University of North Carolina at Charlotte
Department of Electrical and Computer Engineering

Project Report #1 Socket Programming

Project 1

Author: Khoa Do

Partner: Jonathan Stevens

Date: February 2, 2018

This report was submitted in compliance with UNCC POLICY 407
THE CODE OF STUDENT ACADEMIC INTEGRITY, Revised November 6, 2014
(http://legal.uncc.edu/policies/up-407) ___KD, JS____. (Student's Initials)

Objectives

The objective of this first project is to create a socket program to get a hostname for a given IP address in any programming language. It means to find the name of the device and send it to the client. A socket program is one that creates the interface between two windows instances (programs) at a same time. The students will work in group of two to finish the project and write a report that includes the process of socket programming, descriptions of function used, codes, and comments.

Procedure and Result

The group decided to do this project in C++ programming language because the sample files for server and client were already given on Canvas in C++ language. The group implemented (modified) the codes to get what they were supposed to output – the name of the client (hostname or computer's name) that can be found in "About your PC" under the "Device name" of "System Settings." For example, the hostname of the computer using to write this report is LAPTOP-DF3P24M1.

First, the group tried to understand the codes. They were complicated because they had a structure inside of a structure in side of another structure, like a nested structure (object). With the help from the resources available on the internet (google, youtube, and forums), the team were able to understand what needed to be added to the original codes.

The instruction was not clear, so the team overthought the project and ended up overkilling it. Instead of outputting the hostname or the device's name, the group implemented the code to get and output the given IP, which was 127.0.0.1, the actual IP that the computer was using, the device's name, and the ability to send texts between the server and the client. Dr. Han and the TA said it was not necessary to output all this information.

Part 1 – Server.cpp

The first job was to implement the server.cpp file. The implementation of not only this file but also client.cpp file was simple. The team had trouble with project simply because we overthought what needed to be output and that we needed to be able to understand the codes (what they meant here and there). Later, it was visible that the team only need six lines of code (few extra libraries were added to the program).

```
char host[NI_MAXHOST];
char service[NI_MAXSERV];
ZeroMemory(host, NI_MAXHOST);
ZeroMemory(service, NI_MAXSERV);
if (getnameinfo((SOCKADDR*)&sockAddr, sizeof(sockAddr), host, NI_MAXHOST,
service, NI_MAXSERV, NI_NUMERICSERV) == 0)
{cout << host << " connected via port " << service << " " << NI_NUMERICSERV <<
endl;}
```

The code above would output the device name. There were some syntax errors that did not make the program work but were fixed later in that day ("[" was used instead of "(" in ZeroMemory lines). That was about it for the first part to implement in the. However, the team overkilled it by displaying the IP given in the files, the actual IP that the computer was using, and the texts sent from the client to the server. To do that, the team implement the function "DWORD WINAPI Protocol(LPVOID IPParameter)" in line 19. It was structure data type. The implementation was done from line 94 to 112.

