|  |  |  |
| --- | --- | --- |
| Number of Orders | Single Thread Processing Time (msec) | Multi-Thread Processing Time (msec) |
| 10 | 475 | 400 |
| 25 | 745 | 773 |
| 50 | 1294 | 1124 |
| 100 | 2046 | 1601 |
| 150 | 2875 | 2144 |
| 200 | 3562 | 2812 |
| 250 | 4498 | 3787 |
| 300 | 5385 | 3933 |
| 350 | 6662 | 4751 |
| 400 | 7305 | 5235 |
| 450 | 8216 | 5496 |
| 500 | 9077 | 6104 |

This table represents the number of orders processed and how much time it took for a single thread to complete the task compared to the multiple threads. I decided to make my table expansive enough where it would represent how fast my code could process a small number of orders compared to a substantial number. As you can see, in the beginning, when the number of orders to process is not large, single thread and multiple thread processing speed are similar. At processing 25 orders, single thread processing was slightly faster even! This is because of the synchronization that I had formed to help my code write a client report by reading the list of items that is shared by all clients. This synchronization will have slowed down the processing a little bit and made it closer to the processing speed of a single thread since my code’s single thread also processes and writes each report one by one. The eventual change in multiple thread processing speed occurs since I refrained from synchronizing the processing of a client’s purchased items because that would not be shared or accessed amongst other threads. I attempted to leave out as much processing as possible from synchronization to make sure my multiple threading processing was most efficient. Since processing purchased items allows my code to access only one file associated with the client, I do not have to worry about data races where multiple threads have access to the same purchase order. Each thread is associated with an instance of the Client class since each client has their own individual order and file to read. As the number of orders to process increases, we can eventually see a clear difference in the processing speeds and it was shown that my code could process 500 orders at a speed of 6,104 milliseconds, while a single thread could process those at 9,077 milliseconds. This would be inferred to be because a single thread can only read and process one file at a time and write a report for it one by one. However, with multiple threads, multiple threads can alternate with each other when reading files that are incredibly large. Some threads will read a file faster than others and that is when multitasking will really come in hand to increase processing speed.