

Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей
Кафедра информатики
Дисциплина «Операционные среды и системное программирование»

ОТЧЕТ
к лабораторной работе №5
на тему:
«ЭЛЕМЕНТЫ СЕТЕВОГО ПРОГРАММИРОВАНИЯ»
БГУИР 6-05-0612-02 67

Выполнил студент группы 353503
КОХАН Артём Игоревич

(дата, подпись студента)

Проверил ассистент каф. информатики
Гриценко Никита Юрьевич

(дата, подпись преподавателя)

Минск 2025

СОДЕРЖАНИЕ

1 Индивидуальное задание.....	3
2 Краткие теоритические сведения	4
3 Описание функций программы.....	5
4 Пример выполнения программы	7
4.1 Запуск программы и процесс выполнения	7
4.2 Описание работы и результатов	7
Вывод.....	9
Список использованных источников	10
Приложение А (справочное) Исходный код	11

1 ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

Программа или несколько программ, демонстрирующие взаимодействие между собой или с иным (внешним) ПО посредством сетевых протоколов и программного интерфейса сокетов. Реализация известных сетевых протоколов прикладного уровня или разработка собственных. Специальных требований к приложениям не предъявляется; в частности, во многих случаях они могут быть не обязательно оконными, но также и консольными.

Разработка и реализация (в виде сервера и, при необходимости, клиента) некоторой сетевой службы (варианты, не вошедшие в предыдущие пункты).

Функциональность службы:

- установка на заданный порт (сервер);
- обнаружение сервера (клиент);
- ввод или генерация данных запроса (клиент);
- обработка запроса и возврат результата (сервер);
- прием и отображение ответа (клиент);
- контроль соединения, обработка ошибок.

2 КРАТКИЕ ТЕОРИТИЧЕСКИЕ СВЕДЕНИЯ

Сетевое программирование позволяет организовывать взаимодействие между процессами, находящимися на разных устройствах в одной сети или в различных сетях. Основой сетевого программирования является иерархическая модель взаимодействия, такая как модель OSI или стек TCP/IP, в которых взаимодействие организуется через различные уровни протоколов. Эти уровни включают транспортный уровень TCP и UDP для обеспечения передачи данных, а также прикладные протоколы, такие как ICMP, HTTP и другие, для реализации конкретной функциональности [1]. Сетевые протоколы обеспечивают стандартизированные методы передачи данных, что позволяет различным устройствам взаимодействовать друг с другом, несмотря на разнообразие платформ и операционных систем.

Программный интерфейс API сокетов предоставляет функции для создания и управления сетевыми соединениями, такие как `socket()`, `bind()`, `listen()`, `connect()`, `send()`, и `recv()`. Сокеты бывают двух основных типов: потоковые TCP и датаграмные UDP. Потоковые сокет обеспечивают надежную передачу данных с подтверждением и контролем ошибок, а датаграмные сокет подходят для быстрой передачи данных, но без гарантий доставки [2]. Это делает TCP предпочтительным выбором для приложений, требующих высокой надежности, таких как веб-сервисы и обмен файлами, тогда как UDP более удобен для приложений, где важна скорость, таких как видеотрансляции и онлайн-игры.

Использование протокола ICMP в утилите Traceroute позволяет отслеживать маршрут пакетов в сети и измерять задержку на каждом узле. Это достигается с помощью отправки ICMP Echo-запросов и получения ICMP Echo-ответов, что позволяет определить последовательность маршрутизаторов между исходной и конечной точками [3]. Разработка подобных приложений помогает лучше понять особенности маршрутизации, диагностики и устранения неполадок в сетях. Знание таких инструментов, как Traceroute, может оказаться полезным для системных администраторов и сетевых инженеров, так как оно позволяет оперативно выявлять и устранять проблемы с подключением.

3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

Программа реализует клиент-серверную модель сетевого взаимодействия с использованием сокетов WinAPI. Архитектура программы состоит из двух основных компонентов: сервера, предоставляющего услуги кодирования/декодирования Base64, и клиента, взаимодействующего с сервером по TCP-протоколу.

Основным классом серверной части является SocketServer, который инкапсулирует логику сетевого взаимодействия. Класс содержит следующие приватные поля: структуру WSADATA для инициализации Winsock, серверный сокет для приема соединений, структуру sockaddr_in для настройки адреса сервера. Дополнительно реализован класс Base64Converter, предоставляющий статические методы для кодирования и декодирования данных в формат Base64.

