Sarah Khasawneh
Alan Sempruch

ReadMe

Usage:

```
./sorter -c <column> -d [starting directory] -o [output directory]
```

<column> - column name to sort by
[starting directory] - directory from which the program will start searching for csv files (by default it is root of program)
[output directory] - directory to output the final sorted file to (by default it is root of program)

Design: The program is similar to that of project 1, however instead of spawning processes, it spawns threads. For each directory found, a new thread is spawned to traverse that directory while the parent thread continues on. In the case of a csv file, a thread is spawned calling on our csv handler function which inserts the data from the file into our shared global data structure, which requires a mutex lock, and sorts it. After all directories have been traversed, mergesort is called on the entire data structure, resulting in one large sorted file.

Assumptions: The assumptions made were that the csv files were properly formatted if the first row is properly formatted.

Difficulties: By far the largest difficulty we experienced when porting project 1 to a multithreaded version was managing the use of global and allocated variables. Since all threads share the same heap while processes do not, we had to ensure that certain global variables where made local and passed as parameters to avoid multiple threads using one variable that is supposed to be unique per thread. We also had some difficulties on waiting on all the threads. We solved this problem by storing thread TIDs in an array.

Description:

Project 2: Basic Data Sorter - Multithreaded involves taking in large data files and spawning threads in order to sort and merge into one large mass file that contains all the entries sorted by the respected column given in the parameters. The program is given at least the category the user would like all the files to be sorted by. The user may also choose which directory to start in, by default if they do not chose to give a directory path, the current directory the program is in will be root. Also, the user may choose where the output file goes. By default if they choose not to include it, it is written in the root directory that is being worked from. The program begins by the traversing through the root level. If it locates a directory, it creates a thread and continues in both new directory and current directory. If it finds a CSV file, it creates a new thread and sorts the file using the merge-sort sorting algorithm. After sorting and traversing until no more, merge-sort is called on the global array and sorts all the files in order and outputs it into one mass file. If it happens to find a nin-CSV file, it does not create a thread and continues traversing through the directory.

To stdout, it prints the Initial Process ID, as well as all thread ID's, and total number of threads created.

Sarah Khasawneh
Alan Sempruch

- Header File :
    - typedef struct movie_{ <all 28 categories are in here>}movie.
        - This is the struct in which all lines are identified /input/ and sorted into as each line is traversed.
    - typedef struct fileParams_{char * fileName; FILE *fp; char* d;}fileParams;
        - This is the struct that is passed into each pthread_create function. It contains the parameters for each file that is being sorted, a filepointer to work through the file, and a char pointer that keeps track of the working directory per thread.
    - char **getString(movie** info, int entry, int element);
        - This function works through each string that needs to be stored in the struct. It takes in the info array, the entry number, and the element number of where it belongs in the struct.
    - getLong *getInt(movie** info, int entry, int element);
        - This function works the same as the getString function but on integers found in each line.
    - getFloat(movie** info, int entry, int element);
        - Same as the getString function but for floats found within each line.
    - void insert(char *line);
        - This function takes in a line and inserts it into the array.
    - int getKey(char* word);
        - This function takes in a char* and find the word/ element number location within the struct.
    - void mergesort(movie **array, int a, int b, char *word);
        - This is the merge-sort function that sorts the input array/ file. It takes in the beginning index, the end index and the category being sorted by.
    - void merge((movie **array, int a, int b, char *word);
        - Recursive function that breaks up the array into smaller arrays before calling the merge-sort function.
    - void print(movie** info, int numOfEnteries, char* fileName, char* path):
        - This function prints the array into the output file.
    - void traverse(void *d);
        - Function takes in a char* directory and traverses it.
    - void csvHandler(void* params);
        - Handles all CSV files found within each directory by inserting the contents into our shared data structure and then calling on mergesort.