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»
БГУИР 6-05-0612-02 23

Выполнил студент группы 353503
СЕБЕЛЕВ Дмитрий Юрьевич

_____
(дата, подпись студента)

Проверил ассистент каф. информатики
ГРИЦЕНКО Никита Юрьевич

_____
(дата, подпись преподавателя)

Минск 2025

СОДЕРЖАНИЕ

1 ПОСТАНОВКА ЗАДАЧИ

Задачей данной лабораторной работы является разработка и реализация утилиты для мониторинга и диагностики сетевых соединений в операционной системе Windows, которая позволит отслеживать активные сетевые подключения, анализировать состояния TCP/UDP портов и идентифицировать процессы, использующие сетевые ресурсы.

Цель: обеспечить эффективный механизм получения информации о текущих сетевых соединениях локальной системы, предоставить пользователю структурированное представление данных о протоколах, адресах, портах и состояниях подключений, а также идентифицировать процессы-владельцы для целей диагностики и мониторинга сетевой активности.

2 ОПИСАНИЕ РАБОТЫ ПРОГРАММЫ

2.1 Инициализация системных ресурсов

Этот модуль отвечает за настройку среды выполнения и инициализацию необходимых системных компонентов для работы с сетевыми интерфейсами Windows. Программа начинает работу с подключения заголовочных файлов, включающих winsock2.h для базовых сетевых функций, ws2tcpip.h для поддержки протоколов TCP/IP и iphlpapi.h для доступа к расширенным функциям управления IP-интерфейсами. Директивы препроцессора pragma comment обеспечивают автоматическое связывание с системными библиотеками iphlpapi.lib и ws2_32.lib, которые содержат реализацию функций для работы с сетевыми таблицами и сокетами. В функции main выполняется инициализация библиотеки Winsock вызовом WSAStartup с запросом версии 2.2, что является обязательным условием для использования любых сетевых функций Windows API. Функция SetConsoleOutputCP устанавливает кодовую страницу UTF-8 для корректного отображения символов в консоли. После успешной инициализации программа выводит заголовок утилиты «NETWORK MONITOR» с декоративным оформлением и формирует шапку таблицы с колонками для протокола, локального адреса, удаленного адреса, состояния соединения и идентификатора процесса, используя манипуляторы форматирования std::setw для выравнивания данных. Программный код представлен в приложении А [1].

2.2 Вспомогательные функции преобразования данных

Этот модуль реализует служебные функции для конвертации системных представлений сетевых данных в удобочитаемый формат. Функция GetTcpState принимает числовой код состояния TCP-соединения типа DWORD и возвращает строковое представление на основе констант MIB_TCP_STATE, определенных в Windows API. Конструкция switch обрабатывает все возможные состояния протокола TCP согласно модели конечного автомата: CLOSED для закрытого соединения, LISTENING для сокета, ожидающего входящие подключения, SYN_SENT и SYN_RCVD для этапов установления соединения, ESTABLISHED для активного соединения с передачей данных, FIN_WAIT1, FIN_WAIT2, CLOSE_WAIT, CLOSING и LAST_ACK для различных фаз корректного закрытия соединения, TIME_WAIT для состояния ожидания перед окончательным закрытием и DELETE_TCB для соединений, помеченных к удалению. Функция IpToString преобразует IP-адрес из внутреннего формата DWORD в стандартную нотацию с точками, где создается структура in_addr для хранения адреса, поле S_un.S_addr заполняется переданным значением, и функция InetNtopA выполняет преобразование в строку с использованием семейства адресов AF_INET и буфера фиксированного размера INET_ADDRSTRLEN, возвращая результат

в виде объекта std::string или сообщение об ошибке в случае некорректного адреса [2].

2.3 Получение и обработка таблицы TCP-соединений

Этот модуль реализует логику извлечения информации о всех активных TCP-соединениях в системе с использованием расширенного API Windows. Функция ShowTcpTable начинает работу с объявления указателя на структуру MIB_TCPTABLE2, которая содержит массив записей о TCP-соединениях, и переменной ulSize для хранения требуемого размера буфера. Программа выполняет первый вызов GetTcpTable2 с нулевым указателем для определения необходимого объема памяти, после чего функция malloc выделяет соответствующий буфер в куче. Второй вызов GetTcpTable2 с параметром TRUE для автоматической сортировки записей заполняет выделенную структуру актуальными данными о соединениях. Поле dwNumEntries структуры указывает количество записей в таблице, и программа итерируется по массиву table, обрабатывая каждую запись типа MIB_TCPROW2. Для каждого соединения извлекается локальный IP-адрес из поля dwLocalAddr функцией IpToString, локальный порт из поля dwLocalPort с преобразованием из сетевого порядка байтов функцией ntohs, и формируется строка в формате «адрес:порт». Аналогично обрабатывается удаленный адрес и порт из полей dwRemoteAddr и dwRemotePort, при этом если удаленный порт равен нулю, что характерно для соединений в состоянии LISTENING, вместо конкретного адреса выводится символ «:». Состояние соединения определяется вызовом GetTcpState с передачей значения поля dwState, а идентификатор процесса-владельца извлекается из поля dwOwningPid. Все данные форматируются с помощью манипуляторов std::setw для выравнивания по столбцам и выводятся в стандартный поток вывода. По завершении обработки всех записей выделенная память освобождается функцией free [3].

