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.253505 Таргонский Д.А.

Проверил:

ассистент кафедры информатики Гриценко Н.Ю.

Минск 2024

**СОДЕРЖАНИЕ**

[1 Цель работы 3](#_Toc179320137)

[2 Краткие теоретические сведения 4](#_Toc179320138)

[3 Инструментальная языковая среда 5](#_Toc179320139)

[4 Описание программного продукта 6](#_Toc179320140)

[4.1 Описание используемых функций 6](#_Toc179320141)

[4.2 Алгоритм работы программы 7](#_Toc179320142)

[5 Результат выполнения программы 8](#_Toc179320143)

[Заключение 9](#_Toc179320144)

[Список использованных источников 10](#_Toc179320145)

[Приложения А (обязательное) исходный код продукта 11](#_Toc179320146)

# **1 ЦЕЛЬ РАБОТЫ**

Цель данной лабораторной работы заключается в изучении и закреплении навыков работы с сетевыми протоколами и программным интерфейсом сокетов, что является важным аспектом сетевого программирования. В условиях современного программирования, где сетевые взаимодействия становятся неотъемлемой частью многих приложений, необходимо понимать, как организовать эффективное взаимодействие между процессами в сетях.

В ходе выполнения лабораторной работы будут исследованы различные подходы к организации сетевого взаимодействия, включая использование протоколов транспортного уровня (TCP и UDP), а также изучение функций API для работы с сокетами. Это включает в себя изучение механизмов, таких как socket(), bind(), connect(), listen(), send(), recv(), sendto(), recvfrom() и других, которые позволяют организовать передачу данных между клиентом и сервером.

Задача состоит в реализации программы, которая:

* Организует сетевое взаимодействие между клиентом и сервером. Это позволит передавать данные по сети, используя различные транспортные протоколы (TCP или UDP).
* Обеспечивает корректное взаимодействие между клиентом и сервером, включая обработку ошибок и управление соединением. Важно реализовать механизмы, которые будут контролировать доступ к сетевым ресурсам, гарантируя, что данные передаются корректно и без потерь.
* Сравнивает эффективность различных транспортных протоколов. В рамках работы будет проведен анализ производительности подходов, использующих TCP и UDP, что поможет понять преимущества и недостатки каждого из протоколов в зависимости от условий задачи.
* Измеряет время выполнения сетевых операций. Программа будет фиксировать время выполнения операций в обоих подходах, что позволит провести количественный анализ и сделать выводы о том, какой протокол более эффективен в различных сценариях.

Таким образом, выполнение данной лабораторной работы не только углубит теоретические знания, но и даст практические навыки, необходимые для разработки сетевых приложений.

2 КРАТКИЕ ТЕОРИТИЧЕСКИЕ СВЕДЕНИЯ

Подсистема сетевого взаимодействия в операционных системах отвечает за организацию связи между процессами, выполняющимися на разных устройствах в сети, и обмен данными между ними. Существует несколько основных методов сетевого взаимодействия: использование сокетов, транспортных протоколов (TCP и UDP), а также сетевых служб и протоколов прикладного уровня. Эти методы позволяют процессам взаимодействовать друг с другом, обеспечивая передачу данных по сети с учетом синхронизации и согласованного доступа к общим ресурсам.

Сокеты представляют собой программный интерфейс для обмена данными между процессами, выполняющимися на разных устройствах в сети. Они позволяют организовать двусторонний обмен данными, используя различные транспортные протоколы, такие как TCP (надежный, ориентированный на соединение) и UDP (ненадежный, ориентированный на датаграммы). Сокеты обеспечивают низкоуровневый доступ к сетевым функциям, что делает их универсальным инструментом для разработки сетевых приложений.

Транспортные протоколы, такие как TCP и UDP, обеспечивают различные уровни надежности и скорости передачи данных. TCP гарантирует доставку данных в порядке их отправки, а также обеспечивает восстановление после потерь пакетов. UDP, напротив, не гарантирует доставку данных и не обеспечивает восстановления после потерь, что делает его более быстрым, но менее надежным.

Сетевые службы и протоколы прикладного уровня, такие как HTTP, FTP, SMTP, POP3 и IMAP, предоставляют высокоуровневый интерфейс для обмена данными между приложениями. Эти службы и протоколы упрощают разработку сетевых приложений, предоставляя готовые решения для выполнения стандартных задач, таких как передача файлов, отправка электронной почты и веб-запросы.

