Havan Patel(hp397)

Charmian Goh(csg102)

Project2(Readme & Analysis)

How to compile:

1. gcc –o mergesort.c mainThreadProcess.c sorter -pthread
2. ./sorter -c column name -d thisdir -o thatdir

   If there are from 1 to 1024 csv files that has 20 lines works fine
   However, if there are from 50 to 1024 csv files that has more than 5000 lines of data it for some weird reason says “killed” on the terminal or it just segfaults. Don’t know why what the reason could be for that.

How it’s Programmed

1. Whenever the program finds a directory or csv files it will create threads for each of them. Once the csv is found that particular thread is going to put the data into a temp movie struct and will be passed into a global movie struct pointer. Once each csv thread is done with the putting data in to original global array. We merge sort the entire global movie struct array and put it inside a path if it’s provided with -o. If -o is not provided we put the sorted file into current directory or the root directory. However, this code will ignore any csv files with “sorted” as name of csv.
2. The concept for this programming is very similar to project 1 as most of the stuff was copy/paste from previous project.

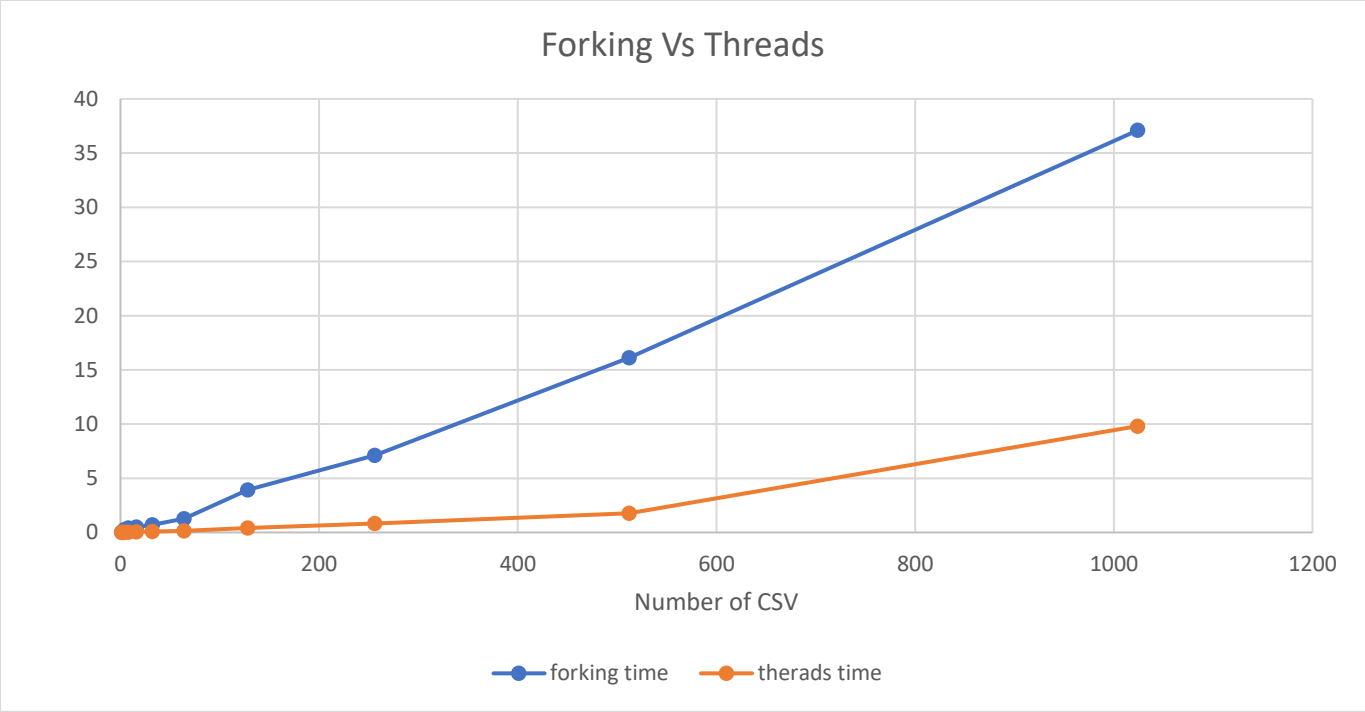
Name of the sorted file

Name of the all sorted output file is called “Sorted.csv”

Analysis

The analysis is done on files with 20 lines in them. Below is the analysis for csv files with 20 lines

Number of csv	Fork(REAL TIME)	Threads(REAL TIME)
1	0.013	0.009
2	0.032	0.011
4	0.250	0.019
8	0.42	0.024
16	0.507	0.054
32	0.701	0.097
64	1.27	0.151
128	3.938	0.42
256	7.109	0.826
512	16.109	1.763
1024	37.1	7.814



Analyzing Data Above

The linux time command utility tells that multi-threading programming is faster program than multiprocessing(fork). Reason being that in our code we put all sorted information in different files as we encounter many csv files. Instead sorting single large file. The amount of times merge-sort is called by the fork process and fopens for output files this lead to slower program in comparing to the threads. The wait function, which requires processes to be active until the child processes die also, can have a minimal effect on run time. The main differences in time is how multi-processing takes more time than multi- threading regarding csv's. The differences between these two programs can also be explained by the different factors as stated above. way to make processing faster than threading is to switch the outputs of the two programs. By doing so the run time of both programs should be drastically different. This will involve multi-processing being faster than multi-threading. I would say as well that the merge-sort algorithm is the best algorithm for a multi-threaded sorting program as it can be easily parallelizable, quicksort is also possible to parallelize but parallelizing quicksort is more difficult than merge-sort on a common basis. It may differ depending on data distribution, but for the most part merge-sort is the right option for multi-threaded programming.