**Задание:**

1) Написать клиент-серверное приложение

2) Используя HTTP- клиент, обратиться методом GET к веб-серверу json.org

3) Обратиться к HTTP-серверу через стандартный браузер

4)С помощью telnet отправить HTTP-запрос к веб-серверу

**Ответы:**

1)Была написана программа, с помощью которой клиент передает серверу структуру, содержащую поля: Фамилия, температура и давление. Сервер выполняет вычисления, после чего создает тело ответа и отправляет его клиенту.(см. Проект 3-Клиент и Проект 3-Сервер).

Код сервера проекта:

#include <iostream>

#include <sstream>

#include <string>

// #define \_WIN32\_WINNT 0x501

#include <WinSock2.h>

#include <WS2tcpip.h>

#define BUF\_SIZE 64

#pragma comment(lib, "Ws2\_32.lib")

using std::cerr;

 struct Person

{

    char name[25];

    double temp;

    int davl;

} A;

int main() {

    SetConsoleCP(1251);

    SetConsoleOutputCP(1251);

    WSADATA wsaData;

    int result = WSAStartup(MAKEWORD(2, 2), &wsaData);

    if (result != 0)

    {

        cerr << "WSAStartup failed: " << result << "\n";    return result;

    }

    struct addrinfo\* addr = NULL;

    // структура для IP-адреса  сервера

    struct addrinfo hints;

    ZeroMemory(&hints, sizeof(hints));

    hints.ai\_family = AF\_INET;

    hints.ai\_socktype = SOCK\_STREAM;

    hints.ai\_protocol = IPPROTO\_TCP;

    hints.ai\_flags = AI\_PASSIVE;

    // Инициализируем структуру addr

    // HTTP-сервер на 8000-м порту localhost

    result = getaddrinfo("127.0.0.1", "8000", &hints, &addr);

    if(result != 0) {

        cerr << "getaddrinfo failed: " << result << "\n";

        WSACleanup(); // выгрузка библиотеки Ws2\_32.dll

        return 1;

    }

    // Создание сокета

    int listen\_socket =  socket(addr->ai\_family, addr->ai\_socktype,  addr->ai\_protocol);

    if (listen\_socket == INVALID\_SOCKET) {

        cerr << "Error at socket: " << WSAGetLastError() << "\n";

        freeaddrinfo(addr);

        WSACleanup();

        return 1;

    }

    // Привязываем сокет к IP-адресу

    result = bind(listen\_socket, addr->ai\_addr, (int)addr->ai\_addrlen);

    if (result == SOCKET\_ERROR) {

        cerr << "bind failed with error: " << WSAGetLastError() << "\n";

        freeaddrinfo(addr);

        closesocket(listen\_socket);

        WSACleanup();

        return 1;

    }

    // Инициализируем слушающий сокет

    if (listen(listen\_socket, SOMAXCONN) == SOCKET\_ERROR)

    {

        cerr << "listen failed with error: " << WSAGetLastError() << "\n";

        closesocket(listen\_socket);

        WSACleanup();

        return 1;

    }

    const int max\_client\_buffer\_size = 1024;

    char buf[max\_client\_buffer\_size];

    int client\_socket = INVALID\_SOCKET;

    for (;;) // Принимаем входящие соединения

    {

        client\_socket = accept(listen\_socket, NULL, NULL);//принятие запроса на установку

        if (client\_socket == INVALID\_SOCKET)

        {

            cerr << "accept failed: " << WSAGetLastError() << "\n";

            closesocket(listen\_socket);

            WSACleanup();

            return 1;

        }

        result = recv(client\_socket, buf, max\_client\_buffer\_size, 0);//получение от клиента

        std::stringstream response; // сюда будет записываться ответ клиенту

        std::stringstream response\_body; // тело ответа

        if (result == SOCKET\_ERROR)

        {   // ошибка получения данных

            cerr << "recv failed: " << result << "\n";

            closesocket(client\_socket);

        }

        else if (result == 0)

        {

            // соединение закрыто клиентом

            cerr << "connection closed...\n";

        }

        else if (result > 0)

