Objective of this assignment:

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

What you need to do:

1. Implement a simple UDP Client-Server application
2. Execute client and server and collect readable screenshots with required information
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 **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. 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 Friend client and server code to implement this simple application. The Friend app was presented during the lectures. Consider using *Wireshark* to check whether the protocols you implement meet this assignment requirements.

Hint: look at how to "How to get started?" below

Programming Assignment

(100 points): UDP "*ShortIntegerToString*" Client-Server

a) **Client**

The client 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 the server Sv binds to.

This client must:

1) Create a UDP 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 number N that is a valid decimal number in the range -32768 to 32767 ($-2^{15}$to $2^{15}-1$).

ii) Send a ***request*** containing the number N as a **short integer** (2 bytes representing the number N). **Just before sending** N, you must display byte per byte in hexadecimal on the client the array of bytes (representing the number N) that the client sends. This display byte per byte will allow you to check whether you are respecting the protocol specified by this assignment. For example, to send a number N = 16735, the client will send bits that we can represent as this array of bytes in hexadecimal:

0x41	0x5F

iii) Receive a **response** containing the string S corresponding to the sent number N. When the server receives the number N, it converts it to a string S and send it back to the client. For example, if the server receives the number N = 16735, it will convert N to the string S = "16735" and send it back to the client using the **UTF-16 encoding scheme**. Below is the array of bytes in hexadecimal representing "16735". **As soon as the client receives the string *S***, you must display byte per byte in hexadecimal on the client the array of bytes (representing the sentence S) that the client received. This display byte per byte will allow you to check whether you are respecting the protocol specified by this assignment. This is the array that the client should receive if the number N it sent to the server was 16735:

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

iv) Measure the duration between the time when the number N is sent and the time a response with the string S is received.

v) Display the following information: the received string S and the time expressed in milliseconds.

vi) Collect the round-trip time for 5 different numbers. Report the min, the max, and the average of the measurements you collected.

The client must be implemented in Java. This program may use additional classes as for the Friend application.

Your code must be neat and pleasant to read. Comment the code appropriately. If starting from some other code, delete all unnecessary instructions (do not just comment out the unnecessary instructions). A code not neat or pleasant will be penalized up to -25 points.

b) **Server**

This server must take one command argument: a port number P. The port number P is any valid port number. The *Engineering Networking Services* office restricts the port numbers to use. On Tux machines, a valid port number is restricted to be in the range 10010-10200. Each team must use Port number 10010+TeamNumber. For example, if your team is Team17, you must use Port # 10027 (= 10010+17) for the server.

The server must:

1) Create a UDP server socket

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

i) **display byte per byte in hexadecimal the array of bytes received**. Note that strictly speaking, the server receives bits.

ii) display the number N for a normal user. The number N must match the number N sent by the client.

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

3) convert the received number N (2 bytes) into a string S (variable number of bytes), **encode the string S using the UTF-16 encoding scheme**, and echo back the string S to the client**). Just before sending back the response**, **display byte per byte in hexadecimal the array of bytes (representing S) you send.**

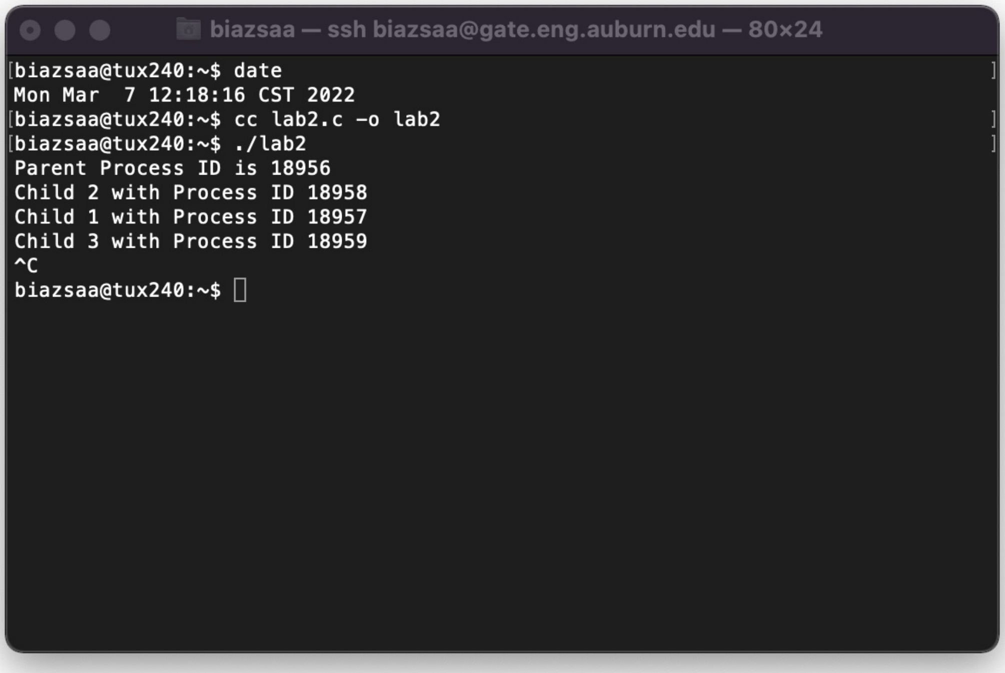
4) **Error**: If the server receives a number of bytes not equal to 2, return the string S = "****".

To implement the server, you should consider starting with the Friend code. **If you implement a successful 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 the Tux machines for your chosen language before you start implementing.**

Your code must be neat and pleasant to read. Comment the code appropriately. If starting from some other code, delete all unnecessary instructions (do not just comment out the unnecessary instructions). A code not neat or pleasant will be penalized up to -25 points.

Data collection and analysis

For **the** client, report separately the min, average, and max round trip time. Include screenshots of your 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**. In other words, if any information (username, date, or tux machine name) is missing, the assigned credit for the assignment will be 0. **You must have two screenshots: one for the server and one for the client.** Here is a screenshot containing the Tux machine, a username, and a date. **Avoid screenshots too small. If screenshots are not easily and conveniently readable, they will be considered missing. 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:~$ 
```

How to get started?

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, FriendEncoder.java, FriendEncoderBin.java, FriendDecoder.java, FriendDecoderBin.java, SendUDP.java, and RecvUDP.java.

2) Compile these files and execute the UDP server and client. Ensure 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 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 (a missing report incurs a 30 points penalty)

- 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 programs
    - the information this assignment asks you to report (minimum, average, and maximum round trip times) and the

required screenshots of the execution of the client and server. **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 programs. **Each** missing or incomplete screenshot will result in **a 50 points penalty**.

What you need to turn in:

- Electronic copy of your source programs (standalone, i.e. **NOT** in a zipped folder)
- Electronic copy of the report (including your answers) (standalone, i.e. **NOT** in a zipped folder). Submit the file as a Microsoft Word or PDF file.
- **In addition**, put all files in a zipped folder and submit the zipped folder.

Grading

1) Client is worth 40% if it works well:

a) **meets** the **protocol** specifications (20%) and the user interface (10%)

b) communicates correctly 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 50 points penalty per missing screenshot

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

The server is graded the same as the client (30% + 10% + 10%).