Министерство образования Республики Беларусь

Учреждение образования «Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа № 4

За 6 семестр

По дисциплине: «Аппаратно-программное обеспечение ЭВМ и сетей»

Тема: «Изучение протокола TCP»

Выполнила:

студентка 3-го курса

группы АС-56

Карпенко М.В.

Проверил:

Булей Е. В.

Брест 2022

Цель работы: 1) изучить основы программирования сетевых приложений Windows на базе библиотеки WINSOCK2.H; 2) приобрести навыки по практическому использованию библиотеки для реализации сетевых приложений в среде C++ на базе протоколов TCP и UDP.

Вариант 4

4	После приема каждой группы из 64 символов от клиента сервер формирует и отсылает клиенту цепочку <«символ1-количество1», «символ2-количество2» и т.д.>, где «символ» - встреченный в группе символ, а «количество» - их количество в группе. Если в очередной группе число разных символов меньше 3, то отсылается соответствующее сообщение клиенту и сервер разрывает соединение.

Код:

СЕРВЕР:

```
#define _WINSOCK_DEPRECATED_NO_WARNINGS
#include <iostream>
#include <cstdio>
#include <string>
#include <fstream>
#include <winsock2.h>
#include <vector>
#include <map>
#include <string>

#pragma comment(lib, "WS2_32.lib")
using namespace std;
DWORD WINAPI serverReceive(LPVOID lpParam) { //Получение данных от клиента
	char buffer[1024] = { 0 }; //Буфер для данных
	SOCKET client = *(SOCKET*)lpParam; //Сокет для клиента
	while (true) { //Цикл работы сервера
		if (recv(client, buffer, sizeof(buffer), 0) == SOCKET_ERROR) {
			//Если не удалось получить данные буфера, сообщить об ошибке и выйти
			cout << "Reciving function failed with error " << WSAGetLastError() << endl;
			return -1;
		}

		map<char, int> mapMessage;
		for (int i = 0; i < strlen(buffer); i++)
		{
			if (int(buffer[i]) != 10)
	//Условие на проверку паразитного символа
			{
				auto mapIterator = mapMessage.find(buffer[i]);			//Формируем нашу
цепочку символов
				if (mapIterator == mapMessage.end())
				{
					mapMessage.insert(make_pair(buffer[i], 1));
				}
				else
				{
					mapIterator->second += 1;
				}
			}
		}

		if (mapMessage.size() <= 3) {
			cout << "Thank you for using the application" << endl;
			break;
		}
		int sum=0;
		for (auto x : mapMessage) { sum += x.second; }
```

```
                float oloNg = float(mapMessage.size())/float(sum);
                cout << "Sum " << sum << ",  N0 " << oloNg << endl;
                string outLine;
                char d[20];                                                     //Конверт символов
                sprintf_s(d, "%d", oloNg);
                int j = 0;
                outLine += "N0=CountUniqueSumbols/CountAllSymbols = ";
                outLine += to_string(oloNg);
                outLine += ", ";
                for (auto x : mapMessage)
                {
                        cout << x.first << " - " << x.second << endl;
                        char s[20];                                                 //Конверт
символов
                        sprintf_s(s, "%d", x.second);
                        outLine += "<<";
                        outLine += x.first;
                        outLine += '-';
                        outLine += s;
                        outLine += ">>";
                        if (j != mapMessage.size() - 1) outLine += ", ";
                        j++;