        {

            result = recv(client\_socket, (char\*)&A, BUF\_SIZE, 0);

            buf[result] = '\0';   // Данные успешно получены

       // формируем тело ответа (HTML)

            if (37.0 <= A.temp || A.davl > 90)

            {

                response\_body << "<title>Проверка здоровья</title>\n"

                    << "<h2>Данные пациента</h2>\n"

                    << "<pre>" << buf << "</pre>\n"

                    << "<p>Имя "<<A.name <<"< / p>\n"

                    << "<p>Температура " << A.temp <<" и давление "<<A.davl<<"< / p>\n"

                    << "<h2>Результат</h2>\n"

                    << "<em><small>Вы больны</small></em>\n";

                // Формируем весь ответ вместе с заголовками

                response << "HTTP/1.1 200 OK\r\n"

                    << "Version: HTTP/1.1\r\n"

                    << "Content-Type: text/html; charset=utf-8\r\n"

                    << "Content-Length: " << response\_body.str().length()

                    << "\r\n\r\n" << response\_body.str();

            }

            else {

                response\_body << "<title>Проверка здоровья</title>\n"

                    << "<h2>Данные пациента</h2>\n"

                    << "<p>Имя " << A.name << "< / p>\n"

                    << "<p>Температура " << A.temp << " и давление " << A.davl << "< / p>\n"

                    << "<h2>Результат</h2>\n"

                    << "<em><small>Вы здоровы</small></em>\n";

                // Формируем весь ответ вместе с заголовками

                response << "HTTP/1.1 200 OK\r\n"

                    << "Version: HTTP/1.1\r\n"

                    << "Content-Type: text/html; charset=utf-8\r\n"

                    << "Content-Length: " << response\_body.str().length()

                    << "\r\n\r\n" << response\_body.str();

            }

            // Отправляем ответ клиенту

            result =  send(client\_socket, response.str().c\_str(), response.str().length(), 0);

            if (result == SOCKET\_ERROR)

            {    // произошла ошибка при отправке данных

                cerr << "send failed:"<< WSAGetLastError() <<"\n";

            }

            // Закрываем соединение к клиентом

            closesocket(client\_socket);

        }

    } // конец for(;;)

     // Убираем за собой

    closesocket(listen\_socket);

    freeaddrinfo(addr);

    WSACleanup();

    return 0;

}

Код клиента проекта:

#include <string>

#define \_WINSOCK\_DEPRECATED\_NO\_WARNINGS

// подавление предупреждений библиотеки winsock2

#include <winsock2.h>

#include <iostream>

#pragma comment (lib,"Ws2\_32.lib")

#pragma warning(disable: 4996)  // подавление предупреждения 4996

using namespace std;

#define request "Get/ http/1.1\r\n host: localhost \r\n\r\n" //html запрос.

#define max\_packet\_size   65535

struct Person

{

    char name[25];

    double temp;

    int davl;

}B;

int main() {

    SetConsoleCP(1251);

    SetConsoleOutputCP(1251);

    WSADATA  ws;

    SOCKET   s;

    sockaddr\_in adr;

    HOSTENT\* hn;

    char    buff[max\_packet\_size];

    if (WSAStartup(0x0202, &ws) != 0) { return -1; }    // error

    // создаём сокет

    if (INVALID\_SOCKET == (s = socket(AF\_INET, SOCK\_STREAM, 0)))

    {

        return -1;

    }

// получаем адрес

    if (NULL == (hn = gethostbyname("localhost")))

    {

        return -1;

    }   // error

// заполняем  поля структуры adr для использование ее в connect

    adr.sin\_family = AF\_INET;

    ((unsigned long\*)&adr.sin\_addr)[0] =

        ((unsigned long\*\*)hn->h\_addr\_list)[0][0];

    adr.sin\_port = htons(8000);

    // устанавливаем соединение с сервером

    if (SOCKET\_ERROR == connect(s, (sockaddr\*)&adr, sizeof(adr)))

    {

        int res = WSAGetLastError(); return -1;

    } //error

// посылаем запрос серверу

    if (SOCKET\_ERROR == send(s, (char\*)&request, sizeof(request), 0))

    {

        int res = WSAGetLastError(); return -1;

