ECEN 602 TCP Echo Server and Client

Sep.17th,2018

Team: 23 Feiyan Yu, Ningze Wang, Boquan Tao

## Server side code

Socket\_server\_yfy.cpp

#include <stdio.h>

#include <string>

#include <stdlib.h>

#include <sys/socket.h>

#include <sys/types.h>

#include <sys/un.h>

#include <signal.h>

#include <sys/wait.h>

#include <netdb.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <errno.h>

//STR\_SIZE is the maxsize of message, MAXLINE is to give a buffer

#define MAXLINE 100

#define STR\_SIZE 50

int readline (int sockfd, char\* buffer, int n);

int writen (int sockfd, char\* buffer, int n);

void sig\_handling(int sig);

int echo\_new(int sockfd);

char\* trim(char \*str);

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

int sockfd;

char recvline[MAXLINE+1];

struct sockaddr\_in servaddr;

int port\_num;

//Command is echo port

if (argc != 2){

printf( "Command error\n");

exit(0);

}

//create a socket

if ((sockfd = socket(AF\_INET,SOCK\_STREAM,0)) < 0){

printf("Socket error\n");

exit(0);

}

//Get port number.

for (char \*p = argv[1]; \*p != '\0'; p++) {

if (\*p >= '0' && \*p <= '9') {

port\_num = port\_num \* 10 + (\*p - '0');

}

else {

printf("Invaild port number error\n");

exit(0);

}

}

printf("Port number is:%d\n", port\_num);

//Socket initialization

bzero(&servaddr, sizeof(servaddr));

servaddr.sin\_family = AF\_INET;

servaddr.sin\_port = htons(port\_num);

//inet\_aton("10.230.169.95", &servaddr.sin\_addr);

servaddr.sin\_addr.s\_addr = INADDR\_ANY;

//printf("%d\n", servaddr.sin\_addr.s\_addr);

socklen\_t servaddr\_size = sizeof(servaddr);

//binding (sockaddr and sockaddr\_in is at same size, so just cast)

if (bind (sockfd, (struct sockaddr\*)&servaddr, servaddr\_size) < 0){

printf("Binding socket error.");

printf("Error number:%d\n",(int) errno);

exit(0);

}

else {

printf("Socket binded successfully.\n");

}

//listen to the client

//backblog: max connected number.

if(listen(sockfd, 10) < 0){

printf("Unable to find client.\n");

printf("Error number%d\n ",(int) errno);

exit(0);

}

else {

//listen to socket and waiting for client

printf("Listening to the cilent.\n");

}

//Prepared for echo.

char received\_str[MAXLINE];

//Process the zombie using sigaction

struct sigaction act;

act.sa\_handler = sig\_handling;

sigemptyset(&act.sa\_mask);

act.sa\_flags = 0;

sigaction(SIGCHLD,&act,0);

while(true){

struct sockaddr\_in clientaddr;

socklen\_t clientaddr\_size = sizeof(clientaddr);

int client\_socket = accept(sockfd, (struct sockaddr\*)&clientaddr, &clientaddr\_size);

if(client\_socket < 0){

if(errno == EINTR){

//EINTR, try it again.

continue;

}

else {

printf("Unable to accept client\n");

printf("Error number: %d\n", (int) errno);

exit(0);

}

}

//create a child process

int pid = fork();

if (pid < 0){

perror("Error of fork a child\n");

exit(1);

}

else if (pid == 0){

//child process

//echo(client\_socket);

close(sockfd);

echo\_new(client\_socket);

close(client\_socket);

exit(0);

}

//main process do nothing but loop.

else {

//sleep(0.2);

}

}

return 0;

}

int readline (int sockfd, char\* buffer, int n){

char str[MAXLINE + 1];

int len;

int letter;

char \*b;

char character;

if (n <= 0 || buffer == NULL) {

printf("Invaild input.\n");

return -1;

}

//Initialization

b = buffer;

len = 0;

for(int i = 0; i < MAXLINE; i++){

//read one letter

letter = (int)read(sockfd, &character, 1);

//printf("%d\n",letter);

if(letter == -1) {

if(errno == EINTR){

//EINTR, try it again.

continue;

}

else {

//Some errors occur

return -1;

}

}

else if(letter == 0) {

//Some letter is read, end with '\0', read is over

break;

}

else {

// only read <= n-1 letters

if (len < n - 1) {

len++;

\*b = character;

b++;

}

// The line is over.

if (character == '\n') {

break;

}

}

}

//Mark the end of the string(line)

\*b = '\0';

return len;

}

int writen (int sockfd, char\* buffer, int n){

int len\_written;

char\* b = buffer;