Основные функции для работы с сетевым взаимодействием включают создание и управление сокетами, такими как socket(), bind(), connect(), listen(), send(), recv(), sendto(), recvfrom(). Функция socket() создает сокет, который может использоваться для обмена данными по сети. Функция bind() связывает сокет с определенным адресом и портом, а connect() устанавливает соединение с удаленным сервером. Функции send() и recv() используются для отправки и получения данных через сокет, обеспечивая синхронное взаимодействие.

Эти методы и функции обеспечивают эффективное и безопасное сетевое взаимодействие, что является ключевым аспектом разработки сетевых приложений.

# **3 ИНСТРУМЕНТАЛЬНАЯ ЯЗЫКОВАЯ СРЕДА**

Для разработки программы был выбран язык программирования C++. Это компилируемый, высокопроизводительный язык программирования общего назначения, поддерживающий как низкоуровневые, так и высокоуровневые парадигмы программирования. C++ используется для разработки системного программного обеспечения, приложений, драйверов устройств, а также других программ, требующих высокой эффективности и прямого доступа к системным ресурсам.

В качестве интегрированной среды разработки (IDE) был выбран Microsoft Visual Studio, что является важным решением для разработчиков. Интегрированная среда разработки — это программа, которая объединяет все необходимые инструменты для написания кода, отладки и тестирования в одном интерфейсе, что значительно упрощает процесс разработки. Microsoft Visual Studio является одной из самых популярных IDE для разработки на C++. Она предлагает широкий набор функций, включая редактор кода с подсветкой синтаксиса и автозавершением, инструменты для отладки, встроенные средства для юнит-тестирования и возможности для профилирования и анализа производительности кода.

Разработка осуществляется на Microsoft Windows 10. Использование этой операционной системы позволяет эффективно работать с API и системными вызовами Windows, что важно для разработки программ, которые будут функционировать в этой среде. Windows 10 предоставляет разработчикам мощные инструменты для управления процессами и ресурсами, что критично для создания высокоэффективных приложений.

Вся работа ведется на ноутбуке, что обеспечивает мобильность и возможность работать в разных условиях — будь то офис, дом или в пути. Современные ноутбуки могут обеспечить достаточную производительность для разработки и тестирования программ, особенно если они оснащены хорошими процессорами и достаточным объемом оперативной памяти.

Таким образом, выбор Microsoft Visual Studio в качестве IDE, использование операционной системы Windows 10 и работа на ноутбуке создают оптимальные условия для разработки программ на C++. Это позволяет эффективно использовать все доступные инструменты и ресурсы, что в свою очередь способствует созданию качественного и производительного программного продукта.

4 ОПИСАНИЕ ПРОГРАММНОГО ПРОДУКТА

Программа реализует функциональность аналога сетевой утилиты **ping**, демонстрируя взаимодействие с сетевыми протоколами и использование программного интерфейса сокетов. В данной реализации используется протокол ICMP (Internet Control Message Protocol) для отправки эхо-запросов (ping) на указанный IP-адрес. Программа позволяет отправлять несколько запросов, собирать статистику успешных и неуспешных ответов, а также измерять время отклика. Рассмотрим ключевые функции и алгоритм работы программы.

**4.1 Описание используемых функций**

Далее представлены функции, используемые для работы с сетевым взаимодействием и отправкой ICMP-запросов:

а) WSAStartup(). Инициализирует библиотеку Winsock, которая предоставляет интерфейс для работы с сетевыми функциями в Windows. Эта функция должна быть вызвана перед использованием любых других сетевых функций.

б) IcmpCreateFile(). Создает дескриптор для работы с ICMP-запросами. Этот дескриптор используется для отправки эхо-запросов и получения ответов.

в) IcmpSendEcho(). Отправляет ICMP-эхо-запрос на указанный IP-адрес. Эта функция принимает параметры, такие как дескриптор ICMP, IP-адрес, данные для отправки, размер данных, а также буфер для хранения ответа.

г) InetPtonA(). Преобразует строку с IP-адресом в двоичный формат, который может быть использован для сетевых операций. Эта функция проверяет корректность введенного IP-адреса.

д) GetLastError(). Возвращает код последней ошибки, возникшей при выполнении сетевых операций. Это позволяет диагностировать проблемы, такие как ошибки соединения или превышение времени ожидания.

