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

ОТЧЁТ

к лабораторной работе №5

на тему

**ЭЛЕМЕНТЫ СЕТЕВОГО ПРОГРАММИРОВАНИЯ**

Выполнил: студент гр.253504 Божко Я.Д

Проверил: ассистент кафедры информатики

Гриценко Н.Ю

Минск 2024

**СОДЕРЖАНИЕ**

[Введение 3](#_Toc6512)

[1 Теоретические сведения 4](#_Toc10206)

[1.1 Взаимодействие процессов 4](#_Toc16048)

[1.2 Реестры 6](#_Toc2035)

[1.3 Журналирование 9](#_Toc6435)

[2 Инструментальная языковая среда 12](#_Toc3814)

[3 Работа приложения 13](#_Toc400)

[3.1 Структура приложения 13](#_Toc26015)

[3.2 Функции приложения 13](#_Toc22336)

[3.3 Анализ данных 14](#_Toc29478)

[Список использованных источников 15](#_Toc11910)

[Приложение А (обязательное) 16](#_Toc27357)

**ВВЕДЕНИЕ**

В современном программировании важнейшую роль играют системы управления конфигурациями, механизмы журналирования, а также эффективное взаимодействие с операционной системой через различные интерфейсы. Эти компоненты являются основой разработки приложений, особенно в среде Windows, где операционная система предоставляет широкий спектр инструментов для работы с реестром, журналами событий и динамическими библиотеками.

Одной из ключевых тем в области системного программирования является использование реестра Windows. Реестр представляет собой центральную базу данных, хранящую настройки операционной системы и приложений. Понимание структуры реестра и методов взаимодействия с ним позволяет разработчикам эффективно управлять конфигурациями, а также обеспечивать высокую гибкость и адаптивность программного обеспечения. Кроме того, журналирование в Windows помогает отслеживать события в системе, фиксировать ошибки и обеспечивать безопасность приложений, что является важной составляющей любой сложной системы.

Другим важным аспектом является работа с динамическими библиотеками (DLL), которые позволяют приложениям загружать и использовать код во время выполнения, что значительно повышает гибкость разработки. Применение смешанных сборок (mixed assemblies) в сочетании с .NET и нативным кодом расширяет возможности взаимодействия с операционной системой и облегчает интеграцию различных технологий в одном проекте.

В данной работе рассматриваются подходы к хранению и использованию конфигураций, особенности работы с реестром и журналами событий Windows, а также вопросы, связанные с динамическими и смешанными библиотеками. Эти темы являются неотъемлемой частью разработки программного обеспечения в экосистеме Windows и оказывают значительное влияние на производительность, безопасность и удобство использования приложений.

Целью данной работы является изучение и анализ теоретических аспектов работы с конфигурациями, реестром, журналами и библиотеками в операционной системе Windows, а также рассмотрение практических методов их применения в реальных проектах.

**1 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

* 1. **Иерархическая модель взаимодействия открытых систем**

Иерархическая модель взаимодействия открытых систем (ISO/OSI) — это теоретическая модель, описывающая процесс передачи данных в компьютерных сетях. Она делит всю систему связи на 7 уровней, каждый из которых решает свою задачу и использует услуги нижележащих уровней. Эта модель помогает стандартизировать взаимодействие между различными системами и сетями. [1]

**1.1.1** Уровни модели OSI

Физический уровень (Physical Layer). Отвечает за передачу битов по каналу связи, определяет физические характеристики среды передачи (проводка, радиоволны и т. д.).

Канальный уровень (Data Link Layer). Обеспечивает надежную передачу данных через физическое соединение. Реализует механизмы обнаружения и коррекции ошибок, а также управление потоком.

Сетевой уровень (Network Layer). Отвечает за маршрутизацию и доставку пакетов данных от источника к получателю, а также за управление логическими адресами (например, IP-адресами).

Транспортный уровень (Transport Layer). Обеспечивает надежную доставку данных от одного процесса к другому, используя протоколы, такие как TCP (Transmission Control Protocol) и UDP (User Datagram Protocol).