    }   // error

// ждём ответа

    int len = 0;

    cout << "Введите запрос: Фамилия, температура, давление" << endl;

    cin >> B.name >> B.temp >> B.davl;

    getchar();

    send(s, (char\*)&B, sizeof(B), 0);

    do

    {

        if (SOCKET\_ERROR == (len = recv(s, (char\*)&buff, max\_packet\_size, 0)))

        {

            int res = WSAGetLastError();     return -1;

        }

        for (int i = 0; i < len; i++)

            cout << buff[i];

    } while (len != 0); //получаем данные по частям, пока не len != 0.

    if (SOCKET\_ERROR == closesocket(s))   // закрываем соединение

    {

        return -1;

    }  // error

    cin.get();

    return 1;

}

**Вывод консоли для здорового клиента:**

Изображение выглядит как текст

Автоматически созданное описание

**Вывод консоли для больного клиента:**

Изображение выглядит как текст

Автоматически созданное описание

2)Обращаемся с помощью метода GET к веб-серверу json.org. (см. Проект 3- Подключение и Проект 3-сервер обычный).

Код HTTP-клиента:

#include "winsock2.h"

#include <iostream>

#define \_WINSOCK\_DEPRECATED\_NO\_WARNINGS

#pragma warning(disable: 4996)

#pragma comment (lib, "Ws2\_32.lib")

#define request "GET / HTTP/1.0 \r\nHost: json.org\r\n\r\n"

#define MAX\_PACKET\_SIZE 4096

using namespace std;

int main() {

    WSADATA ws;

    SOCKET s;   sockaddr\_in adr;   hostent\* hn;

    char buff[MAX\_PACKET\_SIZE]; // Init

    if (WSAStartup(0x0101, &ws) != 0) { return -1; } // Error

    if (INVALID\_SOCKET == (s = socket(AF\_INET, SOCK\_STREAM, 0)))

        // Создаём сокет

    {

        return -1;

    } // Error

          // Получаем адрес

    if (NULL == (hn = gethostbyname("json.org")))

    {

        return -1;

    }  // Error

 // Заполняем структуру с адресом

    adr.sin\_family = AF\_INET;

    // adr.sin\_addr.s\_addr = \*(DWORD\* ) hn->h\_addr\_list[0];

    ((unsigned long\*)&adr.sin\_addr)[0] = ((unsigned long\*\*)hn->h\_addr\_list)[0][0];

    adr.sin\_port = htons(80);

    // Устанавливаем соединение с сервером

    if (SOCKET\_ERROR == connect(s, (sockaddr\*)&adr, sizeof(adr)))

    {

        return -1;

    }

// Посылаем запрос

    if (SOCKET\_ERROR == send(s, (char \*)&request, sizeof(request), 0))

    {

        return -1;

    } // Error

    int len = 0; // ждём ответа

    do {

        if (SOCKET\_ERROR ==

            (len = recv(s, (char\*)&buff, MAX\_PACKET\_SIZE, 0)))

        {

            int res = WSAGetLastError(); return -1;

        }

        for (int i = 0; i < len; i++)

            std::cout << buff[i];

    } while (len != 0);

    //получаем данные по частям, пока len не 0

    // закрываем соединение

    if (SOCKET\_ERROR == closesocket(s))

    {

        return -1;

    }  // error

    cin.get();

    return 1;

}

Код HTTP-сервера:

#include <iostream>

#include <sstream>

#include <string>

// #define \_WIN32\_WINNT 0x501

#include <WinSock2.h>

#include <WS2tcpip.h>

#pragma comment(lib, "Ws2\_32.lib")

using std::cerr;

int main() {

    WSADATA wsaData;

    int result = WSAStartup(MAKEWORD(2, 2), &wsaData);

    if (result != 0)

    {

        cerr << "WSAStartup failed: " << result << "\n";    return result;

    }

    struct addrinfo\* addr = NULL;

    // структура для IP-адреса  сервера

    struct addrinfo hints;

    ZeroMemory(&hints, sizeof(hints));

    hints.ai\_family = AF\_INET;

    hints.ai\_socktype = SOCK\_STREAM;

    hints.ai\_protocol = IPPROTO\_TCP;

    hints.ai\_flags = AI\_PASSIVE;