/\*

write(fp, p1+len, (strlen(p1)-len)

\*/

for (int i = n; i > 0; ){

len\_written = write(sockfd, b, i);

if (len\_written <= 0){

if (len\_written < 0 && errno == EINTR){

//Try it again.

continue;

}

else {

//Some errors occur.

return -1;

}

}

else {

i -= len\_written;

//point to the next location for write

b += len\_written;

}

}

return n;

}

//Handler for process the SIGCHLD when child is terminated

void sig\_handling(int sig) {

int status;

pid\_t pid;

if(sig == SIGCHLD)

{

//waitpid to kill zombie

pid = waitpid(-1,&status,WNOHANG);

if(WIFEXITED(status))

{

printf("process %d exited,return value=%d\n",pid,WEXITSTATUS(status));

}

}

}

int echo\_new(int sockfd) {

int len;

char str[MAXLINE + 1];

memset(str, 0, STR\_SIZE);

//Receive message from client.

while(1){

//len = read(sockfd, str, STR\_SIZE);

len = readline(sockfd, str, STR\_SIZE);

//printf("len: %d\n",len);

//str[STR\_SIZE]='\0';

if(len < 0){

printf("Unable to receive message from client.\n");

break;

}

else if(len == 0) {

printf("Client type EOF, service terminate.");

break;

}

else{

char \*temp = (char \*)malloc((MAXLINE + 1) \* sizeof(char));

strcpy(temp, str);

temp = trim(temp);

//printf("temp: %s\n", temp);

//printf("is quit: %d\n", strcmp(temp, "quit"));

if (strcmp(temp, "\0") != 0){

//printf("len: %d\n",len);

printf("Message from client: %s\n", temp);

writen(sockfd, str, len);

}

if (strcmp(temp, "quit") == 0 ) {

break;

}

}

}

return 0;

}

//Trim a string. remove the head and tail of '\0' '\n' '\t' '\r'

char\* trim(char \*str) {

int first = -1;

int last = -1;

for (int i = 0; str[i] != '\0'; i++) {

if (str[i] != ' ' && str[i] != '\t' && str[i] != '\n' && str[i] != '\r'){

first = i;

break;

}

}

if (first == -1){

str[0] = '\0';

return str;

}

for (int i = first; str[i] != '\0'; i++){

if (str[i] != ' ' && str[i] != '\t' && str[i] != '\n' && str[i] != '\r') {

last = i;

}

}

str[last + 1] = '\0';

return str + first;

}

## Client side code

client\_wnz.cpp

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <errno.h>

#include <iostream>

#include <stdio.h>

#include <string.h>

#include <unistd.h>

#include <stdio\_ext.h>

#include <stdlib.h>

using namespace std;

#define MAXLINE 50

//\*\*\*\*\*\*\*\*readline function\*\*\*\*\*\*\*//

int readline(int fd, char\* buf, int len)

{

int bytes, bytes\_get;

char x;

char \*y;

if (buf == NULL || len <= 0)

{

errno = EINVAL;

return -1;

}

bytes\_get = 0;

y = buf;

while (1)

{

bytes = read(fd, &x, 1); // returns one character per function call

if (bytes == -1)

{

if (errno == EINTR)

continue;

else

return -1;

}

else if (bytes == 0)

{

if (bytes\_get == 0) //read no character

return 0;

else break; //a line terminated with a newline

}

if (bytes\_get<len - 1)

{

bytes\_get++;

\*y = x;

y++;

}

if (x == EOF) return -1; //turn to close the socket

if (x == '\n')

break;

}

\*y = '\0';

return bytes\_get;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*writen function\*\*\*\*\*\*\*\*\*\*//

int writen(int fd, char\* buf, int len)

{

char\* sen = buf;

int bytes\_left = len; //total characters need to be writen

int bytes\_written;

while (bytes\_left>0)

{

bytes\_written = (int)write(fd, sen, bytes\_left);

if (bytes\_written <= 0)

{

if (bytes\_written<=0 && errno == EINTR)

bytes\_written = 0;

else return -1;

}

sen += bytes\_written;

bytes\_left -= bytes\_written;

}

return bytes\_written;

}

void str\_cli(FILE\* stdin, int sockfd)

{

char sendline[MAXLINE], recvline[MAXLINE];

while (fgets(sendline, MAXLINE, stdin) != NULL) //get the message from keyboard to sendline

{

if (feof(stdin))

{

cout << "\n" << "EOF detected, stop sending messages and exit" << endl;

break;

}

sendline[MAXLINE - 1] = '\0';

if (strlen(sendline) == MAXLINE - 1 && sendline[MAXLINE - 1] != '\n')