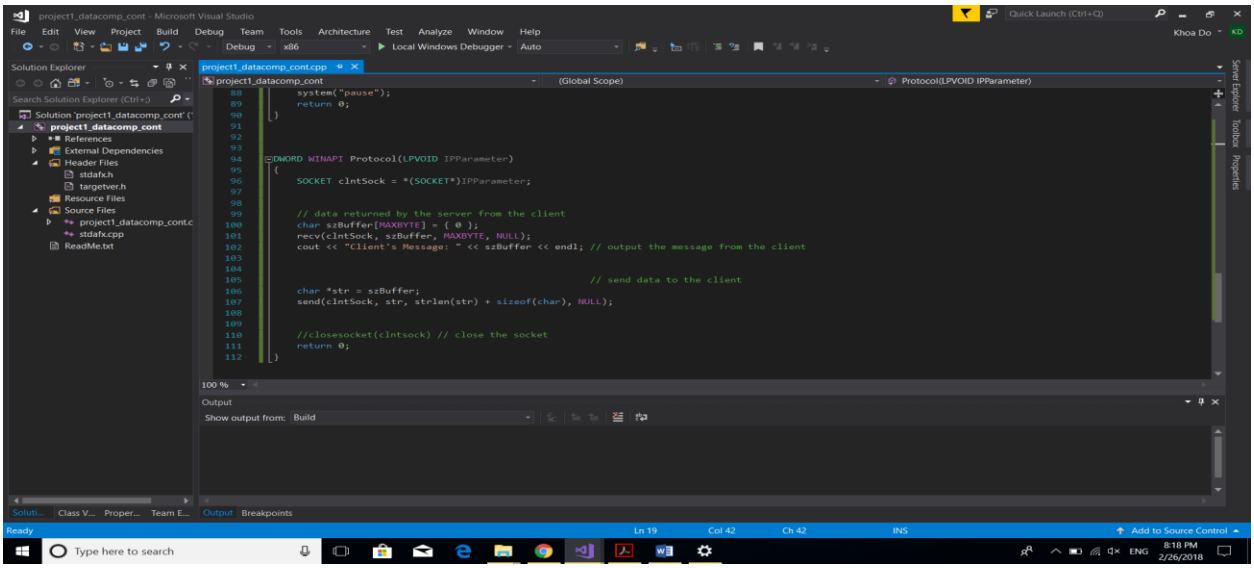

Figure 1-1
A Screenshot of the Implementation of the Server.cpp

Noticing a large part of the code in the main function was modified from the original function because the team was trying to follow a tutorial on the internet. However, the original code and the modified code were equivalent. Some minor syntax errors were pointed out and fixed before the program was successfully run.

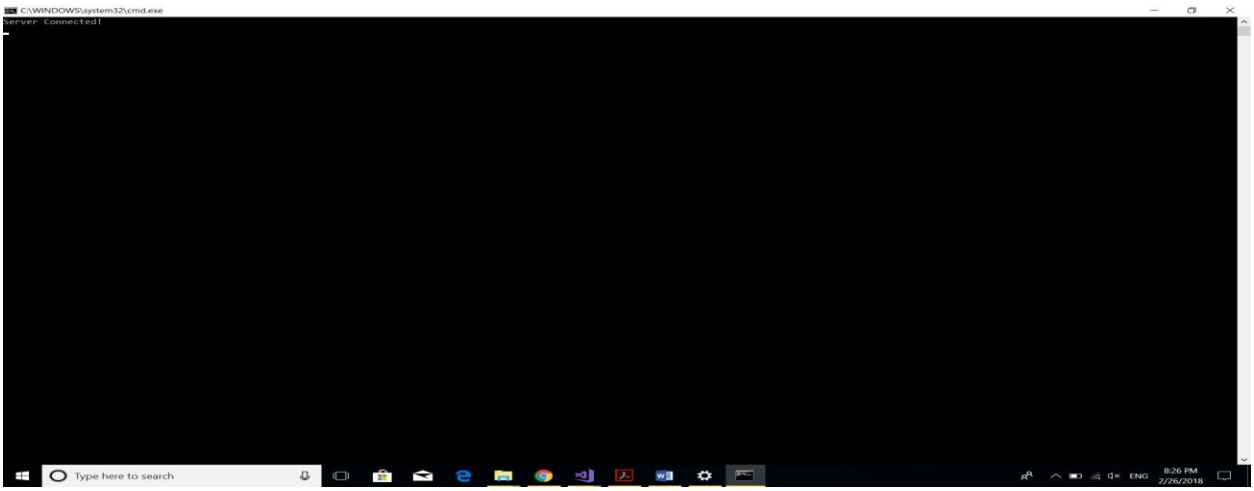

Figure 1-2
A Screenshot of the Working Server.cpp

The figure above shows the output of the server.cpp without running the client.cpp. In this project, both files must be run at a same time in order to create an interface (connection) between them. The strcpy_s that used the #include<string.h> library could not be able to run on DevC++ IDE; therefore, the group had to use CodeBlocks or Visual Studio. To display the actual IP address that the computer was using, the following lines of code were used:

```
int s = 0;
s = (int)ntohs((sockAddr.sin_port));
cout << "Client is using " << inet_ntoa(sockAddr.sin_addr) << s << " IPA" << endl;
```

Part 2 – Client.cpp

For the second part of this project, the group had to implement the client.cpp. There was not that much to add to the client.cpp file. It could have stayed as it was suppsoed to be initially. However, the team modified the client.cpp as well because we did modify a large part of the main function in the server.cpp file. Though, the codes were identical to one another. Moreover, the team had to implement a function that could send a text message to the server's prompted windows since we thought that the program was required to be able to do this.

```
connect(sock, (SOCKADDR*)&sockAddr, sizeof(SOCKADDR));
        cout << "Send a message(no space between words): ";
        char sendString[500]; // max of 500 characters
        cin >> sendString;
        strcat(sendString, "\n");
```

According to the code above, the easiest way to implement was to collect and send the message using "character (char)" data type. The message could have as 500 characters as maximum, and no space between words required; otherwise, the code would only send the first word in the message to the server. The number of maximum characters could be changed. The team could have implemented a code that could sent the whole message with spaces between words as "string (str) data type, but it was way more complicated to do that, and that this part of the project was "optional."

