CS 558: Computer Systems Lab
(January-May 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. The applications' description is given in this document.

Instructions:

- Each group needs to implement one application assigned to you and make one single submission on MSTeams. Only one member from a group needs to make the submission. The information about the assignment of applications to groups is contained in **Table 1** given below.

- The application should be implemented with socket programming in C/C++ programming language only. No other programming language other than 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 Wednesday, 25th January 2023 (hard deadline)**. The **ZIP file's name should be the same as your group number**, for example, "Group_4.zip", or "Group_4.rar", or "Group_4.tar.gz".

- The assignment will be evaluated offline/through viva-voce during your lab session on **Friday, 27th January 2023 (ML-5: 9:00 am to 11:55 am)** where you will need to explain your source codes and execute them before the evaluator

- **Write your own source codes and do not copy from any source. Plagiarism detection tool will be used and any detection of unfair means will be penalised by awarding NEGATIVE marks (equal to the maximum marks for the assignment).**

Reference Text Book:

"*Unix Network Programming*", Volume 1, by W. Richard Stevens (publisher: Prentice Hall) (refer to first few chapters)

Application No	Group No
1	1,6,11,16,21
2	2,7,12,17,22
3	3,8,13,18
4	4,9,14,19
5	5,10,15,20

Table 1

Application #1: *Point-of-Sale Terminal using socket programming*

Use socket programming to implement a simple client and server that communicate over the network and implement a simple application involving Cash Registers. The client implements a simple cash register that opens a session with the server and then supplies a sequence of codes (refer ***request-response messages format***) for some products. The server returns the price of each one, if the product is available, and also keeps a running total of purchases for each clients transactions. When the client closes the session, the server returns the total cost. This is how the point-of-sale terminals should work. You can use a TXT file as a database to store the UPC code and item description at the server end.

You also require implementing a "***Concurrent Server***", i.e., a server that accepts connections from multiple clients and serves all of them *concurrently*.

Request-response messages format

Request_ Type	**UPC-Code**	**Number**

Where
- ***Request_Type*** is either 0 for ***item*** or 1 for ***close***.
- ***UPC-code*** is a 3-digit unique product code; this field is meaningful only if the ***Request_Type*** is 0.
- ***Number*** is the number of items being purchased; this field is meaningful only if the ***Request_Type*** is 0.

For the ***Close*** command, the server returns a number, which is the total cost of all the transactions done by the client. For the ***item*** command, the server returns:

Response_ Type	**Response**

Where:

<Response_type> is 0 for ***OK*** and 1 for ***error*** If ***OK***, then <Response> is as follows:

- if client command was "close", then <response> contains the total amount
- if client command was "item", then <response> is of the form <price><name>
  where
  <price> is the price of the requested item
  <name> is the name of the requested item

If ***error***, then <Response> is as follows: a null terminated string containing the error; the only possible errors are "***Protocol Error***" or "***UPC is not found in database***".

You should accept the IP Address and Port number from the command line (Don't use a hard-coded port number). Prototype for command line is as follows:

Prototypes for Client and Server

Client: <executable code><Server IP Address><Server Port number>
Server: <executable code><Server Port number>

The connection to the server should be gracefully terminated. When the server is terminated by pressing ***control C***, the server should also gracefully release the open socket (Hint: requires use of a signal handler). *Please make necessary and valid assumptions whenever required.

Application #2: *Client-Server programming using both TCP and UDP sockets*

In this assignment, you require to implement two C programs, namely server and client to communicate with each other based on both TCP and UDP sockets. The aim is to implement a simple 2 stage communication protocol.

Initially, server will be waiting for a TCP connection from the client. Then, client will connect to the server using server's TCP port already known to the client. After successful connection, the client sends a Request Message (Type 1 message) to the server via TCP port to request a UDP port from server for future communication. After receiving the Request Message, server selects a UDP port number and sends this port number back to the client as a Response Message (Type 2 Message) over the TCP connection. After this negotiation phase, the TCP connection on both the server and client should be closed gracefully releasing the socket resource.

In the second phase, the client transmits a short Data Message (Type 3 message) over the earlier negotiated UDP port. The server will display the received Data Message and sends a Data Response (type 4 message) to indicate the successful reception. After this data transfer phase, both sides close their UDP sockets.

The messages used to communicate contain the following fields:

Message_Type	**Message_Length**	**Message**

1. Message_type : integer
2. Message_length : integer
3. Message : Character [MSG_LEN], where MSG_LEN is an integer constant

<Data Message> in **Client** will be a **Type 3** message with some content in its message section.

You also require implementing a "***Concurrent Server***", i.e., a server that accepts connections from multiple clients and serves all of them *concurrently*.

You should accept the IP Address and Port number from the command line (Don't use a hard-coded port number). Prototype for command line is as follows:

Prototypes for Client and Server
Client: <executable code><Server IP Address><Server Port number>
Server: <executable code><Server Port number>

*Please make necessary and valid assumptions whenever required.

