Final Project

README  
by Nicholas Jones

CS-250

Network Programming

Prepared for

Professor Pollack

**Descriptions**

**Application.o**

This program is the main interface and menu for the user to use. The application communicates and makes use of the Client.o program to make requests to the server depending on what the user requested. The application creates and ends the client program process for each request sent. The application keeps track of dislikes for quotes so they will not be shown again. This is an executable that will need to be started by the user.

**Client.o**

This program makes the network connections to the server based on what was requested by the application. The client mostly acts as a broker between the server and application and has a buffer for the request to send and a buffer for the information received. The client also makes use of the caesarHelper.c helper program so that it can encrypt the requests before they are sent and decrypt the received requests from the server.

**Server.o**

The server program provides the services based on the request made by the client according to the protocol. The server makes use of caesarHelper.c in order to decrypt incoming connections and encrypt outgoing connections. The server program also makes use of the timeHelper.c, countryHelper.c, and quoteHelper.c programs to provide each service respectively. The server executable will need to be started by the user along with the arguments of “31337” for the port and any number of processes.

**caesarHelper.c**

This program is included within the other programs to provide the Caesar Cipher encryption and decryption functionality in both directions.

**timeHelper.c**

This program is included by the server to provide the local time and GMT time to the server to be provided to the app.

**countryHelper.c**

This program is included by the server in order to provide access to the countryDB.csv file of countries based on the request made by the application.

**quoteHelper.c**

This program is included by the server to provide a random quote from quotes.txt to the client and application.

**Issues Encountered**

**UDP Support**

I ended up having a lot of trouble attempting to combine the UDP and TCP functionality together into a single server and client combo. I was unable to figure out how to create the separation between the two protocols and keep the time service exclusive to UDP connections. Ultimately, the time service is provided over encrypted TCP which doesn’t hurt, however UDP is better for sending something short and not guaranteed such as time. It also isn’t necessary to encrypt time.

**Country Functionality**

In the end after spending a lot of time trying to get this functionality to work, I was unable to come to a working solution. Reading the file was not a problem, but finding a way to read it in a way to recognize the \0 or \n in order to separate the lines was difficult. I wanted to find the requested entry and send the entire record for the specified ID back to the client for it to be parsed on the Application for the specific information the user requested. I would have been able to successfully parse the single record on the application by using the comma delimiters but ultimately, I was unable to implement the solution for searching and extracting a specific record from the database.

**Lessons Learned**

This was definitely the best learning experience I have had in programming so far in my academic career. This project incorporated networking, encryption, menus, protocol development, different types of file reading, and modular design all into one project. Of these, the basic network programming ended up being my strongest skill, and the skills that really held me back were my ability to parse files and my ability to incorporate two different types of protocols into one implementation. One specific thing that I’m sure I improved on throughout this project was my knowledge of using char arrays for strings in C and working with them. I also really enjoyed creating the helper programs in order to separate the functionality for the different needed services. Finally, implementing the execution and arguments of the client by the application was a very good learning experience.

**Protocol Documentation**

The protocol between the client and application and client and server is very simple, and they all use the same protocol together. Depending on what the user selects that they want to do, a “service” string is generated with a single word to describe what it is that the user is requesting. This service string is then given to the client from the application as a command line argument and is then sent to the server from the client. It is important to note that this string is a char array and is encrypted between the client and server. Once the service string arrives at the server it is processed to determine the service to be provided, and the server then sends back the information requested and provided by the helper programs. The information sent back is always the raw char array of the data and is encrypted in transit. The built in address and port is 127.0.0.1 and 31337.

**Service Strings:**

***quote***

This string is sent to the server when the user requests a quote.

***country***

This string is sent to the server when the user is requesting country information.

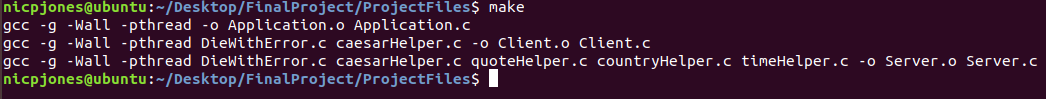
***gmttime***

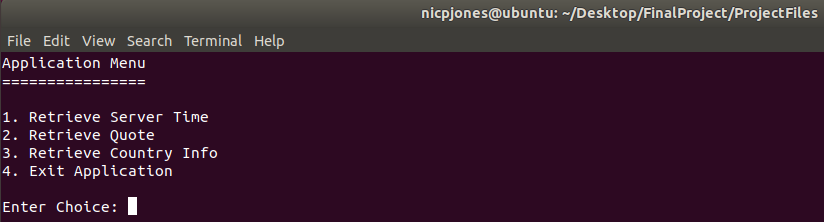
This string is sent to the server when the user is requesting the GMT time information.

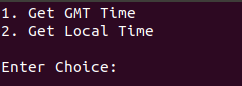
***localtime***

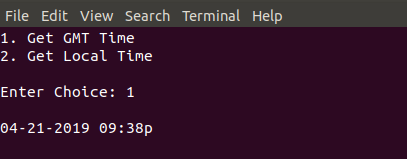
This string is sent to the server when the user requests the local time information.

