Programming Assignment 1

Member: Chenjie Luo (UIN: 324007289)
Yu-Wen Chen (UIN: 227009499)

a. **Description**:

This project is mainly divided into two parts: server and client. Chenjie Luo is responsible for the server while Yu-Wen Chen is responsible for client.

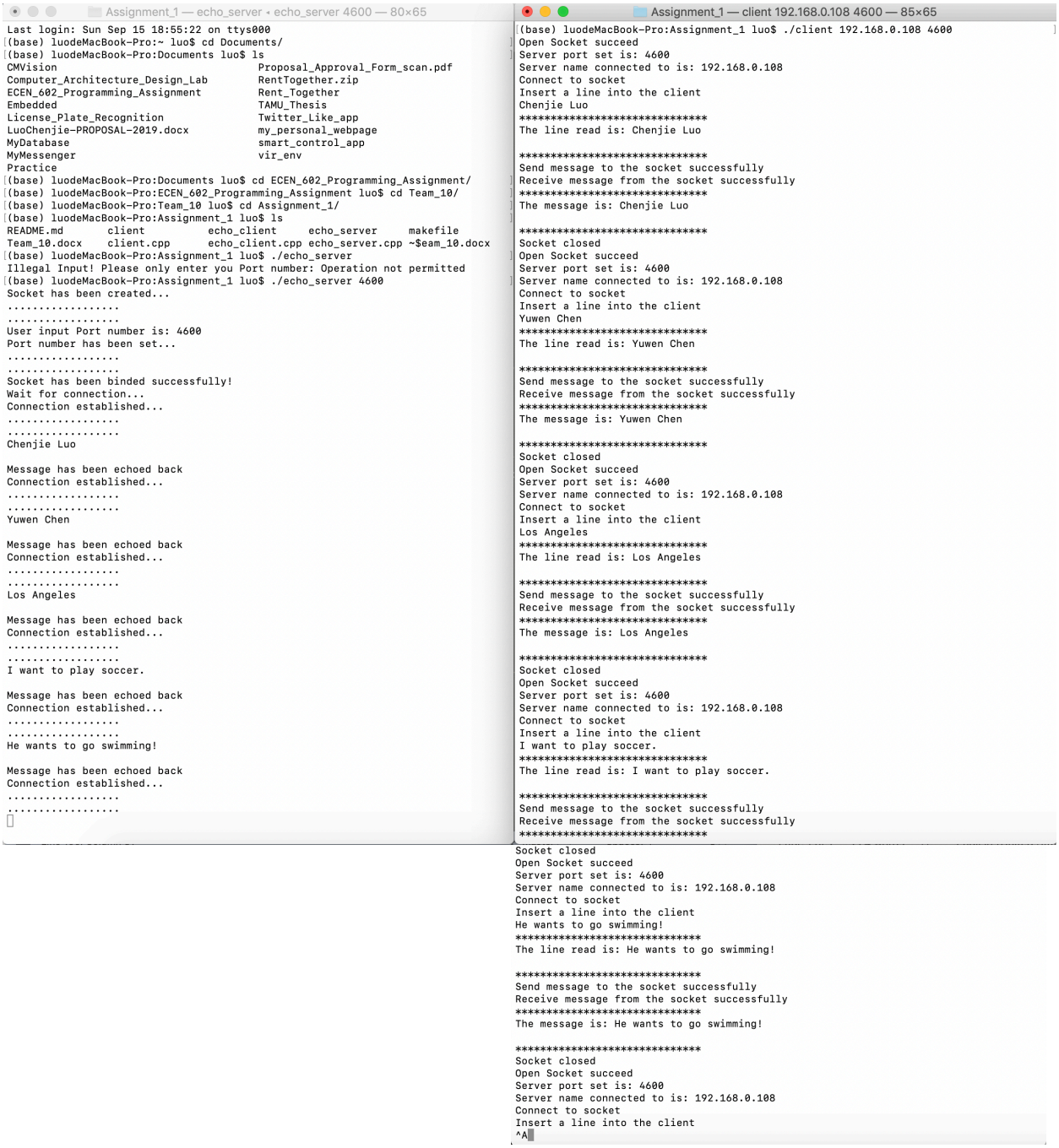
For the server side, when target file is executed, user will input port number and a socket will be created with a socket descriptor socket_fd. If failed to do so, error is generated and this is done with wrapper function socket(). After that the socket will be bineded to the input port number. If failed error code still popped up in terminal. Then in an infinite loop, socket will listen to clients connection until Ctrl + C is pressed to terminate. Accept() function will create a new socket and fork() is used to handle multiple clients. When clients send message to the server, the message will be read and saved for echo back. Here writen() function is required by the professor to write back message to the socket. What it does is basically read all the contents inside buffer instead of part of message. The reason to write this function is write() function may not be able to write chars inside the buffer to socket. If writen() successfully, it returns length of characters it send, otherwise -1 is return as a flag.