{

cout << "size of input string exceed the maximum" << endl;

cout << "restart, the maximum is " << MAXLINE - 1 << endl;

\_\_fpurge(stdin);

continue;

}

writen(sockfd, sendline, strlen(sendline)); //write to the server

if (readline(sockfd, recvline, MAXLINE) == 0)

perror("str\_cli:server shut off");

fputs(recvline, stdout);//echo it on screen

}

}

int main(int argc, char \*argv[])

{

if (argc != 3)

{

printf("command error");

exit(0);

}

int c0,result,port\_num;

struct sockaddr\_in servaddr;

//\*\*\*\*get port number\*\*\*\*\*\*\*//

for (char \*p = argv[2]; \*p != '\0'; p++) {

if (\*p >= '0' && \*p <= '9') {

port\_num = port\_num \* 10 + (\*p - '0');

}

else {

printf("Invaild port number error\n");

exit(0);

}

}

printf("Port number is:%d\n", port\_num);

c0 = socket(AF\_INET, SOCK\_STREAM, 0);

bzero(&servaddr, sizeof(servaddr));

servaddr.sin\_family = AF\_INET;

servaddr.sin\_port = htons(port\_num);// server listen port

result=inet\_pton(AF\_INET, argv[1], &servaddr.sin\_addr);// IP address of server

if (result<0)

{

perror("first parameter error:system doesn't support this protocol type");

close(c0);

exit(1);

}

else if (result == 0)

{

perror("second parameter is not valid format");

exit(1);

}

if (connect(c0, (const struct sockaddr\*)&servaddr, sizeof(struct sockaddr\_in)) == -1)

{

perror("fail to connect");

close(c0);

exit(1);

}

cout << "Connected to server\n";

cout << "Input a sentence(Ctrl+D means to stop connection)" << endl;

cout << "The maximum length is 50" << endl;

str\_cli(stdin, c0);//echo

exit(0);

}

client\_yfy.cpp

#include <stdio.h>

#include <string>

#include <sys/un.h>

#include <sstream>

#include <stdlib.h>

#include <stdio\_ext.h>

#include <sys/socket.h>

#include <netdb.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <errno.h>

//STR\_SIZE is the maxsize of message, MAXLINE is to give a buffer

#define MAXLINE 100

#define STR\_SIZE 50

int readline (int sockfd, char\* buffer, int n);

int writen (int sockfd, char\* buffer, int n);

char\* trim(char \*str);

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

int sockfd;

char str[MAXLINE+1];

struct sockaddr\_in servaddr;

int port\_num = 0;

unsigned int ip;

//command is echo server\_address port

if(argc != 3) {

printf("command error\n");

exit(0);

}

//create a socket

if ((sockfd = socket(AF\_INET,SOCK\_STREAM,0)) < 0){

printf("Socket error\n");

exit(0);

}

//Get a unsigned int32 IPv4 address(aton is ok).

ip = inet\_aton(argv[1], &servaddr.sin\_addr);

if (ip == 0){

printf("IPv4 format invalid!\n");

exit(0);

}

//Get port number.

for (char \*p = argv[2]; \*p != '\0'; p++) {

if (\*p > '0' || \*p < '9') {

port\_num = port\_num \* 10 + (\*p - '0');

}

else {

printf("Invaild port number error\n");

exit(0);

}

}

printf("Connecting to %s:%d.", inet\_ntoa(servaddr.sin\_addr), port\_num);

//Socket initialization

bzero(&servaddr, sizeof(servaddr));

servaddr.sin\_family = AF\_INET;

servaddr.sin\_port = htons(port\_num);

socklen\_t server\_addr\_size = sizeof(servaddr);

//Connect to server

int connection\_status = connect(sockfd,(struct sockaddr\*)&servaddr,server\_addr\_size);

if(connection\_status < 0){

printf("Error when connecting to server.\n");

printf("Error number: %d\n", (int) errno);

exit(0);

}

printf("Server connected.\n");

char buffer[MAXLINE+1];

char buffer\_recv[MAXLINE+1];

while(1) {

//Send message to server

printf("Send Message: ");

if (fgets(buffer, sizeof(buffer), stdin) == NULL && feof(stdin)) {

printf("EOF detected, the client will be terminated.");

break;

}

//buffer[strlen(buffer)] = '\0';

/\*

if(feof(stdin)){

printf("EOF detected, the client will be terminated.");

break;

}\*/

if(strlen(buffer) >= STR\_SIZE){

printf("Exceed limit, try again.(50)\n");

continue;

}

int write\_to\_server = writen(sockfd, buffer, strlen(buffer));

if(write\_to\_server < 0){

printf("Error occur when writing.");

break;

