МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра информационных технологий**

**Отчёт по лабораторной работе №6**

**«Многопоточное сетевое клиент-серверное приложение на основе TCP-сокета»**

Работу выполнил

студент 22 группы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Н.Е. Лучин

Направление подготовки 01.03.02 Прикладная математика и информатика

Проверил

канд. техн. наук, доц.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_А.А. Полупанов

Краснодар 2022

**Цель работы:** реализовать многопоточное сетевое клиент-серверное приложение на основе TCP-сокета, позволяющее передавать данные произвольных типов (структуру) и работающее с клиентами параллельно.

**Задача:** приложение состоит из двух программ: клиент и сервер, взаимодействие которых осуществляется с помощью TCP-сокета. Алгоритм взаимодействия следующий: клиент посылает данные через сокет, сервер в бесконечном цикле слушает свой входной порт и при появлении сообщений от нового клиента выделяет для него отдельный поток, занимающийся обработкой сообщения клиента и отправкой результата обработки клиенту обратно, пока основной поток продолжает слушать сообщения от новых клиентов.

**Практическая часть:**

Листинг клиента:

#include <iostream>

// для работы с сокетами

#include <winsock2.h>

// для работы с Windows API

#include <windows.h>

// для доп. функций (напр-р, inet\_pton)

#include <WS2tcpip.h>

/\*

линкер скомпонует библиотеку в исполняемый файл

.exe =

объектные файлы (smth.cpp -> smth.o)

+

библиотеки

\*/

#pragma comment (lib, "Ws2\_32.lib")

using namespace std;

#define SRV\_PORT 1234

/\*

объявление структуры с коэффициентами

для квадратного трехчлена

\*/

struct Coefficients

{

    int a, b, c;

} coeffs;

/\*

объявление структуры с ответом

от сервера

\*/

struct Equation {

    int roots;

    double r1, r2;

} eq;

int main()

{

    /\*

    инициализация Winsock:

    0x0202 - шестнадцатеричная версия интерфейса 2.2

    можно аналогично написать MAKEWORD(2, 2),

    что мы и сделали

    второй аргумент - структура WSADATA wsData

    в нее будут записаны сведения

    о конкретной реализации интерфейса Winsock

    \*/

    WSADATA wsData;

    // код результата. не ноль в случае ошибки

    int resCode = WSAStartup(MAKEWORD(2, 2), &wsData);

    if (resCode != 0)

    {

        // выведем ошибку

        cout << "Error WinSock initializaion #"

<< WSAGetLastError();

        return -1;

    }

    else

        cout << "WinSock initialization OK" << endl;

    /\*

    создание сокета для протокола IPv4 (AF\_INET)

    с режимом потокового соединения (SOCK\_STREAM)

    IPPROTO\_TCP - выбор протокола TCP для нашего соединения

    \*/

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

    if (clntSock == INVALID\_SOCKET)

    {

        cout << "Error initialization socket #"

<< WSAGetLastError() << endl;

        closesocket(clntSock);

        // освободить выделенную память для Winsock

        WSACleanup();

        return -1;

    }

    else

        cout << "Client socket initialization OK"

<< endl << endl;

    // структура с информацией о клиенте

    sockaddr\_in clntInfo;

    // опустошение памяти (на всякий случай)

    ZeroMemory(&clntInfo, sizeof(clntInfo));

    // указываем версию IP IPv4

    clntInfo.sin\_family = AF\_INET;

    clntInfo.sin\_addr.s\_addr = 0;

    clntInfo.sin\_port = 0;

    resCode = bind(clntSock, (sockaddr\*) &clntInfo, sizeof(clntInfo));

    if (resCode != 0)

    {

        cout << "Error Socket binding. Error # "

<< WSAGetLastError() << endl;

        closesocket(clntSock);

        WSACleanup();

        return -1;

    }

    else

        cout << "Binding socket to client info is OK"

<< endl;

    in\_addr ip\_to\_num;

    resCode = inet\_pton(AF\_INET, "127.0.0.1", &ip\_to\_num);

    if (resCode <= 0) {

        cout << "Error in IP translation to special numeric format" << endl;

        return -1;