For the client side, when target file is executed, user will input the IP address of the server to be connected and the corresponding port number. After opening the client, the program first create a socket using function socket(). Then, the program initiates server_addr based the structure of sockaddr_in and assigns the necessary information (server IP address needs to be transformed to network format) to the object server_addr. Finally, the program connect itself to the server using function connect() with server_addr, and now the client is connected to the server through the created socket. Now, user can input a message to the server and see the same message sent back from the server. Here, we implement a function written() to send a message to the server and a function readline() to read the message from the server. Function written() writes one character at a time to the server so that if the client fails to send one character to the server, the function will know and resend this character again. Function readline() read one character at a time so that when the client fails to read a character, the function will know and read this character again. In the client program, all functions will print the error type based on errno when the functions return -1. Besides, I added an infinite loop to the program so that users can keep sending message to the server until they insert “CTRL + C” to terminate the client.

b. Contributions:

This project is mainly divided into two parts: server and client. Chenjie Luo is responsible for the server side while Yuwen Chen is responsible for the client side.

c. Test Cases Screenshot



```
Assignment_1 — echo_server ◂ echo_server 4600 — 80×65
Last login: Sun Sep 15 18:55:22 on ttys000
(base) luodeMacBook-Pro:~ luo$ cd Documents/
(base) luodeMacBook-Pro:Documents luo$ ls
CMVision                                Proposal_Approval_Form_scan.pdf
Computer_Architecture_Design_Lab        RentTogether.zip
ECEN_602_Programming_Assignment         Rent_Together
Embedded                                TAMU_Thesis
License_Plate_Recognition               Twitter_Like_app
LuoChenjie-PROPOSAL-2019.docx           my_personal_webpage
MyDatabase                              smart_control_app
MyMessenger                             vir_env
Practice
(base) luodeMacBook-Pro:Documents luo$ cd ECEN_602_Programming_Assignment/
(base) luodeMacBook-Pro:ECEN_602_Programming_Assignment luo$ cd Team_10/
(base) luodeMacBook-Pro:Team_10 luo$ cd Assignment_1/
(base) luodeMacBook-Pro:Assignment_1 luo$ ls
README.md       client          echo_client     echo_server     makefile
Team_10.docx    client.cpp      echo_client.cpp echo_server.cpp ~$eam_10.docx
(base) luodeMacBook-Pro:Assignment_1 luo$ ./echo_server
Illegal Input! Please only enter you Port number: Operation not permitted
(base) luodeMacBook-Pro:Assignment_1 luo$ ./echo_server 4600
Socket has been created...
..................
..................
User input Port number is: 4600
Port number has been set...
..................
..................
Socket has been binded successfully!
Wait for connection...
Connection established...
..................
..................
Chenjie Luo

Message has been echoed back
Connection established...
..................
..................
Yuwen Chen

Message has been echoed back
Connection established...
..................
..................
Los Angeles

Message has been echoed back
Connection established...
..................
..................
I want to play soccer.

Message has been echoed back
Connection established...
..................
..................
He wants to go swimming!

Message has been echoed back
Connection established...
..................
..................
```

```
Assignment_1 — client 192.168.0.108 4600 — 85×65
(base) luodeMacBook-Pro:Assignment_1 luo$ ./client 192.168.0.108 4600
Open Socket succeed
Server port set is: 4600
Server name connected to is: 192.168.0.108
Connect to socket
Insert a line into the client
Chenjie Luo
******************************
The line read is: Chenjie Luo

******************************
Send message to the socket successfully
Receive message from the socket successfully
******************************
The message is: Chenjie Luo

******************************
Socket closed
Open Socket succeed
Server port set is: 4600
Server name connected to is: 192.168.0.108
Connect to socket
Insert a line into the client
Yuwen Chen
******************************
The line read is: Yuwen Chen

******************************
Send message to the socket successfully
Receive message from the socket successfully
******************************
The message is: Yuwen Chen

******************************
Socket closed
Open Socket succeed
Server port set is: 4600
Server name connected to is: 192.168.0.108
Connect to socket
Insert a line into the client
Los Angeles
******************************
The line read is: Los Angeles

******************************
Send message to the socket successfully
Receive message from the socket successfully
******************************
The message is: Los Angeles

******************************
Socket closed
Open Socket succeed
Server port set is: 4600
Server name connected to is: 192.168.0.108
Connect to socket
Insert a line into the client
I want to play soccer.
******************************
The line read is: I want to play soccer.

******************************
Send message to the socket successfully
Receive message from the socket successfully
******************************
Socket closed
Open Socket succeed
Server port set is: 4600
Server name connected to is: 192.168.0.108
Connect to socket
Insert a line into the client
He wants to go swimming!
******************************
The line read is: He wants to go swimming!

******************************
Send message to the socket successfully
Receive message from the socket successfully
******************************
The message is: He wants to go swimming!

******************************
Socket closed
Open Socket succeed
Server port set is: 4600
Server name connected to is: 192.168.0.108
Connect to socket
Insert a line into the client
^A
```

