Assignment description: This is a C++ project. The comms class containing the code that is common to both the client and server. The classes should contain the functionality required to establishing a **TCP** WinSock2 connection between two applications running within two processes on a single machine, one being a client and the other a server. They should also implement the functionality that will allow messages (array of char) to be exchanged between the Client and the Server.

Using these classes develop a chat application that when run, requests the client enter a message which is then echoed on the server interface. A message can then be typed into the server interface and this is then echoed on the client interface.

This process should repeat indefinitely until the client types QUIT, at which point both the client and server programs should terminate.

The classes and application should include exception handling code, that replaces much of the original error handling statements found within the practical exercises.

Pre-processor directives should be embedded within your project to enable / disable compilation of client and server code. This should allow you to develop a single project that contains both the Client and Server code.

Solution:

Comms class:

Comms.h

```
#pragma once
#pragma comment(lib, "ws2_32.lib") // library for winsock2

#include <winsock2.h> // winsock2 library
#include <iostream> // cin/cout
#include <string> // string operations
#define SERVER 1
#define CLIENT 2
#define DefaultPort 8888
#define DefaultServerAddress "127.0.0.1" // local host
#define BUFSIZE 100 // maximum length of text message

using std::cin;
using std::cout;
using std::endl;

class Comms {
public:
	Comms(); // default constructor
	static void InitServerSocket(); // initializes socket, address, etc.
	static unsigned long __stdcall Receive(void * pParam); // processes input
messages
	void Send(); // sends messages

protected:
	SOCKET serverSocket; // server socket
	sockaddr_in addr; // address
};
```

Comms.cpp

```
#include "Comms.h"

Comms::Comms() { // default constructor
	InitServerSocket();
}

void Comms::InitServerSocket() { // empty function
}

unsigned long __stdcall Comms::Receive(void * pParam) { // empty function
	return 0;
}

void Comms::Send() { // empty function
}
```

Server class

Server.h

```
#pragma once

#include "Comms.h"

class Server: public Comms {
public:
	Server(); // default constructor
	void InitServerSocket(); // initializes socket, binds it with address, etc.
	static unsigned long __stdcall Receive(void* pParam); // processes input
message
```

```
        void Send(); // sends messages to client

private:
        SOCKET clientSocket; // stores client's socket
};
```

Server.cpp

```
#include "Server.h"

namespace serv { // to avoid name collisions
        bool IsQuitRequested = false;
}

Server::Server() { // default constructor
        InitServerSocket();
}

void Server::InitServerSocket() {
        serverSocket = socket(AF_INET, SOCK_STREAM, 0); // create socket for
internetwork: UDP, TCP, etc.; stream socket; 0 - TCP protocol
        addr.sin_family = AF_INET; // internetwork: UDP, TCP, etc.
        addr.sin_port=htons(DefaultPort); // set port
        addr.sin_addr.s_addr=inet_addr(DefaultServerAddress); // set address

        bind(serverSocket, (sockaddr *)&addr,sizeof(addr)); // binding address with
socket
        int c = listen(serverSocket,1);                 //  waiting for connection; 1 client
only excepted
        cout<<"Server receive ready"<<endl;
}

unsigned long __stdcall Server::Receive(void* pParam) {
```

```
	Server * server = (Server *)pParam; // server class, stores info about client socket
	char buf[BUFSIZE]; // buffer for text message
	memset(buf, 0, BUFSIZE); // clear buffer

	try {
		while(recv(server->clientSocket, buf, BUFSIZE - 1, 0) > 0) { // while receive messages from client
			if(strcmp(buf, "QUIT") == 0) { // if QUIT received
				cout<<"Client quit command received!"<<endl;
				serv::IsQuitRequested = true; // set flag to close all the sockets and terminate the program
				break;
			}
			else {
				cout<<"Client: "<<buf<<endl; // display client's message
				memset(buf, 0, BUFSIZE); // clear buffer
			}
		}
		throw -1; // client disconnected, throw exception
	}

	catch (int a) {
		if (a == -1) {
			cout<<"Client disconnected.\n";
			closesocket(server->clientSocket); // close sockets
			closesocket(server->serverSocket);
			exit(0); // exit server type program
		}
	}
}

void Server::Send()
```

```
{
        char buf[BUFSIZE]; // buffer for messages

   sockaddr_in client_addr;
   int client_addr_size = sizeof(client_addr);

        fd_set socksToCheck;
        TIMEVAL timeOut;
        timeOut.tv_sec = 1; // period for checking new client connections

        while(1) {
                FD_ZERO(&socksToCheck);
                FD_SET(serverSocket, &socksToCheck);
                int res = select(1, &socksToCheck, 0, 0, &timeOut); // organizes
checking process
                if(res == 1) { // user connected
                        clientSocket = accept(serverSocket,(sockaddr *)&client_addr,
&client_addr_size); // accept client connection
                        cout<<"Client connected.\n"<<endl;
                        strcpy(buf, "You connected to server\n");
                        send(clientSocket, buf, strlen(buf), 0); // inform client about
connection
                        DWORD threadId;
                        CreateThread(NULL,NULL, &Server::Receive, this, NULL,
&threadId); // receive message from client to server in another thread thread
                        break;
                }
   }

        while (!serv::IsQuitRequested) {
                cin.getline(buf, BUFSIZE);
                if (send(clientSocket, buf, strlen(buf), 0) == SOCKET_ERROR) break; //
send messages from server to client
```

```
        }

        cout<<"Server shutting down."<<endl;
        closesocket(clientSocket); // closing all the sockets
        closesocket(serverSocket);
}
```

