**CIS-427**

**With Dr. Zheng Song**

**Student: Demetrius Johnson**

**Program Assignment 1: Socket Programming**

**21 October 2022**

# Introduction and Special Notes

As in the next section, make sure to add the ws2\_32.lib library. This must be linked before compiling otherwise the header files used will be missing the files with definitions (in ws2\_32.lib) of their declarations. This is a Windows C++ server-client application that I wrote. I would like to recognize the udemy.com course: “*Socket Programming A to Z – Windows and Linux in C and C++”*  by [**Sonali Shrivastava**](https://www.udemy.com/user/sonali-das-19/). I used the online course to help me to understand socket programming on both Linux and Windows machines, and she even does a demonstration and provides sample code for a basic server-client program. All of my program I wrote in the main function and the comments are very well done; you can easily follow along and see how I implemented and explain everything, and the code for both client and server is not that long at all (only about 200 lines each, including comments).

Lastly, I would like to mention that the README.TXT file in the *source code* folder provides information given here in this report and some additional information in the event that you run into any compilation issues. Please reach out to me if you have any issues as my program and its compilation on my machine works without any issues.

# Setting up the IDE (integrated develop environment) with *Visual Studio 19 (Visual C++ 19)*

First, I needed to link the ws2\_32.lib library to my project so that I can use the socket program functions defined in the library for Windows 32-bit machines and so that when I compile my program, I will not getting any errors due to missing code/libraries.

* I had to go to project🡪properties🡪Linker🡪Input🡪Additional Dependencies🡪click drop down and select “edit” so that I could add *ws2\_32.lib* library:
* Graphical user interface

  Description automatically generated
* Graphical user interface, text, website

  Description automatically generated
  + Then click “OK” and “Apply” accordingly. Now, I have successfully linked the library (where the source code is) so that I can #include the appropriate header files and use the methods that are a part of that library.
  + Notice under “inherited values” which libraries are already added to the project by default, such as *kernel32.lib* or *user32.lib*.
* *Note to self:*

**LIB vs DLL**

* A DLL is a library that contains functions that can be called by applications at run-time while LIB is a static library whose code needs to be called during the compilation.
* Using LIB would result in a single file that is considerable bigger while you end up with multiple smaller files with DLL's.

# Source code (client and server programs)

## **ServerSocket.cpp**

//Author: Demetrius Johnson

//For: CIS-427 With Dr. Zheng Song at UM-Dearborn

//Date created: 10-21-22

//Date modified: 10-22-22

//purpose: Create the server for the client-server socket programming application

//to send and update BANNER message of the day service ran on the server.

#include <iostream>

#include <Winsock2.h> //need this for socket programing in Windows

#include <Ws2tcpip.h> //need this for updated functions that replace depracated functions (such as inet\_addr() function)

#define APP\_SOCKET\_PORT 5555

using namespace std;

int main()

{

WSADATA WSAData; //WinSock data structure

SOCKET serverSock, clientSock; //note: SOCKET is a descriptor to identify SOCKADDR\_IN structures: use serverSock to listen to incoming connection requests; use clientSock for accepting incoming requests;

//We do this so that the server can listen and accept requests at the same time.

SOCKADDR\_IN serverAddr, clientAddr; // The sockaddr\_in structure specifies the address family, IP address, and port of the server to be connected to.

int check\_WSAStartup = WSAStartup(MAKEWORD(2, 0), &WSAData); //initiates use of the Winsock DLL by a process; we are using Winsock v 2.0 --> 2,0

if (check\_WSAStartup != NO\_ERROR) {

wprintf(L"WSAStartup function failed with error: %d\n", check\_WSAStartup);

return 1;

}

//initialize socket and get descriptor

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

//socket initialization error checking:

if (serverSock == INVALID\_SOCKET) {

wprintf(L"socket function failed with error: %ld\n", WSAGetLastError());

WSACleanup();

return 1;

}

//inet\_pton() function converts ASCII string to binary IP address to be stored in addr object --> assign the IP to the serverAddr socket.