    }

    sockaddr\_in srvInfo;

    ZeroMemory(&srvInfo, sizeof(srvInfo));

    srvInfo.sin\_family = AF\_INET;

    srvInfo.sin\_addr = ip\_to\_num;

    srvInfo.sin\_port = htons(SRV\_PORT);

    resCode = connect(clntSock, (sockaddr\*)&srvInfo, sizeof(srvInfo));

    if (resCode != 0) {

        cout << "Connection to Server FAILED. Error # " << WSAGetLastError() << endl;

        closesocket(clntSock);

        WSACleanup();

        return -1;

    }

    else

        cout << "Connection established SUCCESSFULLY. Ready to send a message to Server"

        << endl;

    // размер отправляемого и получаемого пакета

    short packet\_size = 0;

    while (true)

    {

        cout << "Input coeffs: a b c"

<< " (or enter all zeros to exit)" << endl;

        cin >> coeffs.a >> coeffs.b >> coeffs.c;

        /\*

        передаем сокет,

        преобразовываем указатель на структуру

        в указатель на char (первая ячейка памяти, занимаемая структурой)

        передаем размер памяти структуры

        0 - стандартный флаг

        передаем указатель на структуру

        srvInfo (инф-я о том, куда мы отправляем)

        и ее размер

        \*/

        packet\_size = send(clntSock, (char\*) &coeffs, sizeof(coeffs), 0);

        if (packet\_size == SOCKET\_ERROR)

        {

            cout << "Faild sending message. Error #" << WSAGetLastError() << endl;

            closesocket(clntSock);

            // освободить выделенную память для Winsock

            WSACleanup();

            return -1;

        }

        // аналогично send, но получаем

        packet\_size = recv(clntSock, (char\*) &eq, sizeof(eq), 0);

        // интерпретация ответа от сервера

        if (eq.roots == -1)

            break;

        else if (eq.roots == 0)

            cout << "There're no roots in real nums"

<< endl;

        else if (eq.roots == 1)

        {

            cout << "1 root: " << eq.r1 << endl;

        }

        else

        {

            cout << "2 roots: " << eq.r1 << " " << eq.r2 << endl;

        }

        cout << endl;

    }

    cout << "exit" << endl;

    closesocket(clntSock);

    WSACleanup();

    return 0;

}

Листинг сервера:

#include <iostream>

#include <winsock2.h>

#include <windows.h>

#include <WS2tcpip.h>

#include <cmath>

#pragma comment (lib, "Ws2\_32.lib")

using namespace std;

#define SRV\_PORT 1234

#define PRINTUSERS if (nclients)\

cout << "User online " << nclients << endl;\

else cout << "No user online\n";

DWORD WINAPI ConToClient(LPVOID client\_socket);

int nclients = 0;

struct Coefficients

{

    int a, b, c;

};

struct Equation {

    int roots;

    double r1, r2;

};

int main() {

    WSADATA wsData;

    int resCode = WSAStartup(MAKEWORD(2, 2), &wsData);

    if (resCode != 0)

    {

        cout << "Error WinSock initializaion #"

<< WSAGetLastError();

        return -1;

    }

    else

        cout << "WinSock initialization OK" << endl;

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

    if (servSock == INVALID\_SOCKET)

    {

        cout << "Error initialization socket #"

<< WSAGetLastError() << endl;

        closesocket(servSock);

        WSACleanup();

        return -1;

    }

    else

        cout << "Server socket initialization OK"

<< endl;

    // информация о сервере

    sockaddr\_in servInfo;

    ZeroMemory(&servInfo, sizeof(servInfo));

    servInfo.sin\_family = AF\_INET;

    // готовы принимать сообщения от любого адреса (INADDR\_ANY)

    servInfo.sin\_addr.s\_addr = 0;

    servInfo.sin\_port = htons(SRV\_PORT);