Client class

Client.h

```
#pragma once

#include "Comms.h"

class Client: public Comms {
public:
        Client(); // default constructor
        void InitServerSocket(); // init socket
        static unsigned long __stdcall Receive(void* pParam); // process messages
from server
        void Send(); // send message to server
};
```

Client.cpp

```
#include "Client.h"

Client::Client() { // default constructor
        InitServerSocket();
}

void Client::InitServerSocket() {
```

```
	serverSocket = socket(AF_INET, SOCK_STREAM, 0); // create socket for internetwork: UDP, TCP, etc.; stream socket; 0 - TCP protocol
	addr.sin_family = AF_INET; // internetwork: UDP, TCP, etc.
	addr.sin_port=htons(DefaultPort); // set port
	addr.sin_addr.s_addr=inet_addr(DefaultServerAddress); // set address
}

unsigned long __stdcall Client::Receive(void* pParam)
{
	SOCKET * sock = (SOCKET *)pParam; // stores info about server socket
	char buf[BUFSIZE]; // buffer for text message
	memset(buf, 0, BUFSIZE); // clear buffer

	try {
		while(recv(*sock, buf, BUFSIZE - 1, 0) > 0) // while receive messages from server
		{
			cout<<"Server: "<<buf<<endl; // display message
			memset(buf, 0, BUFSIZE); // clear buffer
		}

		throw "Server disconnected"; // throw exception if cannot access to server
	}

	catch (const char * s) { // catch server disconnected exception
		printf("%s\n", s);
		closesocket(*sock); // close socket
		return 1;
	}
}

void Client::Send()
```

```
{
	try {
		if(connect(serverSocket,(sockaddr *)&addr,sizeof(addr)) != 0)
			throw "Error connecting to server."; // exception
	}

	catch (const char * s) {
		printf("%s\n", s); // report about exception
		return;
	}

	DWORD threadId;
	CreateThread(NULL,NULL,&Client::Receive, &serverSocket, NULL,
&threadId); // receive from server in another thread thread

	char buf[BUFSIZE]; // buffer for messages

	while(1)
  {
		cin.getline(buf, BUFSIZE); // read message from stdin
		if (send(serverSocket, buf, strlen(buf), 0) == SOCKET_ERROR ||
(strcmp(buf, "QUIT") == 0)) break; // send message to server
  }

  closesocket(serverSocket); // close server socket
}
```

main.cpp (server project)

```
#include "Client.h"
#include "Server.h"

#define PROJECT_TYPE SERVER // set project type
```

```
WORD wVersionRequested; // needed to init sockets lib
WSADATA wsaData;
int err;

bool InitSocketsLib();
bool ReleaseSocketsLib();

int main (int argc, char ** argv) {
	try {
	if (InitSocketsLib() == false)
		throw "Cannot initialize sockets library."; // exception
	}

	catch (const char * s) {
		cout<<s<<endl; // report about exception
		return 1;
	}

	try {
		if (PROJECT_TYPE == SERVER) {
			Server obj;
			obj.Send();
		}
		else
			if (PROJECT_TYPE == CLIENT) {
				Client obj;
				obj.Send();
			}
			else
				throw "Invalid project type."; // exception
	}
```

```
        catch (const char * s) {
                cout<<s<<endl; // report about exception
                return 1;
        }

        try {
                if (ReleaseSocketsLib() == false)
                        throw "Cannot release sockets library."; // exception
        }

        catch (const char * s) {
                cout<<s<<endl;  // report about exception
                return 1;
        }

        return 0;
}

bool InitSocketsLib() // initializes sockets library
{
        wVersionRequested = MAKEWORD(2, 2);
        err = WSAStartup(wVersionRequested, &wsaData);
        return (err == 0);
}

bool ReleaseSocketsLib() // releases socket library
{
        err = WSACleanup();
        return (err == 0);
}
```

main,cpp (client project)

```
#include "Client.h"
#include "Server.h"

#define PROJECT_TYPE CLIENT // set project type

WORD wVersionRequested; // needed to init sockets lib
WSADATA wsaData;
int err;

bool InitSocketsLib();
bool ReleaseSocketsLib();

int main (int argc, char ** argv) {
	try {
	if (InitSocketsLib() == false)
		throw "Cannot initialize sockets library."; // exception
	}

	catch (const char * s) {
		cout<<s<<endl; // report about exception
		return 1;
	}

	try {
		if (PROJECT_TYPE == SERVER) {
			Server obj;
			obj.Send();
		}
		else
			if (PROJECT_TYPE == CLIENT) {
				Client obj;
```