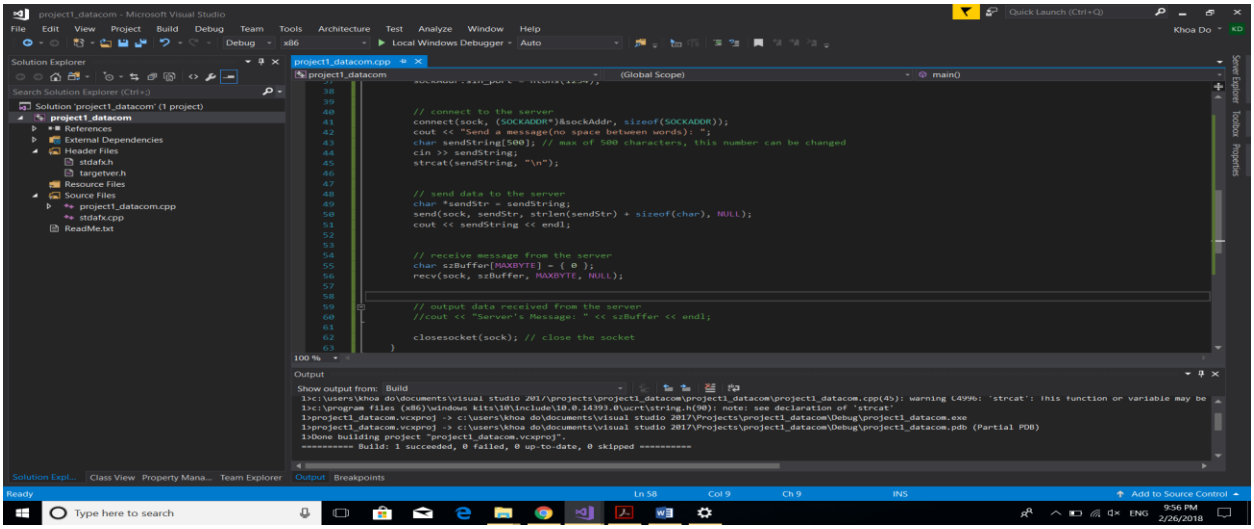

Figure 2-1
A Screenshot of the Implementation of the Client.cpp

Then, using the following code, the client would be able to receive the text message back from the server.

```
// send data to the server
char *sendStr = sendString;
send(sock, sendStr, strlen(sendStr) + sizeof(char), NULL);
cout << sendString << endl;
```

However, this could only display the same text message that the client had sent to the server before. Due to the time constrain, the team decided to leave the code as it was and would go back to it some time in the future to make the program look even better. The group did not have any trouble in writing the code for the second file such as syntax errors or trying to understand the code since we had spent many hours to work on the first file.

As a result, the group obtained the following prompted window when ran the program without running the server.cpp.

