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

Фроленко К.Ю.

Проверил: ассистент кафедры информатики Гриценко Н.Ю.

Минск 2024

СОДЕРЖАНИЕ

[1 Формулировка задачи 3](#_Toc181800310)

[2 Краткие теоритеческие сведения 4](#_Toc181800311)

[3 Описание функция программы 5](#_Toc181800312)

[3.1 Функция parseCommandLine 5](#_Toc181800313)

[3.2 Функция resolveHostname 5](#_Toc181800314)

[3.3 Функция traceroute 5](#_Toc181800315)

[3.4 Функция isValidTTL 6](#_Toc181800316)

[3.5 Функция printHelp 6](#_Toc181800317)

[3.6 Функция main 6](#_Toc181800318)

[4 Пример выполнения программы 7](#_Toc181800319)

[4.1 Запуск программы и процесс выполнения 7](#_Toc181800320)

[Вывод 8](#_Toc181800321)

[Список использованных источников 9](#_Toc181800322)

[Приложение А (обязательное) 10](#_Toc181800323)

1 ФОРМУЛИРОВКА ЗАДАЧИ

Целью лабораторной работы является разработка и анализ приложения, демонстрирующего основы сетевого программирования, включая взаимодействие между процессами в сетевой среде посредством сокетов и сетевых протоколов. Основной задачей является исследование методов организации связи и передачи данных в сети, включая использование различных сетевых протоколов и программного интерфейса сокетов. Это позволит улучшить понимание архитектуры сетей и механизмов взаимодействия приложений, а также получить практический опыт работы с сетевыми протоколами и интерфейсами.

В рамках лабораторной работы предполагается выполнение следующих задач:

1**Реализация взаимодействия процессов через сетевые протоколы**. Разработать приложение или несколько приложений, демонстрирующих взаимодействие с использованием сокетов. Это может быть реализация протокола прикладного уровня, такого как *ICMP*, *TCP* или *UDP*, для обмена данными между клиентом и сервером.

2 Изучение и демонстрация сетевых взаимодействий. Провести проверку и демонстрацию различных способов организации серверов и клиентов, таких как потоковые сокеты *TCP* и датаграмные сокеты *UDP*. Рассмотреть методы передачи данных, обеспечения надежности и контроля ошибок.

3 **Анализ и оценка эффективности работы сетевых взаимодействий.** Измерение и сравнение различных подходов к организации связи, таких как использование протоколов *TCP* и *UDP*, их особенности и преимущества в различных сценариях. Анализировать время отклика, пропускную способность и влияние параметров сети на взаимодействие между клиентом и сервером.

В ходе выполнения работы предполагается создание приложения, демонстрирующего сетевое взаимодействие, основанное на использовании протоколов и программного интерфейса сокетов. Это может быть аналог сетевой утилиты, такой как *Traceroute*, которая использует протокол *ICMP* для отслеживания маршрута до удаленного узла, или любая другая утилита для диагностики сети, сканирования портов, измерения задержек и т.д.

Данный проект позволит углубить знания в области сетевого программирования и изучить на практике стеки протоколов, а также методы организации связи между процессами. Также будет проведен анализ особенностей работы различных протоколов и способов их применения в сетевых приложениях, что способствует разработке более надежных и эффективных сетевых решений.

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 сокеты обеспечивают надежную передачу данных с подтверждением и контролем ошибок, а датаграмные сокеты 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

В рамках разработки программы для демонстрации работы с файловой базой данных были реализованы следующие функции:

– Функция *parseCommandLine*;

– функция *resolveHostname*;

– функция *traceroute*;

– функция *isValidTTL*;

– функция *printHelp*;

– функция *main*.

3.1 Функция parseCommandLine

Функция *parseCommandLine* предназначена для обработки аргументов командной строки и выполнения следующих задач:

1 **Анализ аргументов командной строки**, функция принимает параметры командной строки, проверяет их на корректность и извлекает значения начального и максимального *TTL*, а также адрес назначения.

2 **Проверка корректности входных данных**, обеспечивается проверка корректности аргументов, таких как диапазон *TTL* (должен находиться в пределах от 1 до 255).

3.2 Функция resolveHostname

Функция *resolveHostname* выполняет разрешение *DNS*-имени в *IP*-адрес, что необходимо для последующего использования в сетевых операциях:

1 **Инициализация и получение информации о хосте**, с помощью системных вызовов происходит разрешение имени хоста в *IP*-адрес, что позволяет использовать его в последующих сетевых вызовах.

