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

Lab Assignment 1
Groups of 2, Due by Sep 21, 2018
Programming Assignment (**Group,** 100 pts)

Lab 1: Introduction to Socket Programming

First, it is assumed that by today, you already have your AU Engineering account, so that you can edit, compile, and execute C programs on Engineering Unix machines (Tux machines) and on Mallard of AU.

Second, **y**ou must implement and test your programming assignments on Mallard of AU and a Tux machine in the engineering network. Your work will be tested and graded on the same system setup (i.e., Mallard and Tux).

Third, find a teammate and request from me a group ID (GID). In your request email please indicate the names of both group members.

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

Important Notice: For all labs in this semester, you must use C for the implementation. Moreover, your implementation must follow the protocol agreement specified in Lab11-RFC.pdf (for Lab 1.1) and Lab12-RFC.pdf (for Lab 1.2), so that servers and clients from different groups can interact with each other.

Lab 1.1: Introduction to Socket Programming

The objective is to get familiar with basic socket programming function calls. You will design and implement two separate programs: a server and a client. Server and clients will be running on local loop and on different machines and communicate on the Internet. You will also make some basic measurements of the round trip time for requests.

Datagram socket programming (**warm-up**)

a) Write a **concurrent** datagram (UDP) **echo** server **S** (**server11.c**). This server must send back any message you send to it. The server must "listen" on port 10010. The server must be able to handle concurrently multiple requests.
b) Write a datagram client (**client11b.c**) which:
    i. Accepts as command line the **name** of the echo server
    ii. Prompts the user to enter a string **A**
    iii. Sends the string A to Server S
    iv. Listens for a response from Server S
    v. Prints out the response received from Server S
    vi. Prints out the round trip time (time between the transmission and the reception)
c) Write a client (**client11c.c**) with **two** processes:
    i. The first process runs a loop which sends continuously the numbers from 1 to 10,000 as strings (so 10,000 strings in total)
    ii. The second process receives the responses from the server and reports any missing echo.

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

Your implementation should be able to run on the following two setups: (1) **local-loop** and (2) **networked**. In (1), both the server and all clients are running on the same Tux machine. In (2), the server is running on Mallard, and all clients are running on different Tux machines.

Lab 1.2: TCP Calculator

Write a TCP calculator server (**server12.c)** and a TCP calculator client (**client12.c**).

a) The TCP client must :

i. accept as command line two 32 bit unsigned integers (***a*** and ***b***) and a character ***c*** that can be: '+', '-', 'x', or '/'.

ii. bundle ***a***, ***b***, and ***c*** into an application **message *M*.** The size of the application message ***M*** should not exceed 10 bytes.

iii. send ***M*** to the server

b) The TCP server must:

i. receive message ***M***

ii. extract a, b, and c

iii. perform the operation requested

iv. send back the result with repeating the operands and operation. The size of the message ***M*** (at the application layer) should not exceed 16 bytes.

Your implementation should be able to run on the following two setups: (1) **local-loop** and (2) **networked**. In (1), both the server and the client are running on the same Tux machine. In (2), the server is running on Mallard, and the client is running on a Tux machine.

Grading:

1) 20 points per program (3 clients and 2 servers)
2) Code does not compile on Tux machine: 0% credit
3) Code compiles on Tux machines but does work: 10% credit
4) Code interacts correctly with counterpart from the same group in local loop: 60% credit
(5) Code interacts correctly with counterpart from the same group in networked setup: 90% credit
5) **Code compiles and interacts correctly with counterpart from other groups: 100% credit**

Advice to complete these exercises:

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

Step 1: download, compile, and execute Beej's sample programs for Section 6.3 (talker.c (client) and listener.c (server) for the datagram (UDP) sockets)
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 received.
Step 4: improve the client to receive the echo and print it out.
Step 5: SAVE the improved versions (you may need to roll back to them when things will go bad)
Step 6: All you need now is "forming" your messages based on the specified format.

For the TCP socket programming, redo Step 1-6 using Beej's stream sample programs (Sections 6.1-6.2)

What to turn in?

1) Electronic copy of your report and code. These source codes must be named as shown above. The report is named as report.doc or report.pdf. The code and your report must be put in a

folder named lab1_XX where XX is your group ID. Zip the folder and post it on Canvas. Each group please only submits one copy. **Failing to submit the proper format will result in 20% penalty.**

2) Your code MUST compile and execute on engineering machines tuxXYZ.
3) Your report must:
    a. include GID, student names, and email addresses at the beginning.
    b. state whether your code works
    c. explain how to compile and execute your code
    d. report bugs/problems

If it is unable to access/compile/execute your work, no credit will be awarded.