Department of Computer Science and Engineering
Institute of Technical Education & Research, SOA, Deemed to be University

MINOR ASSIGNMENT-07

UNIX Network Programming (CSE 4042)

Publish Date: 09-07-2022 **Submission Date:** 18-07-2022

Working with elementary UDP sockets

This assignment is designed to practice with development of client server application(s) using UDP Socket API.

1. Write an UDP client server program to send a string **ITER** from client to server. The server will display the string as well as the client protocol address.

2. The client reads a number and sends to the server. The server doubles it and sends back to the client.

3. The client reads a line of words separated by white space and sends to the server. The server transforms the line of words in which the words appear in the reverse order and sends back to the client. The client displays original line and the received line from the server. For example *Alice likes Bob* transforms to *Bob likes Alice*. Implement a function for reversing the words in a string *s*.

4. The client writes a datagram of length 0 and sends to the server. Verify that it is acceptable and display the same in server side.

5. Let an UDP client sends a number to UDP server and the server finds the sum of the digits of the received number. The server sends the sum to the client. You have to modify the 4th parameter, **from**, argument to **`recvfrom`** is a null pointer and the corresponding 5th parameter, **addrlen**, also to be a null pointer to indicate that the server is not interested to know the protocol address of who send the data.

6. Design a TCP client server for sending and receiving a string application using **`recvfrom`** and **`sendto`** instead of **`read`** and **`write`**. State the reason, why normally it is not required to do this in TCP client server application.

7. The client reads 10 numbers and sends them to the server one by one. The server displays them one after another.

8. Design an UDP server program and two UDP client programs. The first client will send a string in small case to the server. The server will display the string along with the clients who sends the data. Now the server will send the upper case of the string to the second client and the second client will display the string forwarded from the server.

9. [**Verify received response:**] Let a UDP client sends a message, **`''COVID-19''`**, to UDP server. The server will display the string and also the server will also display the protocol address of the client. Here any other process that know the client's ephemeral port number could send datagrams to the previos client. Show the case that another client send datagram to the client. Now modify the original client code to ignore any received datagrams that are not from the server to whom you sent the datagram.

10. Write a connected UDP client-server program using **`connect`** to exchange a number in between them.

11. Write an unconnected UDP client-server program to examine a case in which the client is started without starting the server. If you do so and type a line to the client, then conclude what happens to the client. Now modify the unconnected UDP to connected UDP to realise the case of an *asynchronous error*.

12. [**Determining outgoing interface with UDP:**] Develop an UDP client-server program that uses **`connect`** to determine outgoing interface for the client. For example:

```
1. Run client as $./client1 server-port server-IP
2. Output at client: local protocol address 127.0.0.1:5678
3. Where 5678 is the client's ephemeral port and 127.0.0.1
    is the client's IP
```

13. A process with a connected UDP socket can call **`connect`** again for that socket for one of two reasons:

    - To specify a new IP address and port
    - To unconnected the socket

    Develop a program to demonstrate the above cases for a connected UDP socket.

14. [**UDP Echo Server:**] Design an UDP client/server program for an echo server that performs the following steps: [*Refer chapter 8.3 of your text book*]

    i) The client reads a line of text from its standard input and write the line to the server.

    ii) The server reads the line from its network input and echoes the line back to the client.

    iii) The client reads the echoed line and prints its on its standard output.

    iv) The server will keep a track of the client number and the protocol address that is connected to the server.

    v) The UDP echo server will use a user-defined function, **`datagram_echo()`**, to echo lines on a datagram socket.

    vi) The client will also call a user-defined function, **`datagram_client()`**, to send line of text to the server.

    vii) In the client side, the function **`datagram_client()`** must meet the given four steps such as (1) read a line from the standard input using **`fgets`**, (2) send the line to the server using **`sendto`**, (3) read back the server's echo using **`recvfrom`** and (4) print the echoed line to the standard output using **`fputs`** respectively in a loop till you type EOF character (CTRL+D), which terminate the client.

15. [**Optional**] **Lack of flow control with UDP:** With reference to **Figure 8.19** of the Text Book, use the function **`dg_cli`** in UDP client to send a fixed number of datagrams to the UDP serevr without reading from the standard input. Assume that the function writes more than 2000 1400-byte to the server. Also, refer Text Book **Figure 8.20** for **`dg_echo`** function in server to receive the datagrams and count the number received. The server no longer echoes datagram back to the client. Terminate the server with terminal interrupt key (`SIGINT`) after successful execution of client and server. Now for the above set up for the client and serevr, **examine the effect of UDP not having any flow control**. Run the command **`netstat -s`** on the server, both before and after, as the statistics that are output tell how many datagram were lost. Attach the screenshot of the output on the server for the command **`netstat -s`**.

You are required to show the sample input and output for various part of the assignment.