}

//trim temp and compared to "quit"

char \*temp = (char \*)malloc((MAXLINE + 1) \* sizeof(char));

strcpy(temp, buffer);

temp = trim(temp);

if(!strcmp(temp,"quit")){

printf("Quit detected, the client will be terminated.");

break;

}

readline(sockfd, buffer\_recv, STR\_SIZE);

printf("Echoed message from server: %s\n", buffer\_recv);

}

close(sockfd);

printf("You shut off the connection.\n");

}

int readline (int sockfd, char\* buffer, int n){

char str[MAXLINE + 1];

int len;

int letter;

char \*b;

char character;

if (n <= 0 || buffer == NULL) {

printf("Invaild input.\n");

return -1;

}

//Initialization

b = buffer;

len = 0;

for(int i = 0; i < MAXLINE; i++){

//read one letter

letter = (int)read(sockfd, &character, 1);

//printf("%d",letter);

if(letter == -1) {

if(errno == EINTR){

//EINTR, try it again.

continue;

}

else {

//Some errors occur

return -1;

}

}

else if(letter == 0) {

//Some letter is read, end with '\0'

break;

}

else {

// only read <= n-1 letters

if (len < n - 1) {

len++;

\*b = character;

b++;

}

// The line is over.

if (character == '\n') {

break;

}

}

}

//Mark the end of the string(line)

\*b = '\0';

return len;

}

int writen (int sockfd, char\* buffer, int n){

int len\_written;

char\* b = buffer;

/\*

write(fp, p1+len, (strlen(p1)-len)

\*/

for (int i = n; i > 0; ){

len\_written = write(sockfd, b, i);

if (len\_written <= 0){

if (len\_written < 0 && errno == EINTR){

//Try it again.

continue;

}

else {

//Some errors occur.

return -1;

}

}

else {

i -= len\_written;

//point to the next location for write

b += len\_written;

}

}

return n;

}

char\* trim(char \*str) {

int first = -1;

int last = -1;

for (int i = 0; str[i] != '\0'; i++) {

if (str[i] != ' ' && str[i] != '\t' && str[i] != '\n' && str[i] != '\r'){

first = i;

break;

}

}

if (first == -1){

str[0] = '\0';

return str;

}

for (int i = first; str[i] != '\0'; i++){

if (str[i] != ' ' && str[i] != '\t' && str[i] != '\n' && str[i] != '\r') {

last = i;

}

}

str[last + 1] = '\0';

return str + first;

}

client\_tbq.cpp

#include <stdio.h>

#include <string>

#include <sys/un.h>

#include <sstream>

#include <stdlib.h>

#include <stdio\_ext.h>

#include <sys/socket.h>

#include <netdb.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <errno.h>

#define maxlen 20

//describe errors

void err\_sys(const char\*x){

perror(x);

exit(1);

}

//write from buf to sockfd

int writen(int sockfd,char\* buf,int N){

int NumLeft=N;

int Numcount;

char\* ptr=buf;

while(NumLeft){

Numcount=write(sockfd,ptr,NumLeft);

if(Numcount<0){

if(errno==EINTR)

continue;

else

return -1; //errors occur

}

NumLeft-=Numcount; //number left to write to sockfd

ptr+=Numcount; //pointer ptr pointing to left characters

}

return N;

}

//read from sockfd to buf

int readline(int sockfd,char\* buf,int N){

int i;

int len=0;

int readf;

char\* ptr=buf;

char c;

if(N<0||buf==NULL){

printf("input N or buf error!\n"); //invalid input

}

//read each character

for(i=1;i<N;i++){

readf=read(sockfd,&c,1);

if(readf<0){

if(errno==EINTR){

i--;

continue;

}

else

return -1; //errors occur

}

else if(readf==0) //break when '\0' is found

break;

else {

if (len< N - 1) {

\*ptr=c;

len++;

ptr++;

}

if (c == '\n') {

break;

}

}

}

\*ptr='\0';

return len;

}

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

int sock\_client;

int pton;

int writeto;

int readfrom;

struct sockaddr\_in server\_addr;

unsigned short int portnum=0;

char\* port=argv[2];

char sendbuf[maxlen+1];

char recvbuf[maxlen+1];

if(argc!=3){

printf("command line error!\n");

exit(1);

}

pton=inet\_pton(AF\_INET,argv[1],&server\_addr.sin\_addr);

if(pton==0){

printf("IP address invalid!\n");

exit(1);

}

while(\*port!='\0'){

if(\*port>='0'&&\*port<='9'){

portnum=portnum\*10+(\*port-'0');

port++;

}

else{

printf("portnumber error!\n");

exit(1);

