Jon Cobi Delfin and Jeffrey Pham
Systems Programming
Project 1: Multiprocess CSV sorter

Design, Assumptions, and Difficulties

At first, we did not have much experience with navigating through directories or forking to create multiple processes. Professor Francisco helped in outlining what he would do if he were to implement this project and we followed his outline. First, we worked on creating a simple program to navigate all the directories/files starting at ".", and to print out there names. Once we had that working, we implemented it into our code to either start at a user specified directory (if the user specified directory does not exist, it will let the user know the directory cannot be found and exit the program) and if there was not a specified directory, to start at the directory the program was called in using getcwd().

From there, any directory or file it ran into, it would increment an ongoing count and print out the PID so that we could keep track of the number of processes. If it read another directory, it would recursively call itself to search into found directory. If it read a file, our program sent it to a file handler that would check to make sure it was a CSV file that was not already sorted and sent that file to our mergesort.

To functionalize our mergesort, we changed it to take in the filename, the column name to be sorted, and the name of the output directory. Therefore, instead of reading from stdin like in project 0, we utilized open() to open the filename sent to mergesort in read only mode. From there, most of the code was the same in terms of splitting the file into tokens and sorting it on the column name specified. Another change was that since we weren't going to print to stdout anymore, we had to create a new filename that followed the format of <filename>-sorted-<column name specified>. Once we had that created, we used open() on that filename and made it so that we could write to it and printed the sorted output to the file.

Within main, we changed all the input error checking since there were more inputs to account for. At the minimum, the program needs '-c' and the column name that the user wishes to sort on. From there, it is optional for the user to either specify '-d' and a directory they wish to start from and/or '-o' and a directory they wish to send all the sorted csv files to. The first 2 parameters must be '-c' and a column name, but the following parameters may be in any order.

Testing

We tested our code using the test cases from the last project. We started off just with the minimum of '-c' and a column name to make sure it created a new csv file with the correct format and correct sorted data in it. We made sure it would place those new csv files within the directory that the original csv files were found.

We then moved on to specifying a starting directory and after that we made sure we could specify an output directory. The last step was to test for errors while sorting the csv files which includes: the file not being a proper movie csv file (does not contain the correct headings), the csv file has more headings than columns, the csv file only having one row, the column name to be sorted on is not found in the csv file. We made sure all the error statements printed out to stderr and that no new csv file was created if such errors were found.

Using Our Program

We have included a makefile, so please call make in order to create the executable named simpleCSV sorter. From there, you may run it in the following fashion:

No specified starting directory or output directory, sorted csv files will be placed in the directory the original files were found in:

./simpleCSVsorter –c <columnName>

Only starting directory specified, sorted csv files will still be placed in the directory the original files were found in:

./simpleCSVsorter –c <columnName> -d <startingDirectoryPath>

Only output directory specified, sorted csv files will be placed in the output directory:

./simpleCSVsorter –c <columnName> -o <outputDirectoryPath>

Starting directory and output directory specified, sorted csv files will be placed in the output directory:

./simpleCSVsorter –c <columnName> -d <startingDirectoryPath> -o <outputDirectoryPath>

OR

./simpleCSVsorter –c <columnName> -o <outputDirectorypath> -d <startingDirectoryPath>

If you are expecting an error in any of the csv files, PLEASE include 2>nameOfErrorLog.txt at the end of the input so that the error statements are not printed in the terminal itself. The reasoning for this is that we made it so that the error statements print out the name of the csv file that the error occurred on, so we could not use write() as the name of the csv file changes from one to the next and we instead used fprintf() to stderr, which will also print to the terminal if not redirected. The input would look like the following:

./simpleCSVsorter –c <columnName> 2>errorLog.txt

./simpleCSVsorter –c <columnName> -d<startingDirectoryPath> 2>errorLog.txt

And so on.