Cameron Sparks 101181932

NOTE: All the functions mentioned within the pseudocode have already been described in the assignment 3 design doc.

Program Pseudo Code

server.h

```
// Include the necessary header files (stdio, stdlib, string, unistd, sys/socket, netinet/in, arpa/inet)
// Define the server ip and port. Server ip="127.0.0.1", Port=6000.
// Include some of the structures and functions from assignment 3.
struct QueryServer{
    //Variables needed during the query process along with a variable to store the file location of
“pokemon.csv”
}
```

---

client.h

```
// Include the necessary header files (stdio, stdlib, string, unistd, sys/socket, netinet/in, arpa/inet,
semaphore, pthread)
// Define the server ip and port. Server ip="127.0.0.1", Port=6000.
// Include some of the structures and functions from assignment 3.
struct QueryClient{
    //Variables needed to store and save query data.
}
```

---

serverFunctions.c

```
// Include server.h
// Contains the functions from assignment 3 needed to query the given type 1.
```

---

clientFunctions.c

```
// Include client.h
// Contains the saveAll function from assignment 3 needed to save queried data to a file.
```

---

server.c

```
// Include server.h
int main(int argc, char **argv){
    // Create and initialize a QueryServer variable.
    // Create server related variables
    // Enter the enterCsv() function and pass in the QueryServer variable.
    // Create the server socket
    // Setup the server address
    // Bind the server socket
    // Set up the line-up to handle up to 1 clients in line (Doesn't really matter as no more then 1 user
will be connecting at a time)

    START INFINITE LOOP 1:
    // Accept the incoming client
    START INFINITE LOOP 2:
    // Receive and store the pokemon type 1 that was sent from client to be queried
    // If the received type 1 is the string “done” break out of the loop.
    // Enter the query function typeSearch() and pass in the QueryServer variable.
    // Send over the total amount of pokemon queried and the total amount of queries to the user.
    // If the total amount of pokemon is not zero then send over the queried pokemon array.
    END OF INFINITE LOOP 2:

    // Close the client socket
    // If the total amount of pokemon is not zero free the queried pokemon array and zero the QueryServer variable (Specifically the totalPokemon and totalQueries variables).

    END OF INFINITE LOOP 1:
    //The code will never reach this part as the infinite loop never breaks.
    //Keeping this code as a reminder of what should happen if the server were to actually close.
    //Close the server socket
    //Return success / exit program.
}
```

client.c

```
// Include client.h
int main(int argc, char **argv){
    //Create and initialize a QueryClient variable
    //Create other variables to store things like (users choice, save file names, number of threads / saved files and the input query information (type 1).
    //Create network related variables.
    //Dynamically allocate memory for the queried pokemon array, threads and the save file name array.
    //Initialize a mutex
    //Create the client socket, setup the address and connect to the server

    START INFINITE LOOP 1:
    // Prompt for the user to select an option (1,2 or 3).
    if(userOption==1){
        //Prompt the user to enter a pokemon type.
        //Send the pokemon type to the server.
        //Receive the total amount of queried pokemon and total amount of queries from the server.
        //If the total amount of queried pokemon is not zero, reallocate more memory for the queried pokemon array and receive the queried array from the server.
    }
    //Functionality wise userOption=2 is the same as assignment 3.
    //userOption=3 is the same as assignment 3 except it sends the “done” string to the server (Closes the connection).
    //Close the client socket, threads and free dynamic data.
    //Print total queries and save files names.
    // Return success / exit program.
}
```

END OF PSEUDO CODE

1. The program will contain 4 C files and two header files. The client.c file contains the client side functionality of the program and includes options for sending a query to the server, saving the query data and exiting the program. Alongside client.c is the clientFunctions.c file which contains all the functions,headers and structures that client.c uses. The server.c file contains the server functionality of the program which is essentially code to communicate with the client and query the data that was sent as well as sending the queried data back to the user. Alongside server.c is the serverFunctions.c file which contains all the functions,headers and structures that server.c uses. Finally the server.h and client.h files contain all the structures, function declarations and header files that both the server and client need respectively. The reason for 4 C files is to modularize the program and separate it into the server side and the client side with two C files being for the server and the other two being for the client. The two headers files are also to modularize the program further by providing the includes, structures and function declarations that both the server side and client side need.
2. FUNCTIONAL REQUIREMENTS:
    a. The program implements the first use case by entering a function called enterCsv() which prompts the user to enter a pokemon.csv file. If the pokemon.csv file path exists then the function returns and the program continues on with setting up the required server-client connection. If the pokemon.csv file path does not exist the program prompts the user to enter it again or quit by typing “q”.
    b. The program implements the second use case by first establishing a connection to the server (using the example code from the textbook). Then the program uses the same code as assignment 3 to display the menu.
    c. The program implements the third use case by essentially using the same menu code from assignment 3 however instead of running a function to query the data, a user prompted query is sent to the server instead. The server then receives the user query and uses the assignment 3 query functions to query the data and send it back to the user. The server is not multithreaded due to it not needing to perform other tasks other than querying. Even if the querying process was multithreaded it would still take the same amount of time to complete as two back to back non multithreaded queries.
    d. The program implements the fourth use case exactly the same way as assignment 3.
    e. The program implements the fifth use case almost exactly like assignment 3 except it sends a “done” string to the server so that the server can close the client's socket and restart.
3. NON-FUNCTIONAL REQUIREMENTS:
    a. NFR1: The PQC program uses threads during the saving process so that the program remains responsive and lets the user perform other tasks while the data is being saved to a file. However for queries the PQC program waits for the server to respond back before proceeding (Professor said this was fine in Discord).
    b. NFR2: As stated above the client and server side of the program has been split into separate C and headers files.
4. sys/socket, netinet/in and arpa/inet are the libraries needed to provide network functionality for both the client and server side of the program. They contain the required functions and structures needed to establish the connection between the server and client and ultimately to pass query data between the two. Besides the network libraries, all the C language features and libraries that were used in assignment 3 apply here as well.