Programming Assignment 3

Objective of this assignment:

- Develop and implement in your **preferred**[1] language a simple application using UDP and TCP sockets. The application using UDP sockets must be developed for **this** *Programming Assignment 2*. The application using TCP sockets will be due for *Programming Assignment 3*. Insure the language is already available on Tux machines. It is your responsibility to check. **10 bonus** if client and server are developed with different languages.

What you need to do:

1. Implement a simple UDP Client-Server application (*Programming Assignment 2*)
2. **Implement a simple TCP Client-Server application (*Programming Assignment 3*)**

Objective:

The objective is to implement a client-server application using a safe method: start from a simple **working** code for the client and the server. **You must slowly and carefully *bend* (modify) little by little the client and server alternatively until you achieve your ultimate goal. You must bend and expand each piece alternatively the way a black-smith forges iron. From time to time save your working client and server such that you can roll-back to the latest working code in case of problems**. Failing to follow this incremental approach may result in a ball of wax impossible to debug in case your program does not behave or work as expected.

If you plan to use Java for this programming assignment, you are advised to start from the *Friend* client and server application to implement the calculator server. You will first implement the calculator server using UDP (*Programming Assignment 2*), and then TCP (*Programming Assignment 3*). If using a language other than Java, you are on your own. Insure that you preferred language is already available on Tux machines. It is your responsibility to check.

Part A: Datagram socket programming (*Programming Assignment 2*)

The objective is to design a ***Calculating Server (CS)***. This calculating server performs bitwise boolean and arithmetic computations requested by a client on 16-bit signed integers. Your server must offer the following operations: 1) addition (+), 2) subtraction (-), 3) bitwise ***OR*** (|), 4) bitwise ***AND*** (&), 5) division, and 6) multiplication.

A **client *request*** will have the following format:

Field	TML	Operand 1	Op Code	Operand 2	Request ID
Size (bytes)	1	2	1	2	2

Where

1) **TML** is the Total Message Length (in bytes) including TML. It is an integer representing the **total** number of bytes in the message.
2) **Request ID** is the request ID. This number is generated by the client to differentiate requests. You may use a variable randomly initialized and incremented each time a request is sent.
3) **Op Code** is a number specifying the desired operation following this table

Operation	+	-	|	&	/	*
OpCode	0	1	2	3	4	5

4) **Operand 1:** this number is the first or unique operand for all operations.
5) **Operand 2:** this number is the second operand.

Operands are sent in the **network byte order** (i.e., big endian).
Hint: create a class object *Request* like "Friend", but with the information needed for a request.....

Below are two examples of requests

[1] The language must be available on Tux Machines. Check before developing/implementing.

Request 1: suppose the Client requests to perform the operation 240 + 4: (This is the 5th request)

0x08	00x	0xF0	0x00	0x00	0x04	0x00	0x05

Request 2: suppose the Client requests to perform the operation 240 – 160 (if this is the 9th request):

0x08	0x00	0xF0	0x01	0x00	0xA0	0x00	0x09

The **Server** will respond with a message with this format:

Total Message Length (TML)	Result	Error Code	Request ID
one byte	4 byte	1 byte	1 byte

Where

1) **TML** is the Total Message Length (in bytes) including TML. It is an integer representing the **total** numbers of bytes in the message.
2) **Request ID** is the request ID. This number is the number that was sent as Request ID in the request sent by the client.
3) **Error Code** is **0** if the request was valid, and **127** if the request was invalid (Length not matching TML).
4) **Result** is the result of the requested operation.

In response to ***Request 1*** below

0x08	0x00	0xF0	0x00	0x00	0x04	0x00	0x05

the server will send back:

0x07	0x00	0x00	0x00	0xF4	0x00	0x05

In response to ***Request 2***,

0x08	0x00	0xF0	0x01	0x00	0xA0	0x00	0x09

the server would send back:

0x07	0x00	0x00	0x00	0x50	0x00	0x09

a) **Repetitive Server**: Write a datagram **Calculating Server (ServerUDP.xxx)** in **your preferred language**. This server must respond to requests as described above. The server must bind to port (10010+*TID*) and could run on any machine on the Internet. ***TID*** is your Canvas team #. The server must accept a command line of the form: ***prog ServerUDP portnumber*** where ***prog*** is the executable, ***portnumber*** is the port the server binds to. For example, if your Team ID (GID) is Team 13 then your server must bind to Port # 10023.
   Whenever a server gets a request, it must:
   i. print the request one byte at a time in hexadecimal (for debugging and grading purpose)
   ii. print out the request in a manner convenient for a typical Facebook user: the request ID and the request (operands and required operation)

b) Write a datagram **client** (**ClientUDP.xxx**) in your preferred language:
   i. Accepts a command line of the form: ***prog* ClientUDP *servername PortNumber*** where ***prog*** is the executable, ***servername*** is the server name, and ***PortNumber*** is the port number of the server. Your program must prompt the user to ask for an ***Opcode, Operand1*** and ***Operand2*** where ***OpCode*** is the opcode of the requested operation (See the opcode table). ***Operand1 and Operand2*** are the operands. For each entry from the user, your program must perform the following operations:

ii. form a message as described above
iii. send the message to the server and wait for a response
iv. print the message one byte at a time in hexadecimal (for debugging and grading purpose)
v. print out the response of the server in a manner convenient for a typical Facebook user: the request ID and the response
vi. print out the round trip time (time between the transmission of the request and the reception of the response)
vii. prompt the user for a new request.

