Programming Assignment 1

Member: Chenjie Luo (UIN: 324007289)

Yu-Wen Chen (UIN: 227009499)

1. **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.

1. **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.

1. **Test Cases Screenshot**

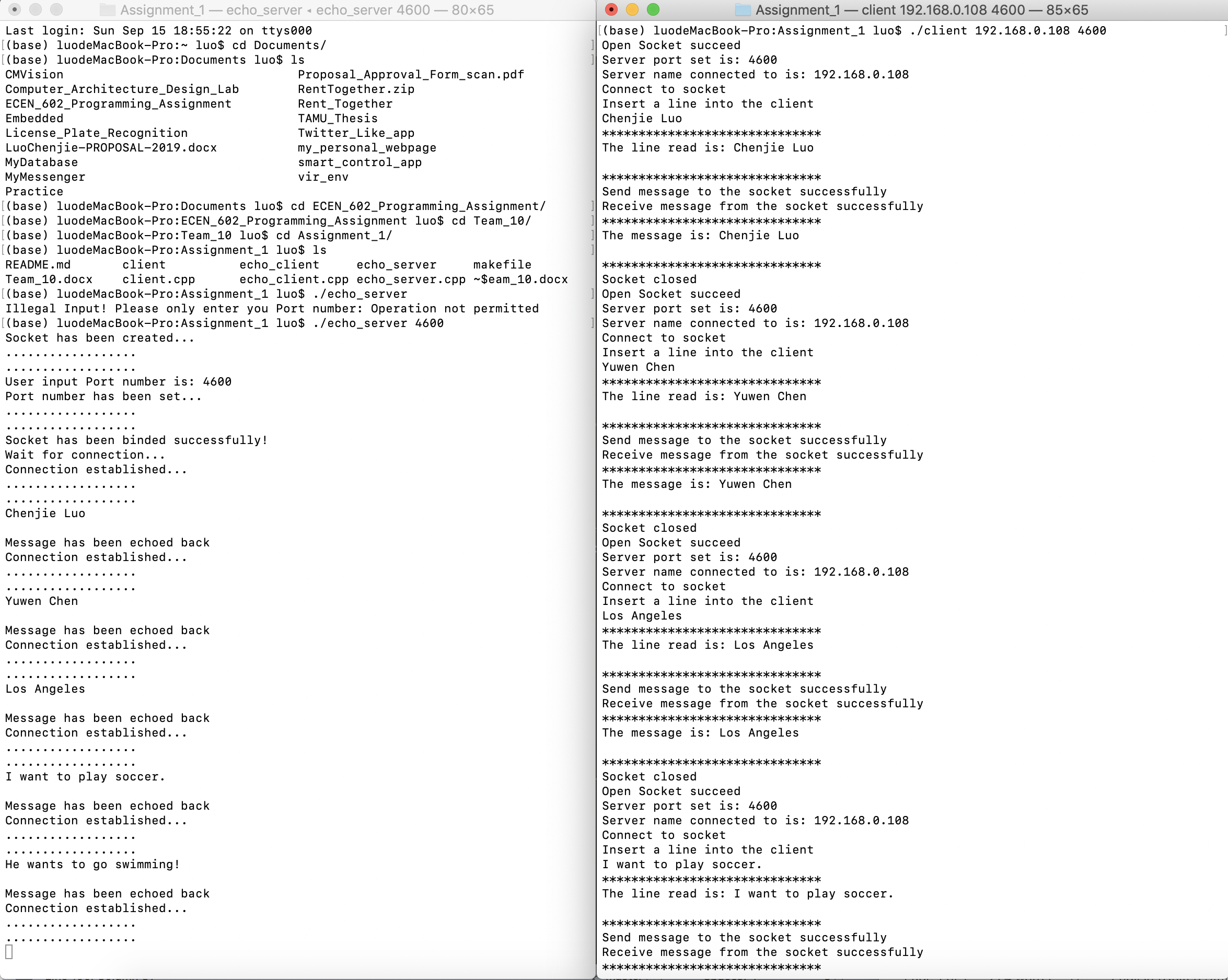
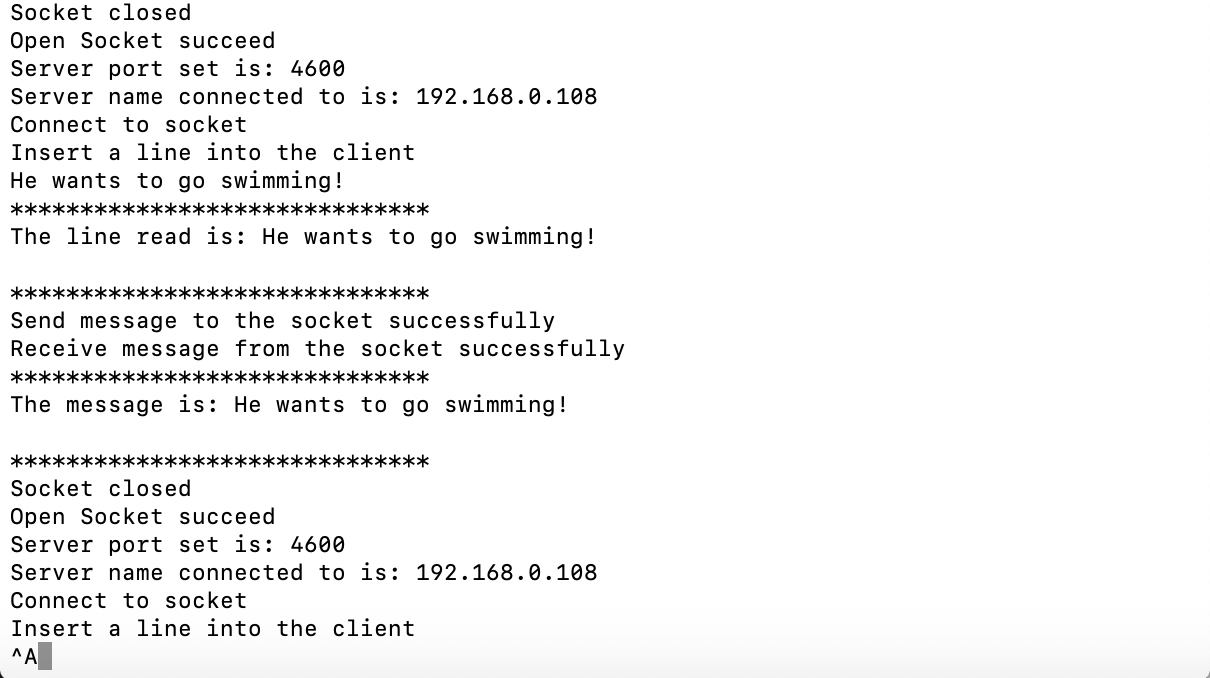
 

