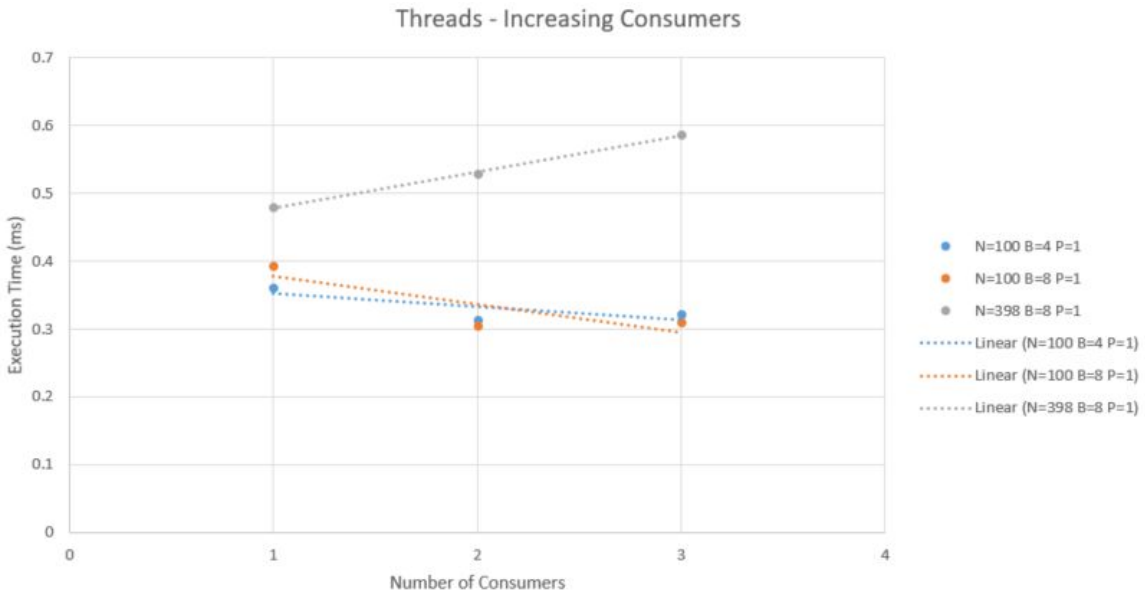

Figure 3.2

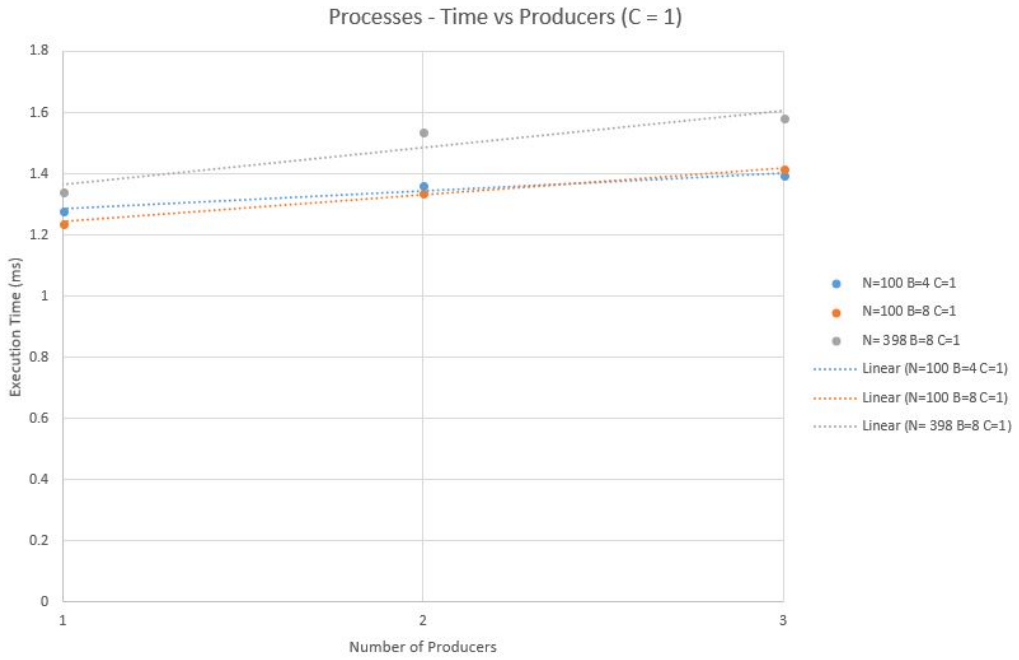

Figure 3.3

As seen in figures 3.1 and 3.2, as consumers increase with the number of producers remaining one, the time decreases (with the exception of the outlying set “N=398 B=8”). This is because as later discussed the consumers are the slower task so adding more of them in parallel will speed up the execution time. This holds for both the thread and process implementation. As seen in figure 3.3 when the number of consumers is one and there is an increase in producers there is no time decrease, instead there is actually a time increase this is because consumers are significantly slower due to their call to `printf()` which is a blocking call. This means that there is no advantage to adding more producers because the bottleneck is the number of consumers, yet there is still the cost of both the creation of more producers and the process or thread switching required for all producers to produce their designated values. This cost without benefit adds more time to the total execution of the program. This time increase is more significant for a process implementation as creation and switching is much more expensive for processes than threads. The ideal scenario is to have a ratio of consumers to producers such that consumers can consume at an equal rate that producers can produce, with this ratio achieved it is also ideal to have the max number of producers and consumers running such that the benefit of concurrency still outweighs the cost of process/thread creation and switching.

