Lawrence Yong, Danny Rivera

INSTRUCTIONS

To use this application, follow the correct command line instructions

1. Build the application: **make**
2. Execute application: **./sorter -c <column\_name> -d <input\_directory> -o <ouput\_directory> -h <host\_name> -p <port\_number>**

Command line arguments can be in any order.

* 1. <column\_name> is the name of the column in the csv by which to sort the data
  2. <input\_directory> is the directory path to search for csv files
  3. <ouput\_directory> is the directory path that sorted csv files will be saved to
  4. <host\_name> is the name of the server you are trying to connect to
  5. <port\_number> the port number that is shared between the host and client

Design:

We followed the same structure as the previous assignment with the use of recurseDir() that searches the given input directory. For every directory entry (dirent) that it finds, it checks to see if the entry is a regular file or a directory, DT\_REG or DT\_DIR, respectively.

If the directory entry is DT\_REG, a regular file, then the file extension is checked to see if it is a .csv file. If it is, a socket is constructed and the arguments of conServ\* args are casted and passed onto pthread\_create() along with the clientToServer function pointer. Unlike the previous assignment, the client sends a message to the server via the function pointer and the server is in charge of sorting the message with the data and sending it back to the client. The sending of messages through this function pointer is mutex locked in order to control the traffic of messages.

However, if the entry found is a DT\_DIR, a directory type, the same procedure is used from the last assignement where the recurseDirArgs arguments are casted and passed along with pthread\_create along with the recurseDir() function pointer. This continues recursively until all the directories of the input directory have been traversed.

If no input directory is given, then the input directory is the current working directory. If no output directory is given, then the output directory is the current working directory.

DIFFICULTIES

DIFFICULTIES

While writing this code we needed to make sure we didn't just use the name of the entry found in the directory, but its absolute file path, both for regular file entries and directory entries. We solved this by prepending the path of the directory that the entry was found in to the name of the entry itself. We also needed to make sure that we did not call recurseDir() on the "." and ".." directories or else we would have fork bombs that would crash the system. Throughout this assignment we had troubles with several memory leaks due to complications of threads trying to access memory that had been freed from other threads. And we also had trouble with correctly sending/receiving messages between the client and server as well as segmentation faults and infinite loops

ASSUMPTIONS

We assumed that all csv formatted files have columns in similar orders. Assuming the command line argument for port number is a number and that the port number doesn’t exceed the max or is less than the min for port number values.

HEADER FILE

* typedef struct \_regFileArgs
  + arguments/data for regular files: char\* filePath, char\* targetColName,

int (\*mergeData)(Record\*\*, int, int);

* typedef struct recurseDirArgs:
  + arguments to be passed on to recurseDir() : char\* tarColName, char\* inputDir, char\* outputDir
* typedef struct conServArgs:
  + arguments to be used for message sending/receiving
  + char\* dataToSort
  + char\* colName
  + char\* action
  + int socketDesc
  + int collecId
* char\* outputFile
  + holds the string name of the output file.
* Record\*\* gRecs
  + holds all the records as a global 2d array.
* int gNumOfRecs
  + keeps track of the number of recs encountered in total.
* pthread\_t\* gTid
  + global array of thread ids.
* int gNumOfTids
  + keeps track of number of thread ids produced in total
* pthread\_mutex\_t rec\_lock & tid\_lock
  + used to regulate access to gRecs and gTid global variables
* enum DataType
  + represents type of data being stored in the field, using for data interpretation during sorting
* typedef struct Field
  + represents a column in a record
* typedef struct Record
  + represents a row in a csv file
* short determineTypeofData(char \*dataString)
  + takes in a string of data as an argument, returns a number representing DataType
* void trim(char \*string)
  + trims trailing and leading whitespace from given string
* void mergeSortInt(Record\*\* records, int numberofrecords, int columnIndex)
  + sorts a two-dimensional array of records, requires the number of records, and the index of the column to sort on within a record’s field array
* void mergeInt(Record\*\* leftSub, Record\*\* rightSub, Record\*\* records, int lengthOfLeft, int lengthOfRight, int columnIndex)
  + takes in a left and right sub array of records which merges the two subarrays together into the larger array of records
  + the column index is used to find which column in a record to sort on and it uses the appropriate comparison algorithm dpending on the type of data of that field
* void printArray(Record\*\*records, int numberofrecords, FILE \*fptr)
  + prints all the records in a 2D records array, in csv format
  + fptr specifies the file pointer to the file that the sorted data will be saved to
* Record \*csvToRec(char \*rowString)
  + Converts a csv formatted string representing a row in a csv file to a Record
* int sortCSV(char\* inFileName, char\* inCsvStr, char\* tarColName, FILE\* fptr)
  + sorts a csv given the name of the csv file, its contents, the column to sort by and the file pointer to the file to write to when finished sorting
* void freeRec(Record\* rec)
  + frees the memory allocated to one record
* char \*readCmdArgs(int, char\*\*)
  + Reads in and concats all command line args
* char \*readSTDIN()
  + Reads in and concats data from STDIN
* char\* readFile(char\* filename)
  + returns the contents of a file specified by the given filename
* void clientToServer(conServArgs \*)
  + used for the sending and receiving of messages from client to server
* int readSocket(int, char \*\*)
  + used to read the contents of a specified socket
* xmlparse file
  + used for parsing xml formatted messages from client to server and back.

.