Marima Andrew Mambondiumwe

Concept Portfolio

Networking

I enjoyed learning about the whole concept of Server/client communication under the TCP/IP. We started the course by undertaking simple implementations of trying to create a connection between the client and the server and we gradually started to add on and build up complexity from that point onwards. One of the first things or concepts that I learnt in class was the socket. A socket is defined as one endpoint of a 2-way communication link between a client and server running on the network. When a server is run or initiated, it opens up a socket that will listen to connections from clients, and a socket is bound to a port and can be accessed through a specific port number. On the client side, if the connection is accepted, a socket is successfully created and the client can use the socket to communicate with the server. The client and server can now communicate by writing to or reading from their sockets.

I was introduced to working with virtual machines (VMs). The first task was to create a server and a client on one virtual machine and get them to communicate with each other. After that, I progressed to setting up two different virtual machines and having one act as the client and the other as the server. Other modifications that I made to the server and to the client were to add functionalities such as the checksum, check moduli, threading and retry/resend. A checksum is a count of the number of bits in a transmission unit that is included with the unit so that the receiver can check to see whether the same number of bits arrived. If the counts match, it's assumed that the complete transmission was received. TCP communication layers provide a checksum count and verification as one of their services. Threading basically allows multiple people/clients to connect to the same server at the same time and be able to input a message and see it instantly. A retry/resend implementation was added to the code in case if the server prematurely closes connection or in order for the application to handle failures when trying to connect to a service. If the specific fault reported is unusual, it might have been caused by unusual circumstances such as a network packet becoming corrupted while it was being transmitted. In this case, the application could retry the failing request again immediately because the same failure is unlikely to be repeated and the request will probably be successful. I met a lot of challenges when trying to add all these different implementations but with the aide of internet tutorials and consultations of peers and TA’s we eventually figured it out. All my code was done in python and was object orientated programming.

I created two sets of code, one for the client side and the other for the server side. I worked with Sher and Jonathan in most of the implementations and we practiced good pair-programming techniques. We ensured that everyone got to be both the driver and also the navigator at some point. Or on other instances, we would divide the workload and have everyone work on a certain part of the code and then combine the code once we were all done. We made sure that at all instances we were always on the same wavelength in terms of understanding the work that we were doing.

For the server code, in the final implementation, I created a class which I named the S impleMailServerProtocol. I initialized three dictionaries for this class, one was the MBX, which was the user personal mail box, and the other was LOGIN, which had the purpose of storing the registered accounts and the passwords. The last dictionary that I used was the ID, this stored the cookies and usernames, the cookies functioned as session ids and their role was important in that they would allow the communication to be secure since each individual user would have a randomly generated unique cookie that couldn’t be easily hacked. A cookie is destroyed once the user logs out and a new one is assigned once they log back in. Two lists were created too in addition to the three dictionaries. The first one is the IMQ, which is the incoming message queue, this stores all messages temporarily before they are sent to their recipients in their user mail boxes. The last list is the ASSIGNED COOKIES, this checks the IDs and which session cookie has already been assigned. The code starts by starting the server, which will cause it to create a socket and listen for connections from clients, two parameters are important in the client being able to make a connection with the server, these are the host and the port number, the client should specify a corresponding hostname and port number that matches that which is on the server. If they match then the client then establishes connection and the client is prompted to either login or to register. If the client is new then, it should register by inputting a username and a password. After registration or logging in the client is next prompted to select one from several options, these options include add message, store message, count, delete message, logout and dump. All these options are class methods of the class SimpleMailServerProtocol that I created.

The method get message in my server code is the only method that included the check sum implementation. I included the check sum implementation here so that I wouldn’t have to put it in all the other methods. It occupies the position 0 on the parameters for the get message method. The threading part was achieved by importing the threading library. The threading is initiated from the handle\_method which is a class method of the mail\_server class. The handle\_message method takes two arguments which are message and connection. After a semester taking Network, I feel that I understand the whole concept of Computer networks and how computers can communicate over a network, Design was an important thought process that we had to do before starting any coding or implementation. It makes it so much easier when you have everything mapped out and are just following an outline .I can implement a client/server relationship and can write code that allows several clients from different computers to communicate with one server as long as they know the host name and the port number. I can also modify both the client and server to make them more robust and have several implementation. It has been a fruitful semester and while this course was extremely challenging at times the professor taught it well and I learnt a great deal.