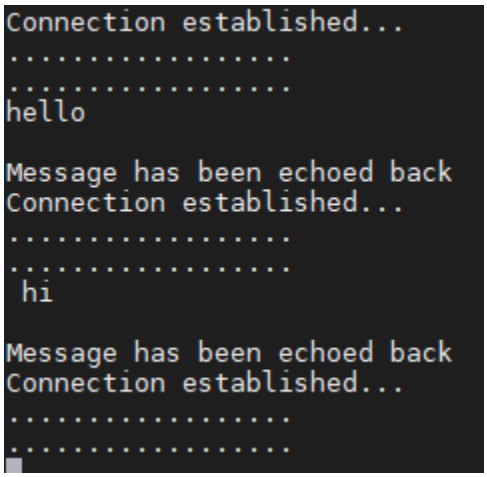
Fig.1 Test cases

1. Chenjie Luo
2. Yuwen Chen
3. Los Angeles
4. I want to play soccer.
5. He wants to go swimming!

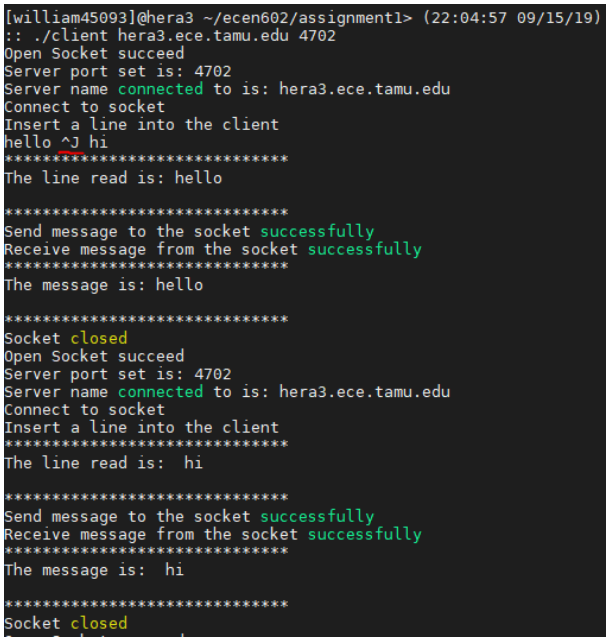
Special test case

1. Line of text terminated by a newline

   Server

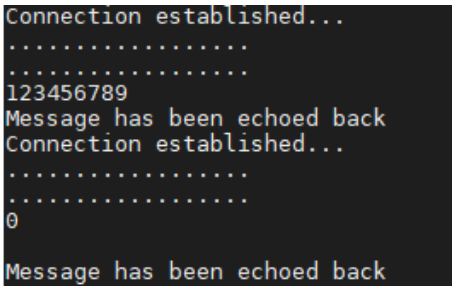


   Client

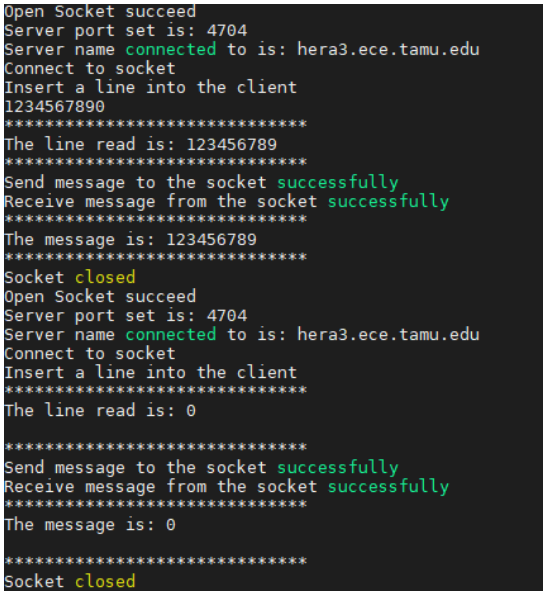


2. Line of text the maximum line length without a new line by setting buffer size to 10 (last char is EOF)

   Server

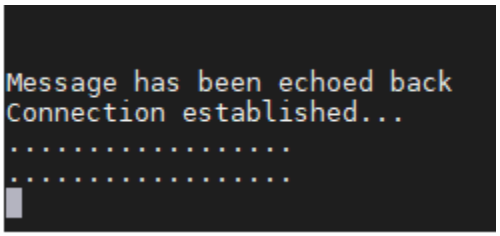


   Client

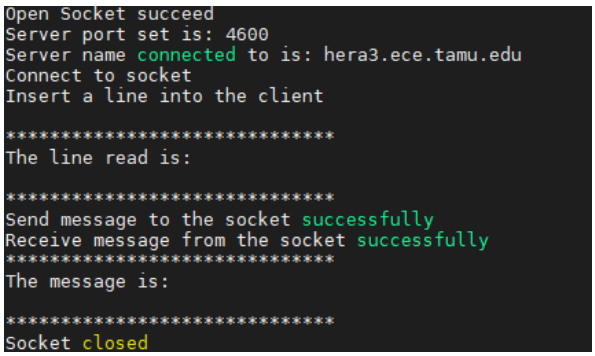


3. Line with no characters and EOF

   Server

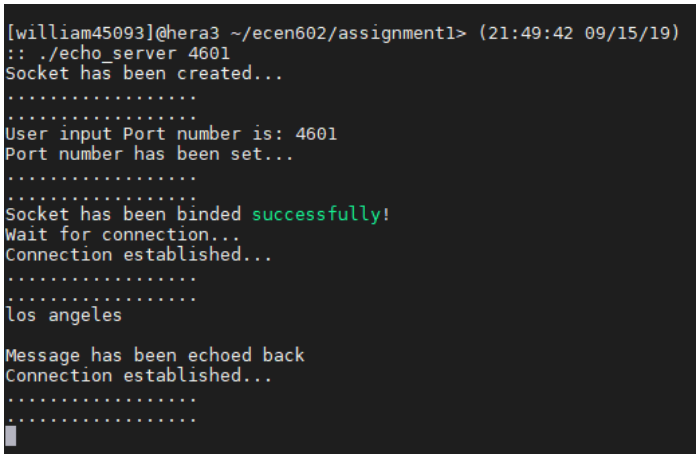


   Client

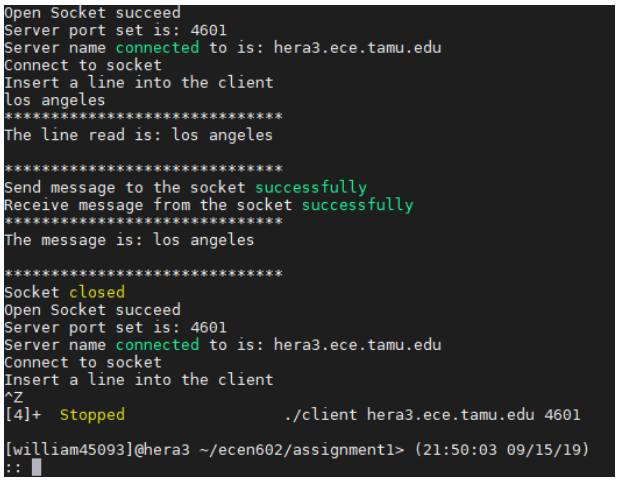


4. Client terminated after entering text

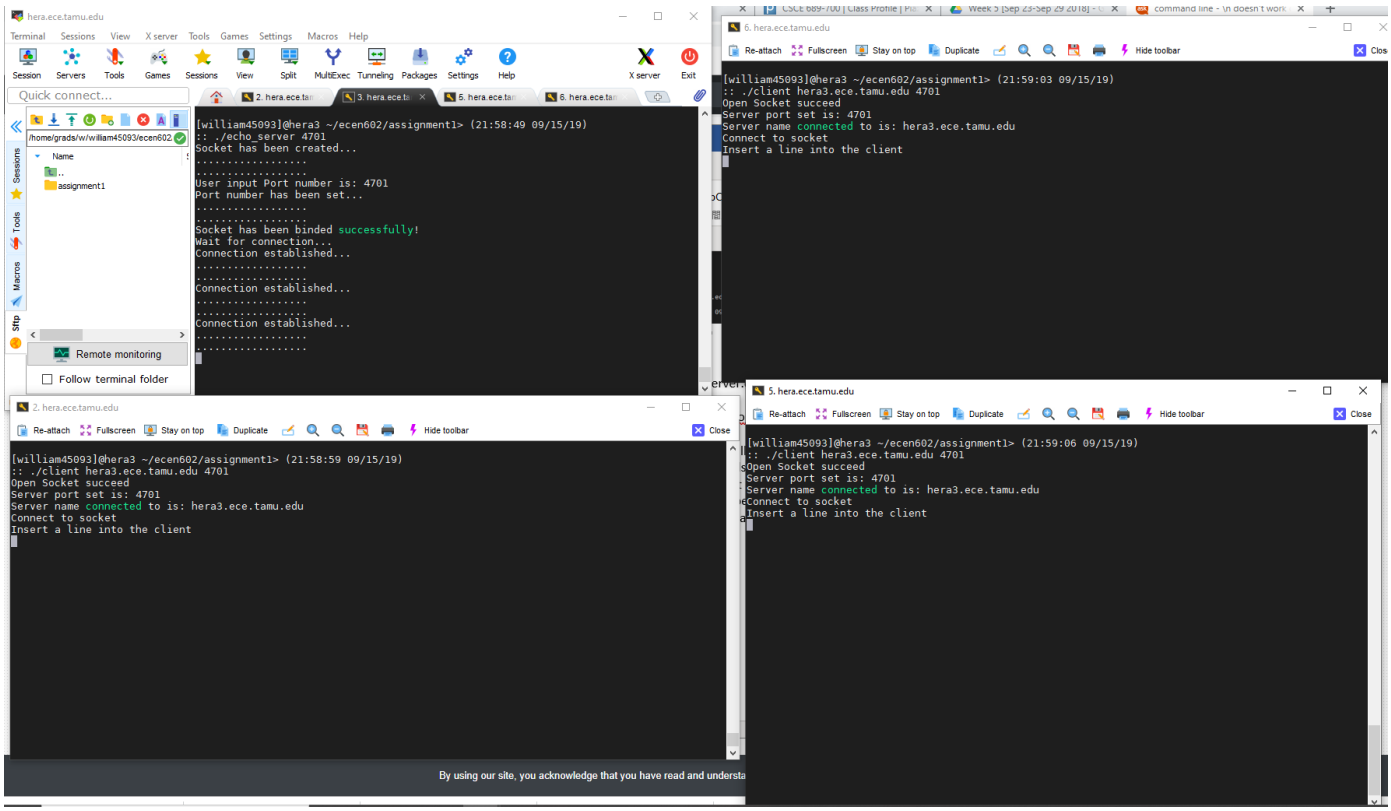
Server



Client



5. Three clients connected to the server



d. Instructions to run our code:

1. Open 2 Terminals, compile echo_server.cpp and client.cpp files with following commands:
   g++ -std=c++11 echo_server.cpp -o echo_server