Функция SafePrint обеспечивает потокобезопасный вывод сообщений в консоль с использованием мьютекса cout_mutex. Это предотвращает перемешивание вывода от разных потоков и обеспечивает читаемость логов выполнения программы, что особенно важно в многопоточном серверном окружении.

Метод initialize класса SocketServer выполняет инициализацию сетевой подсистемы. Последовательность действий включает: инициализацию Winsock через WSStartup, создание TCP-сокета функцией socket, настройку адреса сервера (порт 8888, любой интерфейс), привязку сокета функцией bind и перевод сокета в режим прослушивания функцией listen. В случае ошибки на любом этапе метод возвращает false с выводом диагностической информации.

Метод processCommand реализует обработку клиентских запросов. Команды от клиентов поступают в формате "КОМАНДА:ДАННЫЕ". Поддерживаются три типа команд: ENCODE - кодирование строки в Base64, DECODE - декодирование из Base64, TEST - тестовая команда для проверки связи. Метод анализирует команду, вызывает соответствующий метод класса Base64Converter и формирует ответ в формате "РЕЗУЛЬТАТ:ДАННЫЕ".

Метод run реализует основной цикл сервера. В бесконечном цикле сервер ожидает подключений через accept, принимает данные от клиента через recv, обрабатывает команды через processCommand и отправляет ответы через send. Особенностью реализации является поддержка команды QUIT для корректного завершения соединения.

Класс Base64Converter содержит алгоритмы преобразования данных. Метод encode преобразует входную строку в Base64-формат путем группировки битов и преобразования 6-битных последовательностей в соответствующие символы алфавита Base64. Метод decode выполняет обратное преобразование, используя предварительно заполненную таблицу соответствия символов. Оба метода корректно обрабатывают padding символы '='.

Клиентская часть представлена классом `SocketClient`. Метод `connectToServer` устанавливает TCP-соединение с сервером по указанному IP-адресу и порту. Процесс включает инициализацию Winsock, создание клиентского сокета, настройку адреса сервера и установление соединения через `connect`.

Метод `sendCommand` обеспечивает отправку команды серверу и получение ответа. Отправка данных осуществляется через `send`, прием ответа – через `recv`. Метод обрабатывает различные сценарии сетевого взаимодействия, включая разрыв соединения и ошибки передачи.

Метод `interactiveMode` реализует интерактивный режим работы клиента. В цикле осуществляется чтение команд из стандартного ввода, их отправка на сервер и вывод полученных ответов. Режим поддерживает команды выхода "quit" и "exit".

Функция `main` серверной части инициализирует сервер и запускает его основной цикл. Сервер выводит информационные сообщения о своем состоянии, принимаемых командах и подключениях клиентов.

Функция `main` клиентской части устанавливает соединение с `localhost` на порт 8888 и запускает интерактивный режим. При невозможности подключения выводится диагностическое сообщение с рекомендациями по устранению проблемы.

Программа демонстрирует эффективное использование WinAPI сокетов для организации клиент-серверного взаимодействия, реализует простой текстовый протокол прикладного уровня и предоставляет практический пример работы с сетевыми соединениями в среде Windows. Архитектура приложения обеспечивает модульность и возможность расширения функциональности.

4 ПРИМЕР ВЫПОЛНЕНИЯ ПРОГРАММЫ

4.1 Запуск программы и процесс выполнения

Программа была запущена в среде разработки для тестирования клиент-серверного взаимодействия с использованием TCP-сокетов. Первым был запущен сервер Base64 Encoding Service Server, который успешно инициализировал Winsock, создал серверный сокет и начал прослушивание порта 8888. Сервер отобразил информационное сообщение о доступных командах и перешел в режим ожидания подключений.

Затем был запущен клиент Base64 Encoding Service Client, который установил TCP-соединение с сервером по адресу 127.0.0.1:8888. После успешного подключения клиент перешел в интерактивный режим работы, предоставив пользователю возможность ввода команд в формате "COMMAND:DATA".

