Name: Sean Bartholomew
Date: 20160626
Current Module: Network Programming with C
Project Name: UCT

Project Goals:

The goal of the project is to identify the open port on the devprep machine. From there we are to reverse engineer the service running on the port and build a basic client to connect with the server.

Considerations:

What is the service running on the machine?
How much detail will be implemented in the client?
Are there any special protocols that will need to be adhered to?
Should the client have a GUI or be command based?
What on earth is IRC?

Initial Design:

*netstat -pant | less*

looking though the resultant data I determined that there are three services running on the machine – on ports 3389 – Remote desktop, 22 – SSH, and 6667 – whatever Liam put there.

*Ps -elf | grep wechlin*

On port 6667 there is a process running called mimircd. The server is an IRC server.

*Google what is IRC?*

Apparently IRC is a legacy chat service that is somehow still used in some places on the internet like its 1989, It's like every other chat service – just older.

*Nc localhost 6667*

When connecting to the service the user is immediately prompted to identify themselves. Typing anything that is not a recognized command results in an error message referencing the help page. Help lists the known commands: Away, Ison, Help, Info, Join, List, Lusers, Mode, Motd, Nick, Notice, Part, Ping, Pong, Privmsg, Quit, Topic, Wallops, Who, Whois.

*Google IRC*

I honestly had no idea what IRC was, and had to spend a lot of time talking to other people and searching for what was going on.

*Design the program*

I chose to use Java because I could honestly, and I know how much Liam loves Oracle. Java also makes the client server communications as well as threading much easier than C. The original intent was to create a GUI based client, however that proved to be overly ambitious. I had intended to keep track of the different channels that a user is a member of and display all relevant information about the channels on separate tabs. If given another day in class, I'm sure that I could have achieved this, however, my toddler does not appreciate coding.

As such, the client is strictly Command based. It will also only allow you to be connected to ONE channel at a time. The server allows for multiple connections however, to

implement a command line solution that is also somewhat intuitive, only one channel is allowed.

To execute the program from command line download the bin folder on github and run the command “java uct.Uct”

Data Flow:

When the program is started, the user is automatically logged in with the nickname CamelCase username HaHaHa and placed on the channel #irchacks. The main process starts a child thread which handles the communication from the server. When the server sends a ping it will automatically be replied to with a pong. All other messages will be translated (if possible) and displayed to the screen. Both processes will loop continuously until the child thread stops or the user types quit.

Communication Protocol:

CMD line options: none. The database, username, and nickname are all specified.
Signals handled: None.
Potential Pitfalls: I’m sure that there are some IRC commands that I am unaware of that will do some wired things to my program.

Test Plan:

Test user’s ability to send and receive messages, both group and private.
Leave the program running for a prolonged period of time to verify proper translation of server data.

Expected Result:

I expect the behavior outlined in the initial design to be appropriately implemented. The program will not crash.

Conclusion:

You said to make a “primitive” client. This program is very much primitive. I would have liked to have more time in class to work on it, and hash out more details about what it should be doing, and to implement a gui solution. However, it does enable the user to communicate through the server, although in a very limited capacity.