Objective of this assignment:

- To get you familiar with developing and implementing TCP or UDP sockets.

What you need to do:

1. Implement a simple TCP Client-Server application
2. Implement a simple UDP Client-Server application
3. Collect and analyze round trip time measurements for each client.

Objective:

The objective is to implement a simple 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 like 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. Not using this "baby steps" strategy may leave you with a ball of wax hard to debug.

For this programming assignment, you are advised to start from the simple echo client and server to implement a very simple application. Consider to use Wireshark to check whether the protocols you implement meet this assignment requirements.

Programming Assignment 1 (TCP)

(100 points + Possible 10 Bonus Points): TCP "*StringToShortInteger*" Client-Server

Implement the following Client-Server application that will use two programs: a client program myFirstTCPClient.java and myFirstTCPServer.java

a) **Client: myFirstTCPClient.java**

This program must take two arguments: a hostname H and a port number P. The hostname h is a name or a decimal dotted-quad IP address of the server Sv. The port number P is any valid port number where the server Sv binds to. On Tux machines, a valid UDP or TCP port number is in the range 10010-10200. Each team must use Port number 10010+TeamNumber. For example, if your team is Team17, you must use Port # 10010+17=10027.

This client program must:

1) Create a TCP client socket connected with the server Sv running on the machine with hostname (or IP address) h bound to Port number P.

2) Repeatedly perform the following actions:

i) Prompt the user to enter a sentence S that is a valid decimal number in the range 0 to 2^{16}-1.

ii) Send the sentence S as is to the server Sv. If S = "16735", the client must send 5 characters using the UTF-16 encoding scheme. Below is the array of bytes in hexadecimal representing "16735". **Just before sending**, you must display byte per byte in hexadecimal on the client the array of bytes (representing the sentence S) that the client sends. This display byte per byte will allow you to check whether you are respecting the protocol specified by this assignment:

0xFE	0xFF	0x00	0x31	0x00	0x36	0x00	0x37	0x00	0x33	0x00	0x35

iii) Receive the response from the server that will be a **short integer** (2 bytes) representing the sentence S the client sent. **As soon as the client receives the response, display byte per byte in hexadecimal the array of bytes the client receives**. If S = "16735", below are the bytes the client should receive.

The server will send back S as a short integer. For example, to respond to S = "16735", the server will send back this array of bytes in hexadecimal:

0x41	0x5F

iv) Measure the duration between the time when the sentence S was sent and the time a response was received.

v) Display the following information: the received short integer and the time expressed in milliseconds.

vi) Collect the round-trip time for 7 different sentences. Report the min, the max, and the average of the 7 measurements you collected.

To implement the client myFirstTCPClient.java, you should consider starting with the program *TCPEchoClient.java* (provided on Canvas with this programming assignment). Do not forget to change the name of the class inside the program *TCPEchoClient.java* to match your program name. **The client must be implemented in Java.**

b) **Server: myFirstTCPServer.java**

This server program must take one argument: a port number P. The port number P is any valid port number.

This server program must:

1) Create a TCP server socket

2) Wait for a client to connect. When the server receives a request (message):

i) **display byte per byte in hexadecimal the array of bytes received**.

ii) display the received string S for a normal user. This string must match the string S sent by the client

iii) display the IP address and port # of the client,

3) convert the received string S into a short integer A (2 bytes), display the string S, display the short integer A, and echo back the short integer A (2 bytes). Just before sending back the response, **display byte per byte in hexadecimal the array of bytes you send.**

4) **Error**: If the received string S contains characters that are not digits (0, 1, 2, ... ou 9), return the integer A = -1 = 0xFFFF.

In any case, the server must send back exactly 2 bytes.

