МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра ИИТ

Отчёт
о лабораторной работе №5
по дисциплине «Веб-технологии»

Тема: «Клиент-сервер.»

Выполнил студент 2 курса
группы ПО-11 Сымоник И.А.

Проверил: Михняев А.Л.

Брест 2024

Цель работы: Изучить клиент-серверную архитектуру.

Ход работы

Задание 1.

Полученные числа разные, так как при обращении к 80 порту мы получили html страницу, а при обращении к 443 мы получаем 56 битный ключ.

Исходный код:

```
#include <winsock2.h>
#include <ws2tcpip.h>
#include <iostream>
#include <string>
#include <vector>

#include <openssl/err.h>
#include <openssl/ssl.h>

#pragma comment(lib, "Ws2_32.lib")

class Client
{
public:

    constexpr static auto MAX_BUFFER_SIZE = 4092;

    enum class Protocol
    {
        TELNET = 23,
        HTTP = 80,
        HTTPS = 443,
    };

    Client() = default;

    ~Client()
    {
        WSACleanup();
    }

    static bool Init()
    {
        WSADATA wsaData;
        if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0)
        {
            std::cerr << "WSAStartup failed: " << WSAGetLastError() << std::endl;
            return false;
        }

        SSL_library_init();

        return true;
    }

    void RefreshAddress()
    {
        serverAddress.sin_family = AF_INET;
        serverAddress.sin_port = htons(port);
        memcpy(&serverAddress.sin_addr.s_addr, hostIp->h_addr_list[0], hostIp->h_length);
    }

    int MakeConnection()
    {
        if (hostIp == nullptr)
```

```
    {
        std::cerr << "Host ip incorrect: " << WSAGetLastError() << std::endl;
        return WSAGetLastError();
    }

    client = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (client == INVALID_SOCKET)
    {
        std::cerr << "socket failed: " << WSAGetLastError() << std::endl;
        return WSAGetLastError();
    }

    if (serverAddress.sin_addr.S_un.S_addr == 0)
    {
        RefreshAddress();
    }

    if (connect(client, (SOCKADDR*)&serverAddress, sizeof(serverAddress)) == SOCKET_ERROR)
    {
        closesocket(client);
        return WSAGetLastError();
    }

    return WSAGetLastError();
}

int MakeHTTPSConnection()
{
    if (MakeConnection())
    {
        return WSAGetLastError();
    }

    ctx = SSL_CTX_new(SSLv23_client_method());
    if (ctx == nullptr) {
        std::cerr << "Error creating SSL context." << std::endl;
        HandleOpenSSLErrors();
        return WSAGetLastError();
    }

    ssl = SSL_new(ctx);
    SSL_set_fd(ssl, client);

    if (SSL_connect(ssl) != 1) {
        std::cerr << "Error establishing SSL connection." << std::endl;
        SSL_free(ssl);
        closesocket(client);
        return WSAGetLastError();
    }

    return WSAGetLastError();

}

void Send(std::string_view message) const
{
    if (send(client, message.data(), message.length(), 0) == SOCKET_ERROR)
    {
        closesocket(client);
    }


    char buffer[MAX_BUFFER_SIZE];
    int bytesRead;
    while ((bytesRead = recv(client, buffer, MAX_BUFFER_SIZE, 0)) > 0) {
        std::cout.write(buffer, bytesRead);
    }
    if (bytesRead < 0) {
        std::cerr << "Error receiving HTTP response." << std::endl;
    }
    else if (bytesRead == 0)
        std::cerr << "SERVER CLOSE." << std::endl;

    closesocket(client);
```

```
    }


    void SendHttps(std::string_view message) const
    {
        if (SSL_write(ssl, message.data(), static_cast<int>(message.length())) !=
static_cast<int>(message.length())) {
            std::cerr << "Error sending HTTPS request." << std::endl;
            SSL_free(ssl);
            closesocket(client);
            WSACleanup();
        }

        char buffer[MAX_BUFFER_SIZE];
        int bytesRead;
