Lab 2 Report

Conclusion:

In this lab I learned a lot about TCP and socket programming. More specifically TCP socket programming. We had two C files we had to implement which were the server and client.

For the server we had to follow an instruction set for communicating via TCP. Our first step was to create a socket. We specified that we were using IP and that we wanted a Socket stream. Next we had to bind our socket to a port. We had to specify that we were using IP and which IP to bind to in which we used ANY because we didn't want to specify a specific address. We also gave the socket a port to bind to. We then called the listen function to listen on the port. We then call accept which is a blocking function which will wait for a client to connect. Once the client connects it will keep their address in a struct. It will also give us back a new socket that we can use to communicate. The server then can call read in order to receive a char array from the client. The server then can write in order to send back a char array. When the server is done communicating it can call the close function on with the reference to the socket to end the communication.

For the client we have to do less work because it does not need to set up a socket on a specific port. First the client creates a socket to communicate with. We specify that we will be using IP protocol and want a stream socket. We then use the connect function. We give it our socket along with the ip address of the server. This attempts to make a 3 way TCP handshake. If the handshake worked we are able to start communicating with the server. We can then call the write function to send a char array to the server. In a similar way we can call the read function to receive a char array. When we are done the client closes the connection by using the close function with the socket as an argument.

This lab has taught me a lot about socket programming and how TCP works. I have learned how to create both a server and a client for TCP.

Prelab:

One way is to to parse the "uptime" file. The first step is creating a C file and setting it by calling popen("uptime", "r"). This gives us the file in read only allowing us to parse it. You can then use fgets with a buffer string in order to grab chars in "uptime". Afterwards make sure to close the file. It is also a good idea to error check each operation, like popen != null and fgets != null.

Another way is having a struct called sysinfo. You can then pass the structs address into a sysinfo function to update the struct. From there you can access the uptime attribute via myStruct.uptime. You have to make sure to include the sys/sysinfo.h header file though.

Joseph Schmidt
jschm333@iastate.edu