CS 342: Computer Networks Lab

(July-November 2023)

Assignment – 1: Socket Programming

This assignment is a programming assignment where you need to implement an application using socket programming in C/C++ programming language. This is an **INDIVIDUAL** assignment; each student needs to create and submit the assignment individually. No group work will be accepted. The applications' description is given in this document.

Instructions:

- The application should be implemented with socket programming in C/C++ programming language only. No other programming language other than C/C++ will be accepted.
- Submit the set of source code files of the application as a zipped file on MSTeams (maximum file size is 1 MB) by the deadline of 11:55 pm on < 14.08.2023 > (hard deadline). The ZIP file's name should be the same as your role number.
- The assignment will be evaluated offline/through viva-voce during your lab session on Monday, <14.08.2023> where you will need to explain your source codes and execute them before the evaluator. The details of the venue and time will be communicated to you later, on Teams.
- Write your own source code and do not copy from any source. Plagiarism detection tool will be used, and any detection of unfair means will be penalized by awarding NEGATIVE marks (equal to the maximum marks for the assignment).

Question 1: Concurrent Base64 Encoding Communication with TCP Sockets (25 marks)

Scenario: You are required to implement a Concurrent Base64 Encoding Communication protocol using TCP sockets. The communication will take place between a server and multiple clients. The server will wait for connections from clients, and once connected, clients will send messages to the server after encoding them in Base64. The server will then decode the messages, print them, and send an acknowledgment back to the client. Clients can send multiple messages until they decide to close the communication gracefully.

Assumptions:

- The maximum length of a message is fixed and defined as MSG_LEN.
- The server IP address and port number will be provided as command-line arguments for both the server and the client programs.

Prototypes for Client and Server:

Client: <executable_code> <Server_IP_Address> <Server_Port_Number>

Server: <executable_code> <Server_Port_Number>

Message Format: The messages exchanged between the client and server consist of three fields:

The messages used to communicate contain the following fields:

< Message_ Type > < Message >

I. Message_Type: An integer indicating the type of the message.

Type 1: Regular message

Type 2: Acknowledgment message

Type 3: Close communication message

II. Message: A character array of length MSG_LEN to hold the content of the message.
III. < Message> content of the message in Type 3 message can be anything.

Question 2. Write a C/C++ program to create a simple chat server and develop a client to exchange messages with the server. The instructions for the program are given below. (20 marks)

Instructions:

1) Set up the Server:

Write a program to create a simple chat server that binds to a specific IP address and port. The server should listen for incoming connections from clients. When a client connects, the server should display a message indicating the successful connection.

2)Handle Multiple Clients:

Modify the server program to handle multiple clients simultaneously. Ensure that each client connection is managed independently, and messages from one client are not mixed with messages from another.

3)Implement Client Communication: Write the program for the client application. The client should connect to the server and display a message indicating a successful connection.

4)Client-Server Message Exchange:

Implement a simple chat mechanism between the client and the server. The client should be able to send messages to the server, and the server should display those messages. Similarly, the server should be able to send messages to the client, and the client should display those messages.

5)Graceful Exit:

Implement a command in the client and server applications to allow users to exit the chat gracefully. For example, typing "/exit" in the client should terminate the connection and close the client application.

Students can explore ways to enhance the user interface of the client application. This could include adding colors, timestamps, or custom messages for certain actions and Encourage students to add error handling to the application to deal with unexpected situations gracefully. For instance, handle cases where the server or client disconnects unexpectedly.

Question 3) Network Calculator (20+15 marks)

This assignment may be completed using either C or C++.

Assignment The goal of this assignment is to implement a TCP client and server and a UDP client and server (for a total of four different programs). Your TCP or UDP client/server will communicate over the network and exchange data. The user interface (i.e., what's displayed to the user) should look the same for both the TCP and UDP applications. Write a program named CalcClientTCP.c (or .cpp) that performs the following functions:

1. Take a server hostname and a port number as command-line arguments. Note: Your client must resolve the hostname into an IP address.
2. Connect to the server at the given hostname and port using TCP.
3. Print the IP address and port of the server.
4. Ask the user for a simple arithmetic expression to calculate.

Note: The user prompt must contain instructions for the user, including the proper format of the arithmetic expression and the sentinel value that will be used to indicate that the user wishes to quit.

5. Send the expression to the server.
6. Read the answer from the server.
7. Display the answer to the user.
8. Repeat steps 4-7 until the user enters the sentinel value given in the user prompt. When the user enters the sentinel, the client must close the connection to the server and quit.

Write a program named CalcServerTCP.c (or .cpp) that performs the following functions:

1. Take a port number as a command-line argument.
2. Listen for a TCP connection on the port specified.
3. Print the IP address and port of the connected client.
4. Receive data from the client.
5. Evaluate the arithmetic expression.

Note: You must allow the user to add, subtract, divide, multiply, and raise a number to a power. You must also be able to handle negative numbers and numbers with decimals.

6. Send the result back to the client.
7. If the connection is still open, repeat the steps 4-6 until the user presses Ctrl-C. If the connection is closed, repeat steps 2-6 until the user presses Ctrl-C. Though not required, it may be helpful for you to output debugging information from your server.

The UDP application should perform all of the functions as the TCP application in the aforementioned part and it should use UDP as the transport protocol instead of TCP. The UDP client should be named CalcClientUDP.c (or .cpp), and the UDP server should be named CalcServerUDP.c (or .cpp).

After you have completed your programs, write the answers to the following questions in a file named REPORT.txt:

1. Start your TCP client application without the TCP server running. What happens? Why?
2. Start your UDP client application without the UDP server running. What happens? Why?

Make sure you do sufficient error handling such that a user cannot crash your client or server. For instance, what will you do if the user provides invalid input or does not provide any command-line arguments?

Example The following is an example of execution of the TCP version assuming that the client is compiled with

```
g++ CalcClientTCP.c –o CalcClientTCP –lnsl –lsocket
```

and the server is compiled with

```
g++ CalcServerTCP.c –o CalcServerTCP –lnsl –lsocket
```

Server

Client

```
% ./CalcServerTCP 50000

Server listening on port 50000

Client 128.82.4.75 on port 5983 connected






Received from client: 3.5 * -4
Sending to client: -14








Client closed connection

Server listening on port 50000
```

```
% ./CalcClientTCP cash 50000

TCP client connected to 128.82.4.7 on
port 50000

Enter an expression in the following
format:
operand1 operator operand2
Valid operators are + - * / ^.
To quit, enter -1.
3.5 * -4

ANS: 3.5 * -4 = -14

Enter an expression in the following
format:
operand1 operator operand2
Valid operators are + - * / ^.
To quit, enter -1.
-1
Bye!

%
```