    // привязываем заполненную структуру к сокету

    resCode = bind(servSock, (sockaddr\*) &servInfo, sizeof(servInfo));

    if (resCode != 0)

    {

        cout << "Error Socket binding. Error #"

<< WSAGetLastError() << endl;

        closesocket(servSock);

        WSACleanup();

        return -1;

    }

    else

        cout << "Binding socket to Server info OK"

<< endl << endl;

    // 0x100 (16-ая сист. счисл.) = 256 (10)

    resCode = listen(servSock, 0x100);

    if (resCode != 0) {

        cout << "Can't start to listen to. Error # "

<< WSAGetLastError() << endl;

        closesocket(servSock);

        WSACleanup();

        return -1;

    }

    else

    {

        cout << "Listening..." << endl;

    }

    SOCKET clientConn;

    // информация о клиенте

    sockaddr\_in clientInfo;

    ZeroMemory(&clientInfo, sizeof(clientInfo));

    int clientInfo\_size = sizeof(clientInfo);

    short packet\_size = 0;

    while ((clientConn = accept(servSock, (sockaddr\*)&clientInfo, &clientInfo\_size)))

    {

        if (clientConn == INVALID\_SOCKET) {

            cout << "Client detected, but can't connect to a client. Error # " << WSAGetLastError() << endl;

            closesocket(servSock);

            closesocket(clientConn);

            WSACleanup();

            return 1;

        }

        else

            cout << "Connection to a client established successfully" << endl;

        nclients++;

        cout << "new connection!\n";

        PRINTUSERS

        cout << endl;

        DWORD thID;

        CreateThread(NULL, NULL, ConToClient, &clientConn, NULL, &thID);

    }

    return 0;

}

DWORD WINAPI ConToClient(LPVOID clientConn)

{

    SOCKET sock;

    sock = ((SOCKET\*) clientConn)[0];

    short packet\_size = 0;

    Coefficients coeffs;

    Equation eq;

    while (true)

    {

        // получаем информацию о клиенте (clientInfo) и его сообшение

        packet\_size = recv(sock, (char\*) &coeffs, sizeof(coeffs), 0);

        // прерывание работы при нулевых коэффициентах

        if (!coeffs.a && !coeffs.b && !coeffs.c)

            break;

        cout << "Coeffs: " << coeffs.a << ", "

<< coeffs.b << ", " << coeffs.c << endl;

        // считаем дискриминант

        double D = coeffs.b \* coeffs.b - 4 \* coeffs.a \* coeffs.c;

        // просчет вещ-ых корней (если они есть)

        if (D > 0)

        {

            eq.roots = 2;

            eq.r1 = (-coeffs.b + sqrt(D)) / (2 \* coeffs.a);

            eq.r2 = (-coeffs.b - sqrt(D)) / (2 \* coeffs.a);

            cout << "Roots: " << eq.r1 << ", "

<< eq.r2 << endl << endl;

        }

        else if (D == 0)

        {

            eq.roots = 1;

            eq.r1 = -coeffs.b / (2 \* coeffs.a);

            cout << "Root: " << eq.r1 << endl << endl;

        }

        else

        {

            eq.roots = 0;

            cout << "No roots" << endl << endl;

        }

        packet\_size = send(sock, (char\*) &eq, sizeof(eq), 0);

        if (packet\_size == SOCKET\_ERROR)

        {

            cout << "Can't send message. Error #"

<< WSAGetLastError() << endl;

            closesocket(sock);

            WSACleanup();

            return -1;

        }

    }

    // флаг, указывающий на прерывание работы

    eq.roots = -1;

    packet\_size = send(sock, (char\*) &eq, sizeof(eq), 0);

    if (packet\_size == SOCKET\_ERROR) {

        cout << "Can't send message. Error # "

<< WSAGetLastError() << endl;

        closesocket(sock);

        WSACleanup();

        return 1;

    }

    cout << "The client has disconnected" << endl;