<pre>=== Base64 Encoding Service Server === Commands format: COMMAND:DATA Available commands: ENCODE, DECODE, TEST, QUIT Server started on port 8888 Waiting for connections... Client connected! Received command: ENCODE:Hello World Received command: dsfdf Received command: DECODE:SGVsbG8gV29ybGQ= Received command: TEST:Some text -</pre>	<pre>=== Base64 Encoding Service Client === Connected to server 127.0.0.1:8888 Interactive mode started. Type 'quit' to exit. Command format: COMMAND:DATA Examples: ENCODE:Hello World DECODE:SGVsbG8gV29ybGQ= TEST:Some text Enter command: ENCODE:Hello World Server response: ENCODED:SGVsbG8gV29ybGQ= Enter command: dsfdf Server response: ERROR:Invalid command format Enter command: DECODE:SGVsbG8gV29ybGQ= Server response: DECODED:Hello World Enter command: TEST:Some text Server response: TEST:Server is working! Your data: Some text Enter command:</pre>
---	---

Рисунок 4.1 – Результат выполнения программы

4.2 Описание работы и результатов

В процессе тестирования программы было проведено несколько последовательных операций взаимодействия между клиентом и сервером, которые наглядно демонстрируют работоспособность всей системы. Первой выполнялась команда кодирования текста "Hello World" в формат Base64. Сервер успешно принял команду "ENCODE:Hello World", обработал её и вернул закодированную строку "ENCODED:S6Vsb68gV29yb6Q=", что подтвердило корректную работу алгоритма кодирования. Следующим тестом стала проверка обработки ошибок – клиент отправил некорректную команду "dsfdf" без разделителя и параметров. Сервер корректно идентифицировал ошибку формата и вернул сообщение "ERROR:Invalid command format", что свидетельствует о надёжной валидации входных данных на стороне сервера.

Для проверки симметричности работы алгоритмов была отправлена команда декодирования ранее полученной строки "DECODE:S6Vsb68gV29yb6Q=". Сервер успешно выполнил обратное преобразование и вернул исходный текст "DECODED:Hello World", что подтвердило корректность работы обоих алгоритмов – кодирования и декодирования. Завершающим тестом стала проверка общей работоспособности системы с помощью команды "TEST:Some text", на которую сервер ответил подтверждением "TEST:Server is working! Your data: Some text", продемонстрировав стабильность сетевого соединения и готовность к обработке запросов. Все операции выполнялись практически мгновенно, без задержек, что говорит об эффективной работе TCP-соединения и оптимальной реализации сетевого протокола. Программа корректно обрабатывает различные сценарии использования – как штатные операции кодирования/декодирования, так и ошибочные запросы, обеспечивая стабильную работу без аварийных завершений. Результаты тестирования полностью подтверждают соответствие программы заявленным требованиям и демонстрируют реализацию клиент-серверного взаимодействия с использованием сокетов WinAPI.

ВЫВОД

В ходе выполнения лабораторной работы была успешно разработана и протестирована клиент-серверная система, реализующая сетевое взаимодействие с использованием сокетов WinAPI. Программа продемонстрировала эффективную организацию TCP-соединения между клиентом и сервером, а также корректную работу службы кодирования и декодирования данных в формат Base64.

Результаты тестирования показали, что разработанная система успешно обрабатывает все типы запросов: кодирование текстовых данных, декодирование Base64-строк, тестовые команды и обработку ошибочных запросов. Сервер стабильно работает на порту 8888, принимает подключения от клиентов и обеспечивает мгновенный ответ на все корректные команды.

Практическая ценность работы заключается в приобретении опыта работы с сетевым программированием на платформе Windows, включая инициализацию Winsock, создание и настройку сокетов, установку TCP-соединений и организацию обмена данными между процессами. Разработанный простой текстовый протокол прикладного уровня доказал свою эффективность для решения задач клиент-серверного взаимодействия.

Полученные результаты наглядно демонстрируют принципы построения сетевых приложений и могут служить основой для разработки более сложных распределенных систем. Программа успешно решает поставленную задачу и представляет собой законченный пример реализации сетевой службы с использованием интерфейса сокетов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Cisco Documentation: Understanding Networking Fundamentals [Электронный ресурс]. – Режим доступа: <https://www.cisco.com/c/en/us/support/docs/ios-nx-os-software/ios-software-releases-122-mainline/13769-10.html>. – Дата доступа: 15.11.2025.

[2] IBM Documentation: Overview of Socket Programming [Электронный ресурс]. – Режим доступа: <https://www.ibm.com/docs/en/zos/2.4.0?topic=designs-overview-socket-programming>. – Дата доступа: 15.11.2025.

[3] RFC 792: Internet Control Message Protocol (ICMP) [Электронный ресурс]. – Режим доступа: <https://tools.ietf.org/html/rfc792>. – Дата доступа: 15.11.2025.