Performance of Threads vs Processes

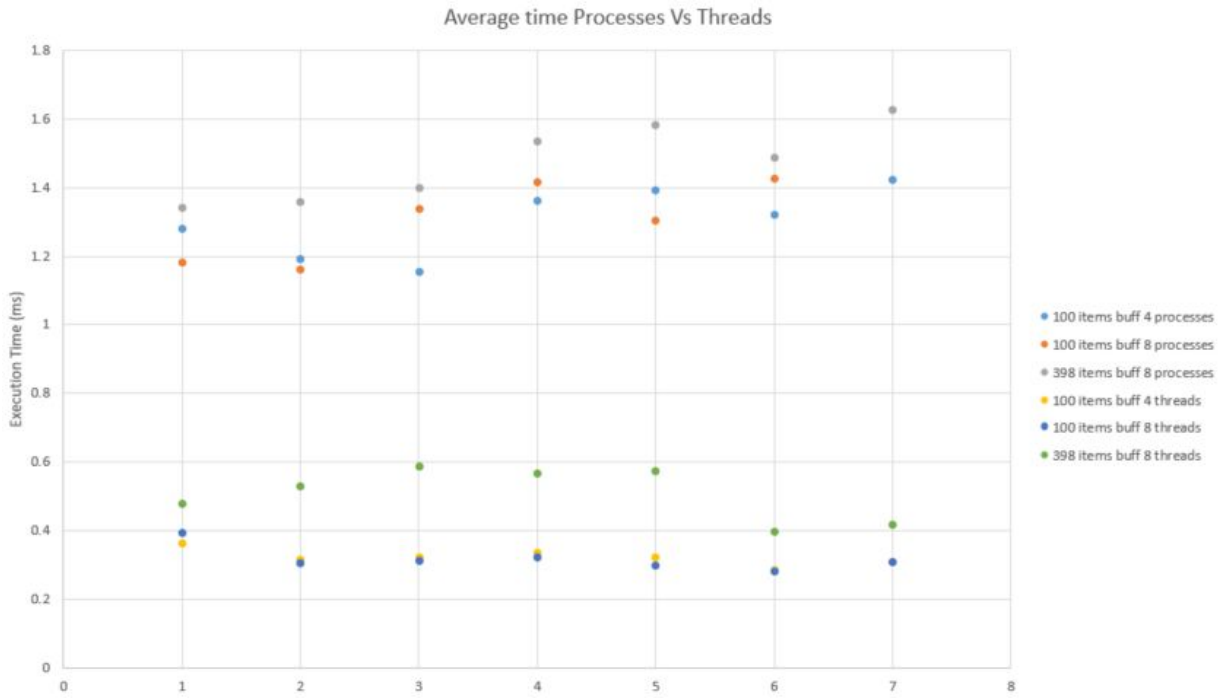

Figure 4.1

As seen in figure 4.1 threads offer a much stronger performance across every test case compared to processes. This is mostly due to the creation switching time being much longer for processes than threads. Which means that creating more producers and consumers takes more time for processes and everytime a producer or consumer is switched out by the operating system more time is wasted with a process implementation than a thread implementation. It is also worth noting that threads are communicating through shared memory which although needs extra logic such as semaphores and mutexes is still faster on average than a message queue as used for communication by processes. As figure 4.1 suggests both threads and processes react similarly to parameter changes but sometimes to various degrees. For example test six to seven switched two producers and two consumers to three of each. This caused an increase in time as the expense to create the new threads/processes was costly and there are not enough items needing to be created to take advantage of the extra consumers and get a return on the investment of creation. This change affects processes significantly more than threads even after accounting for the difference in scale. This is again do to the significant time increase to create a processes vs a thread.

Conclusion

As the data suggests in most scenarios threads will provide better performance than processes, This is, as discussed, do to the significantly reduced creation and switching time of threads compared to processes. Although threads offer performance up to an order of magnitude faster, there are scenarios where it would be ideal to run processes such as mission critical tasks that may have the potential to crash, a thread that crashes may crash the entire processes whereas if a processes crashes the other processes can still continue. These results will vary depending on the operating system and what other threads and processes currently running in the system, this will affect how much time the operating system dedicates to each resource/thread.