Duke UNIVERSITY

ECE/COMPCSI 356: Computer Network Architecture

Lab 1: An Echo Server

Introduction:

This lab is a warm-up for the following two labs. In this lab you need to implement a simple **echo server** using the ANSI C socket API. An echo server can listen on a specific TCP port, and wait for a client to connect. After a TCP connection is established between the echo server and a client, the client can send arbitrary messages to the server and the server will echo back the same message to the client.

Your task is to implement the echo server. We provide the source code of the echo client for you.

This lab needs to be done individually. Two subsequent labs can be done in groups of up to 2.

Textbook reference material: PD Section 1.4.

Environment:

You are required to finish all labs in Ubuntu environment. We have set up an Ubuntu 12.04 virtual machine image for you. You can download it following the link on Sakai. We will test your code in this virtual machine. Currently we have tested this virtual machine image in the following platforms:

- Windows 10 - VirutalBox 5.2.4
- Ubuntu 16.04 - VirutalBox 5.2.4
- Mac OS X - VirtualBox 5.2.4

Starter code:

In Sakai's Lab 1 folder, we provide the skeleton code of the echo server (**not implemented**) and the client (**implemented**) code in one C program. The source code contains a **main** function which is the entry point of the program, a **client** function, which implements the client, and a **server** function, which you should modify to implement the server. You should submit the source code and make sure that it compiles. You can compile your source code with following command:

```
gcc main.c -o binary_name
```

In the same folder, we also provide the binary code that **includes full functions of both the client and server** for you to test your echo server. You can execute the binary code

in either the client mode or the server mode with the following command (suppose the file name of the binary code is "myprog"):

```
myprog s <port>
```

It will start an echo server and listen on the TCP port whose port number is <port>. For example, "myprog s 9999" will start an echo server and listen on port 9999.

```
myprog c <port> <address>
```

It will start a client to connect to an echo server whose IP address is <address> and port is <port>. For example, "myprog c 9999 127.0.0.1" will start a client and try to connect to port 9999 in your own machine. Then you can type any message and press "Enter" to send to the server that is running at your machine's port 9999. The client will print the message received from the server on the screen. You can disconnect the client from the server by terminating the process of the client.

We also provide a simple test script by Python2.7 (linked on Sakai). You can use it to test your code. To use this script, you can use the following command:

```
python lab1_test_script.py binary_path (e.g. python lab1_test_script.py
./sample)
```

If your code is right, you will see the following output:

```
Client1:  Testing short word: PASSED

          Testing long sentence: PASSED

          Testing multiple sentences: PASSED

          Testing EOF: PASSED

Client2:  Testing short word: PASSED

          Testing long sentence: PASSED

          Testing multiple sentences: PASSED

          Testing EOF: PASSED
```

For client1, each test case is worth 2 points. For client2, each test case is worth 0.5 point, for the total of 10 points for this lab.

You can also find those files on /home/ubuntu/projects/Lab1/ in the virtual machine.

Requirements:

- Your server can listen on an arbitrary TCP port (from 1024 to 65535).
- Your server can wait for a client to connect. You can assume that only one client is served at a time.
- After connected to a client, your server can receive a message no longer than 512 bytes from the client, and echo back the same message, until the client closes the TCP connection.
- Your server needs to handle the EOF sent by client. The client will quit after sending the EOF to server. You can send EOF in shell by pressing Ctrl+D.
- After the previous client closes its TCP connection, your server could wait for another client to connect.

Submitting:

Submit a compressed file on Sakai, with **main.c**, and a README file containing your name in it.