2 **Проверка корректности разрешения**, если указанный хост не может быть разрешён, функция выводит соответствующее сообщение об ошибке.

3.3 Функция traceroute

Функция *traceRoute* выполняет основную функциональность утилиты Traceroute, отслеживая путь пакетов через сеть:

1 **Создание дескриптора *ICMP***, создается дескриптор с помощью функции *IcmpCreateFile()*, необходимый для отправки *ICMP*-запросов.

2 **Отправка *ICMP Echo*-запросов**, функция использует цикл для отправки *ICMP Echo*-запросов с нарастающим значением *TTL*, что позволяет определить маршрут следования пакетов до конечного узла.

3 **Анализ полученных ответов**, на каждом шаге проверяется успешность отправки пакета и его прохождение через маршрутизаторы. Функция выводит IP-адрес каждого узла и время задержки (*Round Trip Time, RTT*).

4 **Обработка ошибок и завершение работы**, если отправка пакета неудачна, выводится символ '\*', указывающий на неуспешную передачу. После завершения всех операций освобождаются все выделенные ресурсы.

3.4 Функция isValidTTL

Функция *isValidTTL* выполняет проверку корректности значения *TTL*, переданного в командной строке. Она анализирует, соответствует ли значение установленным ограничениям (от 1 до 255), что позволяет предотвратить использование некорректных параметров и возможные ошибки в работе программы.

3.5 Функция printHelp

Функция *printHelp* выводит на экран справочную информацию по использованию утилиты. Она включает описание всех возможных параметров командной строки, что делает использование программы более интуитивным и удобным для пользователя.

3.6 Функция main

Функция *main* является входной точкой программы и выполняет следующие основные действия:

1 **Инициализация параметров и разбор командной строки**, на этапе инициализации функция считывает параметры из командной строки, такие как начальный *TTL* *(-f)* и максимальный *TTL (-m)*. Также проверяется корректность указанных значений и назначается *IP*-адрес целевого узла.

2 **Инициализация библиотеки *Winsock***, выполняется инициализация библиотеки *Winsock* для обеспечения работы сетевых функций *Windows*.

3 Запуск функции *traceRoute*, после успешной инициализации всех параметров запускается функция *traceRoute*, которая отслеживает маршрут пакетов и выводит результаты на экран.

4 **Освобождение ресурсов**, после завершения работы библиотеки *Winsock* происходит её корректное завершение, чтобы освободить все задействованные ресурсы.

4 ПРИМЕР ВЫПОЛНЕНИЯ ПРОГРАММЫ

4.1 Запуск программы и процесс выполнения

Программа начинается с инициализации параметров запуска, таких как значение *TTL* и *IP*-адрес назначения. Для демонстрации работы утилиты *Traceroute* был произведен запуск программы с указанием целевого хоста *google.com*. На этапе инициализации выполняется разрешение имени хоста в IP-адрес, инициализация библиотеки *Winsock* и подготовка к отправке *ICMP Echo*-запросов.

После инициализации программа начинает отправлять *ICMP Echo*-запросы с возрастающим значением *TTL*, что позволяет отслеживать маршрут следования пакетов через сеть. На каждом этапе выводится *IP*-адрес промежуточного маршрутизатора и время отклика (*Round Trip Time, RTT*) в миллисекундах. В случае отсутствия отклика выводится символ '\*'.

Рисунок 4.1 демонстрирует результат выполнения программы при трассировке маршрута к *google.com*. На скриншоте показан процесс выполнения команды и все промежуточные узлы, через которые проходят пакеты на пути к целевому узлу.