        while ((bytesRead = SSL_read(ssl, buffer, sizeof(buffer))) > 0) {
            std::cout.write(buffer, bytesRead);
        }
        if (bytesRead < 0) {
            std::cerr << "Error receiving HTTPS response." << std::endl;
        }
    }

    void Close()
    {
        closesocket(client);
        serverAddress = {};
    }

    bool SetHostIp(std::string_view hostName) {
        hostIp = nullptr;
        hostIp = gethostbyname(hostName.data());
        if (hostIp == nullptr)
            return false;

        return true;

    };

    int MakeConnectionAny()
    {
        switch (protocol)
        {
        case Protocol::TELNET:
        case Protocol::HTTP:
            return MakeConnection();

        case Protocol::HTTPS:
            return MakeHTTPSConnection();

        default:
            break;
        }
    }


    void SendAny(std::string_view message)
    {
        switch (protocol)
        {
        case Protocol::TELNET:
        case Protocol::HTTP:
            Send(message);
            break;

        case Protocol::HTTPS:
            SendHttps(message);
            break;

        default:
            break;
        }
    }
```

```
    inline void SetProtocol(Protocol protocol) { port = static_cast<uint16_t>(protocol); this-
>protocol = protocol; };

private:

    static void HandleOpenSSLErrors() {
        ERR_print_errors_fp(stderr);
        abort();
    }

    SSL_CTX* ctx = nullptr;
    SSL* ssl = nullptr;

    hostent* hostIp = nullptr;

    SOCKET client = INVALID_SOCKET;
    Protocol protocol = Protocol::HTTP;
    uint16_t port = static_cast<uint16_t>(Protocol::HTTP);

    sockaddr_in serverAddress = {};
};



int main()
{
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    Client::Init();
    auto client = new Client();

    int selectedProtocol = 0;
    int selectedPort = 0;
    std::string address  = "";


    while (true)
    {
        while(selectedProtocol == 0)
        {
            std::cout << "Выберите протокол или порт" << std::endl;
            std::cout << "1. HTTP" << std::endl;
            std::cout << "2. HTTPS" << std::endl;
            std::cout << "3. TELNET" << std::endl;
            std::string temp;
            std::getline(std::cin, temp);

            selectedProtocol = std::stoi(temp);

            switch (selectedProtocol)
            {
            case 1:
                client->SetProtocol(Client::Protocol::HTTP);
                break;
            case 2:
                client->SetProtocol(Client::Protocol::HTTPS);
                break;
            case 3:
                client->SetProtocol(Client::Protocol::TELNET);
                break;

            default:
                selectedProtocol = 0;
                break;
            }
        }

        while (address == "")
        {
            std::cout << "Введите адрес" << std::endl;
            std::getline(std::cin, address);
            if (!client->SetHostIp(address))
            {
                std::cout << "Неверный адрес. Попробуйте снова." << std::endl;
```

```
                address = "";
            }
        }

        while (true)
        {
            std::cout << "Введите запрос или введите exit для выхода" << std::endl;
            std::string str;

            //std::cin.ignore();
            std::getline(std::cin,str);

            if (str == "exit")
                return 0;

            str += '\n';
            if (client->MakeConnectionAny() != 0)
            {
                std::cout << "Соединенение с сервером утеряно" << std::endl;
                break;
            }
            else
                client->SendAny(str);
        }


        client->Close();

    }

    return 0;
}
```

