Lab 1: Socket Programming Basics

Due date: April 24, 2023 (Anywhere on Earth, i.e., April 25, 5 AM PDT)

TA Contact Info: Stan Lyakhov, lyakhovs@oregonstate.edu

Before starting, please download and read through the template files included with the lab: *lab1_client.py* and *lab1_server.py*. Make sure you understand how the sockets are set up from both the client and the server. Note that all tasks are marked using *TODO* comments within the code.

1. **Server Introduction message** It is fairly common for a server to send a welcome message when receiving a client connection. Implementing this is an easy way to get our feet wet with using sockets.

   To make receiving a little simpler, we will be assuming that the welcome message ends in a newline character, '\n'. Then the client can receive data 1 byte at a time until it encounters the newline character!

   (a) Open the file *lab1_server.py* in your favorite code editor. Finish implementing the *send_intro_message()* function by following the instructions given in the comments. Make sure to insert your ONID and major in the *intro_message* variable.

   (b) Now finish up the code that will allow the client to receive the message. Open *lab1_client.py* and finish implementing *recv_intro_message()*. Be sure to print out the intro message you received in the *main()* function!

   Run the server and client (in that order!) in separate windows. The client should have received the welcome message!

2. **Large Data Transfer** While the protocol for Part 1 works fine for small amounts of data that always ends in a newline, we need do better than receiving one character at a time for larger data transfers. We propose the following protocol: the client sends the length of the data they are about to send, encoded as 8 hexadecimal digits, before sending the data itself. Once the server learns the expected size of the message, it can receive the data in larger chunks, and only stop once it has received as much as the client promised.

   (a) In *lab1_client.py*, implement the *send_long_message()* function as described above.

   (b) In *lab1_server.py*, finish implementing the *recv_long_message()*. Make sure to print the data received in the *main()* function after receiving it!

   Run the server and client (in that order!) in separate windows. On the client side, enter a message you would like to send. Even large messages should work well here. Make sure you receive the same message on the server side!

3. **Wireshark Capture** Now that the server and client program are implemented, we will try to monitor this traffic using Wireshark.

   (a) Open Wireshark and make sure you are listening on the *loopback* interface! Begin capturing traffic

   (b) Run the server and client in separate windows. Enter an input on the client-side like in Part 2.

   (c) Stop capturing traffic on Wireshark. Find the communication that took place between the client and the server. Press *Follow TCP Stream.*

   (d) Please **attach a screenshot** showing all the data sent between the server and the client during communication.