Сеансовый уровень (Session Layer). Управляет установлением, поддержанием и завершением сеансов связи между приложениями, регулирует обмен данными между ними.

Представительский уровень (Presentation Layer). Отвечает за преобразование данных в формат, который понимает прикладное приложение, а также за их шифрование и сжатие.

Прикладной уровень (Application Layer). Этот уровень взаимодействует непосредственно с приложениями, предоставляет интерфейсы и службы для работы с данными (например, HTTP, FTP, SMTP).

**1.1.2** Стек протоколов OSI, TCP/IP

Стек протоколов OSI. Модель OSI является абстракцией, и в реальности не существует полного стека протоколов, строго следуя этому представлению. Однако, в основе многих современных сетевых протоколов лежат концепции из этой модели.

Стек протоколов TCP/IP. Стек TCP/IP — это более практическая реализация сетевых протоколов, которая используется в большинстве современных компьютерных сетей, включая интернет. Он включает 4 уровня:

Прикладной уровень (Application Layer) включает протоколы, такие как HTTP, FTP, SMTP и DNS.

Транспортный уровень (Transport Layer) включает TCP и UDP, которые обеспечивают передачу данных между приложениями.

Интернет-уровень (Internet Layer) включает протокол IP (Internet Protocol), который отвечает за маршрутизацию данных.

Уровень канала передачи данных (Link Layer) управляет физической и канальной передачей данных.

**1.1.3** Протокол IP

Протокол IP (Internet Protocol) — это основополагающий протокол для адресации и маршрутизации пакетов в сети. Он используется для передачи данных по сети между компьютерами, которые могут быть на разных физических сетях.

IPv4 — более старый формат IP-адресов, состоящий из 32 бит (4 байта).

IPv6 — более новый формат, который использует 128 бит (16 байтов), обеспечивая гораздо большее количество уникальных адресов.

**1.1.4** Идентификация абонентов в IP-сетях

Идентификация устройств в IP-сетях осуществляется с помощью уникальных IP-адресов, которые делятся на два типа:

1. локальные ip-адреса — используются внутри локальных сетей (например, 192.168.x.x);
2. публичные ip-адреса — используются в глобальной сети интернет.

Кроме того, в IP-сетях используется концепция маски подсети для разделения сетевой части и части, относящейся к устройствам внутри этой сети.

* 1. **Протоколы транспортного уровня TCP и UDP**

**1.2.1** TCP (Transmission Control Protocol)

TCP — это протокол с установлением соединения, который гарантирует надежную доставку данных между приложениями. Он обеспечивает:

1. разбиение данных на пакеты;
2. проверку целостности данных;
3. управление потоком;
4. управление перегрузкой сети.

**1.2.2** UDP (User Datagram Protocol)

UDP — это протокол без установления соединения, который используется для передачи данных без гарантии доставки. Он быстрее, но менее надежен, чем TCP. Применяется в ситуациях, где важна скорость, а не гарантированная доставка (например, в потоковом видео или онлайн-играх). [2]

* 1. **Программный интерфейс сокетов**

Сокеты (sockets) — это абстракции, которые позволяют приложениям взаимодействовать с сетью. Это программные интерфейсы, предоставляющие функции для обмена данными между процессами в разных узлах сети.

**1.3.1** Типы сокетов

Сокеты бывают:

1. Потоковые сокеты (Stream Sockets) — используют TCP для надежной передачи данных. Обычно применяются в приложениях, требующих гарантированной доставки данных.
2. Датаграммные сокеты (Datagram Sockets) — используют UDP для быстрой передачи данных без гарантии доставки.