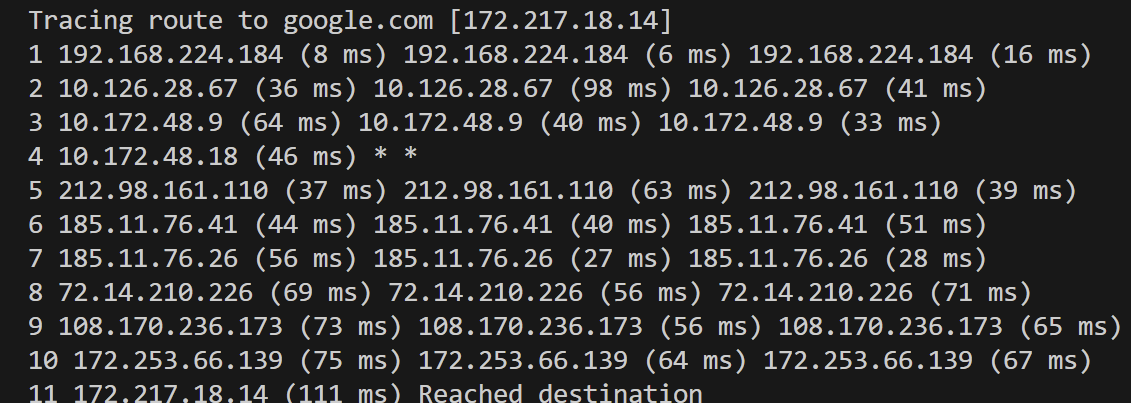
****

Рисунок 4.1 – Консоль с результатами выполнения трассировки маршрута

В ходе выполнения программы *Traceroute* были отслежены промежуточные маршрутизаторы на пути от источника до *google.com*. В каждом из шагов, начиная с минимального значения *TTL* и увеличивая его на единицу, программа последовательно отправляла *ICMP* *Echo*-запросы и получала *ICMP Echo*-ответы, что позволило выявить все маршрутизаторы на данном пути.

Результаты выполнения показали общее количество узлов, через которые проходит маршрут, а также время отклика от каждого из них. Это позволяет оценить задержки на каждом из этапов маршрута и выявить потенциальные проблемы, такие как узлы с высоким временем отклика или потерей пакетов.

# ВЫВОД

В ходе выполнения лабораторной работы была разработана утилита для анализа маршрута сетевых пакетов, основанная на использовании протокола *ICMP* и программного интерфейса сокетов *Windows (WinSock)*. Утилита *Traceroute* позволяет отслеживать маршрут следования пакетов через сеть, обеспечивая получение информации о каждом промежуточном узле, через который проходят пакеты на пути к заданному адресу.

В процессе разработки были изучены и реализованы основные методы взаимодействия с сетевыми протоколами, такие как отправка *ICMP Echo*-запросов и обработка *ICMP Echo*-ответов. Программа позволяет оценить задержки на каждом узле маршрута, выявить потенциальные проблемы с сетевым соединением, такие как высокие задержки или потеря пакетов.

Полученные результаты подтвердили эффективность использования *ICMP* для диагностики сетевых маршрутов и показали, что подобные утилиты могут быть полезны для системных администраторов и сетевых инженеров при выявлении и устранении сетевых неполадок. Разработка утилиты *Traceroute* дала возможность углубить понимание сетевых протоколов и механизмов взаимодействия в сети, что является важным навыком для создания более надежных и производительных сетевых решений.

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: 06.11.2024.

[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: 06.11.2024.

[3] RFC 792: Internet Control Message Protocol (ICMP) [Электронный ресурс]. – Режим доступа: https://tools.ietf.org/html/rfc792. – Дата доступа: 06.11.2024.

ПРИЛОЖЕНИЕ А

(обязательное)

Исходный код программы

#include <bits/stdc++.h>

#include <winsock2.h>

#include <iphlpapi.h>

#include <ws2tcpip.h>

#include <icmpapi.h>

#pragma comment(lib, "Ws2\_32.lib")

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

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

#define PACKET\_SIZE 32

#define MAX\_HOPS 30

#define TIMEOUT\_SEC 1000

#define ll long long

using namespace std;

using namespace chrono;

bool isValidTTL(const string &ttl\_str)

{

try

{

size\_t pos = 0;

ll ttl\_value = stoi(ttl\_str, &pos);

if (pos != ttl\_str.size())

{

return false;

}

return (ttl\_value >= 1 and ttl\_value <= 255);

}

catch (const invalid\_argument &e)

{

return false;

}

}

void printHelp(const char \*programName)

{

cerr << "Usage: " << programName << " [-f first\_ttl] [-m max\_ttl] <destination\_host>" << endl;

cerr << "Options:" << endl;

cerr << " -f, --first-ttl=VALUE Start from the first\_ttl hop (instead from 1)" << endl;

cerr << " -m, --max-ttl=VALUE Set the max number of hops (max TTL to be reached). Default is 30" << endl;

;

cerr << " -h, --help Read this help and exit" << endl;

}

bool parseCommandLine(ll argc, char \*argv[], ll &first\_ttl, ll &max\_ttl, string &destination\_host)

{

for (ll i = 1; i < argc; ++i)

