Devansh Mishra
CSCI4211

Gohper Chat Room Project

How to Compile and Run Code:

Server:
Compile:
```
gcc -o server server.c
```

Run:
```
./server [port number]
./server reset
```

Example:
```
./server 6001
```

Client:
Compile:
```
gcc -o client client.c
```

Run:
```
./client [ip address] [port number] [testfile]
```

Example:
```
./client 3.21.113.0 6001 instructions.txt
```

General Overview of Strategy:
In this implementation, I implemented a non-blocking i/o strategy similar to that of the alphabet client/server example in class. The client first reads the input of the text file line by line, parsing it for the type of package and the data(username, password, message, filename, etc). The client validates the data provided in terms of length requirements, printing an error message and continuing to the next step if it is invalid. The client then stores this data in a series of different packet structs representing the message type, preceded by a header struct that contains the length and type of the packet. The client then casts the data into a char buffer, encrypts it by calling a function, and sends both to the server(header struct first).

The server receives the char buffer and and decrypts it(simply a function call). The server uses the header to determine how many bytes to read in order to get all of the data. It then reads the bytes(non-blocking) until it has read all of the data into the connStat.rec of that client. Each connection is represented by a connStat, containing a rec buffer, send buffer, trackers for the two buffers, id, state(logged in or logges out) and status(represents which step of receiving/sending the connection is in). Once read, it determines if it is a register or login. It validates that it is not part of the cred_table(a list of username and passwords that are initialized from a file). If it is a register, it will add the username to the cred_table and file. Login will validate if the login was correct. Logout will simply log someone out. An error will be generated if any of the validations fail and will be put in the receive buffer of the connStat.

For the rest, it will go to attempt to transfer the receive buffer into the necessary send buffers. This means, for example, that if a SENDF is sent, it will check if all those intended to receive it have space in their send buffer for the file. If so, the rec buffer contents are transferred. If not, then it wont transfer the data. (Error messages are connections sending themselves an error message). The LIST is generated here as the size may vary.

The contents are then sent back to the client until the buffer is empty. After each send attempt, the transfer to receive buffer is attempted again to see if space has cleared up for another message.The send buffer is then encrypted and sent in a non-blocking way.

The client then receives the commands during the DELAY command, using a poll and a timer to ensure that the full time is used. The buffer is decrypted, the header telling how much more data to expect. The data is then parsed and the necessary actions are taken to print the message based on the packet contents.

Corner Cases:

- If an invalid step in the client is found, it will print an error and continue. This is under the assumption that the syntax of the input file (all the spaces and newlines) are kept consistent.
- Logout will not return an error if you are not logged in. Did not feel that sending an error message was necessary in this case.
- I print out the messages that the client send locally(NOT PROCESSED BY SERVER) to make it seem more like a chat room.
- There are no known errors currently, however testing with more than 4 clients was difficult to keep track of.
- “Connection closed” and “Connection removed” mean similar things but this is not an misprint. Connection closed means the client disconnected but connection removed means that the removal of data was properly handled(if there are ten users, a user can join after this message)..

Encryption Extra Feature:

I implemented an encrypt and decrypt function in both my server and client. The key is stored locally on both. The encrypt and decrypt are called before sending and after receiving the char buff array. I used a simple XOR cipher to encrypt and decrypt my data. There is no additional command for the additional feature, however any traffic should be encrypted as well as the credential file saved by the server.