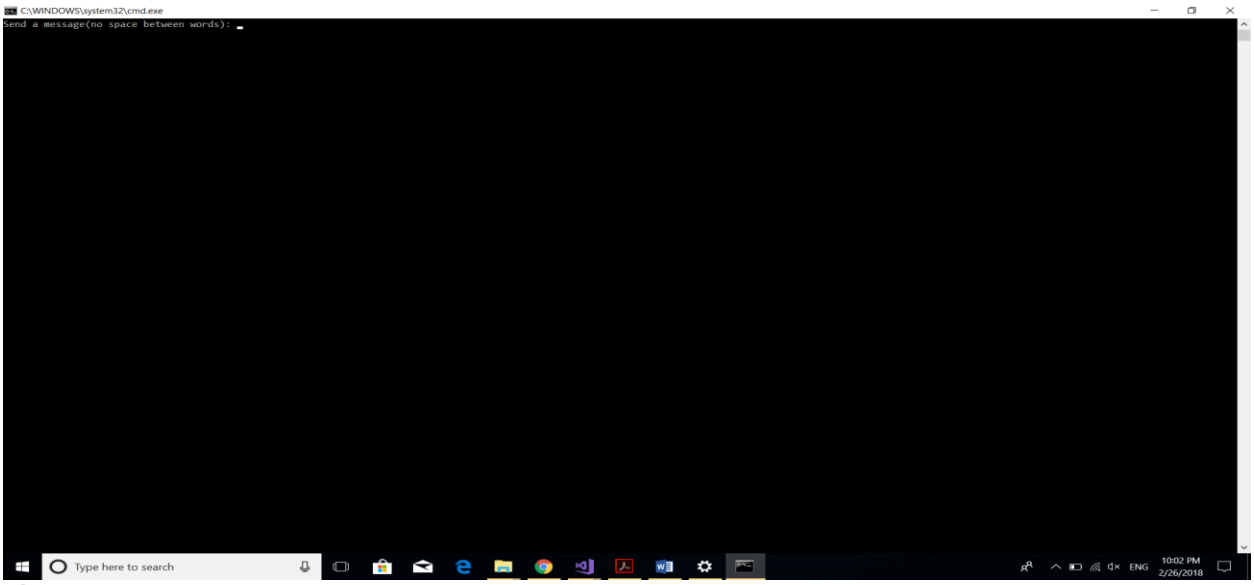

Figure 2-2
A Screenshot of the Working Client.cpp

Part 3 – The Outcome

After successfully writing up, compiling, and running the two files - server.cpp file and client.cpp file separately, the team was ready to do the final demo. The team ran both files at the same time and the outputs on both files were as what had been expected – the given IP address, the actual IP address, device name, message sent between the server and the client. Noticing that the sever.cpp must be run first in order to recognize and connect with the clien.cpp, which is run after the sever.cpp. The following figure shows the final result of the project.

