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 Снежко М.А.

Проверил: ассистент кафедры информатики Гриценко Н.Ю.

Минск 2024

СОДЕРЖАНИЕ

[1 Формулировка задачи 3](#_Toc179742261)

[2 Краткие теоритические сведения 4](#_Toc179742262)

[3 Описание функций программы 5](#_Toc179742263)

[Заключение 6](#_Toc179742264)

[Список использованных источников 7](#_Toc179742265)

[Приложение А (обязательное) 8](#_Toc179742266)

# 1 ФОРМУЛИРОВКА ЗАДАЧИ

Цель лабораторной работы – изучение методов и инструментов сетевого программирования, включая основы построения сетей, стеков протоколов и программных интерфейсов для взаимодействия процессов через сеть. В процессе выполнения лабораторной работы также будут рассмотрены особенности использования сетевых протоколов и сокетов для организации связи между приложениями, а также их применения для диагностики и управления сетями.

В рамках работы необходимо разработать консольное приложение на языке *C++*, моделирующее аналог сетевой утилиты для диагностики сети. Программа выполняет функции сканера портов, который проверяет доступность указанных портов на заданном IP-адресе. В основе работы утилиты используется программный интерфейс сокетов и механизм многопоточности, позволяющий одновременно проверять несколько портов, что значительно ускоряет процесс. Программа должна запрашивать у пользователя IP-адрес и диапазон портов для сканирования. В цикле создаются потоки, каждый из которых проверяет доступность конкретного порта путем попытки установления TCP-соединения. Для обработки результатов используется синхронизация потоков с помощью мьютексов, чтобы обеспечить корректный и упорядоченный вывод данных о состоянии портов (открыт или закрыт).

В реализации предусмотрена инициализация библиотеки WinSock и создание TCP-сокетов для проверки соединения. Для повышения производительности и минимизации задержек задаются ограничения времени ожидания соединения. Благодаря использованию потоков проверка портов выполняется параллельно, что позволяет значительно увеличить скорость работы программы. Для обеспечения корректности вывода данных используется синхронизация с применением мьютексов, что исключает возможность конфликтного доступа к общему ресурсу, например, консоли.

В процессе выполнения работы предполагается исследование параметров утилиты, таких как количество потоков, время отклика для различных диапазонов портов и общая производительность программы при обработке широких диапазонов адресов. Статистика, собранная в ходе выполнения программы, позволит оценить ее эффективность, включая скорость сканирования, количество проверенных портов и точность определения их состояния.

Таким образом, данная лабораторная работа направлена на освоение принципов сетевого взаимодействия, изучение возможностей программного интерфейса сокетов и применение многопоточности для работы с сетевыми приложениями. Полученные знания могут быть использованы для создания более сложных инструментов, таких как анализаторы трафика, сетевые мониторы или утилиты управления сетью.

2 КРАТКИЕ ТЕОРИТИЧЕСКИЕ СВЕДЕНИЯ

Объект синхронизации – это объект, дескриптор которого можно указать в одной из функций ожидания для координации выполнения нескольких потоков. Несколько процессов могут иметь дескриптор одного и того же объекта синхронизации, что делает возможной синхронизацию между процессами [1].

Win32 API (Windows API) представляет собой набор функций и интерфейсов, предоставляемых операционной системой Windows для разработки приложений. Этот мощный набор инструментов обеспечивает доступ к различным функциональным возможностям Windows, включая создание и управление окнами, обработку сообщений, работу с файлами и реестром, а также многие другие операции. Некоторые формы IPC упрощают разделение труда между несколькими специализированными процессами. Другие формы IPC упрощают разделение труда между компьютерами в сети. Win32 API играет ключевую роль в разработке приложений для Windows и обеспечивает высокую степень контроля над поведением приложений [2].

Операционная система Windows предоставляет механизмы для упрощения обмена данными и обмена данными между приложениями. В совокупности действия, включенные этими механизмами, называются межпроцессными коммуникациями (IPC). Некоторые формы IPC упрощают разделение труда между несколькими специализированными процессами. Другие формы IPC упрощают разделение труда между компьютерами в сети.

Сокеты Windows – это независимый от протокола интерфейс. Он использует преимущества возможностей обмена данными базовых протоколов. В сокетах Windows 2 дескриптор сокета можно использовать в качестве дескриптора файлов со стандартными функциями ввода-вывода файлов. Приложение, использующее сокеты Windows, может взаимодействовать с другой реализацией сокетов в других типах систем. Однако не все поставщики услуг транспорта поддерживают все доступные варианты. Cокеты в Windows – это независимый от протокола интерфейс, поддерживающий текущие и новые возможности сети.

Начиная с сборки предварительной оценки Windows 17063, можно использовать семейство адресов сокета UNIX (AF\_UNIX) в Windows для обмена данными между процессами Win32. Сокеты Unix позволяют взаимодействовать между процессами на одном компьютере [3].

3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

Разработанная программа представляет собой многопоточное приложение, выполняющее задачи анализа доступности портов на заданном IP-адресе. Она предназначена для сканирования диапазона портов с использованием сетевых сокетов. Основная задача программы — определить, какие порты открыты и доступны для подключения, а какие закрыты. Для этого каждый порт проверяется с помощью потоков, которые параллельно обрабатывают заданный диапазон, что повышает скорость выполнения сканирования.