    // Инициализируем структуру addr

    // HTTP-сервер на 8000-м порту localhost

    result = getaddrinfo("127.0.0.1", "8000", &hints, &addr);

    if(result != 0) {

        cerr << "getaddrinfo failed: " << result << "\n";

        WSACleanup(); // выгрузка библиотеки Ws2\_32.dll

        return 1;

    }

    // Создание сокета

    int listen\_socket =

        socket(addr->ai\_family, addr->ai\_socktype,

            addr->ai\_protocol);

    if (listen\_socket == INVALID\_SOCKET) {

        cerr << "Error at socket: " << WSAGetLastError() << "\n";

        freeaddrinfo(addr);

        WSACleanup();

        return 1;

    }

    // Привязываем сокет к IP-адресу

    result = bind(listen\_socket, addr->ai\_addr, (int)addr->ai\_addrlen);

    if (result == SOCKET\_ERROR) {

        cerr << "bind failed with error: " << WSAGetLastError() << "\n";

        freeaddrinfo(addr);

        closesocket(listen\_socket);

        WSACleanup();

        return 1;

    }

    // Инициализируем слушающий сокет

    if (listen(listen\_socket, SOMAXCONN) == SOCKET\_ERROR)

    {

        cerr << "listen failed with error: " << WSAGetLastError() << "\n";

        closesocket(listen\_socket);

        WSACleanup();

        return 1;

    }

    const int max\_client\_buffer\_size = 1024;

    char buf[max\_client\_buffer\_size];

    int client\_socket = INVALID\_SOCKET;

    for (;;) // Принимаем входящие соединения

    {

        client\_socket = accept(listen\_socket, NULL, NULL);

        if (client\_socket == INVALID\_SOCKET)

        {

            cerr << "accept failed: " << WSAGetLastError() << "\n";

            closesocket(listen\_socket);

            WSACleanup();

            return 1;

        }

        result =

            recv(client\_socket, buf, max\_client\_buffer\_size, 0);

        std::stringstream response;

        // сюда будет записываться ответ клиенту

        std::stringstream response\_body; // тело ответа

        if (result == SOCKET\_ERROR)

        {

            // ошибка получения данных

            cerr << "recv failed: " << result << "\n";

            closesocket(client\_socket);

        }

        else if (result == 0)

        {

            // соединение закрыто клиентом

            cerr << "connection closed...\n";

        }

        else if (result > 0)

        {

            buf[result] = '\0';   // Данные успешно получены

       // формируем тело ответа (HTML)

            response\_body << "<title>Test C++ HTTP Server</title>\n"

                << "<h1>Test page</h1>\n"

                << "<p>This is body of the test page...</p>\n"

                << "<h2>Request headers</h2>\n"

                << "<pre>" << buf << "</pre>\n"

                << "<em><small>Test C++ Http Server</small></em>\n";

            // Формируем весь ответ вместе с заголовками

            response << "HTTP/1.1 200 OK\r\n"

                << "Version: HTTP/1.1\r\n"

                << "Content-Type: text/html; charset=utf-8\r\n"

                << "Content-Length: " << response\_body.str().length()

                << "\r\n\r\n"

                << response\_body.str();

            // Отправляем ответ клиенту

            result =

                send(client\_socket, response.str().c\_str(),

                    response.str().length(), 0);

            if (result == SOCKET\_ERROR)

            {    // произошла ошибка при отправке данных

                cerr << "send failed:" << WSAGetLastError() <<"\n" ;

            }

            // Закрываем соединение к клиентом

            closesocket(client\_socket);

        }

    } // конец for(;;)

     // Убираем за собой

    closesocket(listen\_socket);

    freeaddrinfo(addr);

    WSACleanup();

    return 0;

}

**Вывод консоли:**

Изображение выглядит как текст

Автоматически созданное описание

3) Обращаемся к HTTP-серверу через стандартный браузер

Изображение выглядит как текст

Автоматически созданное описание

4) С помощью telnet отправляем HTTP-запрос к веб-серверу

Изображение выглядит как текст

Автоматически созданное описание