   g++ -std=c++11 client.cpp -o client
   (or you can type the command “make all”)
2. Execute the target files on two terminals with following commands:
   ./echo_server [You could input any port number you want]
   ./client [server’s IP address] [port number matched above]
3. Following the commands on the terminal.

e. echo_server.cpp Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <sys/errno.h>
#include <string.h> // string in mac; sting.h in Linux
#include <iostream>
#include <sys/types.h>
#include <arpa/inet.h>
```

```
//#define PORT 4500

//using namespace std;

//PRINT OUT SUCCESS MESSAGE FOR CREATING SOCKETS
void socket_created(){
        std::cout << "Socket has been created..." << std::endl;
        std::cout << "..................." << std::endl;
   std::cout << "..................." << std::endl;
}

//USER INPUT TO GET PORT NUMBER
void get_PORT(int &PORT, std::string &INPUT){
   PORT = stoi(INPUT);
        std::cout << "User input Port number is: " << PORT << std::endl;
   PORT = stoi(INPUT);
   std::cout << "Port number has been set..." << std::endl;
   std::cout << "..................." << std::endl;
   std::cout << "..................." << std::endl;
}

//WRITEN FUNCTION WRITE len BYTES TO THE socket_fd.
//IF FAILED, IT SHOULD RETURN -1. OTHERWISE, len SHOULD BE RETURNED.
int writen(int &socket_fd, char* buffer, int len){
   int currptr = 0;
   int has_written = 0;
   while (currptr < len){
      has_written = write(socket_fd, buffer, len - currptr);
      if (has_written <= 0)
         return -1;
      buffer += has_written;
      currptr += has_written;
   }
   return currptr;
}

int main(int argc, char **argv){
   if (argc != 2){
      errno = EPERM;
      perror("Illegal Input! Please only enter you Port number");
      exit(EXIT_FAILURE);
   }
   int PORT = -1;
   std::string str = argv[1];
   int server_fd;
```

```
    int new_socket;
    int val_read;
    struct sockaddr_in address;
    int writenout = 0;
    int addrlen = sizeof(address);
    char buffer[1024] = {0};
    //::string str_to_send;
    pid_t child;

