COMP 4320 Introduction to Computer Networks

Assignment 1

Groups of up to 3 Due by Labor Day

Programming Assignment (**Group,** 100 pts)

IMPORTANT:

1) *Your code will be tested and graded on the Engineering Unix (Tux) machines. If the code does not work on those machines, you will not get **any** credit even if your code works on every other machine in the universe.*
2) *A late submission will get a 50% penalty if submitted after 9:00pm on the due day. After 9:00pm the next day, you cannot submit the lab.*
3) *One submission per group.*
4) *Writing and presentation of your report are considered to grade your lab.*
5) *The quality of your code will be evaluated.*

Programming Assignment (Turned in by one group mate)

First, it is assumed that by Labor day, 1) you have an engineering Unix account, 2) you can edit, 3) you can compile, and 4) you can execute C programs on the Unix (Tux) machines. You can use any personal computer or computing lab to remotely access the Engineering Unix machines.
Second, it is assumed that by Labor Day, you have your group partners. Past labor day, you will be on your own with a **penalty of 5 points per lab**: I want you to learn to work on a team and deal with **human** problems.

Objectives:
Look at "How to get started?" at the very end of the lab.

Lab 1: Introduction to Socket Programming

First, find two groupmates and request from me a group id (*GID*). You must implement and test your programming assignments on a machine (Tux machines) on the engineering network system. Your work will be tested and graded on an engineering machine (Tux machine).

I strongly advise you to read beej's guide at :
http://beej.us/guide/bgnet/ *(Simple, easy-to-understand tutorial)*

FOR ALL THREE LABS IN THIS SEMESTER, you must use C for one of the programs and a different language for the second.

Example: if you write a client in C, then you must use another language for the server.

Part A: Datagram socket programming

The objective is to design a ***Calculating Server (CS)***. This *CS* performs logical and arithmetic computations requested by a client on 16 bit integers. Your server must offer the following operations: 1) addition (+), 2) subtraction (-), 3) logical ***OR*** (|), 4) Logical ***AND*** (&), 5) Shift Right (>>), and 6) Shift Left (<<).

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

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

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 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) **Number Operands** is the number of operands: 2 for (+, -, |, &) and shifts.
5) **Operand 1:** this number is the first or unique operand for all operations.
6) **Operand 2:** this number is the second operand for operations (+, -, |, &). It is the second operand of the addition or a subtraction. It is the number of bits to shift by for the shift operations.

Operands are sent in the **network byte order** (i.e., big endian)

Below are two examples of requests

Request 1: suppose the Client requests to perform the operation 240 >> 4, i.e., shift the number 240 right by 4 bits (if this is the 7th request):

0x08	0x07	0x04	0x02	0x00	0xF0	0x00	0x04

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

0x08	0x09	0x01	0x02	0x00	0xF0	0x00	0xA0

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

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

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.
4) **Result** is the result of the requested operation.

In response to ***Request 1***,

0x08	0x07	0x04	0x02	0x00	0xF0	0x00	0x04

the server would send back:

0x07	0x07	0x00	0x00	0x00	0x00	0x0F

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

0x08	0x09	0x01	0x02	0x00	0xF0	0x00	0xA0

the server would send back:

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

a) **Repetitive Server**: Write a datagram **Calculating Server (ServerUDP.xxx)**. This program must be written in any language (other than C or C++) you prefer. This server must respond to requests as described above. The server must run on port (10010+*GID*) and could run on any machine on the Internet. ***GID*** is your group ID. The server must accept a command line of the form: ***server portnumber*** where ***portnumber*** is the port where the server should be working. For example, if your Group ID (GID) is 13 then your server must listen on Port # 10023.

b) Write a datagram **client** (**ClientUDP.c**) in **C or C++** which:

   i. Accepts as command line of the form: ***client servername PortNumber*** where ***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,*** an ***Operand1*** and if needed an ***Operand2*** where ***OpCode*** is the opcode of the requested operation (See the opcode table). ***Operand1 and Operand2*** *(if applicable)* are the operands. For each entry from the user, your program must perform the following operations:

   ii. forms a message as described above

   iii. Sends the message to the server and waits for a response

   iv. Prints out the response of the server: the request ID and the response

   v. Prints out the round trip time (time between the transmission of the request and the reception of the response)

   vi. Prompt the user for a new request.

Part B: **TCP socket programming**

Repeat part A using TCP sockets to produce **(ServerTCP.c, ClientTCP.xxx)**. The server must be written in C or C++. The client must be written in any language other than C or C++.

Grading:

1) 25 points per program (2 clients and 2 servers)
2) Code does not compile on Tux machine: 0% credit
3) Code compiles on Tux machines but does work: 5% credit
4) Code compiles and interacts correctly with counterpart from the **same** group: 60% credit
5) Code compiles and interacts correctly with counterpart from other groups: 100% credit

"How to get started?":

This is just an introduction to socket programming: I advise you to work **ACTIVELY** to implement these programs.

Step 1: download, compile, and execute Beej's sample programs (UDP-client.c and UDP-server.c)
Step 2: get familiar with this code: study key socket programming calls/methods/functions
Step 3: improve the server to echo (send back) the string it receives.
Step 4: improve the client to receive the echo and to print it out.
Step 5: SAVE the improved versions (you may need to roll back to them if things turn bad)
Step 6: All you need now is "forming" your messages based on the specified format (lab 1 protocol).

For the TCP socket programming, redo Step 1-6 using Beej's stream sample programs TCP-server.c and TCP-client.c.

Help Tools:

Besides this lab, I posted the following programs:

1) **TCP-server.c** (Beej's stream server)
2) **TCP-client.c** (Beej's stream client)
3) **UPD-server.c** (Beej's datagram server)
4) **UDP-client.c** (Beej's datagram client)
5) **TCPServerDisplay.c** : this program is a modification of Beej's TCP server: it takes as input a port number to run on and displays in **hexadecimal** what it receives.
6) **UDPServerDisplay.c :** this program is a modification of Beej's UDP server: it take as input a port number to run on and display in **hexadecimal** what it receives.
7) **packedStruct.c** : this program demoes how to create a "packed" struct in C.

What to turn in?

1) **Electronic copy** of your report (**standalone**) and source code (**zipped**) of your programs. All programming files (source code) must be put in a **zipped folder** named lab1-name1-name2 where *name1* and *name2* are the last names of two teammates. Zip the folder and post it on Canvas. Submit **separately** (not inside the zipped folder) the report as a Microsoft or PDF file. **A penalty of 10 points will be applied if these instructions are not followed.**

2) Your report must:
    a. state whether your code works
    b. Clearly explain how to compile and execute your code. If the TA needs your presence to compile and execute your code, then a 30% penalty will be applied.
    c. If needed, report/analyze (as appropriate) the results. The quality of analysis and writing is critical to your grade.

Good writing and presentation are expected.

If the TA is unable to access/compile/execute your work based on your report, 30% will be applied.
If the turnin instructions are not followed, 10 pts will be deducted.