Brian Schillaci - netid: bcs115
Jesse-James Black - netid: jcb295
Tyson Morris - netid: tem97
Adeeb Kabir - netid: ask171

CS214 Project 3: Basic Data Sorter - server/client

Design

Our program works as follows: we have a client program that first creates a socket and connects with the server, then traverses directories starting from the given -d parameter (or the current directory if not specified), looks for csv files, and then spawns threads for each file which send the rows of that file over the initial socket created in the beginning. Once the client is done traversing all the directories and sending all the rows from all the files, it sets up the final result output file in the specified -o directory and then sends a dump request to the server to let the server know that it is ready to receive all of its sorted rows back. On the server side, we setup a multithreaded server that listens on the given port, and creates new sockets for every client that connects using the accept() function, and for this new connection we spawn a thread, passing the new socket as a parameter. This client handler function will do all the work of reading the column name, reading all of the rows from the client, reading the dump request, sorting all of the rows, and then writing the solution back to the client when the client is ready.

For sending data over the sockets, we used a 6 byte metadata section that we would send over each time before sending any data over. This 6 byte metadata section had the length of the data that was coming next in string form, that way when the server converts this metadata to an integer, it can use that integer as the length of the next read statement, this way we read the correct amount of bytes everytime. We also used this same metadata section to write and read the dump request, which was just a special string that we agreed upon between the client and server. Also, we chose to only connect to the server once per client, and use this socket to send all of the rows from all of the files through. This helps to make sure that a client will only receive the rows that it sent over. After the client received all its rows, both the server and the client closed the socket they were using, and freed all malloc memory used the that server/client interaction.

Compilation and Running Code

To compile you can just enter:

- make

*Make sure to have the -lm flag and -pthread flag if you don't compile with "make".

To run server: ./server -p <port_number>

To run client:
./client -c <column> -h <host_name> -p <port_number> -d <in_directory> -o <out_directory>

The client requires at minimum the following three arguments: -c -h -p

Assumptions

- **We ignored csv files with the string: "-sorted-" in the name of the file, because we don't want to resort the AllFiles-sorted- file.**
- **To check for if a csv file was valid or not**, we just checked if the header of the csv contained the column that we were trying to sort by. This means if the rest of the csv has random data, the sorting could fail, but Professor Tjang said it was okay to just do this check.
- If there is a -d, and no -o, we output to the folder of where the executable is run from.
- We chose to keep the sorted data file on the server even after the dump request.
- **Each client only connects with the server once**. The socket that they get from that connection is what all unsorted files are sent through. It is also the socket used for transmitting the final sorted file. Professor Tjang said this was okay to do.

Difficulties

One of the major difficulties this project created was figuring out how to transfer data between the server and the client. At first we were writing 1024 bytes at a time from the client and reading 1024 bytes at a time from the server. This was problematic and did not always work correctly. After changing the code to use a dynamic pattern with a header and data following the header, all problems were fixed.

Another difficulty had to do with running the server and client on the same computer. This was how we were testing the code to begin with and we were having no problems. Once we put the server and client on separate computers we were getting segfaults. The reason for this was that the write speed must have been different and not all the characters in the write buffer were being written completely. This problem was fixed by creating a loop that ensured each buffer was written completely before moving to the next.

Extra credit 2

Each time a client connects, we assign them a session ID number, called clientID, which is a global variable that we locked and incremented each time a new client connected. This would be passed into the client handler function and a file for that client is created to store all of that

clients sorted data. This helps segregate the data of individual clients that connect, this way when there is a dump request from a client, it will only get the data back that it sent. It will not receive all of the other clients data when sending a dump request.