2.4 Получение и обработка таблицы UDP-соединений

Этот модуль обрабатывает информацию о UDP-подключениях, которые имеют отличную от TCP природу и характеристики. Функция ShowUdpTable работает с типом данных MIB_UDPTABLE_OWNER_PID, который предоставляет информацию о UDP-сокетах с включением идентификатора процесса-владельца. В отличие от TCP, для получения расширенной информации о UDP используется функция GetExtendedUdpTable с параметрами UDP_TABLE_OWNER_PID для типа таблицы и AF_INET для семейства адресов IPv4. Первый вызов с нулевым указателем определяет требуемый размер буфера, который сохраняется в переменной ulSize, после чего функция malloc выделяет необходимую память. Повторный вызов GetExtendedUdpTable с параметром TRUE для сортировки записей заполняет структуру данными о всех активных UDP-сокетах в системе. Итерация по массиву table обрабатывает каждую запись типа MIB_UDPROW_OWNER_PID, откуда извлекается локальный адрес через

поле dwLocalAddr и локальный порт через поле dwLocalPort с необходимым преобразованием порядка байтов. Поскольку UDP является протоколом без установления соединения и не поддерживает концепцию состояний, поле удаленного адреса всегда заполняется значением «:», а поле состояния выводится как «—» для обозначения отсутствия состояния соединения. Идентификатор процесса извлекается из поля dwOwningPid для каждой записи. Форматирование выходных данных выполняется аналогично функции ShowTcpTable с использованием манипуляторов std::left и std::setw для создания выровненных столбцов таблицы. После завершения обработки всех UDP-записей память, выделенная под таблицу, освобождается вызовом free для предотвращения утечек памяти [4].

2.5 Координация вывода и завершение работы программы

Этот модуль отвечает за последовательную организацию вывода информации и корректное завершение работы утилиты. После инициализации библиотек и вывода заголовка таблицы главная функция main последовательно вызывает функцию ShowTcpTable для отображения всех TCP-соединений с их полными характеристиками, включая адреса, порты, состояния и идентификаторы процессов. Непосредственно после завершения вывода TCP-данных программа вызывает функцию ShowUdpTable, которая дополняет общую картину сетевой активности системы информацией о UDP-сокетах. Такая последовательность обеспечивает логическую группировку данных по типу протокола, что упрощает восприятие и анализ информации пользователем. Единый формат таблицы с фиксированной шириной столбцов позволяет визуально разделить TCP и UDP записи, сохраняя при этом согласованность представления. После завершения вывода всей информации программа вызывает функцию WSACleanup для корректного освобождения ресурсов библиотеки Winsock и закрытия всех внутренних структур данных, что является обязательным требованием Windows API.

3 ХОД ВЫПОЛНЕНИЯ ПРОГРАММЫ

3.1 Пример выполнения задания

На рисунке 3.1 представлен вывод заголовка утилиты и шапки таблицы с названиями столбцов.

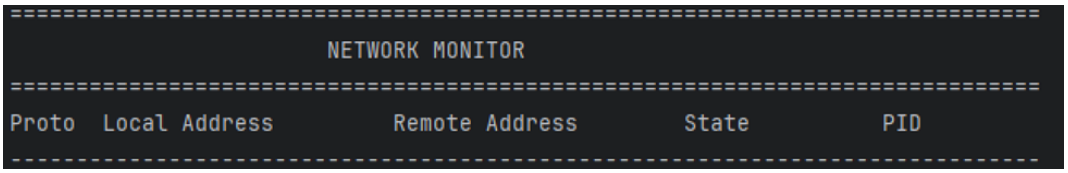

Рисунок 3.1 – Вывод заголовка программы и структуры таблицы

Результат отображения активных TCP-соединений представлен на рисунке 3.2.