//AF\_INET specifies IPV4

//addr.sin\_port = htons(port #) assigns port# to the addr socket.

serverAddr.sin\_addr.s\_addr = INADDR\_ANY; //INADDR\_ANY means listen to all ip addresses of all interfaces on the local machine

serverAddr.sin\_family = AF\_INET;

serverAddr.sin\_port = htons(APP\_SOCKET\_PORT);

//serverSock descriptor to serverAddr structure

bind(serverSock, (SOCKADDR \*)&serverAddr, sizeof(serverAddr));

//bind error checking:

if (serverSock == SOCKET\_ERROR) {

wprintf(L"bind function failed with error %d\n", WSAGetLastError());

serverSock = closesocket(serverSock);

if (serverSock == SOCKET\_ERROR)

wprintf(L"closesocket function failed with error %d\n", WSAGetLastError());

WSACleanup();

return 1;

}

//listen on the serverSock and check for errors when we try to listen

if (listen(serverSock, 0) == SOCKET\_ERROR)

wprintf(L"listen function failed with error: %d\n", WSAGetLastError());

//WSAGetLastError allows you to get the most recent error code (if there were any) stored in a winsock2.h variable

if (WSAGetLastError() == 10060) //error code 10060 means connection timed out (failed)

{

cout << "...connection timed out...\n...exiting program...\n";

return 1;

}

cout << "Listening for incoming connections..." << endl;

//create a buffer for sending and receiving data

char buffer[1024];

int clientAddrSize = sizeof(clientAddr); //get size of clientAddr struct size

//now accept a client that arrives at the serverSock that we are listening on; set clentSock descriptor

//by setting clientAddr structure equal to the connection/structure (SOCKADDR\_IN) that serverSock is listening to.

if((clientSock = accept(serverSock, (SOCKADDR \*)&clientAddr, &clientAddrSize)) != INVALID\_SOCKET)

{

cout << "Client connected!" << endl;

//receive initial message from client:

memset(buffer, 0, sizeof(buffer)); //reset buffer (zero it --> set all values in the array to 0)

recv(clientSock, buffer, sizeof(buffer), 0);

cout << "Client says: " << buffer << endl;

//send initial message to client:

cout << "--SENDING a message to Client--" << endl;

memset(buffer, 0, sizeof(buffer)); //reset buffer (zero it --> set all values in the array to 0)

strcpy\_s(buffer, "\n\tWelcome to the Yet Another Message of the Day (YAMOTD) Banner server.\n\t~We are connected over TCP/IP.~\n");

send(clientSock, buffer, sizeof(buffer), 0);

cout << "--message SENT to Client--" << endl;

}

else {//error checking output

wprintf(L"accept failed with error: %ld\n", WSAGetLastError());

cout << "\n...invalid socket...closing connection...\n";

closesocket(serverSock);

closesocket(clientSock);

WSACleanup();

return 1;

}

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

//CONNECTION IS NOW ESTABLISHED AND TESTED, WE CAN NOW SEND AND RECEIVE ON ClientSocket - BEGIN "YET ANOTHER MESSEAGE OF THE DAY BANNER" (YAMOTD) PROGRAM:

//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

char YAMOTD[200] = "An apple a day keeps the doctor away.\n"; //default banner message

char MSGGET[20] = "MSGGET\n";

char MSGSTORE[20] = "MSGSTORE\n";

char QUIT[20] = "QUIT\n";

char MSG\_200\_OK[20] = "200 OK\n\t";

//NOTE: if strings are EQUAL, strcmp (string compare) returns 0

//run this loop until client wishes to close the connection

while (true) {

//receive message from client:

memset(buffer, 0, sizeof(buffer)); //reset buffer (zero it --> set all values in the array to 0)

recv(clientSock, buffer, sizeof(buffer), 0);

cout << "Client says: " << buffer << endl;

//MSGGET REQUEST:

//if buffer == MSGGET

if (strcmp(buffer, MSGGET) == 0) {

//send 200 OK AND YAMOTD message to client:

cout << "--SENDING YAMOTD to Client--" << endl;

memset(buffer, 0, sizeof(buffer)); //reset buffer (zero it --> set all values in the array to 0)