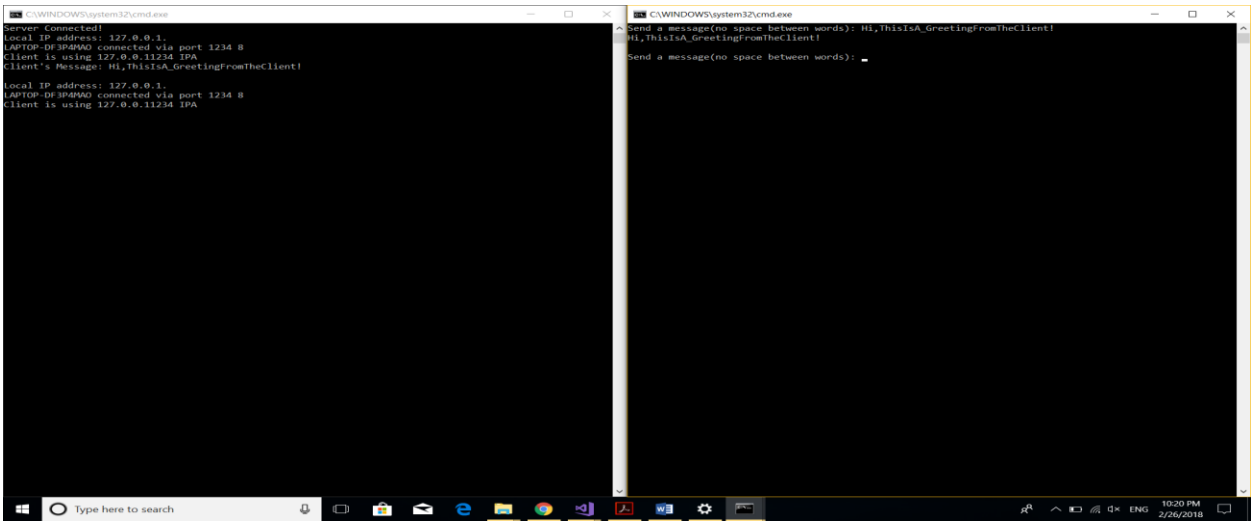

Figure 3-1
Final Result 1

However, the goal was to send the hostname to the client. The team did not obtain that objective because we overthought the project. Instead of adding another function to send the host name to the client, which made the project more complicated to the team, we decided to create another project to implement the codes from there. It took couple lines of code implemented in the sever.cpp to display and send the hostname to the client.cpp (which kept the same as it was). The team was not sure if this was enough for the project because there were just 4 to 5 lines of code and the project was finished. This was the reason why the team overthought the project because we thought this was too short for a project, we needed to do something else extra.

```
        char host[NI_MAXHOST];
        char service[NI_MAXSERV];
        ZeroMemory(host, NI_MAXHOST);
        ZeroMemory(service, NI_MAXSERV);
        if (getnameinfo((SOCKADDR*)&sockAddr, sizeof(sockAddr), host, NI_MAXHOST,
        service, NI_MAXSERV, NI_NUMERICSERV) == 0)
{cout << host << " connected via port " << service << " " << NI_NUMERICSERV << endl;}

        int s = 0;
        s = (int)ntohs((sockAddr.sin_port));
cout << "Client is using " << inet_ntoa(sockAddr.sin_addr) << s << " IPA" << endl;
```

The same codes used in part 1 and 2 were “reused” for the “new project,” which was much shorter and cleaner than the “old one.” Running the program, the team obtained the following result:

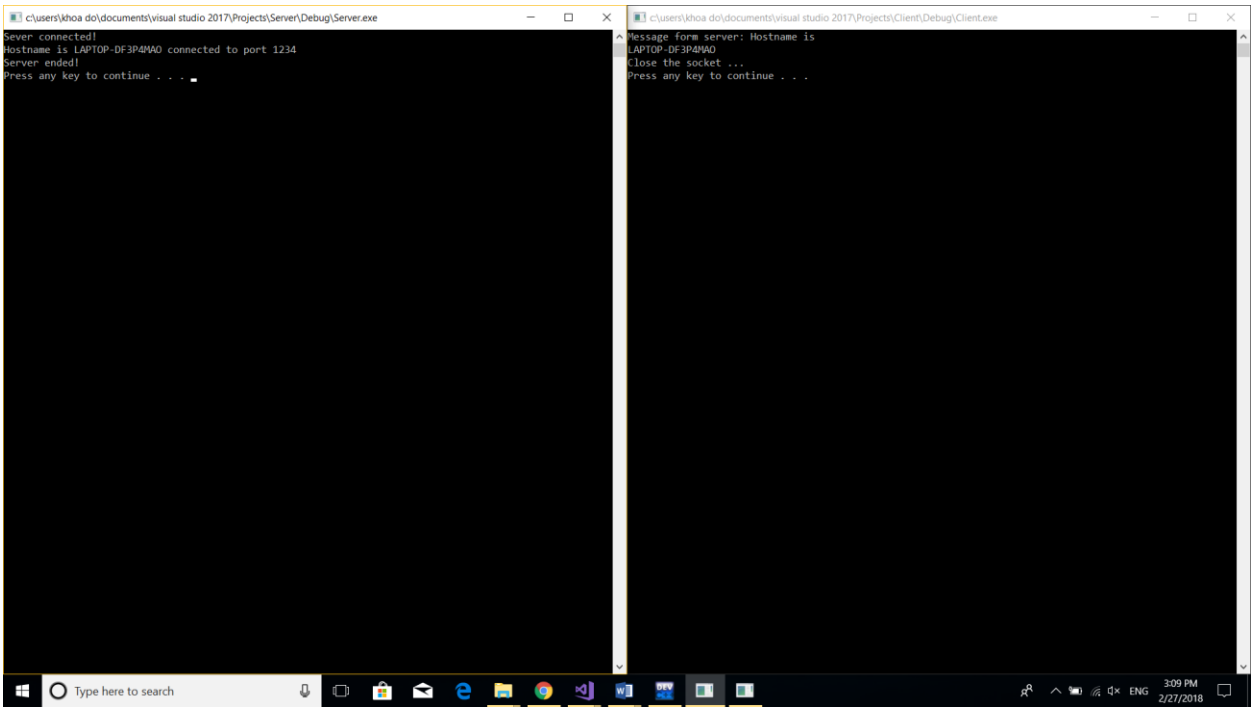

Figure 3-1
Final Result 2

Conclusions

We felt overwhelm at the beginning because we did not know where to start on the project. We are familiar with C++ programming but there were many syntaxes in the project that we did not understand. After teaching ourselves from the resources available online, we started the project and finished it quite fast, which was around 40 minutes. We overthought the project and did many unnecessary extra functions while we only needed to send the device name to the clien.cpp. However, in return, we learned a lot from this project. We will definitely go back to this project some time in the future to "integrate" the "new project" with the "old one" and fix some minor defects such as sending text without spaces between words, the ability to text between the server and the client instead of the client only.