}

}

printf("connecting to %s ; %d\n", inet\_ntoa(server\_addr.sin\_addr), portnum);

sock\_client=socket(AF\_INET,SOCK\_STREAM,0);

if(sock\_client<0){

printf("socket not created!\n");

printf("error number is :%d\n",errno);

err\_sys("client:socket");

exit(1);

}

memset(&server\_addr,0,sizeof(server\_addr));

server\_addr.sin\_family=AF\_INET;

server\_addr.sin\_port=htons(portnum);

socklen\_t server\_addr\_size = sizeof(server\_addr);

if(connect(sock\_client,(struct sockaddr\*)&server\_addr,server\_addr\_size)==-1){

printf("connection failed!\n");

printf("error number is :%d\n",errno);

err\_sys("client:connect");

exit(1);

}

printf("connected to server!\n");

while(fgets(sendbuf,sizeof(sendbuf),stdin)){

printf("input a text\n");

if(feof(stdin)){

printf("terminated text transmission!\n");

break;

}

sendbuf[maxlen]='\0';

if(strlen(sendbuf)==sizeof(sendbuf)-1&&sendbuf[maxlen-1]!='\n'){

printf("Exceed limit!\n");

continue;

}

else{

writeto=writen(sock\_client,sendbuf,strlen(sendbuf));

if(writeto<0){

printf("no sending text!\n");

break;

}

readfrom=readline(sock\_client,recvbuf,maxlen);

if(readfrom<0){

printf("read error!\n");

exit(1);

}

printf("message received from server :%s\n",recvbuf);

}

}

close(sock\_client);

printf("connection is closed!\n");

}

## Makefile

all: server client1 client2 client3

server: Socket\_server\_yfy.cpp

g++ -o server Socket\_server\_yfy.cpp

client1: client\_wnz.cpp

g++ -o client1 client\_wnz.cpp

client2: client\_yfy.cpp

g++ -o client2 client\_yfy.cpp

client3: client\_tbq.cpp

g++ -o client3 client\_tbq.cpp

clean:

rm -rf \*.o

## Readme

ECEN 602

Contribution:

Feiyan Yu:

For groupwork, I coded the client\_yfy.cpp and server. I made a lot of tries to create socket server and socket client.

My experience:

This assignment provides a good way for us to have a real understanding of the socket programming. I trained my C++ programming ability and got familiar with Linux Operating. I also learned how to use GDB to debug.

My first version is to echo a single line of message by a child process. The client need to reconnect the server after it finishes sending a line. This actually works, but not a good method because the server need to create a child for a single line.

Ningze Wang

For groupwork, I coded the client\_wnz.cpp For myself, I practiced coding the server and the client.

Boquan Tao

I coded the client\_tbq.cpp for group work. Also I coded part of the server for my own testing.

Reward: I learned how to use diffenrent socket functions like bind, connect... I learned how to use fork function to creat a child process. I enhanced my understanding of IP protocal and TCP protocal.

Usage

Server:

./server any port number bigger than 1023

Client:

./client1 IPv4 address of the server the port number you put the server listen to

then input string from keyboard.

./client2 IPv4 address of the server the port number you put the server listen to

then input string from keyboard.

./client3 IPv4 address of the server the port number you put the server listen to

then input string from keyboard.

Architecture

Client:

1.create a socket and initialize it

2.get the Ipv4 address and port number

3.connect server

4.input string from keyboard to the socket

5.ready to receive the character from the socket

6.show the echo on screen

Server:

1.create a socket and initialize it

2.get the port number

3.bind, listen and ready to accept and fork.

4.readline and writen contribute the echo.

5.check for EOF and when that happens kill the childprocess

Details:

Client:

1. int readline (int sockfd, char\* buffer, int n):

Read a line from 'sockfd', and put the string in buffer. Using 'read' function to get 1 character a time until '\n', and add a '\0' as mark of termination for a string.

Return the length of buffer.

2. int writen (int sockfd, char\* buffer, int n):

Write n character to buffer from 'sockfd', terminated when have read n character or read a '\0'.

Return n if writen successfully(actually no use).

3.char\* trim(char \*str):

Remove the blank part of head a tail of a string.

It is used to process the situation when user type something like " quit " to close the client. This function could return "quit".

Server:

1. int readline (int sockfd, char\* buffer, int n):

Same as client.

2. int writen (int sockfd, char\* buffer, int n):

Same as client.

3. void sig\_handling(int sig):

To process the zombie process, using 'waitpid' function. This function is set to a sigaction struct.

4. int echo\_new(int sockfd);

The name is 'echo\_new' because this is the second version of 'echo' function.

