sorterPart3

Group Members:

Srihari Venkatesan - sv443

Boris Onufriyev - beo19

Abhishek Arya - aa1488

Ryan Lee - rl630

Project Logic

This is the general flow of logic in the sorter:

1. Check params.
2. Get directory that is to be sorted.
3. Call the traverse() function on the directory.
   1. Open the directory.
   2. While there are more files in the directory:
   3. If a folder is encountered, a new thread is created to call the traverse() function on the subdirectory
   4. If not, the traverse() function continues and gets the absolute path of the file.
   5. If the file is a CSV, a new thread/socket is created which executes the sortFile() function. This will send the contents of the file to the server, which then stores the file into a global array
   6. The IP address is printed out for each connection
4. When the directory has been completely traversed, the client sends the server a dump request. The server then returns the contents of the final sorted file
5. The client then prints this sorted output into the final output file.

Project Methods and Structs

- Struct: Node. Contains the id of the node, the value of the cell and a pointer to the next node.
  - *Note* There are 3 different variables for the value of the node. Two of them will always be null, depending on what type the value is.
- Struct: trav_param. Holds the parameters that get passed to traverse()
- Struct: sort_param. Holds the parameters that get passed to sortFile()
- `int main()` : checks params, gets directory, calls the initial traverse() function, and prints metadata.
- `char* isDir(char*, char*)` : determines if a path leads to a file of a filder.
- `void* traverse(void*)` : traverses folders and forks new processes if it encounters a subdirectory or CSV file.
- `void* sendFile(void*)` : sends the contents of a CSV to the server
- `Node* insertAtHead(Node*, Node*)` : inserts node at head of linked list that will be sorted
- `char* getRow()`: returns the current row as char*
- `int getColNum(char*, char*)`: returns the column number of a certain cell in a row.

- `char* getCellAtInd(char*, int)` : returns the cell in the given row at the provided column number.
- `char* trim(char*)`: removes leading and trailing spaces from a string.
- `char** moreCapacity(char**, int)`: allocates more memory for the strings array if required.
- `Node* splitList(Node*)`: splits a linked list in half, used for mergesort.
- `Node* mergeInts(Node*, Node*)`: merges the nodes back together in sorted order, based on integer comparison.
- `Node* mergeFloats(Node*, Node*)`: merges the nodes back together in sorted order, based on float comparison.
- `Node* floatMergeSort(Node*)`: encapsulates the splitting and merging for floats.
- `Node* intMergeSort(Node*)` : encapsulates the splitting and merging for ints.
- `Node* mergeStrings(Node*, Node*)`: merges the nodes back together in sorted order, based on strcmp().
- `Node* stringMergeSort(Node*)`: encapsulates the splitting and merging for strings.
- `Node* mergeSort(Node*, int)`: determines which mergesort method to use, integer parameter indicates data type.
- `void insertArr(char*)`: inserts string into global list of strings.
- `pthread_t* moreThreads(pthread_t*, int)`: resizes the thread array.
- `int sendall(int, const void*, size_t, int)`: sends CSV file contents to server
- `char* recvall(int, char*, size_t, int, char*)`: receives CSV file contents from clients
- `void* sort(void*)`: takes file contents from client and stores it
- `void* dump(int*, int)`: sorts all of the server's stored content and sends the resulting data back to client

Issues Encountered and Solutions

- Problem: Determining how to compare values due to them being different types.
  Solution: Create one mergesort which then calls seperate mergesorts for each type.
- Problem: Determining a protocol for dump requests Solution: once a directory is completely traversed, automatically dump. After one client dumps, it does not free all that allocated memory. If one client is run after another, the second dump request will also return the content from the first dump. If two clients run concurrently, then both dumps will include content from both clients.
- Problem: Client doesn't send all bytes at once Solution: wrote sendall() and recvall() functions to make sure entire rows are sent

Notes

- If the provided directory does not exist, the program prints an error message and quits
- If no csv files exist in the provided directory, program prints error message and quits
- For directories that have an abnormally large amount of subdirectory depth (more than 256), there might be Linux limits on max length of path that hinder our program.
- ***No output files will be present if the output directory specified does not exist.***
- There was some confusion as to how the strings should be sorted. Differing answers were given on Piazza (one proposed method was lexicographical sorting, another method was using

strcmp()). All of our strings are sorted using strcmp() as indicated by Prof. Tjang on Piazza in the FAQ.

- Our linked list containing all the elements in the column we must sort by is built using insertion at the head node. This means our sort is not stable --- rather, the nodes that have equivalent data will be printed in reverse order.
- There are some non-UTC characters in the given CSV, and as a result, these characters end up in our output as well. This causes problems if we dump the stdout output of our program into a text file and open it in gedit. We are assuming that, because there are non-UTC chars in the given CSV, this is not a problem.
- Server code was compiled using "gcc -lpthread server.c -o server". After running the server, you run the client. Client compilation command used was "gcc -lpthread client.c -o client"
- Certain rows in the given CSV have movie titles or other columns with lots of trailing spaces. These are printed as is --- we trim them when sorting, but when outputting, the spaces are still there. We are assuming this is fine.
- ***Server does not free its allocated memory after each dump request.***