How to use the scannerCSVsorter.c:

In terminal: make

creates multiThreadSorter (with -lpthread, which is necessary for threads)

Clean:

Erases executable

multiThreadSorter.c complies with three other files: **multiThreadSorter.h**, **sorter.c**, and **sorter.h,** and **global.h**

multiThreadSorter.c:

This is the driving program for the assignment. This is the one complied by the makefile, and the one that is called to execute the processes.

Takes in arguments from the command line:

./multiThreadSorter -c <column-name-to-be-sorted-on>

Optional: -d <directory-path>

Sorts from the given directory. If not given, starts at current directory.

Optional: -o <output-directory-path>

Creates sorted files in output directory. If not given, creates sorted file at in the same location as the unsorted file.

Flags can be in any order.

Functions:

Main: checks to make sure the command line call is correct.

Calls navDir from multiThreadSorter.h

Prints initial PID and total number to stdout

Calls MergeSort on the global linked list

Prints the sorted list to the final file

multiThreadSorter.h:

This is the header file for multiThreadSorter.c, and contains one function:

void **navDir:**

Takes in a struct with the following data:

Takes in the fileName, the column to be sorted on, the name of the output file

Prints thread IDs when necessary, locking std out when doing so with a mutex

This function spawns threads on every subdirectory or file found that is not . or ..

Recursively calls navDir for subdirectories

Calls add_file_to_list for files

sorter.c:

This is the file with the main code from PA0, has one function:

void **add_file_to_list:**

Takes in the filepathand the column of the key to be sorted on.

Calls testCSV to see if it is a valid file

Reads the data from the csv file, storing each line in a struct, as well as the value of the key to be sorted on.

Adds all nodes to the global linked list, locking it with a mutex when doing so

sorter.h:

This is the header file for simple.c with helper functions/definitions.

int **testCSV:**

Takes in the filepath, and the key value to be sorted on.

Tests to see if the file is valid to be sorted.

If the file is not a .csv, does not have a valid csv format, or if the file is already all files sorterd, then it will return that it is not a valid file.

int **column_num:**

Takes in a string and returns the column it belongs on

float **parseInt:**

If the value is numerical, this returns the numerical value of the data sent to it

If no value, returns a -1 to put at beginning of list

char * **removeWhite**:

Takes in a string key, removes trailing and leading white space

Returns trimmed key

struct **datanode * merge**:

The merging function for mergesort, driven by merge

Recursively calls

Takes in pointers to the beginning of each list

Also takes in if the string is a numerical value, to call parseInt for comparing

Returns merged list in ascending order

struct **datanode * merge**

Takes in the start of an unsorted linked list

Calls merge to merge sort the list

global.h

Stores the global variables/struct definitions

Includes: Struct for the data

Struct for thread IDS

Struct for Arguments to navDir

Struct for Arguments to add_file_to_list

Int t_counter: counts number of threads

Int finished: Marker to ensure main doesn't end before threads

pthread_mutex_t list_lock: mutex to lock global linked list of data

pthread_mutex_t out_lock: mutex to lock stdout

pthread_mutex_t counter_lock: mutex to lock thread counter

pthread_mutex_t check_lock: mutex to lock a check within the code

list_end_pointer: pointer to the end of the global linked list to append onto

Difficulties:

1. Learning how to effectively use threads with correct syntax
    i. Took some time, but was able
2. Reading to correct directories, whether absolute or relative

    1. Solution: Constantly have to carry the directory path and output directory path. If the output directory is the same location, we then have to append onto its path whatever subdirectory the thread is in (figured out in Asst1.)
3. Being able to lock the list and append to it correctly without messing up null pointer
4. Being able to return data from a thread to print the data for every subthread
    1.Solution: pass a pointer in the arguments and store the data at that pointer.

Test Cases:

With a directory named test, we have included multiple subdirectories to ensure the program correctly spawns threads for those. We also included invalid csv files that have a .csv extension to ensure the program correctly flags them as invalid, as well as files that do not have a .csv extension to ensure the program flags them to not be sorted as well.

With these test cases, we called the program using only the -d flag, only the -o flag, neither, and both to make sure files were sorted and created in the correct places.

Comparison to Asst1:

See analysis.pdf