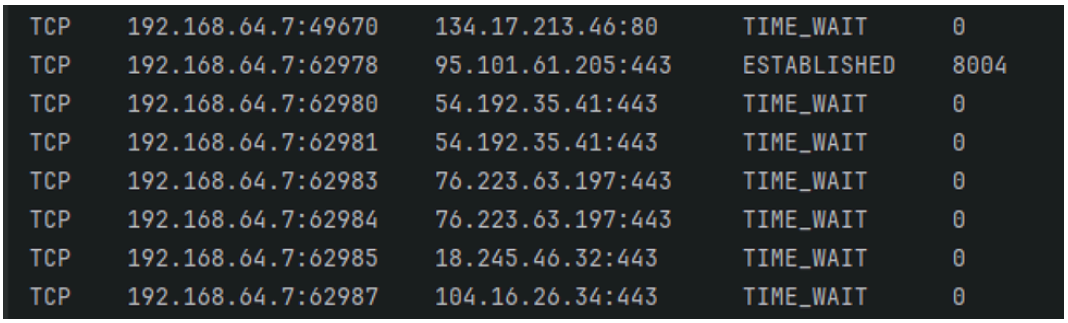

Рисунок 3.2 – Вывод информации о TCP-соединениях

Результат отображения активных UDP-соединений представлен на рисунке 3.3.

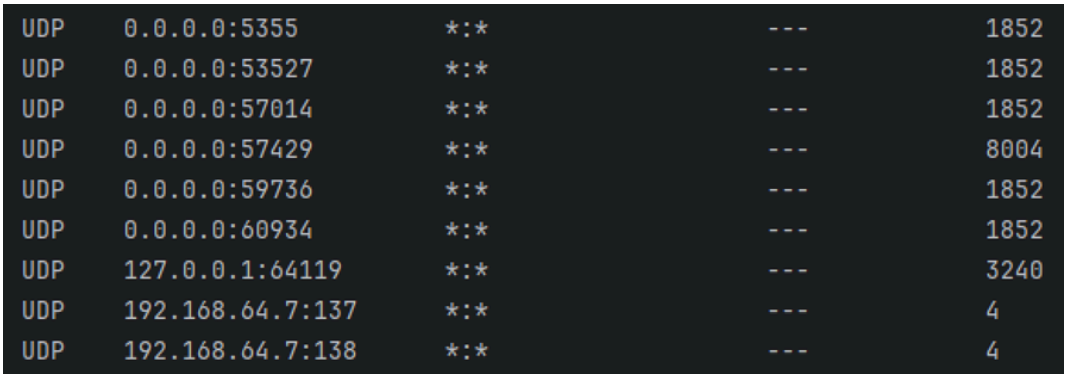

Рисунок 3.3 – Вывод информации о UDP-соединениях

ВЫВОД

В ходе выполнения лабораторной работы были освоены практические навыки разработки сетевых диагностических утилит с использованием Windows API и библиотеки Winsock для получения информации о состоянии сетевых соединений в операционной системе Windows. Была разработана программная утилита мониторинга сетевой активности, демонстрирующая методы работы с системными таблицами TCP и UDP соединений.

Программа реализует функциональность, аналогичную стандартной утилите netstat, позволяя пользователю просматривать все активные сетевые подключения локальной системы. Утилита отображает детальную информацию о каждом соединении, включая тип протокола (TCP или UDP), локальные и удаленные IP-адреса с номерами портов, текущее состояние соединения для протокола TCP и идентификатор процесса-владельца. Данная функциональность является критически важной для задач системного администрирования, диагностики сетевых проблем и мониторинга безопасности.

Для получения информации о сетевых соединениях использовались специализированные функции Windows API из библиотеки IP Helper API. Функция GetTcpTable2 предоставляет расширенную информацию о TCP-соединениях, включая структуру MIB_TCPTABLE2, которая содержит массив записей со всеми характеристиками активных TCP-подключений. Для получения данных о UDP-соединениях применяется функция GetExtendedUdpTable с параметром UDP_TABLE_OWNER_PID, что позволяет получить не только информацию о локальных адресах и портах, но и идентификаторы процессов, использующих UDP-сокеты.

Архитектура программы построена по модульному принципу с выделением отдельных функций для различных задач. Вспомогательные функции преобразования данных обеспечивают конвертацию внутренних системных представлений в удобочитаемый формат: функция IpToString преобразует IP-адреса из числового формата DWORD в стандартную точечную нотацию с использованием функции InetNtopA, а функция GetTcpState транслирует числовые коды состояний TCP-соединений в текстовые описания на основе модели конечного автомата протокола TCP. Функции ShowTcpTable и ShowUdpTable инкапсулируют логику работы с соответствующими таблицами соединений, реализуя двухэтапный алгоритм: определение требуемого размера буфера первым вызовом API-функции, динамическое выделение памяти и повторный вызов для получения актуальных данных.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Windows Sockets 2 Documentation [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/winsock/windows-sockets-start-page-2. – Дата доступа: 27.11.2025.