//next 2 lines --> buffer = MSG\_200\_OK + YAMOTD:

strcpy\_s(buffer, MSG\_200\_OK);

strcat\_s(buffer, YAMOTD);

//send message to client:

send(clientSock, buffer, sizeof(buffer), 0);

cout << "--YAMOTD SENT to Client--" << endl;

}

//MSGSTORE REQUEST:

//if buffer == MSFSTORE

else if (strcmp(buffer, MSGSTORE) == 0) {

//send "200 OK" message to client:

cout << "--SENDING '200 OK' to Client--" << endl;

memset(buffer, 0, sizeof(buffer)); //reset buffer (zero it --> set all values in the array to 0)

strcpy\_s(buffer, "200 OK (input banner message terminated with newline, 200 char or less):\n");

send(clientSock, buffer, sizeof(buffer), 0);

cout << "--'200 OK' SENT to Client--" << endl;

//receive message from client:

memset(buffer, 0, sizeof(buffer)); //reset buffer (zero it --> set all values in the array to 0)

recv(clientSock, buffer, sizeof(buffer), 0);

cout << "Client YAMOTD Banner reads: " << buffer << endl;

//store message:

cout << "~storing message~" << endl;

memset(YAMOTD, 0, sizeof(YAMOTD)); //reset buffer (zero it --> set all values in the array to 0)

strcpy\_s(YAMOTD, buffer);

//send confirmation to client:

cout << "--SENDING banner update confirmation to Client--" << endl;

memset(buffer, 0, sizeof(buffer)); //reset buffer (zero it --> set all values in the array to 0)

strcpy\_s(buffer, "200 OK ~banner updated by server succesfully~\n");

send(clientSock, buffer, sizeof(buffer), 0);

cout << "--banner update confirmation SENT to Client--" << endl;

}

//QUIT REQUEST:

//if buffer == QUIT

else if (strcmp(buffer, QUIT) == 0) {

//send "200 OK" quit message to client:

cout << "--SENDING '200 OK' quit confirmation to Client--" << endl;

memset(buffer, 0, sizeof(buffer)); //reset buffer (zero it --> set all values in the array to 0)

strcpy\_s(buffer, "200 OK...closing socket connection...\n");

send(clientSock, buffer, sizeof(buffer), 0);

cout << "--'200 OK' quit confirmation SENT to Client--" << endl;

//close all sockets and exit server application

closesocket(serverSock);

closesocket(clientSock);

WSACleanup();

cout << "Client disconnected. Exiting server program..." << endl;

system("pause");

return 0;

}

//INVALID REQUEST:

else

{

//send error message to client:

cout << "--SENDING error message to Client--" << endl;

memset(buffer, 0, sizeof(buffer)); //reset buffer (zero it --> set all values in the array to 0)

strcpy\_s(buffer, "~error: invalid request. Please try again (MSGGET or MSGSTORE or QUIT)\n");

send(clientSock, buffer, sizeof(buffer), 0);

cout << "--error message SENT to Client--" << endl;

}

}

}

## **ClientSocket.cpp**

//Author: Demetrius Johnson

//For: CIS-427 With Dr. Zheng Song at UM-Dearborn

//Date created: 10-21-22

//Date modified: 10-22-22

//purpose: Create the client for the client-server socket programming application

//to get and update BANNER message of the day service ran on the remote server.

#include <iostream>

#include <Winsock2.h> //need this for socket programing in Windows

#include <Ws2tcpip.h> //need this for updated functions that replace depracated functions (such as inet\_addr() function)

#define APP\_SOCKET\_PORT 5555

using namespace std;

//this program's main function takes 1 commmandline parameter: server ip address as per the requirement for this assignment.

int main(int argc, char\*\* argv ) //remember: element 0 in the vector is the file name; first commandline argument begins at element 1

{

WSADATA WSAData;

SOCKET serverSock; //note: SOCKET is a descriptor to identify SOCKADDR\_IN structures; we will use serverSock/addr to connect to the server via IP address and TCP port#

SOCKADDR\_IN addr; // The sockaddr\_in structure specifies the address family, IP address, and port of the server to be connected to.