Результат выполнения:

```
Выберите протокол или порт
1. HTTP
2. HTTPS
3. TELNET
1
Введите адрес
wikipedia.org
Введите запрос или введите exit для выхода
GET /
HTTP/1.1 301 Moved Permanently
content-length: 0
location: https:///
server: HAProxy
x-cache: cp3068 int
x-cache-status: int-tls
connection: close

SERVER CLOSE.
Введите запрос или введите exit для выхода
exit
```

Задание 2:

Сервер:

```
#include <iostream>
#include <string>
#include <vector>
#include <thread>
#include <winsock2.h>
#include <sstream>
#pragma comment(lib, "Ws2_32.lib")
enum class State
{
    READY,
    PAUSED,
    STOPED
};
```

```
volatile State serverState = State::READY;

void handleClient(SOCKET clientSocket) {

    int bytesReceived;
    std::string response;
    char buffer[1024];


    do {
        bytesReceived = recv(clientSocket, buffer, sizeof(buffer), 0);
        if (bytesReceived > 0)
        {
            buffer[bytesReceived] = '\0';
            std::string request(buffer);
            std::cout << request << std::endl;

            std::vector <std::string> tokens;
            std::istringstream iss(request);
            std::string s;

            while (std::getline(iss, s, ' '))
            {
                tokens.push_back(s);
            }

            std::string command = tokens[0];


            if (command == "exit") {
                response = "Сервер выключается";
                serverState = State::STOPED;
            }
            else if (command == "pause")
            {
                response = "Сервер на паузе.";
                serverState = State::PAUSED;

            }
            else if (command == "ready")
            {
                response = "Сервер в состоянии готовности";
                serverState = State::READY;
            }
            else if (command == "echo")
            {
                if (tokens.size() > 1)
                {
                    response = tokens[1];
                }
                else
                    response = "Недостаточно аргументов";
            }
            else if (command == "rand")
            {
                response = std::to_string(0 + rand() % 100);
            }
            else
            {
                response = "Неизвестная команда";
            }

            send(clientSocket, response.c_str(), response.size(), 0);
        }
    } while (bytesReceived > 0 && response != "Сервер выключается.");

    closesocket(clientSocket);
}

int main()
{
    srand(time(NULL));

    WSADATA wsData;
```

```
    WORD ver = MAKEWORD(2, 2);

    if (WSAStartup(ver, &wsData) != 0) {
        std::cerr << "Неудалось инициализировать winsock" << std::endl;
        return 1;
    }

    SOCKET serverSocket = socket(AF_INET, SOCK_STREAM, 0);
    if (serverSocket == INVALID_SOCKET) {
        std::cerr << "Неудалось создать сокет" << std::endl;
        return 1;
    }

    sockaddr_in serverAddr;
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_port = htons(54000);
    serverAddr.sin_addr.S_un.S_addr = INADDR_ANY;

    if (bind(serverSocket, (sockaddr*)&serverAddr, sizeof(serverAddr)) == SOCKET_ERROR) {
        std::cerr << "Неудалось привязать сокет" << std::endl;
        closesocket(serverSocket);
        WSACleanup();
        return 1;
    }

    if (listen(serverSocket, SOMAXCONN) == SOCKET_ERROR) {
        std::cerr << "Неудалось запустить прослушивание" << std::endl;
        closesocket(serverSocket);
        WSACleanup();
        return 1;
    }

    std::cout << "Сервер запущен." << std::endl;

    while (true) {

        if (serverState == State::STOPED)
            break;

        if (serverState != State::PAUSED)
        {
            SOCKET clientSocket = accept(serverSocket, nullptr, nullptr);
            if (clientSocket == INVALID_SOCKET) {
                std::cerr << "Неудалось притяль клиената" << std::endl;
                closesocket(serverSocket);
                WSACleanup();
                return 1;
            }

            std::thread clientThread(handleClient, clientSocket);
            clientThread.join();
        }
    }

    closesocket(serverSocket);
    WSACleanup();

    return 0;
}
```

Клиент:

```
#include <iostream>
#include <string>
#include <winsock2.h>
#include <chrono>
#include <limits>
#pragma comment(lib, "Ws2_32.lib")
int main() {
    WSADATA wsData;
    WORD ver = MAKEWORD(2, 2);
```

```
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    if (WSAStartup(ver, &wsData) != 0) {
        std::cerr << "Can't initialize Winsock! Quitting" << std::endl;
        return 1;
    }

    SOCKET clientSocket = socket(AF_INET, SOCK_STREAM, 0);
    if (clientSocket == INVALID_SOCKET) {
        std::cerr << "Can't create socket! Quitting" << std::endl;
        WSACleanup();
        return 1;
    }

    while (true)
    {
        std::string serverAddres = "";
        std::string port = "";
        std::getline(std::cin, serverAddres);
        std::getline(std::cin, port);
        int iPort = 0;

        std::istringstream ss(port);
        ss >> iPort;


        sockaddr_in serverAddr = { };
        serverAddr.sin_family = AF_INET;
        serverAddr.sin_port = htons(iPort);
        serverAddr.sin_addr.S_un.S_addr = inet_addr(serverAddres.c_str());

        if (connect(clientSocket, (sockaddr*)&serverAddr, sizeof(serverAddr)) == SOCKET_ERROR) {
            std::cerr << "Неудалось подключится к серверу!" <<std::endl;
            if (WSAGetLastError() == WSAECONNREFUSED)
            {
                std::cout << "неверный IP или порт" << std::endl;
                continue;
            }
            else
            {
                std::cout << "Неизвестная ошибка: " << WSAGetLastError() << std::endl;
                return -1;
            }
        }

        std::cout << "Connected to server. Type 'end', 'bye', 'stop', or 'exit' to quit." <<
std::endl;
        break;
    }
    std::string userInput;
    do {
        std::cout << "Enter command: ";

        std::getline(std::cin, userInput);
        send(clientSocket, userInput.c_str(), userInput.size(), 0);

        std::chrono::steady_clock::time_point begin = std::chrono::steady_clock::now();

        char buffer[1024];
        int bytesReceived = recv(clientSocket, buffer, sizeof(buffer), 0);
        std::chrono::steady_clock::time_point end = std::chrono::steady_clock::now();
        if (bytesReceived > 0) {
            buffer[bytesReceived] = '\0';
            std::cout << "Server response: " << buffer << std::endl;
            std::cout << "Время обработки: "
                << std::chrono::duration_cast<std::chrono::seconds>(end - begin).count()
                << " секунд "
                << std::chrono::duration_cast<std::chrono::milliseconds>(end - begin).count()
                << " миллисекунд "
                << std::chrono::duration_cast<std::chrono::microseconds>(end - begin).count()
                << " микросекунд "
                << std::chrono::duration_cast<std::chrono::nanoseconds>(end - begin).count()
                << " наносекунд "
                << std::endl;
```

```
        }
    } while (userInput != "stop");

    closesocket(clientSocket);
    WSACleanup();

    return 0;
}
```

