

Programming Lab Assignment 1

COMP 6320: Design and Analysis of Computer Networks

James S. L. Browning (jlb0181@auburn.edu) and Austin Preston (adp0082@auburn.edu)

September 19, 2022

1 Introduction

The purpose of this assignment was to explore basic UDP and TCP socket programming in C.

2 Lab 1.1

Lab 1.1 focused on UDP socket programming.

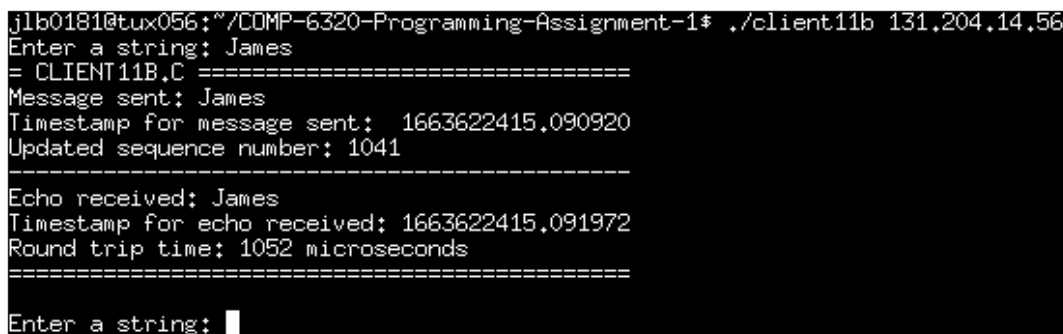
2.1 server11.c

Our first task for this lab was to develop a simple “ping” (echo) server, which would return any message sent to it. This was the simplest part of the lab.

At the time of writing, this program correctly compiles and runs on the Tux servers.

2.2 client11b.c

Our next task was to develop a client that could connect to the server and send it messages. This program takes a hostname as an argument (as this was a local loop assignment, I simply used the IP address SecureCRT listed for the server terminal for this), then prompts the user to enter a string. If the echo is received successfully, the client will repeat the message and display the total round trip time. This can be seen in the screenshot below.



```
jlb0181@tux056:~/COMP-6320-Programming-Assignment-1$ ./client11b 131.204.14.56
Enter a string: James
= CLIENT11B.C =====
Message sent: James
Timestamp for message sent: 1663622415.090920
Updated sequence number: 1041
=====
Echo received: James
Timestamp for echo received: 1663622415.091972
Round trip time: 1052 microseconds
=====
Enter a string: █
```

As was specified in the assignment documents, messages are limited to 1,024 bytes, and packets also include a 2-byte message length integer, a 4-byte sequence number integer, and an 8-byte timestamp integer.

2.3 client11c.c

The final task for this lab was to develop an altered version of the original client, which would loop through all the integers from 1 to 10,000, sending them all to the server and calculating the average round trip time for their echoes. Developing this program did pose a few challenges, but at the time of writing it also compiles and runs on the Tux servers. Initially, we had some issues

with the server only handling about 9,000 pings before apparently freezing, and the client only processing a few thousand before meeting the same fate. This deluge of pings also seemed to inflate the average round trip time of the echoes the client did receive, but by making the client wait a few nanoseconds between sending each message, we were able to ensure that all 10,000 would be correctly sent, received, and processed, as can be seen in the screenshot below.

```
Number of messages so far: 9990
Average round trip time so far: 75.233834 microseconds
Number of messages so far: 9991
Average round trip time so far: 75.230107 microseconds
Number of messages so far: 9992
Average round trip time so far: 75.226381 microseconds
Number of messages so far: 9993
Average round trip time so far: 75.222756 microseconds
Number of messages so far: 9994
Average round trip time so far: 75.219131 microseconds
Number of messages so far: 9995
Average round trip time so far: 75.215608 microseconds
Number of messages so far: 9996
Average round trip time so far: 75.216186 microseconds
Number of messages so far: 9997
Average round trip time so far: 75.216965 microseconds
Number of messages so far: 9998
Average round trip time so far: 75.217644 microseconds
Number of messages so far: 9999
Average round trip time so far: 75.218522 microseconds
Number of messages so far: 10000
Average round trip time so far: 75.219500 microseconds
Number of messages: 10000
Average round trip time: 75.219500 microseconds
```

This lab proved to be an interesting demonstration of what can happen when a server and client are sending and receiving messages faster than they can properly handle them.

3 Lab 1.2

Lab 1.2 focused on TCP socket programming.

3.1 server12.c

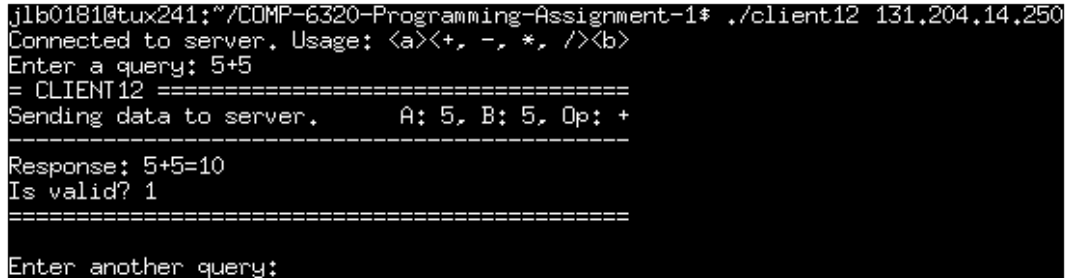
Our first task was to develop a calculator server, which could unpack a message containing two integers and an operator, perform the specified operation, and send the result back to the user. At the time of writing, this program correctly compiles and runs on the Tux servers, as can be seen in the screenshot below.

```
j1b0181@tux243:~/COMP-6320-Programming-Assignment-1$ ./server12
Server is running on port 10020.
A new client has connected!

Bytes received: 9
Bytes sent: 14
```

3.2 client12.c

This lab had just one client, which, like the clients from Lab 1.1, take a hostname as an argument (again, as this was a local loop assignment, we simply used the IP address of the Tux machine the server ran on). Once the client is running, the user is prompted to enter a simple equation, such as 5+5. This string of characters will then be sent to the server, and if the input is valid, the solution will be returned to the user. This can be seen in the screenshot below.



```
jlb0181@tux241:~/COMP-6320-Programming-Assignment-1$ ./client12 131.204.14.250
Connected to server. Usage: <a><+, -, *, /><b>
Enter a query: 5+5
= CLIENT12 =====
Sending data to server.      A: 5, B: 5, Op: +
=====
Response: 5+5=10
Is valid? 1
=====
Enter another query:
```

4 Conclusion

This assignment certainly had its challenges, but it was undoubtedly an effective introduction to socket programming.

In addition to the copies that should be included with this report, the code for this assignment can be found [here](#).