В ходе выполнения программы пользователь вводит IP-адрес и диапазон портов для сканирования. На каждый порт создается поток, который инициирует подключение к серверу через сокет. Если соединение успешно установлено, порт считается открытым, и результат отображается в консоли. В случае ошибки подключения порт считается закрытым. Для синхронизации вывода данных в консоль используется мьютекс, что предотвращает одновременный доступ нескольких потоков к стандартному выводу и гарантирует корректное отображение результатов. Настройки тайм-аутов на отправку и получение данных реализованы через функцию setsockopt, что позволяет ограничить время ожидания для каждого подключения. Это особенно важно при сканировании больших диапазонов портов, так как снижает общую задержку программы на закрытых портах. Итогом работы программы является вывод списка портов, где для каждого порта указано его состояние (открыт или закрыт). Результаты сканирования отображаются непосредственно в консоли, что позволяет оперативно анализировать сетевую активность целевого узла. Результат работы программы представлен на рисунке 3.1.

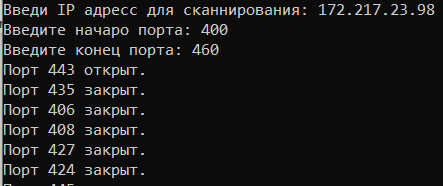


Рисунок 3.1 – Результат работы программы

Таким образом, данная программа демонстрирует использование технологий сетевого программирования и работы с сокетами. Она позволяет изучить основы работы с протоколом TCP, многопоточностью и синхронизацией потоков в рамках реальной задачи сканирования портов. Реализация предоставляет гибкость в выборе диапазона портов и позволяет пользователю наглядно оценить доступность сетевых ресурсов.

# ЗАКЛЮЧЕНИЕ

Таким образом, в рамках данной лабораторной работы была разработана программа для сканирования портов на основе многопоточной обработки. Основная цель работы заключалась в изучении методов сетевого взаимодействия и организации эффективной многопоточной обработки для выполнения сетевых задач. Реализация программы позволила на практике изучить принципы создания параллельных потоков, синхронизации их работы и управления общими ресурсами, такими как консольный вывод.

Разработанная программа представляет собой модель, в которой несколько потоков выполняют сетевые операции, проверяя доступность заданного диапазона портов для указанного IP-адреса. Каждый поток отвечает за попытку подключения к определенному порту, используя сетевые сокеты. В случае успешного подключения порт регистрируется как открытый, иначе он считается закрытым. Для обеспечения корректного доступа к общему ресурсу, каковым является консольный вывод, использовался объект синхронизации.

В ходе выполнения программы исследовались такие параметры, как размер диапазона портов, количество одновременно работающих потоков и настройка тайм-аутов для подключения. Эти параметры оказывали прямое влияние на производительность приложения и время выполнения задачи. Настройка тайм-аутов, реализованная с помощью функций Windows API, таких как setsockopt, позволила минимизировать задержки при обработке закрытых портов и оптимизировать общее время выполнения программы.

Таким образом, лабораторная работа способствовала углублению знаний о принципах работы с протоколом TCP, особенностях многопоточной обработки и использовании инструментов синхронизации. Программа предоставила возможность провести анализ влияния параметров многопоточности на эффективность выполнения задачи и дала практический опыт в проектировании многопоточных приложений с использованием WinAPI.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Microsoft "Synchronization objects" – Электронный ресурс. Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/sync/synchronization-objects.

[2] Щупак Ю. Win32 API. Разработка приложений для Windows. – СПб: Питер, 2008. – 592 с.

[3] Microsoft "Взаимодействие процессов" – Электронный ресурс. Режим доступа: https://learn.microsoft.com/ru-ru/windows/win32/sync/synchronization-objects.

ПРИЛОЖЕНИЕ А

(обязательное)

**Исходный код программы**

#include <iostream>

#include <winsock2.h>

#include <ws2tcpip.h>

#include <thread>

#include <vector>

#include <mutex>

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

std::mutex consoleMutex;

void scanPort(const std::string& ip, int port)

{

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

if (sock == INVALID\_SOCKET)

{

std::cerr << "Error creating socket: " << WSAGetLastError() << std::endl;

return;

}

sockaddr\_in server;

server.sin\_family = AF\_INET;

server.sin\_port = htons(port);

inet\_pton(AF\_INET, ip.c\_str(), &server.sin\_addr);

DWORD timeout = 100;

setsockopt(sock, SOL\_SOCKET, SO\_SNDTIMEO, (const char\*)&timeout, sizeof(timeout));

setsockopt(sock, SOL\_SOCKET, SO\_RCVTIMEO, (const char\*)&timeout, sizeof(timeout));

if (connect(sock, (sockaddr\*)&server, sizeof(server)) == SOCKET\_ERROR)

{

std::lock\_guard<std::mutex> lock(consoleMutex);

std::cout << "Port " << port << " is closed." << std::endl;

}

else

{

std::lock\_guard<std::mutex> lock(consoleMutex);

std::cout << "Port " << port << " is open." << std::endl;

}

closesocket(sock);

}

int main()

{

WSADATA wsaData;

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

{

std::cerr << "WSAStartup failed: " << WSAGetLastError() << std::endl;

return 1;

}

std::string ip;

std::cout << "Enter IP address to scan: ";

std::cin >> ip;

int startPort, endPort;

std::cout << "Enter start port: ";

std::cin >> startPort;

std::cout << "Enter end port: ";

std::cin >> endPort;

if (startPort < 1 || endPort > 65535 || startPort > endPort)

{

std::cerr << "Invalid port range." << std::endl;

WSACleanup();

return 1;

}

std::vector<std::thread> threads;

for (int port = startPort; port <= endPort; ++port)

{

threads.emplace\_back(scanPort, ip, port);

}

for (auto& thread : threads)

{

thread.join();

}

WSACleanup();

return 0;

}