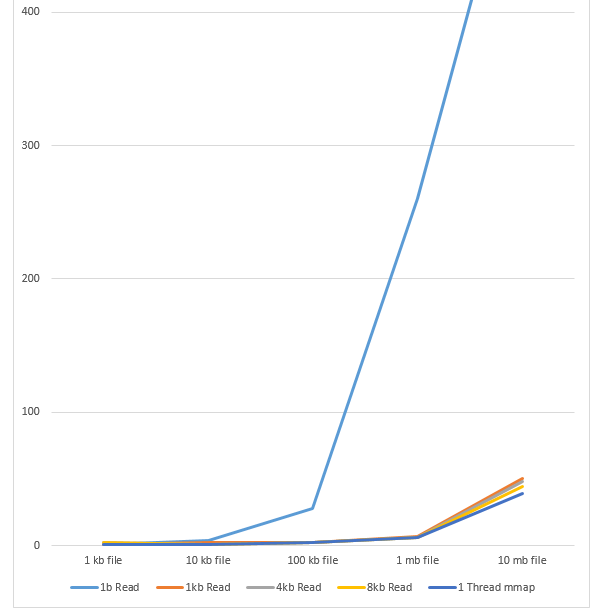
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1 kb file | 10 kb file | 100 kb file | 1 mb file | 10 mb file |
| 1b Read | 1ms | 4ms | 28ms | 260ms | 2582ms |
| 1kb Read | 1ms | 2ms | 2ms | 7ms | 50ms |
| 4kb Read | 2ms | 1ms | 2ms | 6ms | 48ms |
| 8kb Read | 2ms | 1ms | 2ms | 6ms | 44ms |
| 1 thread mmap | 1ms | 1ms | 2ms | 6ms | 39ms |
| 2 thread mmap | 1ms | 1ms | 2ms | 5ms | 40ms |
| 4 thread mmap | 1ms | 2ms | 2ms | 5ms | 43ms |
| 8 thread mmap | 1ms | 2ms | 2ms | 5ms | 53ms |
| 16 thread mmap | 1ms | 2ms | 2ms | 5ms | 52ms |



In order to test the performance of the different file reading systems implemented in homework four, I created 5 files ranging from 1 kilobyte to 10 megabytes and ran a test of each type on each file. All of these tests were run on a virtual machine on my laptop, configured to have one processor. Unfortunately, this virtual machine does not give accurate readings on major page faults, so all of the data I collected was purely wall clock times. For the most part the results of this test are unsurprising; however, there are a few unexpected results.

First, each of these files were run through the standard functionality of this project from phase one, involving 4 read commands (configured at 1b read, 1024b read, 4096b read, and 8192b read), and the basic, single threaded memory mapped input. For the most part, these files were all read without much trouble. The outlier in this group is the 1b read. Because this read system reads each character one by one, the read method is called exponentially more than the other options. As a result of this, the slowdown when switching to the 1 byte read is so drastic that the data doesn’t even fit on the graph! While the jump in speed between other buffer sizes is not nearly as large as this one, there is clearly a correlation between the buffer size and the file reading speed. In every case, the 8kb read was either tied or better than the 4kb and 1kb options. The comparisons between these four input methods and the memory map system are not as obvious, as memory mapped input is handled completely differently. While there is more initial setup before the memory mapped system can be used, afterwards there is never any need to read in more data, as the entire file is already mapped to memory. This allows the memory mapped system to perform either on par or better than the 8kb read option, as it effectively creates a buffer of whatever size is needed, preventing the need for any more read commands once the searching has started.

After running these five tests, I ran four more to test the multithreading capabilities of the memory mapped system. Unfortunately, this virtual machine is slightly restrictive in that it does not have nearly enough cores to take advantage of many of the higher thread counts, causing strange results. The dual threaded option shows almost no change when compared to the single core solution. This is because the machine does not actually have the processing power to run two cores at once, so they still run as if single threaded. However, once more than two cores are added, the program actually begins to slow down. Creating and waiting on a larger number of pthreads causes more overhead, which slightly increases the time these solutions run for. However, the bulk of this slowdown presumably comes from the context switches needed to run up to 16 threads. Because this machine doesn’t actually have multiple processors to run the threads concurrently, it instead just constantly switches between the threads on one processor, causing noticeable slowdowns for the same effect.

The ideal setup for this would be the memory mapped input running multithreaded on a machine that actually has enough cores to run the threads concurrently. This is clearly shown by the fact that the memory mapped solution is faster than any of the normal read solutions, and because obviously more cores running concurrently enables faster processing. However, given my restricted conditions on a virtual machine, the best option is the normal, single-threaded memory mapped input. This solution gives all of the benefits of memory mapped input without any of the drawbacks of the increased number of context switches or overhead of creating threads.