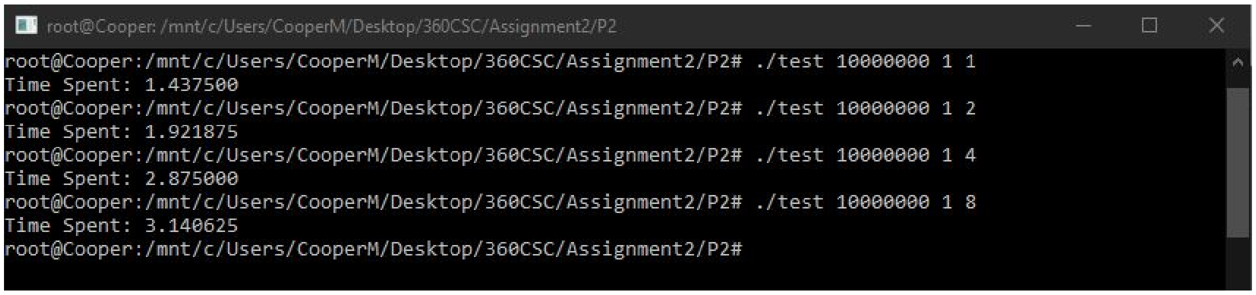

```
root@Cooper:/mnt/c/Users/CooperM/Desktop/360CSC/Assignment2/P2# ./test 10000000 1 1
Time Spent: 1.437500
root@Cooper:/mnt/c/Users/CooperM/Desktop/360CSC/Assignment2/P2# ./test 10000000 1 2
Time Spent: 1.921875
root@Cooper:/mnt/c/Users/CooperM/Desktop/360CSC/Assignment2/P2# ./test 10000000 1 4
Time Spent: 2.875000
root@Cooper:/mnt/c/Users/CooperM/Desktop/360CSC/Assignment2/P2# ./test 10000000 1 8
Time Spent: 3.140625
root@Cooper:/mnt/c/Users/CooperM/Desktop/360CSC/Assignment2/P2#
```

Execution times of running Strikingloo's MapReduce on an input of 10,000,000 items utilizing 1, 2, 4, and 8 threads.

Evaluation Results:
This code was run on my desktop which is rather outdated but still very powerful. The CPU is an i5-4670k (4 cores/threads, 3.4GHz). This means that, in theory, the task would probably benefit most from using 4 threads but we can see that it is fastest with only 1 thread. From what I have read, this is for two reason:

1. I am running the code on the Windows Linux subsystem which is a virtual environment that is run on only one core of my CPU, making 1 thread the most efficient approach
2. The pthreads are being run at the user level instead of the kernel level and this would also limit the program's utilization to only one core

The reason that more threads results in greater execution time is because the time it takes the system to perform the (expensive) context switches outweighs the benefits gained from splitting computations between the threads. This might change with larger datasets, but I encountered crashing issues when trying to work with more than 100,000,000 items.