WSAStartup(MAKEWORD(2,0), &WSAData); //initiates use of the Winsock DLL by a process; we are using Winsock v 2.0 --> 2,0

serverSock = socket(AF\_INET, SOCK\_STREAM, 0); //AF\_INET specifies IPv4, SOCK\_STREAM means use TCP, 0 means do not use a specific protocol (i.e. 1 = ICMP)

char ip\_address[16] = "127.0.0.1"; //default ip will be loopback address of local machine

//only update IP with command line parameter if there is one present and if no more than 1 parameter has been passed in.

if (argc == 2)

strcpy\_s(ip\_address, argv[1]);

//inet\_pton() function converts ASCII string to binary IP address to be stored in addr object --> assign the IP to the addr socket.

//AF\_INET specifies IPV4

//addr.sin\_port = htons(port #) assigns port# to the addr socket.

//inet\_pton() function will convert the IPv4 string into binary and store the address in the the struct in the proper field (replaces the depracated inet\_addr() function)

inet\_pton(AF\_INET, ip\_address, &addr.sin\_addr.s\_addr);

addr.sin\_family = AF\_INET;

addr.sin\_port = htons(APP\_SOCKET\_PORT);

connect(serverSock, (SOCKADDR\*)&addr, sizeof(addr)); //try to connect to server using server IP and Port number we would like to connect on

//WSAGetLastError allows you to get the most recent error code (if there were any) stored in a winsock2.h variable

if (WSAGetLastError() == 10060) //error code 10060 means connection timed out (failed)

{

cout << "...connection timed out...\n...exiting program...\n";

closesocket(serverSock);

WSACleanup();

return 1;

}

if (WSAGetLastError() == 10061) //error code 10061 means connection refused by host/target machine (server refused connection)

{

cout << "...connection refused by target machine...\n...exiting program...\n";

closesocket(serverSock);

WSACleanup();

return 1;

}

cout << "Connected to server!" << endl;

//test sending messages to server on serverSock:

char buffer[1024]={'h', 'e', 'l', 'l', 'o', '.'};

send(serverSock, buffer, sizeof(buffer), 0);

cout << "Message sent!" << endl;

//test receiving messages from server on serverSock:

memset(buffer, 0, sizeof(buffer)); //reset buffer (zero it --> set all values in the array to 0)

recv(serverSock, buffer, sizeof(buffer), 0);

cout << "Server says: " << buffer << endl;

//////////////////////////////////////////////////////////////////////

//READY TO BEGIN REQUESTING YAMOTD Application services from server:

/////////////////////////////////////////////////////////////////////

char REQUEST[20];

//NOTE: if strings are EQUAL, strcmp (string compare) returns 0

do {

//get request input from USER:

cout << "Enter a request to send to YAMOTD banner server (MSGGET, MSGSTORE, or QUIT): ";

cin >> REQUEST; //get input from user

strcat\_s(REQUEST, "\n"); //concatenate newline character to input as required for this client-server application

//send request to server:

memset(buffer, 0, sizeof(buffer)); //reset buffer (zero it --> set all values in the array to 0)

strcpy\_s(buffer, REQUEST); //copy user request message into buffer

send(serverSock, buffer, sizeof(buffer), 0); //send request to server

cout << "Message sent!" << endl;

//Receive request response from server:

memset(buffer, 0, sizeof(buffer)); //reset buffer (zero it --> set all values in the array to 0)

recv(serverSock, buffer, sizeof(buffer), 0);

cout << "Server says: " << buffer << endl;

//case: requesting to update the banner

if (strcmp(REQUEST, "MSGSTORE\n") == 0) {

//send new banner to server:

memset(buffer, 0, sizeof(buffer)); //reset buffer (zero it --> set all values in the array to 0)

cin.ignore(sizeof(buffer), '\n'); //need to flush cin buffer before the next operation

cin.getline(buffer, 200, '\n'); //banner is limited by 200 characters

strcat\_s(buffer, "\n"); //concatenate newline character to input as required for this client-server application

send(serverSock, buffer, sizeof(buffer), 0); //send banner to server

cout << "Message sent!" << endl;