Code

Part 1 – Server.cpp (OLD)

```
// project1_datacomp_cont.cpp : Defines the entry point for the console application.
#include "stdafx".h
#define _WINSOCK_DEPRECATED_NO_WARNINGS
#include <stdio.h>
#include <winsock2.h>
#include <windows.h>
#include <iostream>
#include <cstdlib>
#include <stdlib.h>
#include <string.h>
#include <string>
#include <ws2tcpip.h>
#pragma comment (lib, "ws2_32.lib")  // load ws2_32.dll

DWORD WINAPI Protocol(LPVOID IPParameter);

using namespace std;

int main()
{
	cout << "Server Connected!" << endl;

	// initialize winsock
	WSADATA wsaData;
	WSAStartup(MAKEWORD(2, 2), &wsaData);


	// create a socket
	SOCKET servSock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);


	// bind the socket to IP_ADDRESS:PORT
	sockaddr_in sockAddr;
	memset(&sockAddr, 0, sizeof(sockAddr));  // fill every byte with 0s
	sockAddr.sin_family = PF_INET;  // use IPv4 address
	sockAddr.sin_addr.s_addr = inet_addr("127.0.0.1"); // specify the IP address
	sockAddr.sin_port = htons(1234);  // choosing port
	bind(servSock, (SOCKADDR*)&sockAddr, sizeof(SOCKADDR));
```

```
	// start listening
	listen(servSock, 20);
	// connect to the client
	SOCKADDR_IN clntAddr;


	int nSize = sizeof(SOCKADDR);
	int s = 0;
	char host[NI_MAXHOST];
	char service[NI_MAXSERV];
	ZeroMemory(host, NI_MAXHOST);
	ZeroMemory(service, NI_MAXSERV);

	while (true)
	{
		SOCKET clntSock = accept(servSock, (SOCKADDR*)&clntAddr, &nSize);
		char szIpAddress[16]; // name of the online host
		strcpy_s(szIpAddress, sizeof(szIpAddress), inet_ntoa(clntAddr.sin_addr));
		cout << "Local IP address: " << szIpAddress << "." << endl;

		if (getnameinfo((SOCKADDR*)&sockAddr, sizeof(sockAddr), host,
NI_MAXHOST, service, NI_MAXSERV, NI_NUMERICSERV) == 0)
		{
			cout << host << " connected via port " << service << " " <<
NI_NUMERICSERV << endl;
		}

		s = (int)ntohs((sockAddr.sin_port));
		cout << "Client is using " << inet_ntoa(sockAddr.sin_addr) << s << " IPA" <<
endl;



		HANDLE hThread_1 = CreateThread(NULL, 0, Protocol, (LPVOID)&clntSock,
0, NULL);
		CloseHandle(hThread_1);

	}

	closesocket(servSock); // close the socket
							// clean up winsock
	WSACleanup();
	cout << "Server ended.\n" << endl;
	system("pause");
	return 0;
}
```

```
DWORD WINAPI Protocol(LPVOID IPParameter)
{
	SOCKET clntSock = *(SOCKET*)IPParameter;


	// data returned by the server from the client
	char szBuffer[MAXBYTE] = { 0 };
	recv(clntSock, szBuffer, MAXBYTE, NULL);
	cout << "Client's Message: " << szBuffer << endl; // output the message from the client



// send data to the client
	char *str = szBuffer;
	send(clntSock, str, strlen(str) + sizeof(char), NULL);


	//closesocket(clntsock) // close the socket
	return 0;
}
```