ПРИЛОЖЕНИЕ А

(справочное)

Исходный код

Листинг А.1 – Реализация Server.cpp

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

#pragma comment(lib, "ws2_32.lib")

class Base64Converter {
private:
    static const std::string BASE64_CHARS;

public:
    static std::string encode(const std::string& input) {
        std::string output;
        int val = 0, valb = -6;

        for (unsigned char c : input) {
            val = (val << 8) + c;
            valb += 8;
            while (valb >= 0) {
                output.push_back(BASE64_CHARS[(val >> valb) & 0x3F]);
                valb -= 6;
            }
        }

        if (valb > -6) {
            output.push_back(BASE64_CHARS[((val << 8) >> (valb + 8)) & 0x3F]);
        }

        while (output.size() % 4) {
            output.push_back('=');
        }

        return output;
    }

    static std::string decode(const std::string& input) {
        std::string output;
        std::vector<int> T(256, -1);

        for (int i = 0; i < 64; i++) {
            T[BASE64_CHARS[i]] = i;
        }

        int val = 0, valb = -8;
        for (unsigned char c : input) {
            if (T[c] == -1) break;
            val = (val << 6) + T[c];
            valb += 6;
            if (valb >= 0) {
                output.push_back(char((val >> valb) & 0xFF));
                valb -= 8;
            }
        }
    }
};
```

```

    }

    return output;
}
};

const std::string Base64Converter::BASE64_CHARS =
"ABCDEFGHIJKLMNOPQRSTUVWXYZ"
"abcdefghijklmnopqrstuvwxyz"
"0123456789+/";

class SocketServer {
private:
    WSADATA wsaData;
    SOCKET serverSocket;
    sockaddr_in serverAddr;

public:
    SocketServer() : serverSocket(INVALID_SOCKET) {}

    bool initialize() {
        // Инициализация Winsock
        if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
            std::cerr << "WSAStartup failed: " << WSAGetLastError() <<
std::endl;
            return false;
        }

        // Создание сокета
        serverSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
        if (serverSocket == INVALID_SOCKET) {
            std::cerr << "Socket creation failed: " << WSAGetLastError() <<
std::endl;
            WSACleanup();
            return false;
        }

        // Настройка адреса сервера
        serverAddr.sin_family = AF_INET;
        serverAddr.sin_addr.s_addr = INADDR_ANY;
        serverAddr.sin_port = htons(8888);

        // Привязка сокета
        if (bind(serverSocket, (sockaddr*)&serverAddr, sizeof(serverAddr)) ==
SOCKET_ERROR) {
            std::cerr << "Bind failed: " << WSAGetLastError() << std::endl;
            closesocket(serverSocket);
            WSACleanup();
            return false;
        }

        // Прослушивание порта
        if (listen(serverSocket, SOMAXCONN) == SOCKET_ERROR) {
            std::cerr << "Listen failed: " << WSAGetLastError() << std::endl;
            closesocket(serverSocket);
            WSACleanup();
            return false;
        }

        std::cout << "Server started on port 8888" << std::endl;
        return true;
    }

    void processCommand(SOCKET clientSocket, const std::string& command) {

```

```

std::string response;

// Формат команды: COMMAND:DATA
size_t colonPos = command.find(':');
if (colonPos == std::string::npos) {
    response = "ERROR:Invalid command format";
}
else {
    std::string cmd = command.substr(0, colonPos);
    std::string data = command.substr(colonPos + 1);

    if (cmd == "ENCODE") {
        response = "ENCODED:" + Base64Converter::encode(data);
    }
    else if (cmd == "DECODE") {
        response = "DECODED:" + Base64Converter::decode(data);
    }
    else if (cmd == "TEST") {
        response = "TEST:Server is working! Your data: " + data;
    }
    else {
        response = "ERROR:Unknown command";
    }
}

// Отправка ответа клиенту
send(clientSocket, response.c_str(), response.length(), 0);
}

void run() {
    while (true) {
        std::cout << "Waiting for connections..." << std::endl;

        // Принятие соединения
        SOCKET clientSocket = accept(serverSocket, NULL, NULL);
        if (clientSocket == INVALID_SOCKET) {
            std::cerr << "Accept failed: " << WSAGetLastError() <<
std::endl;
            continue;
        }

        std::cout << "Client connected!" << std::endl;

        char buffer[1024];
        int bytesReceived;

        // Получение данных от клиента
        while ((bytesReceived = recv(clientSocket, buffer, sizeof(buffer)
- 1, 0)) > 0) {
            buffer[bytesReceived] = '\0';
            std::string command(buffer);

            std::cout << "Received command: " << command << std::endl;