**1.3.2** Структуры данных и функции API

Программный интерфейс сокетов предоставляет различные функции для создания, настройки и работы с сокетами. Например:

1. socket() — создает новый сокет;
2. bind() — связывает сокет с локальным адресом (например, портом);
3. connect() — устанавливает соединение с удаленным хостом;
4. listen() — начинает прослушивание входящих соединений на сервере;
5. accept() — принимает входящее соединение;
6. send() — отправляет данные через сокет;
7. recv() — получает данные через сокет;
8. sendto() — отправляет данные по указанному адресу и порту;
9. recvfrom() — получает данные от конкретного адреса и порта.

**1.3.3** Порядок организации взаимодействия через сокеты

Серверный сокет:

1. сервер создает сокет с помощью socket();
2. привязывает сокет к порту с помощью bind();
3. начинает прослушивание с помощью listen();
4. принимает входящие соединения с помощью accept();
5. после этого сервер может отправлять и получать данные через сокет.

Клиентский сокет:

1. клиент создает сокет с помощью socket();
2. устанавливает соединение с сервером с помощью connect();
3. после установления соединения клиент может обмениваться данными с сервером с помощью send() и recv().

**1.3.4** Способы организации серверов

Однопоточный сервер обрабатывает один запрос за раз. При этом использование асинхронных или неблокирующих вызовов (например, с использованием select() или poll()) позволяет обрабатывать несколько клиентов одновременно.

Многопоточный сервер. Каждый запрос обрабатывается в отдельном потоке, что позволяет серверу одновременно обслуживать несколько клиентов.

Сервер с пулом потоков. Создается пул потоков, который распределяет запросы между свободными потоками, улучшая производительность.

Сервер с использованием неблокирующих сокетов использует неблокирующие сокеты и обрабатывает несколько соединений одновременно с помощью select(), epoll() или других механизмов многозадачности.

Понимание и использование иерархической модели взаимодействия открытых систем, стека протоколов TCP/IP и методов работы с сокетами является основой для разработки современных сетевых приложений. Модель OSI предоставляет теоретическую основу для организации сетевых взаимодействий, разделяя их на четко определенные уровни, что упрощает разработку и диагностику сетевых систем.

Стек протоколов TCP/IP, на основе которого функционирует Интернет и другие глобальные сети, обеспечивает необходимую гибкость и эффективность при организации обмена данными между различными устройствами. Протоколы IP, TCP и UDP решают задачи маршрутизации, надежности и скорости передачи данных, в зависимости от требований конкретных приложений.

В целом, эффективная работа с сетями и протоколами, а также использование сокетов и подходов к организации серверов, открывает широкие возможности для разработки гибких и масштабируемых приложений, способных обеспечивать быстрый и надежный обмен данными в разнообразных сетевых условиях. [3]

**2 ИНСТРУМЕНТАЛЬНАЯ ЯЗЫКОВАЯ СРЕДА**

Для выполнения лабораторной работы был выбран язык разработки С++.

Язык программирования С++ представляет высокоуровневый компилируемый язык программирования общего назначения со статической типизацией, который подходит для создания самых различных приложений. На сегодняшний день С++ является одним из самых популярных и распространенных языков. [4]

В качестве среды разработки Visual Studio.

Win32 API (также называемый Windows API) – это исходная платформа для собственных Windows-приложений на языке C/C++, которым требуется прямой доступ к Windows и оборудованию. Он предоставляет интерфейс разработки первого класса без зависимости от управляемой среды выполнения, такой как .NET и WinRT (для приложений UWP для Windows 10). Благодаря этому API Win32 стает оптимальной платформой для приложений, которым требуется самый высокий уровень производительности и прямой доступ к системному оборудованию.

Файлы заголовков для API Windows позволяют создавать 32-разрядные и 64-разрядные приложения. Они включают объявления для версий API Юникода и ANSI.