Part B: **TCP socket programming (Due for *Programming Assignment 3*)**

Repeat part A using TCP sockets to produce **(ServerTCP.xxx, ClientTCP.xxx)**.

How to get started (If using Java)?

1) Download all files (UDP sockets) to run the "Friend" application used in Module 2 to illustrate how any class object can be exchanged: Friend.java, FriendBinConst.java, FriendDecoder.java, FriendDecoderBin.java, SendUDP.java, and RecvUDP.java.

2) Compile these files and execute the UDP server and client. Make sure they work

3) Create a new folder called Request and duplicate inside it ALL files related to the Friend class object

4) Inside the Folder Request, change ALL occurrences of "Friend" with "Request" including the file names.

3) Adapt each file to your calculator application. Replace the fields used by Friend with the fields used by a request.

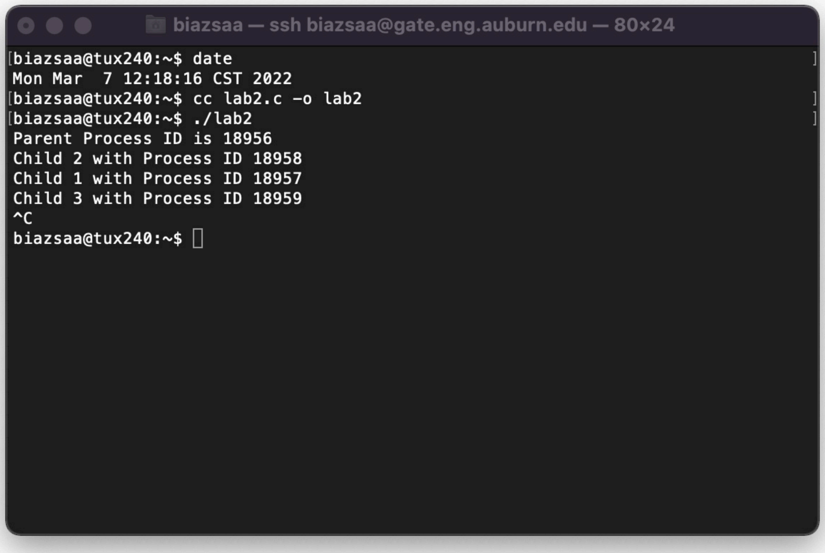
4) Aim to have the client send one request and have the server understand it (just like what we did with a friend object).

5) When your server will receive and print out correctly a request, then you need to send back a response...

6) Create a class object Response....

Report

- Write a report. The report must include screenshots of the client and the server. We must see on the screenshot of the client **four** successful requests for the operations - (subtraction), | (or), & (and), and *. **To receive any credit, the screenshots must clearly show the Tux machine, the username of one of the classmates, and the date**. To get the date, just run the command date before executing your client or server. **Each** missing screenshot will result in **a 25 points penalty**. Your screenshot should have the information on this template:



```
biazsaa — ssh biazsaa@gate.eng.auburn.edu — 80×24
[biazsaa@tux240:~$ date
Mon Mar  7 12:18:16 CST 2022
[biazsaa@tux240:~$ cc lab2.c -o lab2
[biazsaa@tux240:~$ ./lab2
Parent Process ID is 18956
Child 2 with Process ID 18958
Child 1 with Process ID 18957
Child 3 with Process ID 18959
^C
biazsaa@tux240:~$ 
```

- If your program does not work, explain the obstacles encoutered.

What you need to turn in:

- Electronic copy of all your source programs (submit them on Canvas separately).
- **In addition**, put all the source programs in a folder that you name with your concatenated last name and first name. Zip the folder and submit the zipped folder **TOO**. The grader should see on Canvas all your source programs separately **AND** a zip folder containing all the source programs needed to compile and execute your program.
- Electronic copy of the report (including your answers) (standalone). Submit the file as a Microsoft Word or PDF file.

Grading

1) UDP/TCP **client** is worth 40% if it works well: communicates with YOUR server. Furthermore, screenshots of your client and server running on Tux machines must be provided. The absence of screenshots or screenshots on machines other than the Tux machines will incur a 15% penalty.

2) UDP/TCP **client** is worth 10% extra if it works well with a working server from any of your classmates.

1) UDP/TCP **server** is worth 40% if it works well: communicates with YOUR client. Furthermore, screenshots of your client and server running on Tux machines must be provided. The absence of screenshots or screenshots on machines other than the Tux machines will incur a 15% penalty.

2) UDP/TCP **server** is worth 10% extra if it works well with a working client from any of your classmates.