                }
                for (int i = 0; i < sum; i++)
                {
                        float N = oloNg * (1 - oloNg);
                        outLine += "\nN[" + to_string(i + 1) + "] = " + to_string(N);
                        oloNg = N;
                }
                send(client, outLine.c_str(), outLine.length(), 0);             //отправка сообщения на клиент
                mapMessage.clear();

                cout << "Client: " << buffer;
                memset(buffer, 0, sizeof(buffer)); //Очистить буфер
        }
        return 1;
}
DWORD WINAPI serverSend(LPVOID lpParam) { //Отправка данных клиенту
        char buffer[1024] = { 0 };
        SOCKET client = *(SOCKET*)lpParam;
        while (true) {
                fgets(buffer, 1024, stdin);

                if (send(client, buffer, sizeof(buffer), 0) == SOCKET_ERROR) {
                        cout << "Sending failed with error " << WSAGetLastError() << endl;
                        return -1;
                }
                if (strcmp(buffer, "exit\n") == 0) {
                        cout << "End work application!" << endl;
                        break;
                }
        }
        return 1;
}
int main() {
        setlocale(LC_ALL, "EN");
        WSADATA WSAData; //Данные
        SOCKET server, client; //Сокеты сервера и клиента
        SOCKADDR_IN serverAddr, clientAddr; //Адреса сокетов
        WSAStartup(MAKEWORD(2, 0), &WSAData);
        server = socket(AF_INET, SOCK_STREAM, 0); //Создали сервер
        if (server == INVALID_SOCKET) {
                cout << "Socket creation failed with error:" << WSAGetLastError() << endl;
                return -1;
        }
        serverAddr.sin_addr.s_addr = INADDR_ANY;
```

```
	serverAddr.sin_family = AF_INET;
	serverAddr.sin_port = htons(5555);
	if (bind(server, (SOCKADDR*)&serverAddr, sizeof(serverAddr)) == SOCKET_ERROR) {
		cout << "Bind function failed with error: " << WSAGetLastError() << endl;
		return -1;
	}
	if (listen(server, 0) == SOCKET_ERROR) { //Если не удалось получить запрос
		cout << "Listen function failed with error:" << WSAGetLastError() << endl;
		return -1;
	}
	cout << "Listening for incoming connections...." << endl;
	char buffer[1024]; //Создать буфер для данных
	int clientAddrSize = sizeof(clientAddr); //Инициализировать адерс клиента
	if ((client = accept(server, (SOCKADDR*)&clientAddr, &clientAddrSize)) !=
		INVALID_SOCKET) {
		//Если соединение установлено
		cout << "Client connected!" << endl;
		cout << " Enter diffirent char less than 3 to disconnect!" << endl;
		DWORD tid; //Идентификатор
		HANDLE t1 = CreateThread(NULL, 0, serverReceive, &client, 0, &tid); //Создание потока для получения
данных
		if (t1 == NULL) { //Ошибка создания потока
			cout << "Thread Creation Error: " << WSAGetLastError() << endl;
		}
		HANDLE t2 = CreateThread(NULL, 0, serverSend, &client, 0, &tid); //Создание потока для отправки
данных
		if (t2 == NULL) {
			cout << "Thread Creation Error: " << WSAGetLastError() << endl;
		}
		WaitForSingleObject(t1, INFINITE);
		WaitForSingleObject(t2, INFINITE);
		closesocket(client); //Закрыть сокет
		if (closesocket(server) == SOCKET_ERROR) { //Ошибка закрытия сокета
			cout << "Close socket failed with error: " << WSAGetLastError() << endl;
			return -1;
		}
		WSACleanup();
	}
}
```

