COMP 5320/6320/6326 Design and Analysis of Computer Networks

Lab Assignment 1

Group 4

Hugh Kwon (hzk0070), [hzk0070@auburn.edu](mailto:hzk0070@auburn.edu)

Dehao Yu (dzy0023), [dzy0023@auburn.edu](mailto:dzy0023@auburn.edu)

# INTRODUCTION

In this lab, we have implemented two concurrent server/client programs. The first (lab 1.1) is a simple echo server (server11.c) capable of handling multiple clients through UPD protocol. We provide two client programs (client11b.c/client11c.c) for the echo server. The communication between the server and clients follow Lab11-RFC.

The server program (server12.c) uses TCP protocol and functions as a network calculator. For this server, we implemented a client (client12.c). The client will take a server’s IP address, an artithmetic operator (+, -, \*, /), and two operands as command line arguments and pass it to the server, which is expected to respond with a calculated answer. The communication complies to Lab12-RFC.

# COMPILE AND EXECUTE INSTRUCTIONS

## Compiling

All programs were written in a linux environments. Some libraries used are not compatible with Windows. However, we do not believe we have used any special library that requires special linking via command line flag. So, compiling with simple ***gcc <program>.c*** command will suffice, although we recommend using -o flag to specify the name of executable output.

## Executing

Servers do not take any arguments. Any arguments passed will simply be ignored.

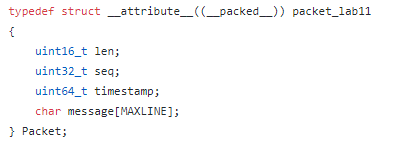
All clients take server IP address as their first argument. Only client12.c require 3 additional arguments. The arguments are expected in a specific order: <sever IP> <op> <operand A> <operand B>, where <op> is one of the following operaotrs: +, -, \*, /. For example, to request a division of 4 by 2, *<client12 executable> <serverIP> / 4 2* is the proper format.

# IMPLEMENTATION

Lab11b:

Client11b.c:

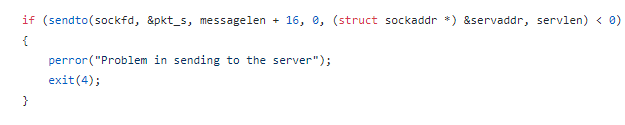
We use this message structure shown below



ii. Prompts the user to enter a string A

printf("Enter a message to send: ");

iii. Sends the string A to Server S



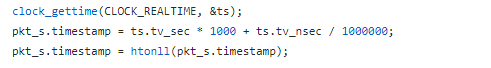
iv. Listens for a response from Server S

C:\Users\DEHAOY~1\AppData\Local\Temp\1537524044(1).png v. Prints out the response received from Server S

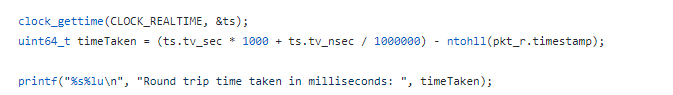
C:\Users\DEHAOY~1\AppData\Local\Temp\1537523682(1).png

vi. Prints out the round trip time (time between the transmission and the reception)

At first, we get the current time and then translate it to right form to send



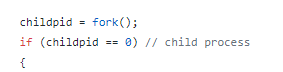
And after the respond we get the current time again and calculate the difference then print it out



Lab11c:

Client11c.c:

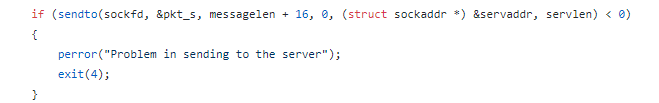
We use fork() to write two processes





1. The first process runs a loop which sends continuously the numbers from 1 to 10,000 as strings (so 10,000 strings in total)



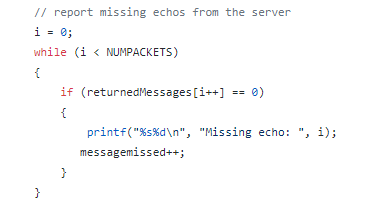


1. The second process receives the responses from the server and reports any missing echo.

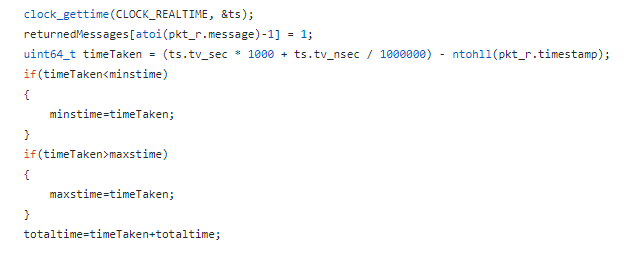
First record missing echo number in a array

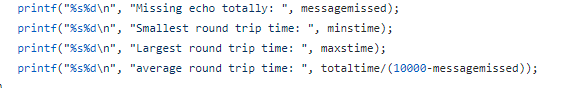
C:\Users\DEHAOY~1\AppData\Local\Temp\1537524657.png

Then we calculate how many



1. At the statistics summary line, report whether there are missing echoes, and the smallest, largest, and average round trip times.





Lab12:

Again, packed structures are used to size the packets as mandated by the RFC. client12.c is much simpler than previous clients. It will take a single arithmetic operation and pass it to the server. The client program will only check for a valid operator and display a response from the server.

Server follows a concurrent TCP model, in case other group’s client can send multiple requests. To ensure interoperability, it still check for a valid operator before performing any arithmetic operation. With a valid operator, the requested operation is carried out and a calculated answer is sent back to the client.

# CONCLUSION

We have implemented two servers and three clients for this lab. All programs performed as expected within our group in both local and networked settings. There aren’t any bugs/problems currently known to us.