**Code Instructions**:

**Compile-**

To compile the code, run these commands in bash, with the current directory set to the folder in which the files reside.

g++ -o3 -w lcovingtAssignment1.cpp -o lcovingtAssignment1

IE the terminal will look something like this.

user@computer:~/folder/Assignment1$ g++ -o3 -w lcovingtAssignment1.cpp -o lcovingtAssignment1

**Run-**

Next, to run the code utilize the following format:

./lcovingtAssignment1 filePath numberOfChildProcesses

**Example Commands-**

./lcovingtAssignment1 file1.dat 1

./lcovingtAssignment1 file1.dat 2

./lcovingtAssignment1 file1.dat 4

./lcovingtAssignment1 file2.dat 1

./lcovingtAssignment1 file3.dat 1

./lcovingtAssignment1 file4.dat 1

**Summary-**

**Table 1**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | File 1 | File 2 | File 3 | File 4 |
| 1 Process | 0.0076s | 0.0107s | 0.0363s | 0.1453s |
| 2 Processes | 0.0063s | 0.0076s | 0.0273s | 0.1303s |
| 4 Processes | 0.0063s | 0.0123s | 0.0296s | 0.1013s |

The results of table one were quite interesting. They were obtained by adding the “time” command to the run command for the assignment in the terminal. Then the times reported in the table are the real values reported from that time command. Each value reported in the table is an average of 3 timings made. This command tracks the execution time of the entire program, which doesn’t make it perfectly accurate. Before seeing the results, one would expect that execution time would decrease as more child processes were created to take advantage of parallelization. For file 1 it did not make a tremendous difference, but adding the children did decrease execution time although there wasn’t a difference in having two or four children. For file 2, having two children processes did prove to yield a decrease in execution time, however, for running four children, the resulting overhead of creating and switching between the processes proved more time intensive than the gained parallelization was worth. The same result was true for file 3 as well. For file 4, it is evident that having more child processes did decrease execution time by quite a bit. The time saved by the parallelization was greater than the overhead was for the parent to pipe and fork to the children.

In this project I learned how to read a file of integers line by line in the c language. I learned how to set up pipes between parent and child processes so they could communicate. I learned that for smaller data sizes the time spent for over head of setting up parallel processes may be better spent serially processing the data instead I learned there can be a sweet spot in how much parallelization is set up vs how much data there is to process. Generally I learned that for increasingly larger sets of data the more parallelization you create the better, and I learned what can create exceptions to this general rule of thumb. I am taking csci 415 as well and learned about how to multithread an application. It was very interesting to learn how to program multiple processes rather than threads in this assignment.

**References**

I ran into compiler errors trying to convert strings to ints using the std::stoi function, what I used in assignment 0. This post gave me insight to use string streams to convert a read string line from the file to an int in my initialize function.

<https://stackoverflow.com/questions/5476616/stringstream-string-to-int>

The ordering of values passed in read and write were used from the provided lecture slides.

Lastly, I attained help from the grader on lab day. He helped me fix my read and write function calls and overall debug my program.