КЛИЕНТ:

```
#define _WINSOCK_DEPRECATED_NO_WARNINGS
#include <iostream>
#include <cstdio>
#include <cstring>
#include <winsock2.h>
#include <vector>
#include <map>

#pragma comment(lib, "WS2_32.lib")
using namespace std;

DWORD WINAPI clientReceive(LPVOID lpParam) { //Получение данных от сервера
	char buffer[10240] = { 0 };
	SOCKET server = *(SOCKET*)lpParam;
	while (true) {
		if (recv(server, buffer, sizeof(buffer), 0) == SOCKET_ERROR) {
			cout << "Reciving function failed with error: " << WSAGetLastError() <<
				endl;
			return -1;
		}
		map<char, int> mapMessage;
		for (int i = 0; i < strlen(buffer); i++)
		{
```

```
                        if (int(buffer[i]) != 10)
        //Условие на проверку паразитного символа
                        {
                                auto mapIterator = mapMessage.find(buffer[i]);                  //Формируем нашу
цепочку символов
                                if (mapIterator == mapMessage.end())
                                {
                                        mapMessage.insert(make_pair(buffer[i], 1));
                                }
                                else
                                {
                                        mapIterator->second += 1;
                                }
                        }
                }
                if (mapMessage.size() <= 3) {
                        cout << "Thank you for using the application" << endl;
                        return 1;
                }

                cout << "Server: " << buffer << endl;
                memset(buffer, 0, sizeof(buffer));
        }
        return 1;
}
DWORD WINAPI clientSend(LPVOID lpParam) { //Отправка данных на сервер
        char buffer[1024] = { 0 };
        SOCKET server = *(SOCKET*)lpParam;
        while (true) {
                fgets(buffer, 1024, stdin);

                map<char, int> mapMessage;
                for (int i = 0; i < strlen(buffer); i++)
                {
                        if (int(buffer[i]) != 10)
        //Условие на проверку паразитного символа
                        {
                                auto mapIterator = mapMessage.find(buffer[i]);                  //Формируем нашу
цепочку символов
                                if (mapIterator == mapMessage.end())
                                {
                                        mapMessage.insert(make_pair(buffer[i], 1));
                                }
                                else
                                {
                                        mapIterator->second += 1;
                                }
                        }
                }
                if (send(server, buffer, sizeof(buffer), 0) == SOCKET_ERROR) {
                        cout << "Sending failed with error: " << WSAGetLastError() << endl;
                        return -1;
                }
                if (mapMessage.size()<=3) {
                        cout << "Thank you for using the application" << endl;
                        break;
                }

        }
        return 1;
}
int main() {
        setlocale(LC_ALL, "EN");
        WSADATA WSAData;
        SOCKET server;
        SOCKADDR_IN addr;
```

```
    WSAStartup(MAKEWORD(2, 0), &WSAData);
    if ((server = socket(AF_INET, SOCK_STREAM, 0)) == INVALID_SOCKET) {
            cout << "Socket creation failed with error: " << WSAGetLastError() << endl;
            return -1;
    }
    addr.sin_addr.s_addr = inet_addr("127.0.0.1"); //коннект к серверу
    addr.sin_family = AF_INET;
    addr.sin_port = htons(5555); //порт
    if (connect(server, (SOCKADDR*)&addr, sizeof(addr)) == SOCKET_ERROR) {
            cout << "Server connection failed with error: " << WSAGetLastError() << endl;
            return -1;
    }
    cout << "Connected to server!" << endl;
    cout << " Enter diffirent char less than 3 to disconnect!" << endl;
    DWORD tid;
    HANDLE t1 = CreateThread(NULL, 0, clientReceive, &server, 0, &tid);
    if (t1 == NULL) cout << "Thread creation error: " << GetLastError();
    HANDLE t2 = CreateThread(NULL, 0, clientSend, &server, 0, &tid);
    if (t2 == NULL) cout << "Thread creation error: " << GetLastError();
    WaitForSingleObject(t1, INFINITE);
    WaitForSingleObject(t2, INFINITE);
    closesocket(server);
    WSACleanup();
}
```

```
Listening for incoming connections....
Client connected!
 Enter diffirent char less than 3 to disconnect!
e - 4
q - 4
r - 4
t - 4
w - 4
y - 4
Client: qqqqwwwweeeerrrrttttyyyy
Thank you for using the application
exit
End work application!

C:\Users\User_Admin\source\repos\Lab4Server\x64\Debug\Lab4Server.exe (process 12536) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

```
Connected to server!
 Enter diffirent char less than 3 to disconnect!
qqqqwwwweeeerrrrttttyyyy
Server: <<e-4>>, <<q-4>>, <<r-4>>, <<t-4>>, <<w-4>>, <<y-4>>
qwe
Thank you for using the application
Server: exit

Thank you for using the application

C:\Users\User_Admin\source\repos\Lab4Client\x64\Debug\Lab4Client.exe (process 1012) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

Вывод: 1) изучили основы программирования сетевых приложений Windows на базе библиотеки WINSOCK2.H; 2) приобрели навыки по практическому использованию библиотеки для реализации сетевых приложений в среде C++ на базе протоколов TCP и UDP.