Microsoft Visual C++ включает копии файлов заголовков Windows, которые были текущими в момент выпуска Visual C++. Таким образом, если вы устанавливаете обновленные файлы заголовков из пакета SDK, на компьютере может возникнуть несколько версий файлов заголовков Windows. Если вы не уверены, что используете последнюю версию файлов заголовков пакета SDK, при компиляции кода, использующего функции, представленные после выпуска Visual C++, будет получен следующий код ошибки: ошибка C2065: необъявленный идентификатор. [5]

**3 РАБОТА ПРИЛОЖЕНИЯ**

Этот код представляет собой клиентское приложение, которое взаимодействует с сервером через сокеты. Сервер принимает команды от клиента, выполняет их, а клиент получает результаты. Давайте разберем, как работает программа и что она делает шаг за шагом.

**3.1 Структура приложения**

Программа состоит из нескольких частей:

1. Инициализация Winsock. Для работы с сетевыми сокетами в Windows используется библиотека Winsock. Перед использованием сокетов нужно инициализировать эту библиотеку. Это делается с помощью функции WSAStartup.
2. Создание сокета клиента. Сокет — это своего рода "канал", через который клиент будет общаться с сервером. Для этого используется функция socket.
3. Подключение к серверу. С помощью функции connect клиент подключается к серверу, указав его IP-адрес и порт.
4. Чтение команд из файла. Команды, которые клиент отправит на сервер, читаются из текстового файла script.txt.
5. Отправка команд на сервер и получение результатов. Каждая команда отправляется на сервер, и клиент ждёт, пока сервер выполнит команду и вернёт результат. Этот результат клиент выводит на экран.
6. Закрытие соединения. После завершения работы с сервером клиент закрывает сокет и освобождает ресурсы.

**3.2 Функции приложения**

Функции приложения:

1. WSAStartup. В самом начале работы программы инициализируем библиотеку Winsock с помощью функции WSAStartup. Это нужно для того, чтобы использовать сокеты на Windows. Если инициализация не удалась, программа выводит ошибку и завершает выполнение.
2. Socket. Создаётся сокет для клиента с помощью функции socket. Это своего рода "точка подключения" к серверу. Для этого указываем, что будем использовать протокол TCP (надежный поток данных).
3. inet\_pton. Чтобы подключиться к серверу, нам нужно преобразовать строковый IP-адрес (например, "127.0.0.1") в формат, который понимает сеть. Для этого используем функцию inet\_pton.
4. Connect. Функция connect отвечает за подключение клиента к серверу. Передаем ей информацию о сервере — его IP-адрес и порт. Если подключение не удастся, программа выведет ошибку и завершит работу.
5. Чтение команд из файла. Дальше программа открывает файл script.txt, из которого она будет читать команды. Каждая строка файла — это команда, которую мы отправим серверу для выполнения.
6. Отправка команд и получение результатов. Каждую строку из файла отправляем на сервер с помощью функции send(). Затем ждём, пока сервер выполнит команду, и получаем результат через recv(). После этого результат выводится на экран.
7. Закрытие соединения. Когда все команды обработаны, программа закрывает сокет, чтобы завершить соединение с сервером.

**3.3 Анализ данных**

В процессе работы с сервером клиент выполняет следующие операции:

1. Отправка команд. Каждая команда, считанная из файла, отправляется серверу. Мы также удаляем лишние пробелы в конце строк команд, чтобы сервер не получил лишние символы.
2. Получение и обработка результатов. После выполнения команды сервер отправляет результат обратно клиенту. Клиент принимает результат через сокет и выводит его в консоль. Мы также удаляем лишние пробелы в полученном результате.

Таким образом, клиент читает команды из файла, отправляет их на сервер, получает результаты и выводит их пользователю.

В данном примере реализовано клиент-серверное приложение, использующее сокеты для обмена данными между клиентом и сервером. Клиент читает команды из текстового файла, отправляет их на сервер, получает результат выполнения команд и выводит его в консоль. Сервер же, в свою очередь, выполняет эти команды и отправляет клиенту результаты. Программа демонстрирует базовые принципы работы с сетевыми сокетами в Windows с использованием библиотеки Winsock.