    //CREATE A SOCKET WITH SOCKET DESCRIPTOR socket_fd. IF socket_fd < 0, IT
FAILED TO CREATE A SOCKET
    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) < 0){
        errno = ETIMEDOUT;
            perror("Failed to create socket...");
            exit(EXIT_FAILURE);
    }
    socket_created();
            get_PORT(PORT, str);

    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(PORT);

    //BIND THE SOCKET TO THE IP ADDRESS AND PORT
    if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0){
        errno = EADDRINUSE;
            perror("Failed to bind...");
            exit(EXIT_FAILURE);
    }
    std::cout << "Socket has been binded successfully! " << std::endl;
    std::cout << "Wait for connection..." << std::endl;

    //SERVER WILL KEEP ACCEPTING
    while (true){
        //SET server_fd TO PASSIVE SOCKET AND COULD ACCEPT CONNECTION, SET
MAXIMUM CONNECTION AT A TIME TO 5
        if (listen(server_fd, 5) < 0)
        {
            errno = ETIMEDOUT;
            perror("Failed to listen...");
            exit(EXIT_FAILURE);
        }
        //WHEN NEW CLIENT CONNECTS, A NEW SOCKET new_socket IS CREATED FOR
COMMUNICATION
            new_socket = accept(server_fd, (struct sockaddr *)&address, (socklen_t*)&addrlen);
            if (new_socket < 0){
```

```
        errno = ETIMEDOUT;
                perror("Failed to accept new client...");
        exit(EXIT_FAILURE);
        }
        std::cout << "Connection established..." << std::endl;
        std::cout << "..................." << std::endl;
        std::cout << "..................." << std::endl;

