CSD2161/CSD2160/CS260/MET3302 2025 Spring *Assignment 1*

Program RTIS/METS Name ______________

DigiPen Institute of Technology

Due Date:	26th January 2025
Topics covered:	Basic Winsock Programming using TCP sockets
Deliverable:	Moodle submission: submit two C++ files `echoserver.cpp` and `echoclient.cpp` to the Moodle. Naming convention: nil, all students use the same file names.

1 Design Requirements

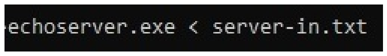
An **echo server** and an **echo client** are to be designed in this assignment. The pair of server and client are able to connect (i.e. establish connection) at first, then the server is able to echo back messages from the client.
The detailed requirements are:

1. The server program will prompt the user for a port and will then will listen on the server's IP address and the specified port, which should be displayed on screen. The server will wait for an incoming connection.
   Note: Some computers may have multiple IP addresses, especially the computers with virtual machines. Your server program should select a working IP address for this communication.

2. The client program can work in one of the following two modes to connect to the server:

   (a) Script Mode: When the client program is executed in Windows Command Prompt and a script file is provided, shown in the Figure 1, the client will prompt the user for the server IP address/port ID and message from this script file.
   Please note:

   - *an example of script file `client_script_example.txt` is provided.*
   - *grading is based on Script Mode.*



```
echoserver.exe < server-in.txt
```

Figure 1: Client Program Execution

   (b) Manual Mode: Otherwise the client program should wait for the user to type the server IP address/port ID and message.

   Noe: the Script Mode applies to the server program as well for auto-grading.

3. When the client is connected to the server and is in Manual Mode, if a user types some text and presses Enter, the text will be sent to the server according to the format shown in Figure 2. Then the server will print the received text on its screen, and send it back to the client, which will also print the text on its screen.

When the client is in Script Mode, the operation is similar except the client should send all text in the script file (excluding the server IP address and port ID) to the server.

4. The message format includes three fields and is defined as follows and illustrated in Figure 2:

   (a) **Command ID**: 1 byte, used to indicate message commands. The list of supported commands is given in Table 1:

Table 1: Message Command List

Command Name	Value	Description
QUIT	1	This message indicates client requests to quit the echo service.
ECHO	2	This message indicates the text in the Text Field should be echoed.

   When the server receives a command other than QUIT and ECHO, it should print "Error invalid command".

   (b) **Text Length** [not valid for **QUIT** message]: 4 bytes, used to indicate the total length of the sent text message in bytes;

   (c) **Text Field** [not valid for **QUIT** message]: The text length is as specified by the above text length field.
   When the server receives a **Text Length** which is different from the size of **Text Field**, the server should print "Error invalid message length". The server will not echo this type of message to the client.

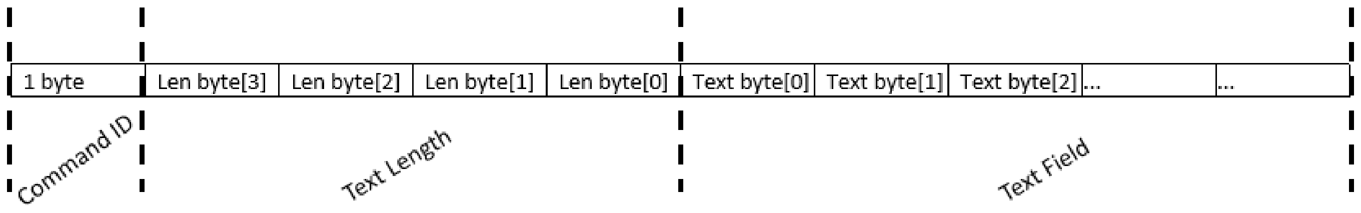


Figure 2: Message Format

In order to set and get the value for the text length field, you should use htonl to convert the host byte order to network byte order at the sender side and use ntohl to convert the network byte order to host byte order at the receiver side. Please note that a hardware platform could be little/big-endian based.

5. There are two **directives** in the input text that may be typed by the user.

   (a) /t: When the client types "/t", it sends the raw data i.e. hexadecimal values as the message to the server for the sake of testing the server behavior. For example, keying in "/t 02000000023334" means the client would ignore "/t" and interpret the subsequent input as the hexadecimal values, and the transmission content would be values of: 0x02, 0x00, 0x00, 0x00, 0x02, 0x33, 0x34, respectively (in network byte order).
   Note: the server program would NOT know anything about this "/t" directive since this directive won't appear in the message.

   (b) /q: When the client types “/q”, it indicates the client requests to quit the communication.
   The server will not echo this message to the client.

6. When the client sends ECHO command with message, the server will echo the received message back to the client, and the client will only print the text field of the message (i.e. excluding the fields of Command ID and Text Length) to the screen.
7. Please note that a (long) message might be sent over multiple TCP packets.
8. Please refer to the sample executable file for the print out format.

2 Rubrics

1. An improper submission (e.g., incorrect file names, missing files etc.) results in a penalty even after re-submission.
2. Your submitted C++ files will be compiled in release mode. Any warnings result in a penalty.
3. Comment & coding style, including:
   - Well-commented in your own words (e.g. using the given demo code's comments or reusing sentences from this PDF does not count.)
   - Indentation
   - Variable naming
   - No magic numbers
4. The server program should be able to prompt for the user to enter a port number.
5. The server program should be able to print the IP address and port number of the listening socket.
6. If in Manual Mode, the client program should be able to prompt for the IP address and port number of the server to which the client will connect.
7. Once the server accepts the connection request, it should print the IP address and port number of the connecting client.
8. Message construction and command processing: The client should be able to construct the correct message following the given format. The command should be able to be processed accordingly and correctly by the server.
9. Echo message
   (a) Whatever is typed into at the client side should first be printed on the server and then printed on the client too.
   (b) If nothing is typed into at the client side but there is something echoed or printed on the server, there will be a penalty.
   (c) If you are not able to support a very long message, there will be a penalty.

10. Correctness and Robustness:

    (a) In our test case, it is expected that send() and recv() may be called multiple times to send/receive the message completely and process accordingly, due to the limited buffer size (of either OS or application). There will be a penalty for no such support.

    (b) Command ID: If the server receives the command other than QUIT and ECHO, it should report the error.

    (c) Invalid message length: If the server receives an invalid message length, it should report the error. There will be a penalty for no such support.

    (d) The client should be able to exit gracefully. This means that if the server is shutdown unexpectedly, the client program should not crash. There will be penalty for no such support.

11. The server should be able to receive a new client connection after the previous client dis- connects.

12. Please also test your programs on two different machines.

The lecturer reserves the right to impose reasonable penalties for code that violates general practices or does not match the specification in an obvious way that has not been mentioned above. In exceptional cases, the lecturer reserves a discretionary right to allow re-submission or submission after the deadline.

3 Design Verification

To have a stable design, you need to verify your design with the following steps:

1. Run the provided echoserver executable design and your echoclient with the provided message script, and verify if the echo function works.

2. Run the provided echoclient executable design with the message script together with your echoserver, and verify if the echo function works.

3. Run your echoserver executable design and your echoclient with the provided client script, and verify if the echo function works.

4. You have to test other message scenarios, since the provided message script only covers bare minimum scenarios.