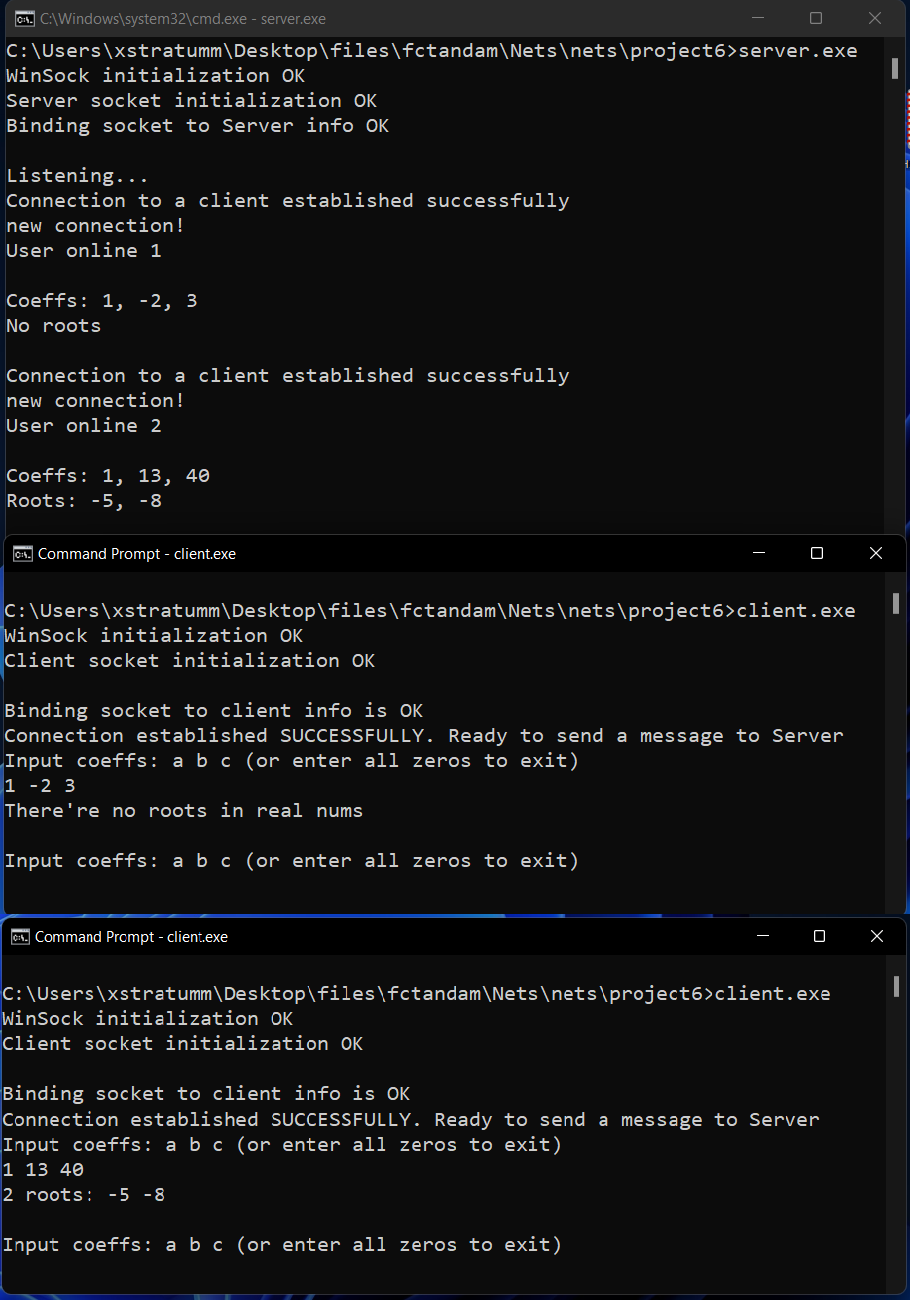
    closesocket(sock);

    nclients--;

    PRINTUSERS

}

Пример работы программ:



*Рисунок 1 – пример работы клиентов с сервером*

**Вывод:**былоразработано многопоточное клиент-серверное приложение на основе TCP-сокетов.