{

if (strcmp(argv[i], "-f") == 0 and i + 1 < argc)

{

if (!isValidTTL(argv[i + 1]))

{

cerr << "First hop value must be in the range 1-255." << endl;

return false;

}

first\_ttl = stoi(argv[++i]);

}

else if (strcmp(argv[i], "-m") == 0 and i + 1 < argc)

{

if (!isValidTTL(argv[i + 1]))

{

cerr << "Max TTL value must be in the range 1-255." << endl;

return false;

}

max\_ttl = stoi(argv[++i]);

}

else if (strcmp(argv[i], "-h") == 0 or strcmp(argv[i], "--help") == 0)

{

printHelp(argv[0]);

return false;

}

else

{

destination\_host = argv[i];

}

}

if (destination\_host.empty())

{

cerr << "Missing destination\_host." << endl;

printHelp(argv[0]);

return false;

}

if (max\_ttl < first\_ttl)

{

cerr << "Max TTL value must be greater than or equal to first TTL value." << endl;

return false;

}

return true;

}

bool resolveHostname(const string &hostname, struct sockaddr\_in &address)

{

struct addrinfo hints = {0};

struct addrinfo \*result = nullptr;

hints.ai\_family = AF\_INET;

hints.ai\_socktype = SOCK\_RAW;

hints.ai\_protocol = IPPROTO\_ICMP;

if (getaddrinfo(hostname.c\_str(), nullptr, &hints, &result) != 0)

{

cerr << "Error resolving destination host." << endl;

return false;

}

address = \*reinterpret\_cast<struct sockaddr\_in \*>(result->ai\_addr);

freeaddrinfo(result);

return true;

}

void traceRoute(const struct sockaddr\_in &addr, ll first\_ttl, ll max\_ttl)

{

HANDLE icmpHandle = IcmpCreateFile();

if (icmpHandle == INVALID\_HANDLE\_VALUE)

{

cerr << "Unable to open ICMP handle: " << GetLastError() << endl;

return;

}

char sendData[PACKET\_SIZE] = {0};

DWORD replySize = sizeof(ICMP\_ECHO\_REPLY) + PACKET\_SIZE;

char \*replyBuffer = new char[replySize];

for (ll ttl = first\_ttl; ttl <= max\_ttl; ++ttl)

{

cout << ttl << " ";

for (ll i = 0; i < 3; ++i)

{

IP\_OPTION\_INFORMATION optionInfo = {0};

optionInfo.Ttl = ttl;

DWORD result = IcmpSendEcho(icmpHandle, addr.sin\_addr.S\_un.S\_addr, sendData, sizeof(sendData), &optionInfo, replyBuffer, replySize, TIMEOUT\_SEC);

if (result != 0)

{

PICMP\_ECHO\_REPLY echoReply = reinterpret\_cast<PICMP\_ECHO\_REPLY>(replyBuffer);

struct in\_addr replyAddr;

replyAddr.S\_un.S\_addr = echoReply->Address;

cout << inet\_ntoa(replyAddr) << " (" << echoReply->RoundTripTime << " ms) ";

if (echoReply->Status == IP\_SUCCESS and echoReply->Address == addr.sin\_addr.S\_un.S\_addr)

{

cout << "Reached destination" << endl;

delete[] replyBuffer;

IcmpCloseHandle(icmpHandle);

return;

}

}

else

{

cout << "\* ";

}

this\_thread::sleep\_for(milliseconds(500));

}

cout << endl;

}

delete[] replyBuffer;

IcmpCloseHandle(icmpHandle);

}

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

{

ll first\_ttl = 1;

ll max\_ttl = MAX\_HOPS;

string destination\_host;

if (!parseCommandLine(argc, argv, first\_ttl, max\_ttl, destination\_host))

{

return 1;

}

WSADATA wsaData;

if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0)

{

cerr << "WSAStartup failed." << endl;

return 1;

}

struct sockaddr\_in dest\_addr;

if (!resolveHostname(destination\_host, dest\_addr))

{

WSACleanup();

return 1;

}

cout << "Tracing route to " << destination\_host << " [" << inet\_ntoa(dest\_addr.sin\_addr) << "]" << endl;

traceRoute(dest\_addr, first\_ttl, max\_ttl);

WSACleanup();

return 0;

}

// g++ -std=c++17 -finput-charset=UTF-8 main.cpp -o build/main -lws2\_32 -liphlpapi -licmp