            // Обработка специальной команды выхода
            if (command == "QUIT") {
                std::cout << "Client requested disconnect" << std::endl;
                break;
            }

            // Обработка команды
            processCommand(clientSocket, command);
        }
    }
}

```

```

        if (bytesReceived == 0) {
            std::cout << "Client disconnected" << std::endl;
        }
        else if (bytesReceived == SOCKET_ERROR) {
            std::cerr << "Recv failed: " << WSAGetLastError() << std::endl;
        }

        closesocket(clientSocket);
    }
}

~SocketServer() {
    if (serverSocket != INVALID_SOCKET) {
        closesocket(serverSocket);
    }
    WSACleanup();
}

};

int main() {
    std::cout << "=== Base64 Encoding Service Server ===" << std::endl;
    std::cout << "Commands format: COMMAND:DATA" << std::endl;
    std::cout << "Available commands: ENCODE, DECODE, TEST, QUIT" << std::endl;

    SocketServer server;
    if (server.initialize()) {
        server.run();
    }

    return 0;
}

```

Листинг А.2 – Реализация Client.cpp

```

#include <winsock2.h>
#include <ws2tcpip.h>
#include <iostream>
#include <string>
#include <windows.h>

#pragma comment(lib, "ws2_32.lib")

class SocketClient {
private:
    WSADATA wsaData;
    SOCKET clientSocket;
    sockaddr_in serverAddr;

public:
    SocketClient() : clientSocket(INVALID_SOCKET) {}

    bool connectToServer(const std::string& serverIP, int port) {
        // Инициализация Winsock
        if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
            std::cerr << "WSAStartup failed: " << WSAGetLastError() <<
std::endl;
            return false;
        }

        // Создание сокета
        clientSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
        if (clientSocket == INVALID_SOCKET) {

```

```

        std::cerr << "Socket creation failed: " << WSAGetLastError() <<
std::endl;
        WSACleanup();
        return false;
    }

    // Настройка адреса сервера
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_port = htons(port);
    inet_pton(AF_INET, serverIP.c_str(), &serverAddr.sin_addr);

    // Подключение к серверу
    if (connect(clientSocket, (sockaddr*)&serverAddr, sizeof(serverAddr))
== SOCKET_ERROR) {
        std::cerr << "Connect failed: " << WSAGetLastError() << std::endl;
        closesocket(clientSocket);
        WSACleanup();
        return false;
    }

    std::cout << "Connected to server " << serverIP << ":" << port <<
std::endl;
    return true;
}

std::string sendCommand(const std::string& command) {
    // Отправка команды серверу
    if (send(clientSocket, command.c_str(), command.length(), 0) ==
SOCKET_ERROR) {
        return "ERROR:Send failed";
    }

    // Получение ответа от сервера
    char buffer[1024];
    int bytesReceived = recv(clientSocket, buffer, sizeof(buffer) - 1, 0);
    if (bytesReceived > 0) {
        buffer[bytesReceived] = '\0';
        return std::string(buffer);
    }
    else if (bytesReceived == 0) {
        return "ERROR:Connection closed";
    }
    else {
        return "ERROR:Receive failed";
    }
}

void interactiveMode() {
    std::string input;

    std::cout << "Interactive mode started. Type 'quit' to exit." <<
std::endl;
    std::cout << "Command format: COMMAND:DATA" << std::endl;
    std::cout << "Examples:" << std::endl;
    std::cout << "  ENCODE:Hello World" << std::endl;
    std::cout << "  DECODE:SGVsbG8gV29ybGQ=" << std::endl;
    std::cout << "  TEST:Some text" << std::endl;

    while (true) {
        std::cout << "\nEnter command: ";
        std::getline(std::cin, input);

        if (input == "quit" || input == "exit") {
            sendCommand("QUIT");

```

```

        break;
    }

    if (!input.empty()) {
        std::string response = sendCommand(input);
        std::cout << "Server response: " << response << std::endl;
    }
}

~SocketClient() {
    if (clientSocket != INVALID_SOCKET) {
        closesocket(clientSocket);
    }
    WSACleanup();
}

};

int main() {
    std::cout << "=== Base64 Encoding Service Client ===" << std::endl;

    SocketClient client;

    // Подключение к localhost на порт 8888
    if (client.connectToServer("127.0.0.1", 8888)) {
        client.interactiveMode();
    }
    else {
        std::cout << "Failed to connect to server. Make sure server is running."
<< std::endl;
        std::cout << "Press Enter to exit...";
        std::cin.get();
    }

    return 0;
}

```