Using 'readline' to read the message from client, save in buffer. And if there is no 'quit' in buffer, use 'writen' function to send message in buffer back to client.

Return 0 if exit successfully.

5. char\* trim(char \*str);

Same as client.

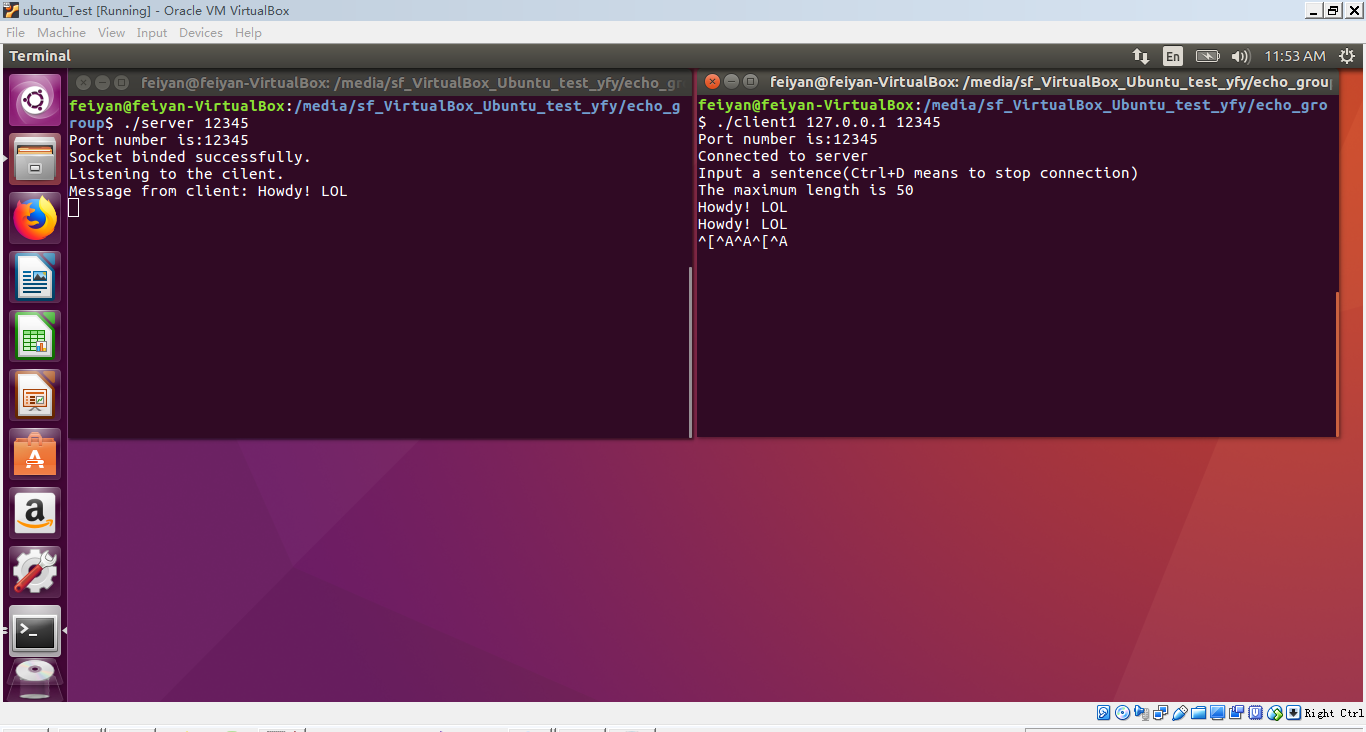
Errta:

In first version, I could not using a loop in echo. I could just reconnect when the same client send message. By debugging, I found the close(client\_socket) is the reason. So echo\_new is created.