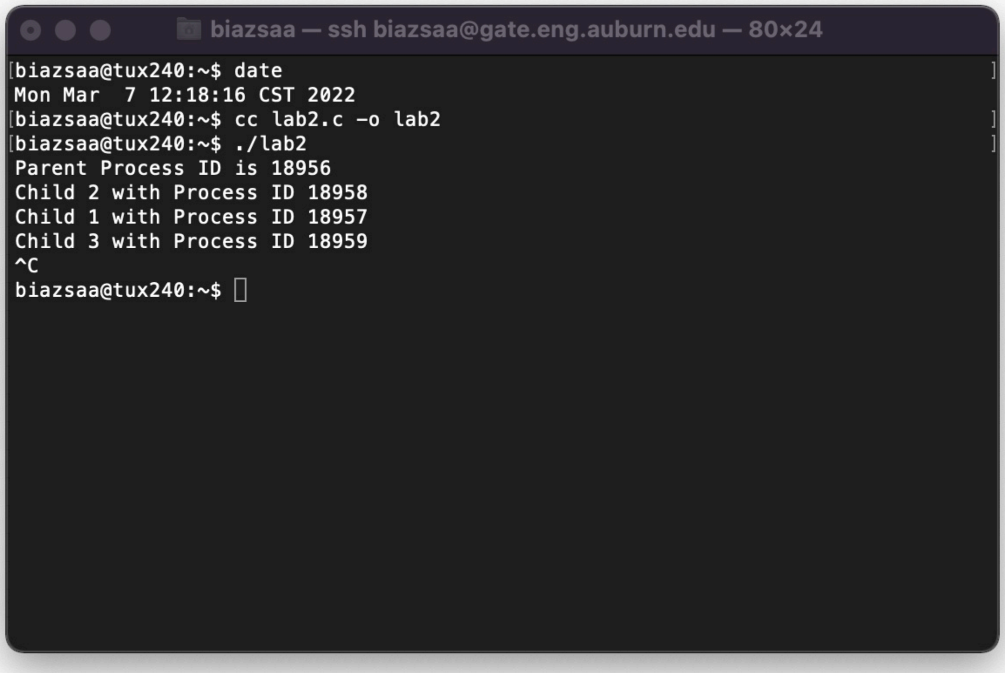
To implement the server myFirstTCPServer.java, you should consider starting with the program *TCPEchoServer.java* (provided on Canvas with this programming assignment). Do not forget to change the name of the class inside the program *TCPEchoServer.java*. **If you implement the server in a language different from Java, you will get 5 points Bonus points. For the language other than Java, the only constraint is that it must already be installed on Tux machines. Check before you start implementing.**

Programming Assignment 2 (UDP)

(100 points + Possible 10 Bonus Points) : UDP "*StringToShortInteger*" Client-Server
Repeat Programming Assignment I using **UDP** sockets. Call the client and server programs myFirstUDPClient.java and myFirstUDPServer.java, respectively.

To implement the server (respectively, client) myFirstUDPServer.java (respectively, myFirstUDPClient.java), you should consider starting with the program *UDPEchoServer.java* (respectively, *UDPEchoClienTimeout.java*) (provided on Canvas with this programming assignment). Do not forget to change the name of the class inside the program.
The client must be implemented in Java. If you implement the server in a language different from Java, you will get 5 points Bonus points. For the language other than Java, the only constraint is that it must already be installed on Tux machines. Check before you start implementing.

Data collection and analysis
For **each** client (UPD or TCP), report separately the min, average, and max round trip time. Include screenshots of your TCP or UDP client and server executing on the Tux machines. Screenshots on machines other than the Tux machines will not receive any credit. **To receive any credit, the screenshots must clearly show the Tux machine name, the username of one of the classmates, and the date**. To get the date, just run the command date before executing your program. **You must have two screenshots per programming assignment: one for the TCP server, the TCP Client for Programming Asssignment I, and the UDP server, and the UDP Client for Programming Assignment 2.** Here is a screenshot containing the Tux machine, the username of one of the classmates, and the date. **Avoid screenshots too small. Screenshots must be easily and conveniently readable.**



```
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:~$ 
```

Report

- Write a report that will report your results..
- Your report must contain the following information:
  - whether the programs work or not (this must be just ONE sentence)
  - the directions to compile and execute your program
  - the information this assignment asks you to report (minimum, average, and maximum round trip times)

required screenshots of the execution of TCP and UDP clients and servers. **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 program. **Each** missing (complete) screenshot will result in **a 25 points penalty**.

What you need to turn in:

- Electronic copy of your source programs (standalone)
- Electronic copy of the report (including your answers) (standalone). Submit the file as a Microsoft Word or PDF file.

Grading

1) TCP client is worth 40% if it works well:

a) **meets** the protocol specifications (30%)

b) communicates with YOUR server. (10%) 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 25 points penalty per missing screenshot

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

All other server and clients (TCP server, UDP client, and UDP server) will be graded the same as the TCP client (40% + 10%).