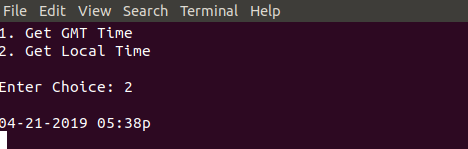
**Screenshots**

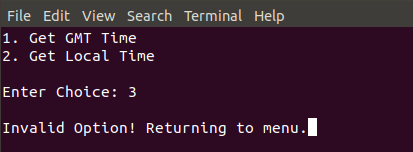
*This screenshot shows my code successfully compiling without warnings or errors.*

  
*This screenshot shows the main menu ready for selection by the user.*

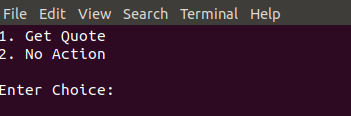
 *This screenshot shows the sub-menu for the time functionality.*

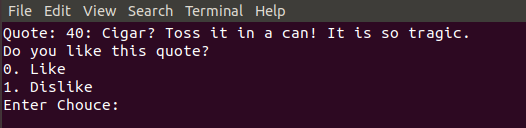
 *This screenshot shows the GMT time functionality.*

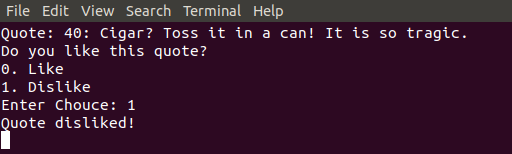
 *This screenshot shows the local time functionality.*

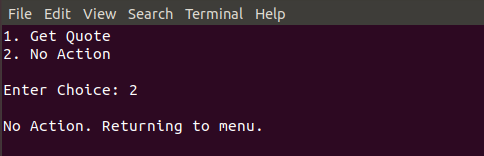


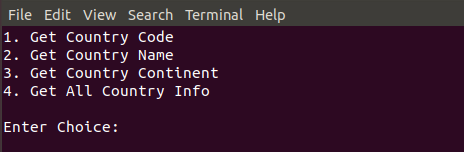
*If the user selects and invalid option in the sub-menu, they are returned to the main menu.*

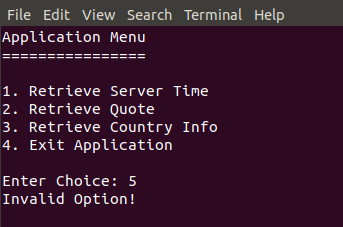
 *This screenshot shows the sub-menu for the quote retrieval functionality.*

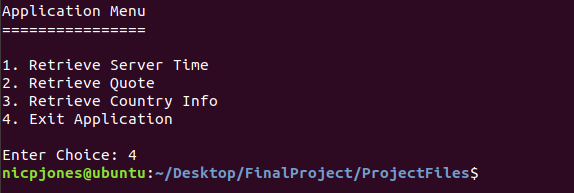
 *This screenshot shows when a quote is retrieved and the option is given to like it or dislike it.*

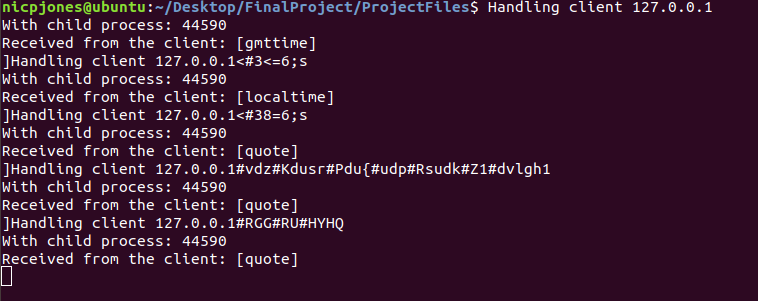
 *This screenshot shows the output when a quote is disliked. Internally, the application only tracks dislikes, and stores them in array according to the quote number seen. There is no need to track likes, because only dislikes modify the functionality.*

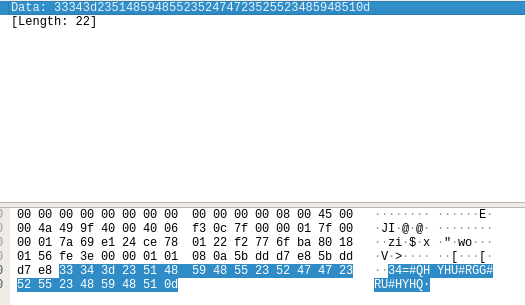
 *This screenshot shows what happens if no action is chosen, and the user is returned to the menu.*

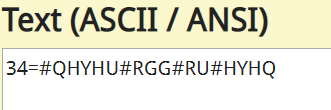
 *This screenshot shows the sub-menu for the country database retrieval functionality.*

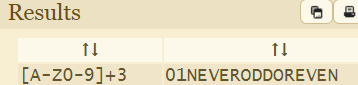
 *This screenshot shows the output when an invalid option on the main menu is entered.*

 *This screenshot shows the application exiting upon choosing the 4th option.*

 *This screenshot shows the server window which isn’t pretty because it shouldn’t be seen by the user of the application. However, it does give insight into the service requests and the encrypted quotes and information being sent.*

 *This screenshot shows some of the encrypted data in wireshark in both hex and ACII form.*

 *This screenshot shows the hex data decoded into ASCII data.*

 *This screenshot shows the encrypted ACII being decoded into the original data using a web tool.*