Mason Walls

CPRE 489

Lab 2: TCP Socket Programming

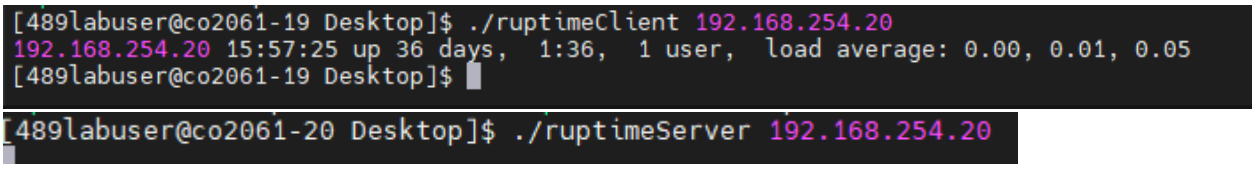
Prelab: The first way I can think of to get system up time would be to use the sys.info functions. These would allow me to see uptime, load times etc. It would not let me see how many users are connected though.

The second way to do this would be to run "uptime" on the server and send the output to the client. This is the option I chose, using forking and dup2.

What I learned: This is the first time I have done C to C socket communication. The only other time I have used a socket in C was C to Java for CPRE 288. With that being said, I learned a good amount about the structure of C sockets, and how they can transfer data. These sockets are much more complex to setup compared to something like socket.io in JavaScript, or standard sockets in Java. These complexities for me, come from all the structs that are given in socket.h. Learning how to use these was a challenge, as I had to do some research on what parameters they have, and how they should be used. One of the most important parts of the structs is pton and htons. I learned that these functions are required to convert port and internet address to byte order, something that can be read by the computer.

Another part of socket programming I learned was manipulating file descriptors for socket output. I read about the dup2 command, which was originally very confusing. Before using dup2, I forked the server programming so that when I ran uptime on execvp, my program would not close. I was surprised to learn that you could have "add" another connection to the socket, as the child process is another entity that can write to it. Dup2 allowed me to change the file descriptor of the client socket to std_out. This made is so execvp in the child process would wright directly to the socket, instead of the screen.

Demo:



Server is running the socket on machine 20's ip, while the client connects from machine 19 to machine 20 using the ip.