      //WHEN NEW CLIENTS CONNECT, CREATE CHILD PROCESS TO HANDLE EACH
CLIENT
        if ((child = fork()) == 0){
                        val_read = read(new_socket, buffer, 1024);
                        std::cout << buffer << std::endl;
                        writenout = writen(new_socket, buffer, strlen(buffer) + 1);
        //IF writenout == -1 IS TRUE, IT MEANS SERVER FAILED TO WRITE BACK TO
SOCKET
        if (writenout < 0){
           errno = ETIMEDOUT;
           perror("Failed to write back to socket...");
           exit(EXIT_FAILURE);
        }
                        std::cout << "Message has been echoed back" << std::endl;
        }
   }
   return 0;
}
```

f. client.cpp Code

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/errno.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include <iostream>

using namespace std;

#define MAXLINE 1024    //***Modified to align with server

int writen(int socketfd, char* str, int num);
int readline(int socketfd, char* buffer, int max_line);
```

```
int main(int argc, char **argv){

   if (argc != 3) {
      errno = EPERM;
      printf("INPUT_ERROR: ERRNO: \t%s\n", strerror(errno));
      return -1;
   }
   char* server_name = argv[1];
   string _server_port = argv[2];
   //const char* server_name = "hera3.ece.tamu.edu"; // hera3.ece.tamu.edu for ece workstation;
192.168.0.108 for Macbook

   int server_port = -1;
   int c;
   int socketfd;
   int maxline = MAXLINE;
   char input[MAXLINE];
   char output[MAXLINE];
   string temp;
   struct sockaddr_in server_addr;

   while (1) {
      if ((socketfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
         printf("SOCKET_OPEN: ERRNO: \t%s\n", strerror(errno));
         return -1;
      }
      printf("Open Socket succeed\n");

      // Input the server port to be connected
      server_port = stoi(_server_port);
      printf("Server port set is: %d\n", server_port);


      //printf("Input the server port to be connected: ");
      //scanf("%d", &server_port);

      // Setup the server address

      memset(&server_addr, 0, sizeof(server_addr)); // Set all bits of server address to zero
      server_addr.sin_family = AF_INET;
      server_addr.sin_port = htons(server_port);
      if (inet_pton(AF_INET, server_name, &server_addr.sin_addr) < 0) {
         printf("ADDR_TRANS: ERRNO: \t%s\n", strerror(errno));
         return -1;
      }
```

```
    printf("Server name connected to is: %s\n", server_name);
    // Connect the socket to the server
    if ((connect(socketfd, (struct sockaddr *)&server_addr, sizeof(server_addr))) < 0) {
        printf("CONNECT: RRNO: \t%s\n", strerror(errno));
        return -1;
    }
    printf("Connect to socket\n");

    // Input string into the client
    printf("Insert a line into the client\n");
    fgets(input, sizeof(input), stdin);

    //*** Modified during server client test
    /*cin.ignore();
     getline(cin, temp); // Store the input string into input array

     for (int i = 0; i < temp.length(); i++){
     input[i] = temp[i];
     }*/
    //*** end
    printf("*******************************\n");
    printf("The line read is: %s\n", input);
    printf("*******************************\n");

    // Written operation