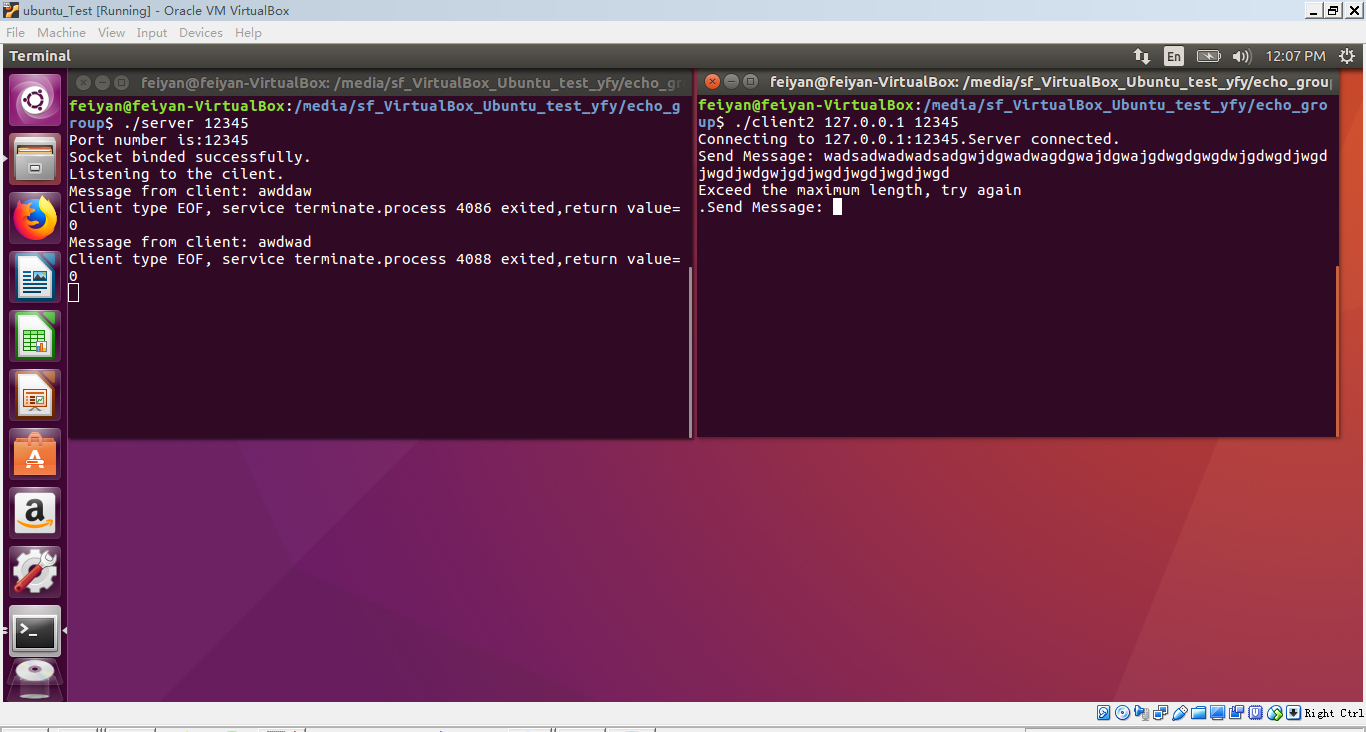
(So many bugs I met, so I did not present other details. Some of the bugs are easy or stupid problems.)

