Code Structure

In order for my code to work it took a long time and a little bit or research. Firstly, my code starts off in the main function with 2 while loops. The first one loops until it gets a correct file name and opens it, or until the user exits. The second while loop is for the options the user has which are altered by 4 switch cases. Obviously the code relies heavily on threads in order to be efficient and carry stuff out in the background while the user has the option to do other stuff. A down side to threads is they can only accept one void pointer as a parameter. So I came up with a way around this, by using containers of data. A container in my code is basically a structure that holds important data that the threads will need to carry out a search or save. Now I needed to come up with a efficient way in order to keep track of searches made in a data structure before they are saved into a file. I found that the best way to carry this out was to create a dynamically allocated me array of strings, which is basically an array of arrays of chars. After analyzing the design document I realized that the longest line was 89 characters. So I decided to take in 100 characters into each line just for safety. It would've been hectic creating a malloc for every single line depending on how long each line was. Then by going through the file twice. Once to get the number of results, and using that number to create the array with exact number of results, and saving that number in the structure for loops later on. And through using tokens and strtok and pointers. I was able to complete a search and have an array with all elements of that type of Pokémon. I then created an array to hold all these structures of search results. I made the array a size of 18, since there is a total of 18 Pokémon types in the entire world, assuming that the user doesn't double search 18 elements was just enough. Then going through that array from a thread I save the results to a file. The code will simply be 1 c file. I found it useless to create a header file since the code only uses to traditional functions. It is mainly using threads to help enhance the performance and deliver a clean fast and more real world like user interface to customer. Obviously the most 2 used libraries were the pthread and semaphore. Pthread was a must in order to create the threads call them and join them, and a semaphore was not really needed, but I don’t know how long the data I am getting will be so it is always better to be safe by locking it and unlocking it when in use by another thread, otherwise we would get a deadlock problem. And as mentioned a dynamic allocation of a double array was used in this code to store the data from the search. When storing strings the best thing to use is definitely pointers, they provide so much flexibility especially with lengths of strings.

Sever Structure

For this assignment I decided to use a UDP sever structure, over a TCP structure for a couple of reasons. This assignment could be easily written in less than 100 lines, but there is a reason why my code is over 400 lines, and that is efficiency and speed. I used multi-threading to ensure that more than one thing were happening at once. That is why I used a UDP instead of a TCP server, because the UDP server has faster runtime. That’s because UDP has no retransmission delays. It also has a higher speed than a TCP because the data packets sent over are much smaller in size thus making it receive and transfer data quickly. In my code I am not transferring that much packets but when the whole objective of the assignment was to focus on speed and efficiency I had go with a UDP based server. Obviously the speed of the UDP comes with a cost, which are the following. There are no guarantees in terms of receiving ordered packets, no protection against duplicate packets, or transmitted bytes or verification reading. However all of these security measures do not benefit or harm the code in what so ever way. That explains my use of UDP. The code in its self is very simple. I made two files a server and a client. The client simply sends over the user input to the server. The server then process the input either asking for input such as Pokémon type or a file name, or it executes a command and returns a success message back to the client. If the user wants to quit the code then the client will send a message back to server asking for the total number of queries and asking the server to quite and close the socket. The client then displays the number of queries and the new files and quits and closes its sockets as well. Obviously memory is checked with valingrd, to make sure all mallocs, and file opening and socket, opening are successfully closed after code termination. Lastely, since there is an scan f the code must be run in 2 terminals and without the & in order for the terminal input to be directed to the file and not terminal commands

**** code must be run in 2 separate terminal windows****

Flow chart and brain storming

Start the server and then connect client → get user input from client → Server process command → Send success message

get user input from client → Server asks for more input

if (a → is input)
↓
ask for type and wait

if (b → is input)
↓
ask for file name

if (c → is input)
↓
send user number of wires over and print exit message to client → Send number by FIFO ~~sta~~ sprintf(); → then send buffer
↓
commit

→ close socket by breaking out of while(1) and returning existing loops