    if (writen(socketfd, input, strlen(input)) < 0) {
        printf("WRITE: ERRNO: \t%s\n", strerror(errno));
        return -1;
    }
    printf("Send message to the socket successfully\n");

    // Read messages from the socket
    if (read(socketfd , output, 1024) < 0) {
        printf("READ: ERRNO: \t%s\n", strerror(errno));
        return -1;
    }
    printf("Receive message from the socket successfully\n");
    printf("*******************************\n");
    printf("The message is: %s\n", output);
    printf("*******************************\n");
    // Reset char array for next turn
    memset(input, 0, sizeof(input));
    memset(output, 0, sizeof(output));
    close(socketfd);
    printf("Socket closed\n");
}
```

```
    return 0;
}

int writen(int socketfd, char* str, int num) {
    int num_write = 0;
    int num_left = num;
    char* ptr = str;

    while (num_left > 0) {
        num_write = write(socketfd, ptr, 1); // Write message to socket with one char at one time
        if (num_write <= 0) {
            // if EINTR(interupt system call) is detected, do the same write operation again because
the original one was blocked
            if (num_write < 0 && errno == EINTR) {
                printf("writen: ERRNO: \t%s\n", strerror(errno));
                num_write = 0;    // no char being written
            }
            // other error detected
            else {
                printf("writen: ERRNO: \t%s\n", strerror(errno));
                return -1;
            }
        }
        num_left = num_left - num_write;
        ptr = ptr + num_write;
    }
    return (num - num_left);
}

int readline(int socketfd, char* buffer, int max_line) {
    int num_read = 0;
    int num_left = max_line - 1; // last char needs to store EOF
    char* buff_ptr = buffer;

    while (num_left > 0) {
        num_read = read(socketfd, buff_ptr, 1); // read one char a time
        if (num_read < 0) {
            // EINTR is detected, read same char again
            if (errno == EINTR) {
                num_read = 0; // no char read
            }
            // other error detected
            else {
                printf("readline: ERRNO: \t%s\n", strerror(errno));
                return -1;
            }
```

```
        }
        // EOF detected, break the read loop
        else if (num_read == 0) {
            break;
        }
        // newline detected, terminate the string by assign this char to null and break the read loop
        else if (*(buff_ptr) == '\n') {
            *(buff_ptr) = '\0';
            break;
        }
        num_left = num_left - num_read;
        buff_ptr = buff_ptr + num_read;
    }
    // buffer full, EOF for the last char
    if (num_left == 0) {
        *(buff_ptr) = '\0';
        num_left = 1; // for returning the correct nnumber
    }
    return (max_line - num_left); // return number of read char
}
```

g. makefile

```
all:server client

#Type "make server" to compile echo_server.cpp
server: echo_server.cpp
	g++ -std=c++11 -o echo_server echo_server.cpp

#Type "make client" to compile client.cpp
client: client.cpp
	g++ -std=c++11 -o client client.cpp

#Type "make clean" to clean output files
clean:
	rm -rf *.o
```