Результат выполнения:

```
127.0.0.1
54000
Connected to server.
Enter command: echo hello
Server response: hello
Время обработки: 0 секунд 0 миллисекунд 211 микросекунд 211600 наносекунд
Enter command: rand
Server response: 41
Время обработки: 0 секунд 0 миллисекунд 170 микросекунд 170000 наносекунд
Enter command: exit
Server response: Сервер выключается
Время обработки: 0 секунд 0 миллисекунд 151 микросекунд 151900 наносекунд
```

Задание 3:

Клиент:

```
#include <iostream>
#include <Winsock2.h>
#include <chrono>
#include <string>
#define SERVER_IP "127.0.0.1"
#define PORT 8888
#pragma comment(lib, "Ws2_32.lib")

enum DataType
{
    INTEGER    = 0x01,
    STRING = 0x02,
    REAL = 0x03,
};


int main()
{
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);


    WSADATA wsaData;
    SOCKET clientSocket;
    struct sockaddr_in serverAddr;
    char buffer[1024] = { 0 };
    int bytesSent, bytesReceived;

    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
        std::cerr << "Failed to initialize Winsock\n";
        return 1;
    }

    if ((clientSocket = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == INVALID_SOCKET) {
        std::cerr << "Failed to create socket\n";
        WSACleanup();
        return 1;
    }

    serverAddr.sin_family = AF_INET;
    serverAddr.sin_addr.s_addr = inet_addr(SERVER_IP);
    serverAddr.sin_port = htons(PORT);
```

```
    while (true) {

        char a = INTEGER;
        while (true)
        {
            std::cout << "Введите тип передаваемых данных: \n1.INT \n2.STRING\n3.DOUBLE\nexit
для выхода" << std::endl;
            std::string dataType = "";
            std::getline(std::cin, dataType);
            if(dataType == "exit")
            {
                return 0;
            }

            if ((a = std::stoi(dataType)) > 3)
            {
                std::cout << "Неверный ввод попробуйте еще раз" << std::endl;
            }
            else
                break;
        }

        std::cout << "Введите количество отправляемых байт" << std::endl;
        std::string temp = "";
        std::getline(std::cin, temp);
        auto sendBytes = std::stoi(temp);

        std::cout << "Введите данные" << std::endl;
        std::string data = "";
        std::getline(std::cin, data);

        std::string sendData;
        sendData.push_back(a);
        sendData += data;


        bytesSent = sendto(clientSocket, sendData.c_str(), sendBytes + 1, 0, (struct
sockaddr*)&serverAddr, sizeof(serverAddr));
        if (bytesSent == SOCKET_ERROR) {
            std::cerr << "sendto() failed with error code : " << WSAGetLastError() << std::endl;
            break;
        }

        std::chrono::steady_clock::time_point begin = std::chrono::steady_clock::now();

        bytesReceived = recv(clientSocket, buffer, sizeof(buffer), 0);
        if (bytesReceived == SOCKET_ERROR) {
            std::cerr << "recv() failed with error code : " << WSAGetLastError() << std::endl;
            break;
        }
        std::chrono::steady_clock::time_point end = std::chrono::steady_clock::now();

        buffer[bytesReceived] = '\0';
        std::cout << "Server response: " << buffer << std::endl;

        std::cout << "Время обработки: "
            << std::chrono::duration_cast<std::chrono::seconds>(end - begin).count()
            << " секунд "
            << std::chrono::duration_cast<std::chrono::milliseconds>(end - begin).count()
            << " миллисекунд "
            << std::chrono::duration_cast<std::chrono::microseconds>(end - begin).count()
            << " микросекунд "
            << std::chrono::duration_cast<std::chrono::nanoseconds>(end - begin).count()
            << " наносекунд "
            << std::endl;
    }

    closesocket(clientSocket);
    WSACleanup();

    return 0;
}
```