//Receive update response from server:

memset(buffer, 0, sizeof(buffer)); //reset buffer (zero it --> set all values in the array to 0)

recv(serverSock, buffer, sizeof(buffer), 0);

cout << "Server says: " << buffer << endl;

}

} while (strcmp(REQUEST, "QUIT\n") != 0);

closesocket(serverSock);

WSACleanup();

cout << "Socket closed." << endl << endl;

system("pause");

return 0;

}

# Test cases:

## Case 1: using default IP address (set to loopback: 127.0.0.1)

* I set the default IP address to 127.0.0.1 in the event that the IP address command line parameter receives no input, or the number of parameters passed in exceeds 2 (first parameter is default as file name, first command line parameter is the second parameter).
* I will show when I specify the default:

A screenshot of a computer

Description automatically generated

* Or when I simply just pass no command line parameters in:

Graphical user interface, text

Description automatically generated

* Notice for both screenshots above that I wrote default communication checks between client and server as they establish a connection; the server listens on one socket while the client attempts to connect; upon connecting, the server accepts the connection on another socket, and it is this socket that they communicate on (named “ClientSock” in my main program of the Server application, and “serverSock” in the Client application main program).
* Once connection is established, they can begin communicating regularly (exchanging messages until connection is closed).

### MSGGET

Text

Description automatically generated

* Notice above, that the default banner is displayed upon first request to server since the banner has not been updated.

### MSGSTORE

Text

Description automatically generated

* Above, I have updated the banner. Now I will call MSGGET again to prove banner has been updated:

Text

Description automatically generated

### Invalid Request input

A screenshot of a computer

Description automatically generated with medium confidence

* Notice above, I have sent several invalid request strings to the server and it responds appropriately and re-prompts the client to send another request. Then I show what happens when I send a valid request again (it works properly as server sends message of the day banner).

### QUIT

Graphical user interface, text

Description automatically generated

* Notice above, that the client requests to QUIT and receives confirmation from server before closing the socket connection. Server side also closes connections and exits program.

## Case 2: using IP address of my ethernet adapter

* Now I will use my local host machine’s ethernet adapter’s private IP address:
* Text

  Description automatically generated
* Notice I have pinged it too to make sure it is working and visible to my machine (as it should be of course since it is assigned to the ethernet adapter on my local machine)L
* Text

  Description automatically generated
* Now I will actually call the client application using my local 192.168.0.17 adapter’s IP; note, I specified IP\_ADDR\_ANY in my server application so that it will listen to all Network Adapters the machine it is running on. Note that for all test cases of this entire report, server and client are running on my same local host machine.

Text

Description automatically generated

### MSGGET, MSGSTORE, invalid input, MSGGET (shows updated banner), invalid input, QUIT

Now I will perform the same tests as I did previously but by connecting using the socket with the specified non-default IP address.

A screenshot of a computer

Description automatically generated with medium confidence

## Case 3: connection timed out

In this scenario, the IP address could not be reached (invalid host) so that the connection timed out code is thrown, which I wrote code to output that error message (and if the error message from connect() function throws a connection timed out error as well – see my code in the Client program).

Text

Description automatically generated

## Case 4: connection refused

Text

Description automatically generated

Notice I wrote an output method for when error code is thrown by winsocket telling the program that the connection was refused by target machine. The server application was not running, therefore, even though I could reach the IP address, no server process was running to allow me to make a TCP connection, so connection was refused by the machine that possessed the IP address (in this case my local machine’s ethernet interface card).

# Summary:

In summary, I used the Winsocket2 library from Windows to write a Windows client-server application using C++. The program works exactly as expected with no errors and even some limited error checking capabilities. It should be fairly easy to grade and compile and run as I have made all instructions, code comments, and screenshots with explanations very clear. Just about anyone with some literacy in computers and programming should be able to replicate my project and the results exactly.