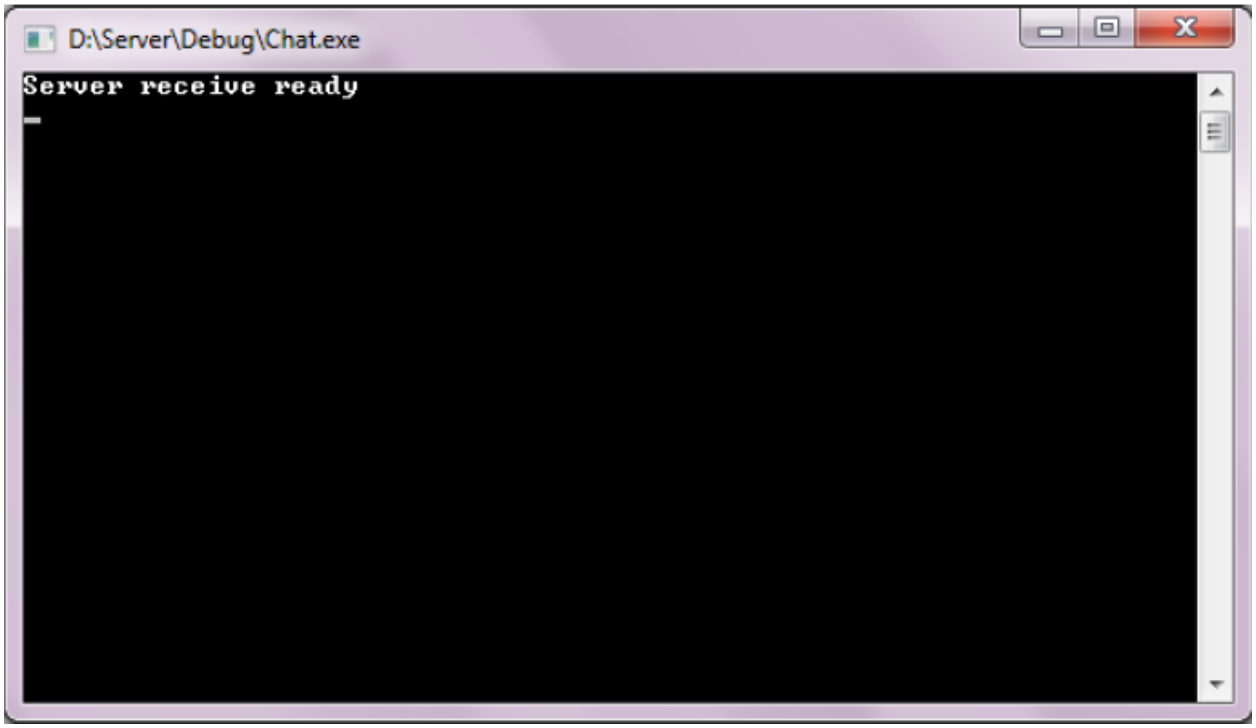
```
                        obj.Send();
                }
                else
                        throw "Invalid project type."; // exception
        }

        catch (const char * s) {
                cout<<s<<endl; // report about exception
                return 1;
        }

        try {
                if (ReleaseSocketsLib() == false)
                        throw "Cannot release sockets library."; // exception
        }

        catch (const char * s) {
                cout<<s<<endl;  // report about exception
                return 1;
        }

        return 0;
}

bool InitSocketsLib() // initializes sockets library
{
        wVersionRequested = MAKEWORD(2, 2);
        err = WSAStartup(wVersionRequested, &wsaData);
        return (err == 0);
}

bool ReleaseSocketsLib() // releases socket library
{
```

```
        err = WSACleanup();
        return (err == 0);
}
```

Below see the test screenshots:

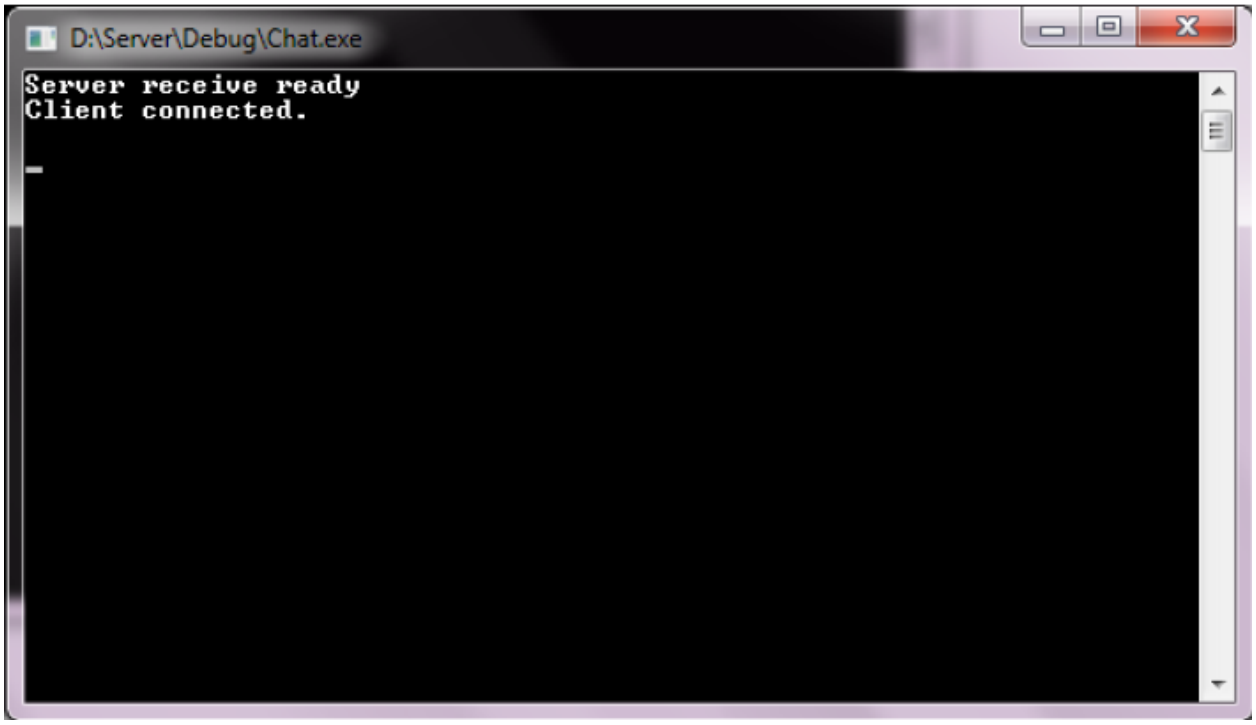
Test 1 (chatting)

1. Launching server:



2. Now server is waiting for client. Launching client:

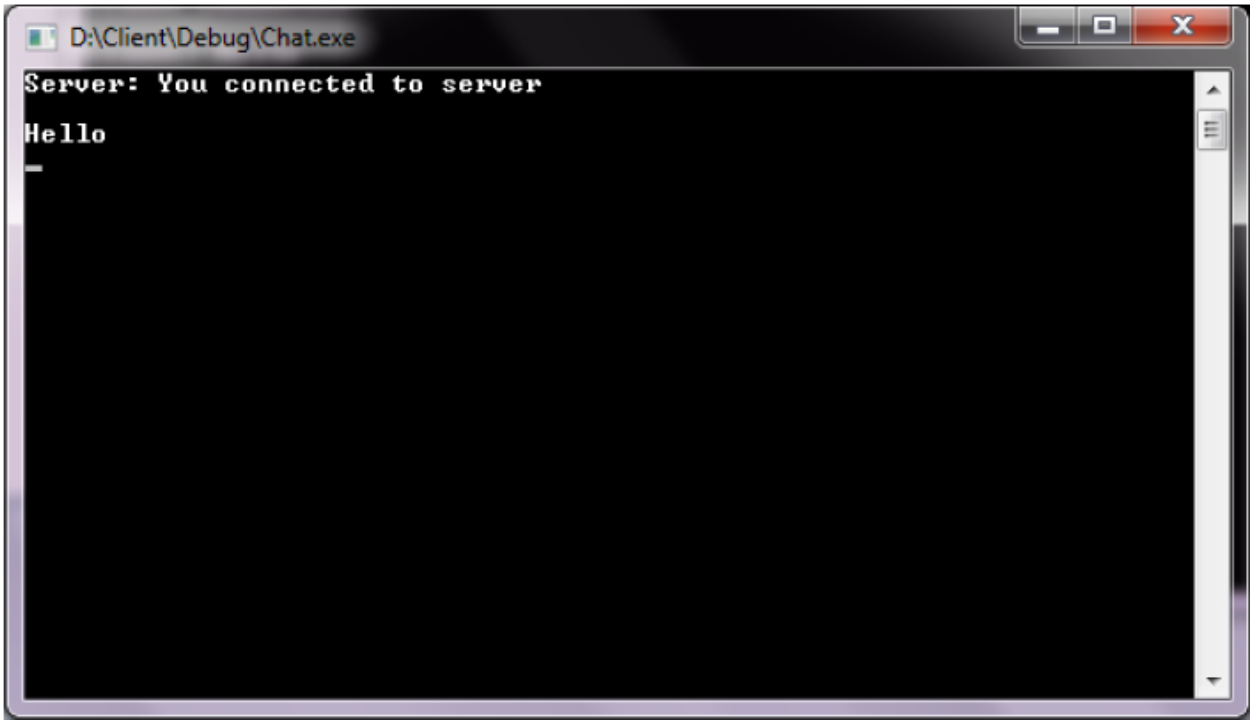
Server output:



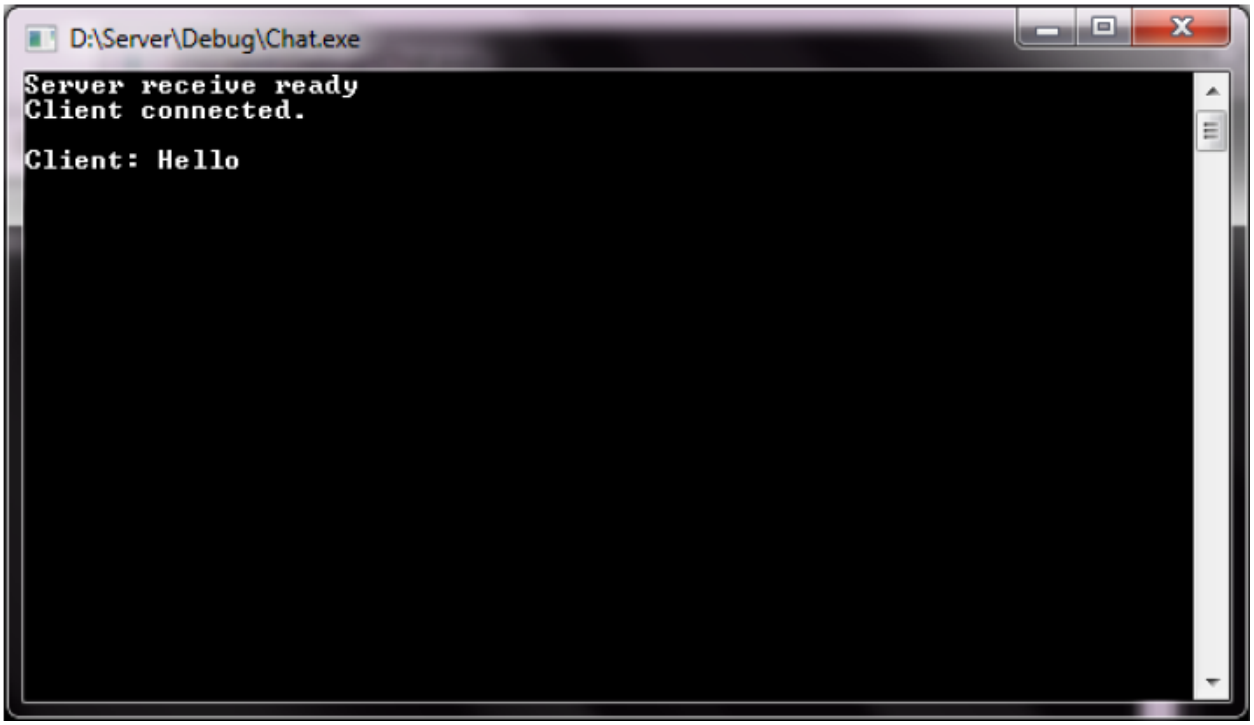
Client output:

3. Client writes “Hello”:

Client output:



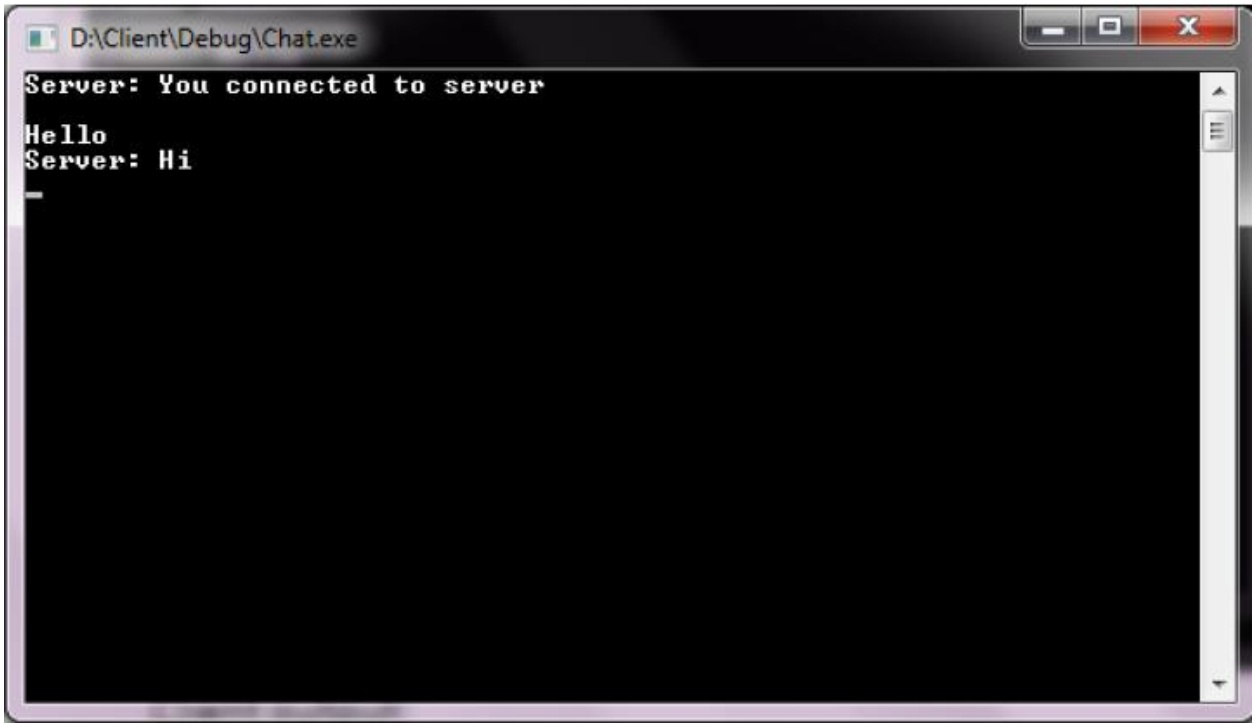
Server output:



4. Server writes “Hi” to client:

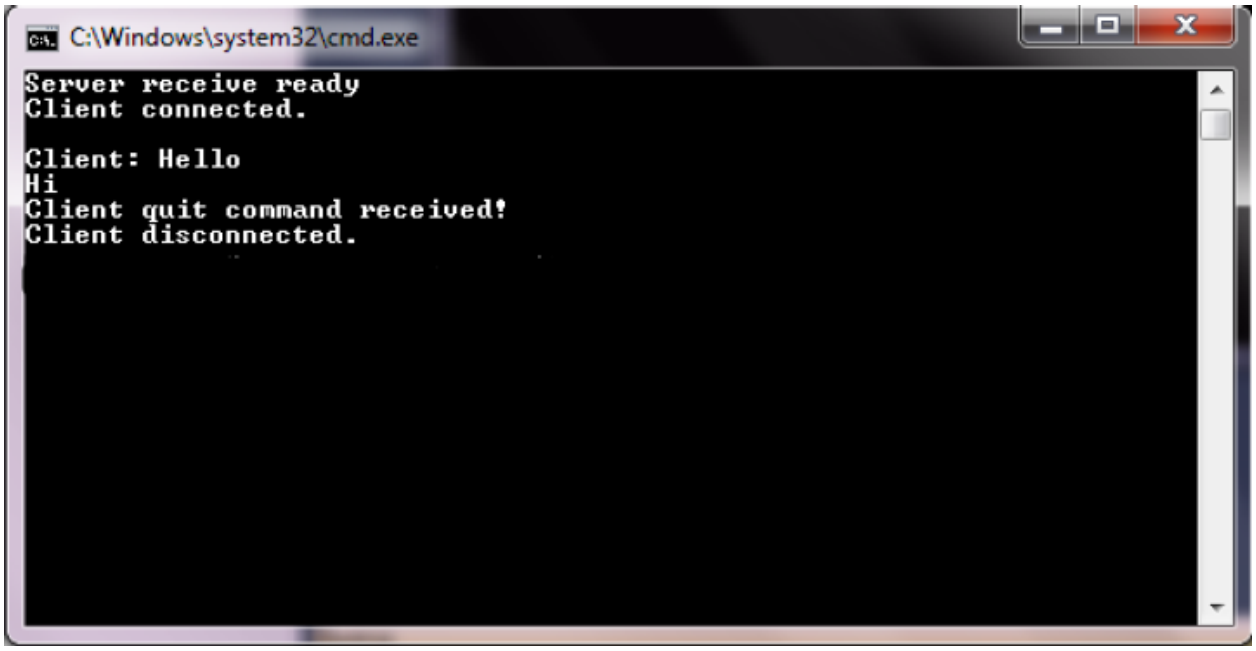
Server output:

Client output:

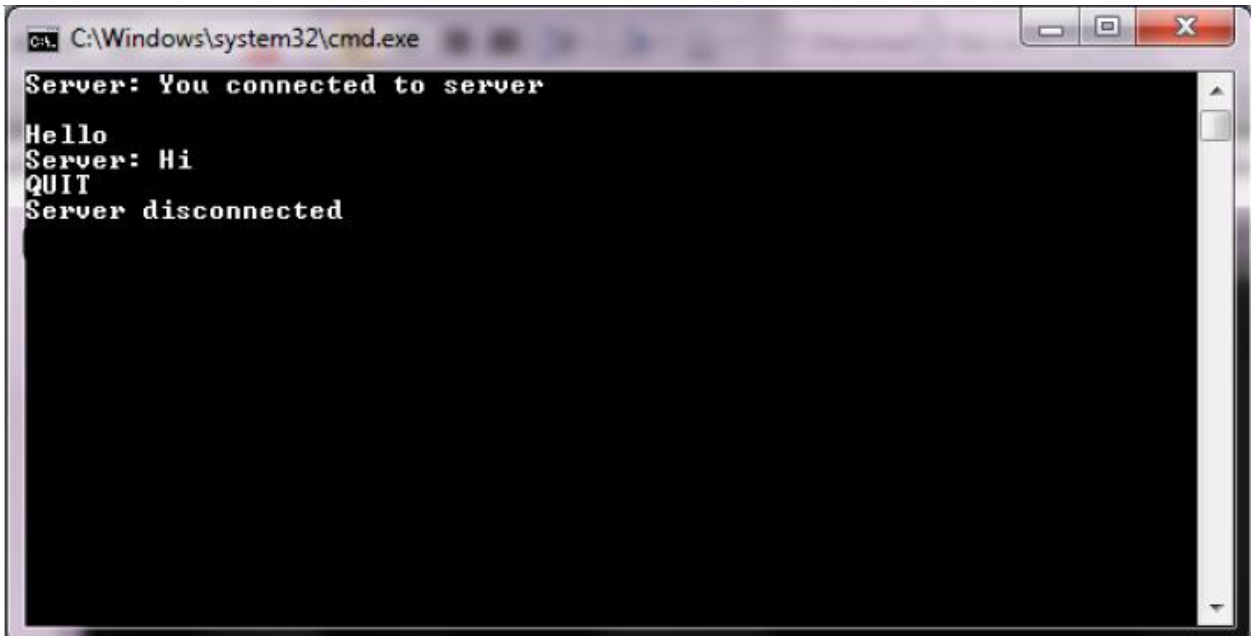


5. Client sends “QUIT” command:

Server output:



Client output:

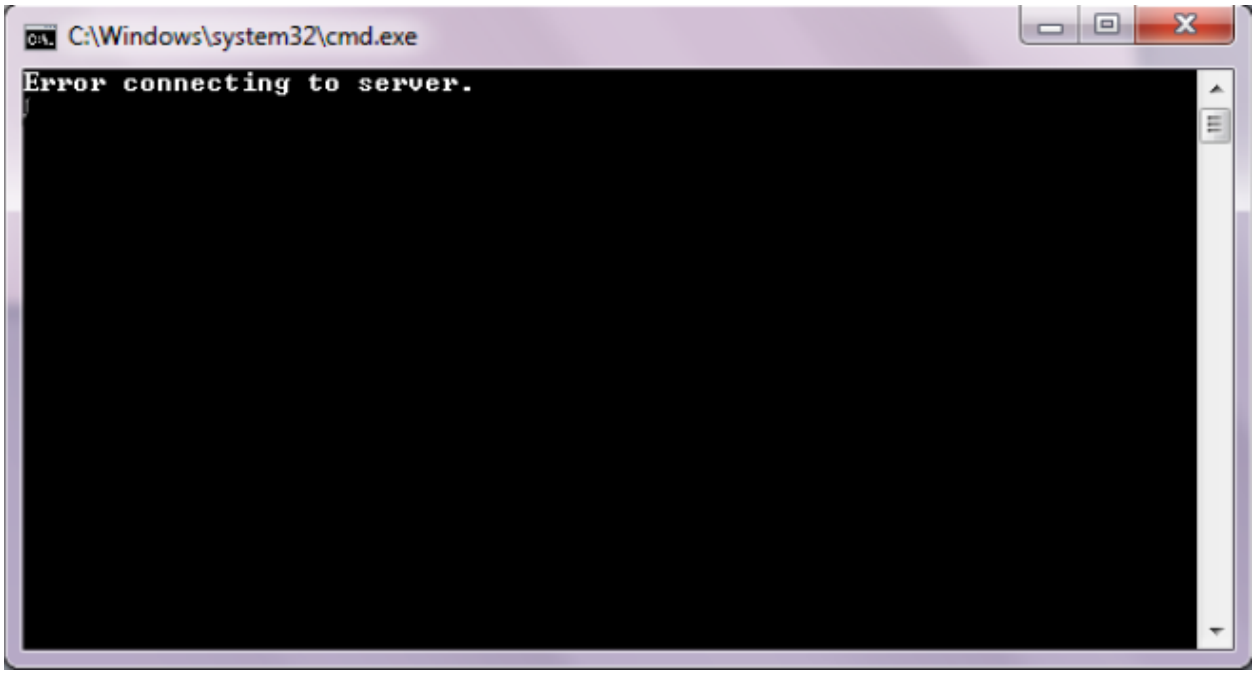


Now both programs terminate.