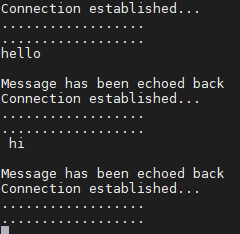
Fig.1 Test cases

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

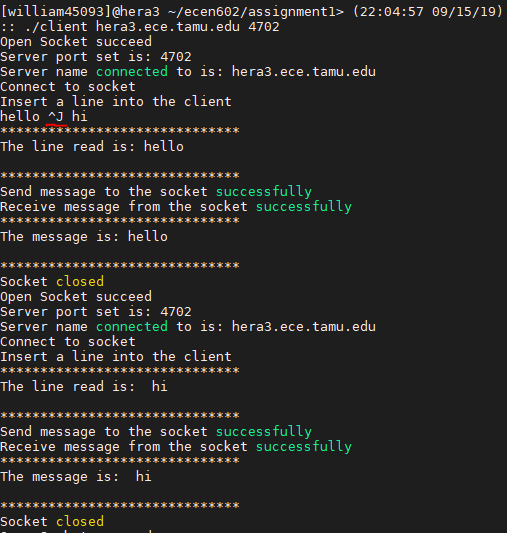
Special test case

1. Line of text terminated by a newline

Server

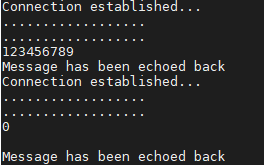


Client

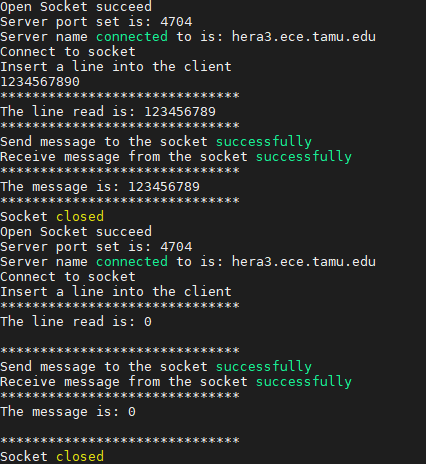


1. Line of text the maximum line length without a new line by setting buffer size to 10 (last char is EOF)

Server

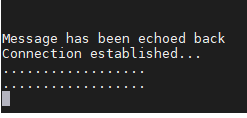


Client

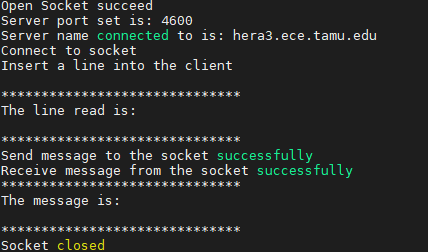


1. Line with no characters and EOF

Server

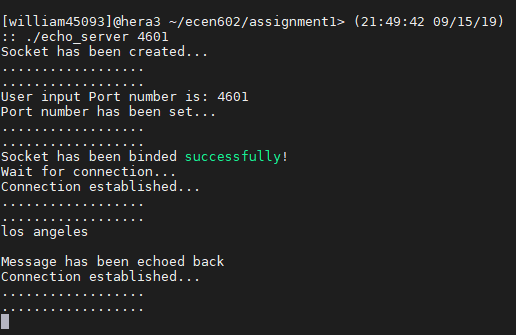