Code

Part 2 – Client.cpp (OLD)

```
	// project1_datacom.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"


#define _WINSOCK_DEPRECATED_NO_WARNINGS
#include <stdio.h>
#include <winsock2.h>
#include <windows.h>
#include <iostream>
#include <cstdlib>
#include <stdlib.h>
#include <cstring>
#include <string>
#pragma comment (lib, "ws2_32.lib")  // load ws2_32.dll

using namespace std;
int main()
{
	// initialize winsock
	WSADATA wsaData;
	WSAStartup(MAKEWORD(2, 2), &wsaData);
	SOCKET sock = NULL;

	while (true)
	{
		// create a socket
		sock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);


		// bind socket to IP_ADDRESS:PORT
		sockaddr_in sockAddr;
		memset(&sockAddr, 0, sizeof(sockAddr));
		sockAddr.sin_family = PF_INET;
		sockAddr.sin_addr.s_addr = inet_addr("127.0.0.1");
		sockAddr.sin_port = htons(1234);


		// connect to the server
		connect(sock, (SOCKADDR*)&sockAddr, sizeof(SOCKADDR));
		cout << "Send a message(no space between words): ";
```

```
            char sendString[500]; // max of 500 characters, this number can be changed
            cin >> sendString;
            strcat(sendString, "\n");


            // send data to the server
            char *sendStr = sendString;
            send(sock, sendStr, strlen(sendStr) + sizeof(char), NULL);
            cout << sendString << endl;


            // receive message from the server
            char szBuffer[MAXBYTE] = { 0 };
            recv(sock, szBuffer, MAXBYTE, NULL);


            // output data received from the server
            //cout << "Server's Message: " << szBuffer << endl;

            closesocket(sock); // close the socket
        }


        cout << "Server ended" << endl;
        // clean up the winsock
        WSACleanup();
        system("pause");
        return 0;
}
```

Code

Part 1 – Server.cpp (NEW)

```
#include <iostream>
#include <winsock2.h>
// load ws2_32.dll
#pragma comment (lib, "ws2_32.lib")
using namespace std;

int main() {
	// initialize winsock
	WSADATA wsaData;
	WSAStartup(MAKEWORD(2, 2), &wsaData);

	// create a socket
	SOCKET servSock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);

	// bind the socket to IP_ADDRESS:PORT
	sockaddr_in sockAddr;
	memset(&sockAddr, 0, sizeof(sockAddr));
	sockAddr.sin_family = PF_INET;
	sockAddr.sin_addr.S_un.S_addr = INADDR_ANY;
	sockAddr.sin_port = htons(1234);  // port
	bind(servSock, (SOCKADDR*)&sockAddr, sizeof(SOCKADDR));
	cout << "Sever connected!" << endl;

	// start listening
	listen(servSock, 20);

	// connect to the client
	SOCKADDR clntAddr;
	int nSize = sizeof(SOCKADDR);
	SOCKET clntSock = accept(servSock, (SOCKADDR*)&clntAddr, &nSize);

	// send message to the client
	char str[MAXBYTE] = { 0 };
	gethostname(str, MAXBYTE);
	int s = 0;
	s = (int)ntohs((sockAddr.sin_port));
	cout << "Hostname is " << str << " connected to port " << s << endl;
	send(clntSock, str, strlen(str) + sizeof(char), NULL);

	// close the socket
	closesocket(clntSock);
	closesocket(servSock);
	cout << "Server ended!" << endl;
```

```
        // clean up winsock
        WSACleanup();
        system("pause");

        return 0;
}
```

Code

Part 2 – Client.cpp (NEW)

```
#define _WINSOCK_DEPRECATED_NO_WARNINGS
#include <iostream>
#include <winsock2.h>
// load ws2_32.dll
#pragma comment (lib, "ws2_32.lib")  //

using namespace std;

int main() {
	// initialize winsock
	WSADATA wsaData;
	WSAStartup(MAKEWORD(2, 2), &wsaData);

	// create a socket
	SOCKET sock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);

	// bind socket to IP_ADDRESS:PORT
	sockaddr_in sockAddr;
	memset(&sockAddr, 0, sizeof(sockAddr));
	sockAddr.sin_family = PF_INET;
	sockAddr.sin_addr.s_addr = inet_addr("127.0.0.1");
	sockAddr.sin_port = htons(1234);

	// connect to the server
	connect(sock, (SOCKADDR*)&sockAddr, sizeof(SOCKADDR));

	// receive message from the server
	char szBuffer[MAXBYTE] = { 0 };
	recv(sock, szBuffer, MAXBYTE, NULL);
	cout << "Message form server: Hostname is " << endl << szBuffer << endl;

	// close the socket
	closesocket(sock);
	cout << "Close the socket ..." << endl;

	// clean up the winsock
	WSACleanup();
	system("pause");

	return 0;
}
```