Test 2 (server not found)

1. Launching client program before server:

Client output:

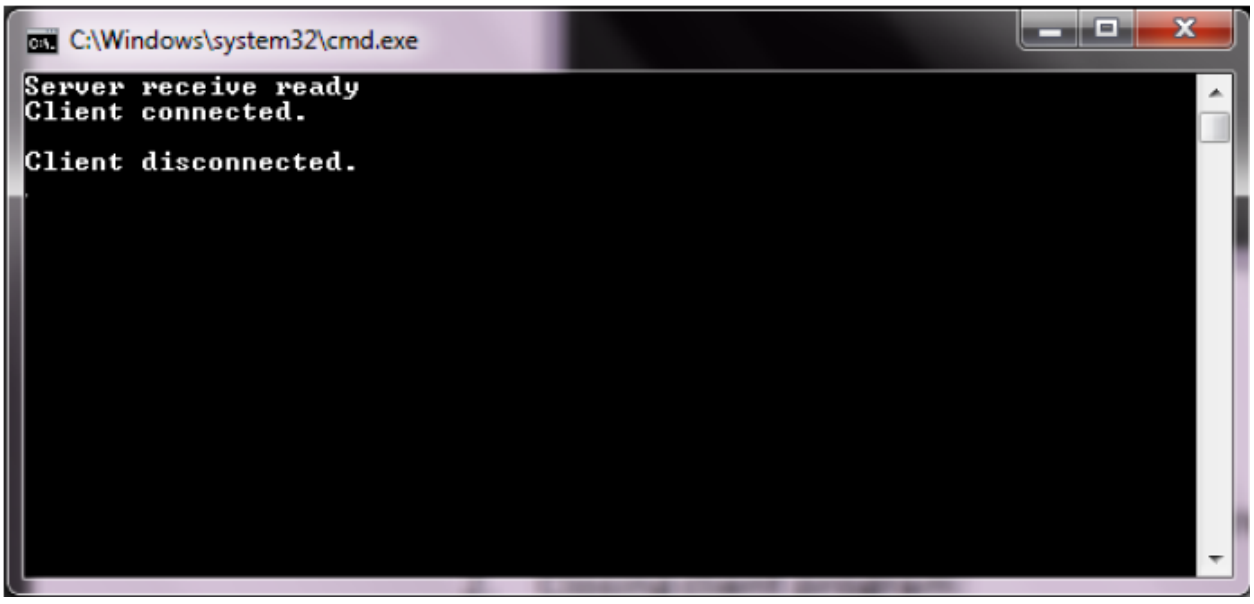


Test 3 (terminating client program)

1. Launching server, client (see test 1 screenshots 1, 2).

2. Closing client program:

Server output:



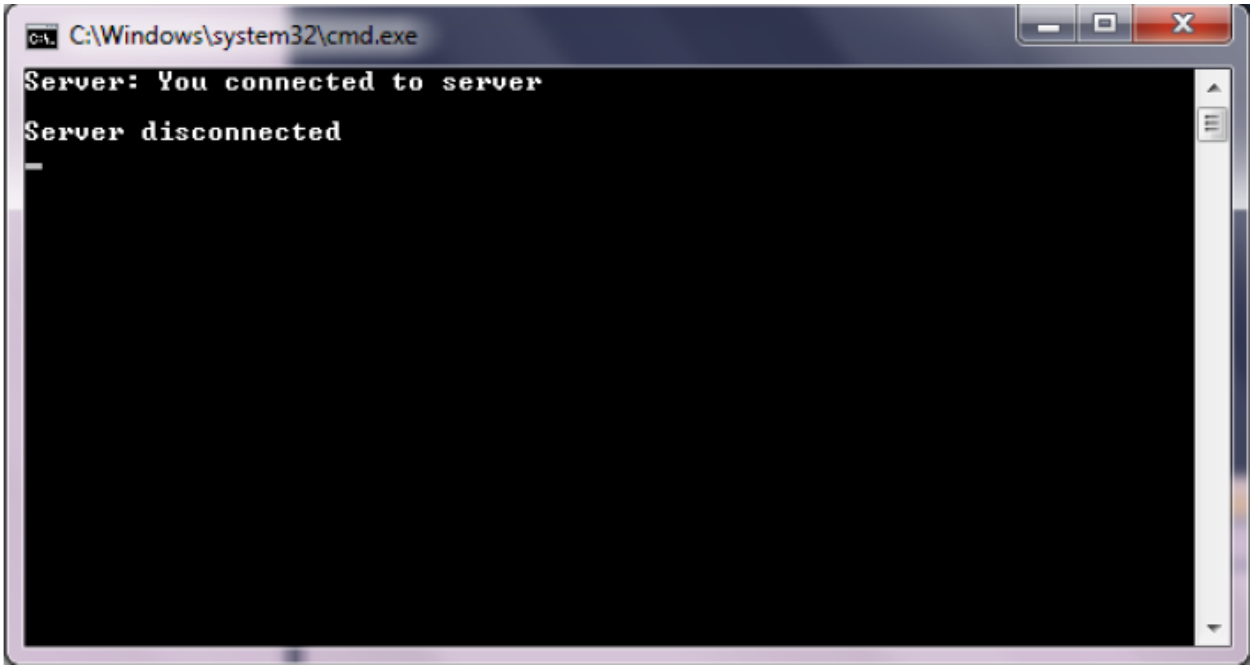
Now server program terminates.

Test 4 (server disconnected)

1. Launching server, client (see test 1 screenshots 1, 2).

2. Closing server program.

Client output:



Now client program terminates.