Client

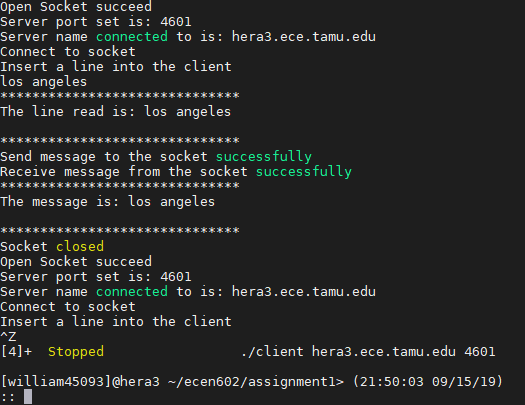


1. Client terminated after entering text

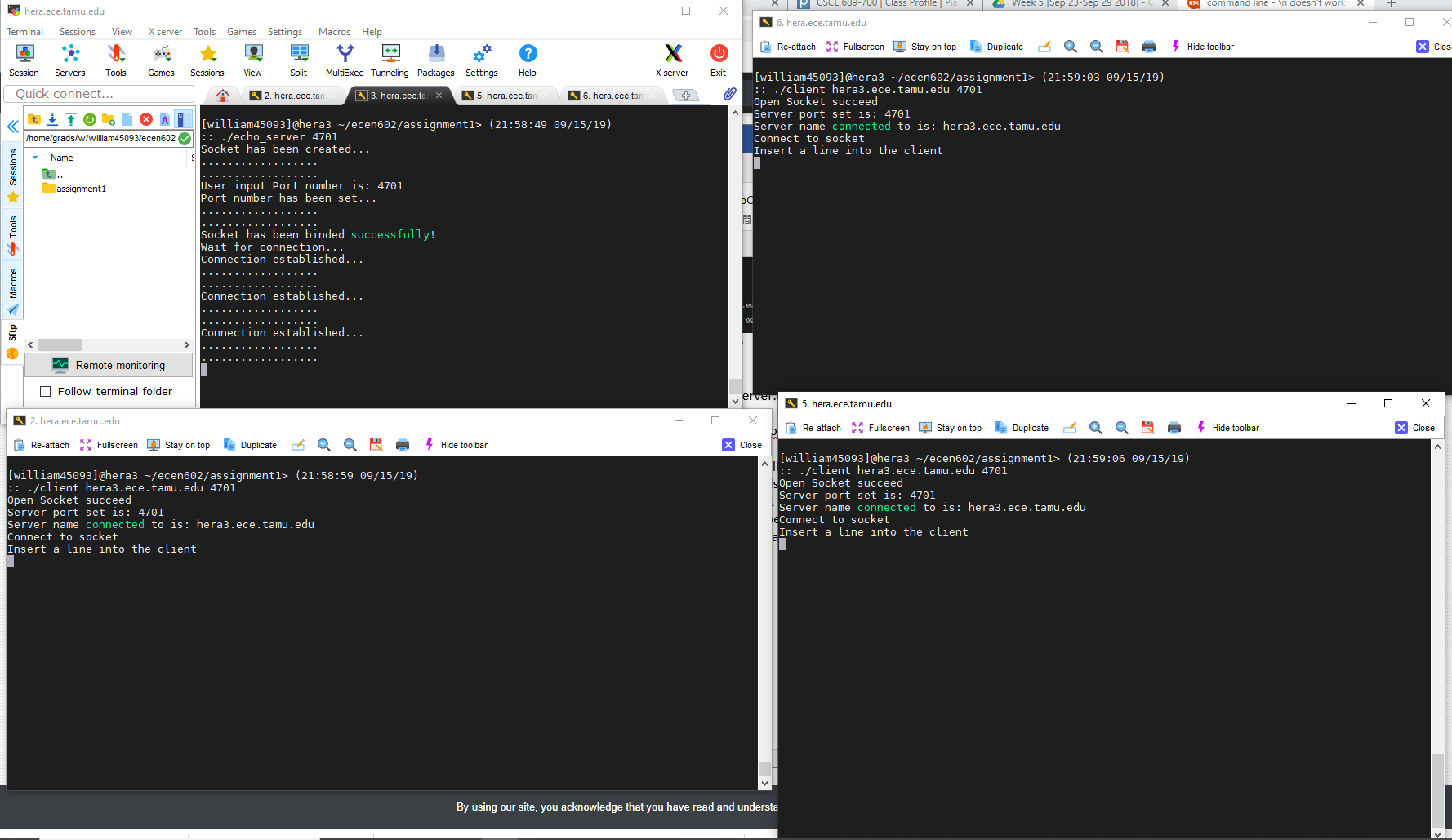
Server



Client



1. Three clients connected to the server



1. **Instructions to run our code:**
2. 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”)

1. 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]

1. Following the commands on the terminal.
2. **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;

}

1. **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

}

1. **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