Brandon Vicinus
Gabriel Sotolongo
Networks 1
GetStock Project

This was a fun project. We learned a lot not only about networks and how to code a simple network, but about cooperation amongst our peers when making a standard and how nothing is perfect, and it takes a long time to create a working protocol. I think that the final version of the protocol is very good and concise. It is definitely good enough to send and receive stocks like intended.

The client proved to be rather simple. We created the client with 4 helper functions: one to create a request for a username, one to create an unregister request of a username, one to request stock quotes, and one to prompt the user to enter any request they want. We made a simple menu that asked the user what they would like to do. Entering options 1, 2, 3, or secretly pressing 4 will allow you to register a username, unregister a username, request a stock quote, and enter any request you would like to send, respectively. Option 4 was required to be able to test sending invalid commands like QQQ, and it also proved to be much faster when trying to create a very long packet. We included an attempt to sleep and resend a packet if the server is offline, but that functionality doesn't work.

The server was the bulk of the work. We began creating the server the same way as the client; with helper functions, that is. It seemed much harder to be able to keep track of scope of variables and lists so we soon moved to creating the server all in the main. We had the entire functionality of the server in a do-while loop that asked the user to enter 1 they want the server to keep listening for more messages or press 0 to kill the server. Inside this loop, the incoming packet interpretation happened. First, the command is validated, then there is a switch statement based off the first char of the verified command: R for REG, U for UNR, Q for QUO, and a default case for invalid commands. Register user option will take in the first 32 characters of the incoming username. That being said, the functionality to not create a username longer than 32 characters lies on the client side. Technically, if a client tries to register a username that is more than 32 characters, the server will take the first 32 characters, register that username, and issue a ROK. This is still in compliance with the protocol, no username more than 32 characters is ever created. The functionality of case sensitivity is on the server side. This is because different clients can and will send uppercase and lowercase requests. We figured it should be the servers job to ensure that all usernames are constantly lowercase. We overlooked the fact that stock names are also case insensitive so we do not have that functionality. All stock names are in caps.

To improve this standard and make it occupy less bandwidth, we could make a few fields shorter. There are only 3 requests, so a simple R, U, Q for request commands would suffice instead of 3 character commands. We can limit the size of the username to 6 or 8 characters, which we suggested on standards day but were denied.

To run our files:

- First note that #define BSD is the first line of code in both files. We did our entire project on Mac machines and compiled with gcc. So we are not 100% sure that the files compile and run in Windows machines, but we did not alter the code within the #ifdef WIN areas so it should still run fine.
- The IP address is set back to the default hardwired address for use on a single machine running both server and client. This would need to be changed in the client if you wish to send to a different location.
- We used command ‘ipconfig getifaddr en1’ to get the IP address of a machine using wireless and ‘ipconfig getifaddr en0’ for IP address of a machine using an Ethernet connection.
- We used command ‘gcc getStockServer.c –o server’ and ‘gcc getStockClient.c –o client’ to compile our files and commands ‘./server’ and ‘./client’ to run the executable files created.
- The user on the server side needs to enter ‘1’ to keep listening for more incoming messages after a packet is received.
- The client executable sends one packet; the executable must be run every time a packet wishes to be sent.