Процесс взаимодействия включает несколько ключевых этапов: инициализация сокетов, создание соединения, отправка и получение данных, а также закрытие соединений по завершении работы. Такой подход позволяет легко масштабировать и адаптировать приложение для различных целей, например, для создания более сложных распределённых систем или инструментов удалённого управления.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] ОСетевая модель OSI. – [Электронный ресурс]. – Режим доступа: <https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%82%D0%B5%D0%B2%D0%B0%D1%8F_>%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB%D1%8C\_OSI.

[2] Отличия TCP- и UDP-протоколов — определяем разницу на примерах. – [Электронный ресурс]. – Режим доступа: <https://selectel.ru/> blog/tcp-vs-udp/.

[3] Сокет (программный интерфейс). – [Электронный ресурс]. – Режим доступа: <https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D0%BA%D0%B5%> D1%82\_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D1%8B%D0%B9\_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81).

[4] C++ Tutorial. – [Электронный ресурс]. – Режим доступа: https://www.w3schools.com/cpp/.

[5] Visual Studio. – [Электронный ресурс]. – Режим доступа: <https://visualstudio.microsoft.com/ru/.>

**ПРИЛОЖЕНИЕ** **А**

**(обязательное)**

**Реализация программы на языке С++**

#define WIN32\_LEAN\_AND\_MEAN

#include <windows.h>

#include <winsock2.h>

#include <iostream>

#pragma comment(lib, "ws2\_32.lib")

// Порт прослушивания

#define PORT 54000

int main() {

setlocale(LC\_ALL, "Russian");

// Инициализация сокетов

WSADATA wsaData;

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

if (result != 0) {

std::cerr << "Ошибка инициализации Winsock" << std::endl;

return 1;

}

// Создание сокета сервера

SOCKET serverSocket = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

if (serverSocket == INVALID\_SOCKET) {

std::cerr << "Ошибка создания сокета сервера" << std::endl;

return 1;

}

// Заполнение структуры sockaddr\_in для сокета сервера

SOCKADDR\_IN serverAddr;

serverAddr.sin\_family = AF\_INET;

serverAddr.sin\_port = htons(PORT);

serverAddr.sin\_addr.s\_addr = htonl(INADDR\_ANY);

// Привязка сокета сервера к адресу

if (bind(serverSocket, (SOCKADDR\*)&serverAddr, sizeof(serverAddr)) == SOCKET\_ERROR) {

std::cerr << "Ошибка привязки сокета сервера" << std::endl;

return 1;

}

// Установка сокета сервера на прослушивание

if (listen(serverSocket, SOMAXCONN) == SOCKET\_ERROR) {

std::cerr << "Ошибка установки сокета сервера на прослушивание" << std::endl;

return 1;

}

// Основной цикл сервера, ожидание подключений

while (true) {

// Прием подключения клиента

SOCKET clientSocket = accept(serverSocket, NULL, NULL);

if (clientSocket == INVALID\_SOCKET) {

std::cerr << "Ошибка приема подключения клиента" << std::endl;

continue;

}

// Получение и выполнение команды от клиента

char command[1024];

int commandLength = recv(clientSocket, command, sizeof(command), 0);

if (commandLength <= 0) {

std::cerr << "Ошибка получения команды от клиента" << std::endl;

}

else {

// Удаляем лишние символы в конце строки

std::string commandString(command, commandLength);

commandString.erase(std::remove\_if(commandString.begin(), commandString.end(), [](unsigned char c) { return std::isspace(c); }), commandString.end());

// Выполнение команды в локальной системе

system(commandString.c\_str());

// Отправка результата клиенту

std::string result = "Команда выполнена успешно";

// Удаляем лишние символы в конце строки

result.erase(std::remove\_if(result.begin(), result.end(), [](unsigned char c) { return std::isspace(c); }), result.end());

send(clientSocket, result.c\_str(), result.length(), 0);

}

// Закрытие сокета клиента

closesocket(clientSocket);

}

// Закрытие сокета сервера

closesocket(serverSocket);

// Освобождение ресурсов Winsock

WSACleanup();

return 0;

}