University of Waterloo

Faculty of Engineering

Department Of Electrical And Computer Engineering

ECE 254

Laboratory 3

Prepared by

Goel, Daivik

UW Student ID Number: 20649169

UW User ID: d3goel @uwaterloo.ca

2B Computer Engineering

and

Schmied, Ryan

UW Student ID Number: 20663175

UW User ID: rmschmie @uwaterloo.ca

2B Computer Engineering

19 November 2018

List of Tables

* Table 1: Average Execution Time and Standard Deviation of Processes and Threads

List of Figures

* Figure 1: Average Execution Time of Processes
* Figure 2: Standard Deviation of Processes
* Figure 3: Average Execution Time of Threads
* Figure 4: Standard Deviation of Threads
* Figure 5: Average Execution Time While Changing Number of Items Produced for Processes
* Figure 6: Average Execution Time While Changing Number of Items Produced for Threads
* Figure 7: Average Execution Time While Changing Buffer Size for Processes
* Figure 8: Average Execution Time While Changing Buffer Size for Threads
* Figure 9: Average Execution Time While Changing Number of Producers for Processes
* Figure 10: Average Execution Time While Changing Number of Producers for Threads
* Figure 11: Average Execution Time While Changing Number of Consumers for Processes
* Figure 12: Average Execution Time While Changing Number of Consumers for Threads

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| N | B | P | C | Average Time (ms) - Processes | Average Time (ms) - Threads | Standard Deviation (ms) - Processes | Standard Deviation (ms) - Threads |
| 100 | 4 | 1 | 1 | 0.485272 | 0.359254 | 0.071053403 | 0.11417776 |
| 100 | 4 | 1 | 2 | 0.469316 | 0.296296 | 0.038420205 | 0.03074183 |
| 100 | 4 | 1 | 3 | 0.504856 | 0.317794 | 0.168967498 | 0.03250852 |
| 100 | 4 | 2 | 1 | 0.485126 | 0.249406 | 0.048230801 | 0.03081385 |
| 100 | 4 | 3 | 1 | 0.536556 | 0.28796 | 0.052840655 | 0.0855923 |
| 100 | 4 | 2 | 2 | 0.479688 | 0.29296 | 0.040167034 | 0.07422047 |
| 100 | 4 | 3 | 3 | 0.581552 | 0.270394 | 0.034656302 | 0.11448563 |
| 100 | 8 | 1 | 1 | 0.482972 | 0.319314 | 0.067576588 | 0.08237851 |
| 100 | 8 | 1 | 2 | 0.470094 | 0.323972 | 0.039035588 | 0.06855992 |
| 100 | 8 | 1 | 3 | 0.496812 | 0.339896 | 0.047661941 | 0.0634619 |
| 100 | 8 | 2 | 1 | 0.488002 | 0.29674 | 0.047628206 | 0.06797532 |
| 100 | 8 | 3 | 1 | 0.53491 | 0.301712 | 0.052394522 | 0.06892954 |
| 100 | 8 | 2 | 2 | 0.485034 | 0.281174 | 0.04157101 | 0.06323578 |
| 100 | 8 | 3 | 3 | 0.57975 | 0.30411 | 0.033365124 | 0.29364384 |
| 398 | 8 | 1 | 1 | 0.699776 | 0.527722 | 0.075305762 | 0.06096547 |
| 398 | 8 | 1 | 2 | 0.800726 | 0.548536 | 0.046497472 | 0.07577261 |
| 398 | 8 | 1 | 3 | 0.78047 | 0.660728 | 0.04344446 | 0.0635981 |
| 398 | 8 | 2 | 1 | 0.825552 | 0.476838 | 0.05074386 | 0.08895064 |
| 398 | 8 | 3 | 1 | 0.851408 | 0.58288 | 0.049324249 | 0.06584702 |
| 398 | 8 | 2 | 2 | 0.664884 | 0.435864 | 0.035085019 | 0.07366088 |
| 398 | 8 | 3 | 3 | 0.81521 | 0.418988 | 0.048460436 | 0.17350999 |

Table 1: Average Execution Time and Standard Deviation of Processes and Threads

Average Time and Standard Deviation

Average times were used to make more accurate conclusions. This is because if a single output of the program was used for comparison, there could be a multitude of factors that result in erroneous answers. Running the program 500 times with the same input and taking the average results in a much more accurate representation of how the processes or threads are working. Averaging removes any outlining erroneous values that could be caused by externalities outside of our control and provide more accurate data. Standard deviation is calculated to show how consistent our data is. This is to show if any erroneous results are present and if so to adjust our analysis accordingly.