## Test Case

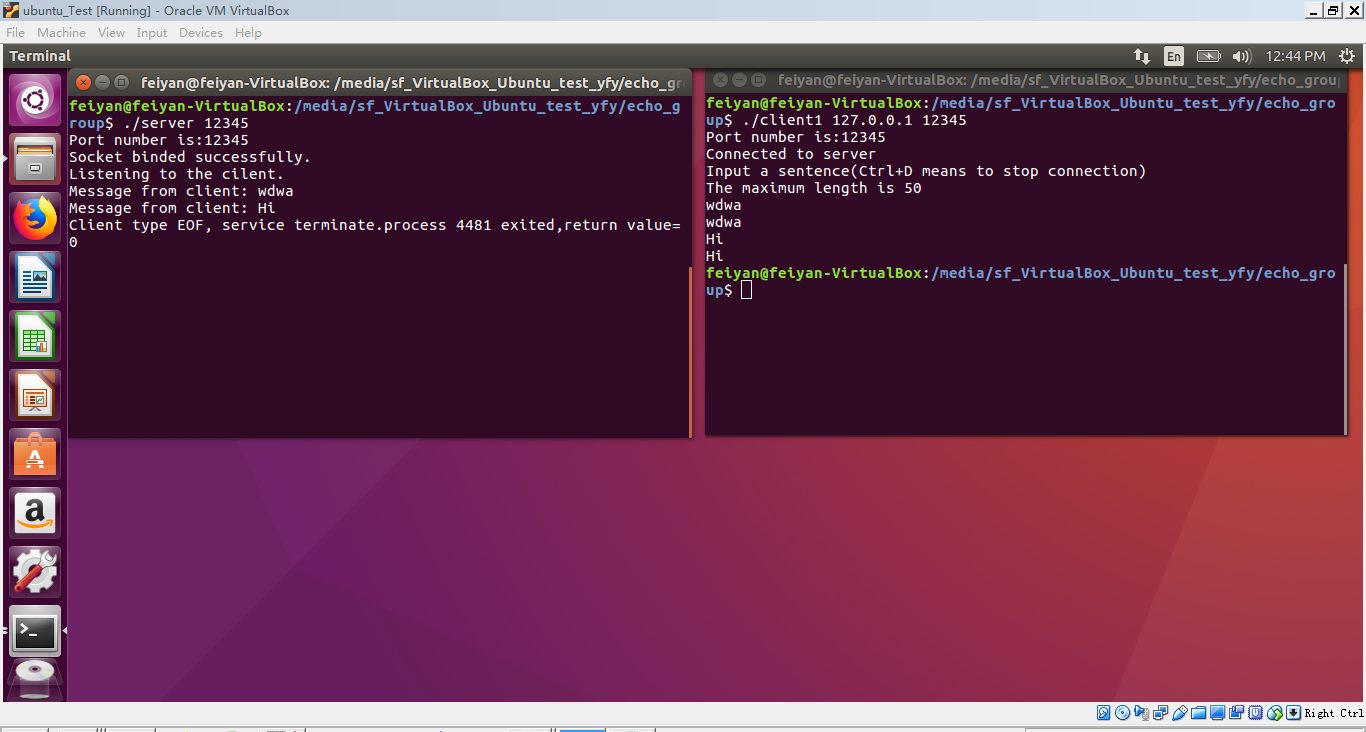
1. line of text terminated by a newline



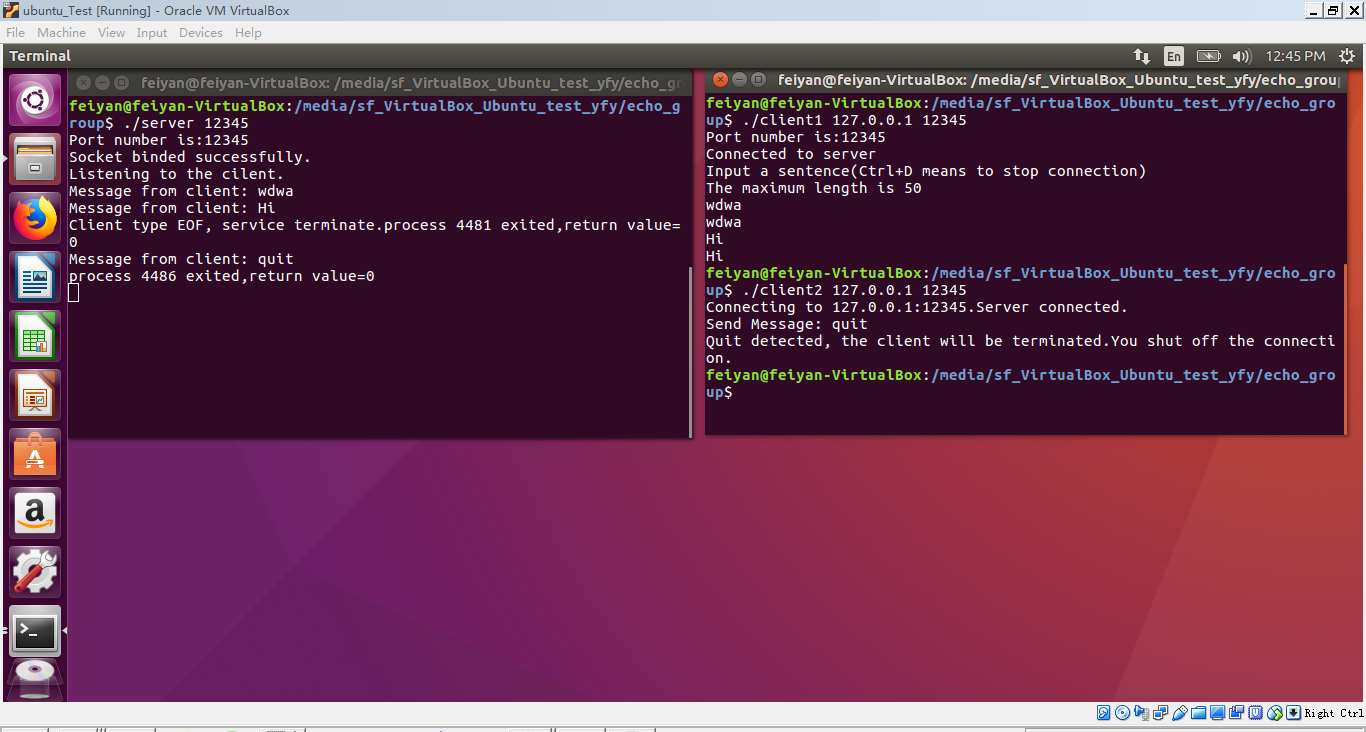
1. line of text the maximum line length without a newline



1. line with no characters and EOF



1. client terminated after entering text



1. three clients connected to theserver