Application #3: *Error Detection using Cyclic Redundancy Code (Using CRC-8)*

In this assignment, your aim will be to implement a simple Stop-and-Wait based data link layer level logical channel between two nodes **A** and **B** using socket API, where node **A** and node **B** are the client and the server for the socket interface respectively. Data link layer protocol should provide the following Error handling technique in Data Link Layer.

- Error Detection using Cyclic Redundancy Code (using CRC-8 as generator polynomial, i.e. $\mathbf{G(x) = x^8+x^2+x+1)}$

Operation to Implement:

Client should construct the message to be transmitted (T(x)) from the raw message using CRC.

At the sender side T(x) is completely divisible by G(x) (means no error), send ACK to the sender, otherwise (means error), send NACK to the sender.

You must write error generating codes based on a user given BER or probability (random no between 0 and 1) to insert error into both T(x)and ACK/NACK.

If NACK is received by the sender, it should retransmit the T(x) again following the above steps.

In the client side also implement Timer Mechanism to detect the timeout (in case of error in ACK/ NACK) and retransmit the message T(x) again once time out happens.

You also require implementing a "***Concurrent Server***", i.e., a server that accepts connections from multiple clients and serves all of them *concurrently*.

You should accept the IP Address and Port number from the command line (Don't use a hard-coded port number). Prototype for command line is as follows:

Prototypes for Client and Server

Client: <executable code><Server IP Address><Server Port number>
Server: <executable code><Server Port number>

The connection to the server should be gracefully terminated. When the server is terminated by pressing ***control C***, the server should also gracefully release the open socket (Hint: requires use of a signal handler). *Please make necessary and valid assumptions whenever required.

Application #4: *Base64 encoding system using Client-Server socket programming*

In this assignment, you require to implement two C programs, namely server and client to communicate with each other based on TCP sockets. The aim is to implement simple Base64 encoding communication protocol.

Initially, server will be waiting for a TCP connection from the client. Then, client will connect to the server using server's TCP port already known to the client. After successful connection, the client accepts the text input from the user and encodes the input using Base64 encoding system. Once encoded message is computed the client sends the Message (Type 1 message) to the server via TCP port. After receiving the Message, server should print the received and original message by decoding the received message, and sends an ACK (Type 2 message) to the client. The client and server should remain in a loop to communicate any number of messages. Once the client wants to close the communication, it should send a Message (Type 3 Message) to the server and the TCP connection on both the server and client should be closed gracefully by releasing the socket resource.

The messages used to communicate contain the following fields:

Message_ Type	**Message**

1. Message_type : integer
2. Message : Character [MSG_LEN], where MSG_LEN is an integer constant
3. < Message> content of the message in Type 3 message can be anything.

You also require implementing a "***Concurrent Server***", i.e., a server that accepts connections from multiple clients and serves all of them *concurrently*.

You should accept the IP Address and Port number from the command line (Don't use a hard-coded port number). Prototype for command line is as follows:

Prototypes for Client and Server

Client: <executable code><Server IP Address><Server Port number>
Server: <executable code><Server Port number>

*Please make necessary and valid assumptions whenever required.

Base64 Encoding System Description:

Base64 encoding is used for sending a binary message over the net. In this scheme, groups of 24bit are broken into four 6 bit groups and each group is encoded with an ASCII character. For binary values 0 to 25 ASCII character 'A' to 'Z' are used followed by lower case letters and the digits for binary values 26 to 51 & 52 to 61 respectively. Character '+' and '/' are used for binary value 62 & 63 respectively. In case the last group contains only 8 & 16 bits, then "==" & "=" sequence are appended to the end.

Application #5: *File Transfer Protocol (FTP) using Client-Server socket programming*

In this assignment, you require to implement two C programs, namely server and client to communicate with each other based on TCP sockets. The goal is to implement a simple File Transfer Protocol (FTP). Initially, server will be waiting for a TCP connection from the client. Then, client will connect to the server using server's TCP port already known to the client. After successful connection, the client should be able to perform the following functionalities:

> **PUT:** Client should transfer the file specified by the user to the server. On receiving the file, server stores the file in its disk. If the file is already exists in the server disk, it communicates with the client to inform it. The client should ask the user whether to overwrite the file or not and based on the user choice the server should perform the needful action.
> **GET:** Client should fetch the file specified by the user from the server. On receiving the file, client stores the file in its disk. If the file is already exists in the client disk, it should ask the user whether to overwrite the file or not and based on the user choice require to perform the needful action.
> **MPUT and MGET:** MPUT and MGET are quite similar to PUT and GET respectively except they are used to fetch all the files with a particular extension (e.g. .c, .txt etc.). To perform these functions both the client and server require to maintain the list of files they have in their disk. Also implement the file overwriting case for these two commands as well.

Use appropriate message types to implement the aforesaid functionalities. For simplicity assume only .txt and .c file(s) for transfer.

You should accept the IP Address and Port number from the command line (Don't use a hard-coded port number). Prototype for command line is as follows:

Prototypes for Client and Server
Client: <executable code><Server IP Address><Server Port number>
Server: <executable code><Server Port number>

*Please make necessary and valid assumptions whenever required.