Figure 1: Average Execution Time of Processes

Figure 2: Standard Deviation of Processes

Figure 3: Average Execution Time of Threads

Figure 4: Standard Deviation of Threads

From Figures 1-4, it can be seen that processes are almost always slower in terms of average execution time although they have a lower standard deviation.

Affect of Number of Items Produced

Figure 5: Average Execution Time Changing Number of Items Produced for Processes

Figure 6: Average Execution Time Changing Number of Items Produced for Threads

As the number of items increase, we see for both threads and processes the average time increasing. This is because it would take longer for each implementation to produce and consume as there are more items to produce and consume thus increasing the average execution time. Figure 5 and 6 both illustrate this. In order to consistently check for both implementations we kept a constant buffer size and kept the increase of items the same. That way we can compare the two without the independent variables being a factor.

Affect of Buffer Size

Figure 7: Average Execution Time While Changing Buffer Size for Processes

Figure 8: Average Execution Time While Changing Buffer Size for Threads

It seems that the buffer size has no noticeable effect on the average time of the threads or processes. One explanation for this is that the consumer function is the slower of the two processes and thus although the buffer may be bigger, the numbers should always be available regardless because the producer will always have a number on the buffer before it is empty. Thus, the buffer size being greater does not affect it at all. This goes for both threads and processes as the consumer is slower for both.

Affect of Number of Producers

Figure 9: Average Execution Time While Changing Number of Producers for Processes

Figure 10: Average Execution Time While Changing Number of Producers for Threads

It can be seen in the above figures (Figure 9-10) that increasing the number of producers has opposite effects for processes than threads. For processes, when the number of producers is increased, the average execution time increases while for threads, the average execution time decreases as number of producers is increased. This is because the threads share memory and can therefore solve the producer consumer problem more efficiently working in parallel without having to interface with the message queue. Since inter process communication can be slow, increasing the amount of producers actually slows down the execution time for processes.

Effect of Number of Consumers

Figure 11: Average Execution Time While Changing Number of Consumers for Processes

Figure 12: Average Execution Time While Changing Number of Consumers for Threads

It can be seen in Figures 11 and 12 that increasing the number of consumers does not have a great affect on the average execution time for both threads and processes. This is because in the producer consumer problem depicted in the lab manual, increasing the number of consumers would not speed up how quickly the numbers being produced onto the buffer are consumed because consumers would simply be blocked more frequently waiting for producers. More consumers in turn leads to the buffer being emptied quicker although this does not necessarily lead to a quicker solution to the problem because the consumers will be blocked more frequently. This is why there is not evident benefit of increasing the number of consumers.

Advantages and Disadvantages of Threads and Processes

Threads and processes both are extremely useful although have their advantages and disadvantages and therefore have their own most applicable use cases. An advantage of processes is that they do not share memory and therefore are protected from each other. If one process corrupts some memory, all other processes will not be affected while for threads this corruption would affect all other threads that are executing since threads share memory. This advantage, along with the fact that processes can be run on different machines while threads must be run on the same machine, makes processes very applicable to online gaming scenarios. Although processes have this advantage over threads, they are at a disadvantage in terms of execution time and memory usage. Processes were proven to be slower compared to threads in every test run (referring to Figure 1 and 3) and since they do not share memory, they do in fact hog up memory as well. This is a huge advantage for threads over processes because of the significant speed up when using threads. If there is a restriction to one machine and corruption of memory is not a concern, threads are definitively the best choice due to the minimal use of memory as well as the execution time increase compared to processes. As mentioned above, corruption of shared memory of threads is a major disadvantage although an there is an advantage to having shared memory other than simply less memory being required. That is context switching between threads is fast due to the fact that theey share memoru