е) IcmpCloseHandle(). Закрывает дескриптор, созданный функцией IcmpCreateFile(), освобождая системные ресурсы.

ж) WSACleanup(). Завершает работу с библиотекой Winsock, освобождая все ресурсы, выделенные при инициализации.

з) std::cout и std::cerr. Используются для вывода информации о результатах работы программы (успешные и неуспешные запросы) в консоль.

и) std::this\_thread::sleep\_for(). Используется для создания паузы между отправками ICMP-запросов, чтобы имитировать реальное поведение утилиты ping.

Таким образом были описаны все используемые функции в лабораторной работе.

**4.2 Алгоритм работы программы**

Алгоритм работы программы включает несколько ключевых этапов, которые обеспечивают корректное выполнение сетевых операций и сбор статистики.

1. Инициализация сетевой библиотеки. Программа начинает с вызова функции WSAStartup(), которая инициализирует библиотеку Winsock. Это необходимо для использования сетевых функций в Windows.
2. Проверка аргументов командной строки. Программа проверяет, передан ли корректный IP-адрес в качестве аргумента командной строки. Если аргумент отсутствует или не является корректным IP-адресом, программа выводит сообщение об ошибке и завершает работу.
3. Создание дескриптора ICMP. Функция IcmpCreateFile() создает дескриптор для работы с ICMP-запросами. Этот дескриптор используется для отправки эхо-запросов.
4. Отправка ICMP-запросов. В цикле программа отправляет 10 ICMP-эхо-запросов на указанный IP-адрес с помощью функции IcmpSendEcho(). Каждый запрос содержит тестовые данные и ожидает ответа в течение 1 секунды.
5. Обработка ответов. Если ответ получен, программа выводит информацию о времени отклика (RoundTripTime) и TTL (Time to Live). Если ответ не получен, программа выводит сообщение об ошибке и увеличивает счетчик неуспешных запросов.
6. Сбор статистики. После завершения цикла отправки запросов программа выводит статистику успешных и неуспешных запросов. Это позволяет оценить доступность указанного IP-адреса.
7. Завершение работы. Программа закрывает дескриптор ICMP с помощью функции IcmpCloseHandle() и завершает работу с библиотекой Winsock с помощью функции WSACleanup().

Таким образом программа демонстрирует эффективное использование сетевых протоколов и программного интерфейса сокетов для реализации функциональности аналога утилиты ping. Она позволяет отправлять ICMP-запросы, обрабатывать ответы и собирать статистику, что делает её полезной для диагностики сетевых проблем и проверки доступности узлов в сети.

**5 РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ ПРОГРАММЫ**

Программа, реализующая функциональность аналога сетевой утилиты **ping**, успешно выполняет отправку ICMP-запросов на указанный IP-адрес и собирает статистику успешных и неуспешных запросов. В результате работы программы пользователь получает информацию о доступности указанного узла в сети, а также данные о времени отклика и количестве успешных и неуспешных запросов.

На рисунке 5.1 представлен фрагмент выполнения программы, где отправляются ICMP-запросы на IP-адрес 8.8.8.8.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 5.1 – Фрагмент выполнения программы

На рисунке 5.2 представлен фрагмент выполнения программы, где отправляются ICMP-запросы на IP-адрес 127.0.0.1.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 5.2 – Фрагмент выполнения программы

Таким образом, демонстрирует эффективное использование сетевых протоколов и программного интерфейса сокетов для реализации функциональности аналога утилиты ping. Она позволяет отправлять ICMP-запросы, обрабатывать ответы и собирать статистику, что делает её полезной для диагностики сетевых проблем и проверки доступности узлов в сети. Результаты выполнения программы подтверждают корректность работы механизма отправки и обработки ICMP-запросов.

# **ЗАКЛЮЧЕНИЕ**

В ходе выполнения лабораторной работы была разработана программа, реализующая функциональность аналога сетевой утилиты **ping**. Программа позволяет отправлять ICMP-запросы на указанный IP-адрес, обрабатывать ответы и собирать статистику успешных и неуспешных запросов. Этот проект позволил глубоко изучить основы сетевого программирования, включая использование транспортных протоколов, программного интерфейса сокетов и работы с сетевыми протоколами, такими как ICMP.

