Name: Stephen Paradis
Date: June 26 2016
Current Module: Network Programming in C
Project Name: UCT

Project Goals:

The goal of this project was twofold, first we were to find a program running on and open port on the Virtual Machine and connect to it. Second we were to reverse engineer what this program was and connect to it. The program was an IRC server and our goal was to create an IRC client to communicate with it.

Considerations:

- We needed to find out what the application was and on what port
- We needed to consider what an IRC client was and what we needed to make one
- We needed to consider the basic format for an IRC client, such as what command shortcuts were supported

Initial Design:

Initially I was attempting this project in C however I changed to python midway through because the parsing of user arguments was made easier in python. I initially created a program that just connected to the open port and communicated with it. From there I did all the threading and parsing so that the input was sent to the server correctly.

Data Flow:

The program first creates a socket with which to connect to port 6667. It then creates threads so that incoming and outgoing data can be handled without causing the other to be affected. Then the program would handle and parse incoming data to do whatever command was input by the user. Once everything was handled and the user wished to quit the program would close the thread and socket and end.

Communication Protocol:

I used threading and sockets to connect and communicate with the IRC server running on port 6667.

Potential Pitfalls:

- Not threading or forking which caused output from the server and input from the user to overlap potentially.
- Not sending the request to the server correctly causing the server to handle the error but break the program.
- Not doing a check to make sure a the “ping” in the /help output was not actually from the server.
- Having input lines overlap or not print where they should

Test Plan:

User Test:

1. Tested input lines to make sure they printed where I wanted them to
2. Catching keyboard interrupts
3. Tested to make sure a user couldn't send a command unless it started with a '/'
4. Tested each command that I supported
5. Tested the '/quit' command

Automated Test:

None

Test Cases:

1. I tested giving input and making sure my input line printed out in the correct place
2. I tested giving keyboard interrupts to make sure they were caught and closed down gracefully
3. Tested the commands with and without the '/' to make sure it did not still send the command if I did not have the '/'.
4. Tested each command I supported to make sure it sent to the server and got the correct output without causing my program to break
5. Tested to make sure that when the '/quit' command was sent out it closed the socket and thread and killed the program correctly instead of looping indefinitely.

Expected Result:

For each of the tests I expected the command to only work with the '/' and to send out a notice otherwise. I also expected that each command would be sent and received back without error and with proper output for that command. I also expected that when the program sent the quit command it would gracefully close the thread and socket and send without looping indefinitely or breaking.

Conclusion:

This project was slightly difficult because the requirements were not laid out to us, we were to make our own decisions on language and how in depth to go. This made it difficult when dealing with an application most of us had no knowledge of. The considerations to be taken in were probably the biggest obstacle to overcome in this project. In the future I think it would be good to go back over IRC basics and make the program a little more robust in what it can do.