Сервер:

```
#include <iostream>
#include <Winsock2.h>
#include <string>
#define PORT 8888
#pragma comment(lib, "Ws2_32.lib")

enum DataType
{
    INTEGER = 0x01,
    STRING = 0x02,
    REAL = 0x03,

};

int main()
{
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);

    WSADATA wsaData;
    SOCKET serverSocket;
    struct sockaddr_in serverAddr, clientAddr;
    int clientAddrLen = sizeof(clientAddr);
    char buffer[1024] = { 0 };
    int bytesReceived;

    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
        std::cerr << "Failed to initialize Winsock\n";
        return 1;
    }

    if ((serverSocket = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == INVALID_SOCKET) {
        std::cerr << "Failed to create socket\n";
        return 1;
    }

    serverAddr.sin_family = AF_INET;
    serverAddr.sin_addr.s_addr = INADDR_ANY;
    serverAddr.sin_port = htons(PORT);

    if (bind(serverSocket, (struct sockaddr*)&serverAddr, sizeof(serverAddr)) == SOCKET_ERROR) {
        std::cerr << "Bind failed with error code : " << WSAGetLastError() << std::endl;
        closesocket(serverSocket);
        WSACleanup();
        return 1;
    }

    std::cout << "UDP server started and listening on port " << PORT << std::endl;

    while (true)
    {
        bytesReceived = recvfrom(serverSocket, buffer, sizeof(buffer), 0, (struct
sockaddr*)&clientAddr, &clientAddrLen);
        if (bytesReceived == SOCKET_ERROR) {
            std::cerr << "recvfrom() failed with error code : " << WSAGetLastError() <<
std::endl;
            break;
        }

        std::cout << "Received from " << inet_ntoa(clientAddr.sin_addr) << ":" <<
ntohs(clientAddr.sin_port) << " - " << buffer << std::endl;
        std::string temp;
        for (int i = 1; i < strlen(buffer); i++)
            temp += buffer[i];

        switch (buffer[0])
        {
        case INTEGER:
            std::cout << "Получен int: " << std::stoi(temp) << std::endl;
```

```
            break;
        case STRING:
            std::cout << "Получена строка: " << temp << std::endl;
            break;
        case REAL:
            std::cout << "Получено вещественное число: " << std::stof(temp) << std::endl;
            break;
        default:
            break;
        }

        if (sendto(serverSocket, buffer, bytesReceived, 0, (struct sockaddr*)&clientAddr,
clientAddrLen) == SOCKET_ERROR) {
            std::cerr << "sendto() failed with error code : " << WSAGetLastError() << std::endl;
            break;
        }
    }
    closesocket(serverSocket);
    WSACleanup();

    return 0;
}
```

Результат выполнения:

```
1
Введите количество отправляемых байт
4
Введите данные
4325
Server response: 4325
Время обработки: 0 секунд 0 миллисекунд 473 микросекунд 473500 наносекунд
Введите тип передаваемых данных:
1.INT
2.STRING
3.DOUBLE
exit для выхода
2
Введите количество отправляемых байт
12
Введите данные
dsdsadasdasdsaasdf
Server response: dsdsadasdasd
Время обработки: 0 секунд 0 миллисекунд 491 микросекунд 491600 наносекунд
Введите тип передаваемых данных:
1.INT
2.STRING
3.DOUBLE
exit для выхода
exit
```

```
UDP server started and listening on port 8888
Received from 127.0.0.1:53344 - 4325
Получен int: 4325
Received from 127.0.0.1:53344 - dsdsadasdasd
Получена строка: dsdsadasdasd
```

Вывод: изучили клиент-серверную архитектуру, UDP и TCP сокеты.