В процессе работы были изучены ключевые функции для работы с сетевымипротоколами,такие как WSAStartup(), IcmpCreateFile(), IcmpSendEcho(), InetPtonA(), GetLastError(), IcmpCloseHandle() и WSACleanup(). Эти функции позволили организовать отправку и обработку ICMP-запросов, что является важным навыком для разработки сетевых приложений. Программа также демонстрирует работу с протоколом ICMP, который используется для диагностики сетевых проблем и проверки доступности узлов. Это позволило понять, как работает утилита ping и как можно реализовать её функциональность с использованием сетевых API.

Программа собирает статистику успешных и неуспешных запросов, что позволяет оценить доступность указанного IP-адреса. Этот аспект работы демонстрирует практическое применение сетевых технологий для решения реальных задач. В процессе выполнения программы были изучены механизмы обработки ошибок, такие как использование функции GetLastError(). Это позволило понять, как диагностировать проблемы, связанные с сетевым взаимодействием.

Выполненная работа улучшила понимание сетевого программирования, а также подготовила к дальнейшему изучению более сложных тем, таких как асинхронное программирование и сетевое взаимодействие. Эти навыки будут полезны в будущих проектах и помогут в разработке приложений, требующих высокой производительности и надежности.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Build desktop Windows apps using the Win32 API Microsoft Software Incorp. USA. [Электронный ресурс]. – Режим доступа : https://learn.microsoft.com/en-us/windows/win32/

[2] Основные сообщения ОС Windows (Win32 API). Программирование в ОС Windows Microsoft Software Incorp. Лекция 1. [Электронный ресурс]. – Режим доступа : https://www.youtube.com/watch?v=wTArIolxch0

[3] Разработка с помощью WinAPI. [Электронный ресурс]. – Режим доступа : https://shorturl.at/BDJW8

ПРИЛОЖЕНИЕ А

(обязательное)

**Исходный код программы**

#include <iostream>

#include <winsock2.h>

#include <ws2tcpip.h>

#include <iphlpapi.h>

#include <icmpapi.h>

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

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

void send\_ping(const std::string& ip\_address) {

HANDLE hIcmpFile;

DWORD dwRetVal = 0;

char send\_data[32] = "Ping test data";

char reply\_buffer[1024];

ICMP\_ECHO\_REPLY\* pEchoReply = reinterpret\_cast<ICMP\_ECHO\_REPLY\*>(reply\_buffer);

struct sockaddr\_in sa;

if (InetPtonA(AF\_INET, ip\_address.c\_str(), &sa.sin\_addr) != 1) {

std::cerr << "Invalid IP address: " << ip\_address << std::endl;

return;

}

hIcmpFile = IcmpCreateFile();

if (hIcmpFile == INVALID\_HANDLE\_VALUE) {

std::cerr << "Unable to create ICMP handle. Error: " << GetLastError() << std::endl;

return;

}

int Sucsess = 0;

int Unsucsess = 0;

for (int i = 0; i < 10; ++i) {

dwRetVal = IcmpSendEcho(

hIcmpFile,

sa.sin\_addr.S\_un.S\_addr,

send\_data,

sizeof(send\_data),

nullptr,

reply\_buffer,

sizeof(reply\_buffer),

1000

);

if (dwRetVal > 0) {

std::cout << "Reply from " << ip\_address << ": bytes=" << pEchoReply->DataSize

<< " time=" << pEchoReply->RoundTripTime << "ms TTL=" << (int)pEchoReply->Options.Ttl << std::endl;

Sucsess++;

}

else {

std::cerr << "Request timed out or failed. Error: " << GetLastError() << std::endl;

Unsucsess++;

}

}

std::cout << "\nStats of Ping " << ip\_address <<"\n";

std::cout << "Sucsess Pockets: "<< Sucsess<<"\n";

std::cout << "Unsucsess Pockets: "<< Unsucsess <<"\n";

IcmpCloseHandle(hIcmpFile);

}

int main(int argc, char\* argv[]) {

if (argc != 2) {

std::cerr << "Usage: " << argv[0] << " <IP\_ADDRESS>" << std::endl;

return EXIT\_FAILURE;

}

WSADATA wsaData;

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

if (result != 0) {

std::cerr << "WSAStartup failed. Error: " << result << std::endl;

return EXIT\_FAILURE;

}

std::string ip\_address = argv[1];

send\_ping(ip\_address);

WSACleanup();

return 0;

}