[2] IP Helper API [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/api/iphlpapi/. – Дата доступа: 27.11.2025.

[3] GetTcpTable2 function [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/api/iphlpapi/nf-iphlpapi-gettcptable2. – Дата доступа: 27.11.2025.

[4] GetExtendedUdpTable function [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/api/iphlpapi/nf-iphlpapi-getextendedudptable. – Дата доступа: 27.11.2025.

ПРИЛОЖЕНИЕ А
(справочное)
Исходный код программы

```
#define _WIN32_WINNT 0x0600

#include <iostream>
#include <winsock2.h>
#include <ws2tcpip.h>
#include <iphlpapi.h>
#include <iomanip>
#include <string>

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

std::string GetTcpState(DWORD state) {
    switch (state) {
        case MIB_TCP_STATE_CLOSED: return "CLOSED";
        case MIB_TCP_STATE_LISTEN: return "LISTENING";
        case MIB_TCP_STATE_SYN_SENT: return "SYN_SENT";
        case MIB_TCP_STATE_SYN_RCVD: return "SYN_RCVD";
        case MIB_TCP_STATE_ESTAB: return "ESTABLISHED";
        case MIB_TCP_STATE_FIN_WAIT1: return "FIN_WAIT1";
        case MIB_TCP_STATE_FIN_WAIT2: return "FIN_WAIT2";
        case MIB_TCP_STATE_CLOSE_WAIT: return "CLOSE_WAIT";
        case MIB_TCP_STATE_CLOSING: return "CLOSING";
        case MIB_TCP_STATE_LAST_ACK: return "LAST_ACK";
        case MIB_TCP_STATE_TIME_WAIT: return "TIME_WAIT";
        case MIB_TCP_STATE_DELETE_TCB: return "DELETE_TCB";
        default: return "UNKNOWN";
    }
}

std::string IpToString(DWORD ipAddress) {
    struct in_addr ipAddr;
    ipAddr.S_un.S_addr = ipAddress;
    char str[INET_ADDRSTRLEN];
    if (InetNtopA(AF_INET, &ipAddr, str, INET_ADDRSTRLEN)) {
        return std::string(str);
    }
    return "Invalid IP";
}

void ShowTcpTable() {
    PMIB_TCPTABLE2 pTcpTable;
    ULONG ulSize = 0;
    GetTcpTable2(NULL, &ulSize, TRUE);
    pTcpTable = (MIB_TCPTABLE2 *) malloc(ulSize);
    if (pTcpTable == NULL) return;
    if (GetTcpTable2(pTcpTable, &ulSize, TRUE) == NO_ERROR) {
        for (int i = 0; i < (int) pTcpTable->dwNumEntries; i++) {
            std::string localIp = IpToString(pTcpTable-
>table[i].dwLocalAddr);
            unsigned short localPort = ntohs((u_short) pTcpTable-
>table[i].dwLocalPort);
            std::string localFull = localIp + ":" +
std::to_string(localPort);

            std::string remoteIp = IpToString(pTcpTable-
>table[i].dwRemoteAddr);
            unsigned short remotePort = ntohs((u_short) pTcpTable-
>table[i].dwRemotePort);
            std::string remoteFull = (remotePort == 0) ? "*:*" :
(remoteIp + ":" + std::to_string(remotePort));

            std::string state = GetTcpState(pTcpTable->table[i].dwState);
            DWORD pid = pTcpTable->table[i].dwOwningPid;
```

```
            std::cout << std::left
                    << std::setw(7) << "TCP"
                    << std::setw(22) << localFull
                    << std::setw(22) << remoteFull
                    << std::setw(15) << state
                    << std::setw(10) << pid << "\n";
        }
    }
    free(pTcpTable);
}

void ShowUdpTable() {
    PMIB_UDPTABLE_OWNER_PID pUdpTable;
    ULONG ulSize = 0;
    GetExtendedUdpTable(NULL, &ulSize, TRUE, AF_INET,
UDP_TABLE_OWNER_PID, 0);

    // 2. Выделяем память
    pUdpTable = (PMIB_UDPTABLE_OWNER_PID) malloc(ulSize);
    if (pUdpTable == NULL) return;
    if (GetExtendedUdpTable(pUdpTable, &ulSize, TRUE, AF_INET,
UDP_TABLE_OWNER_PID, 0) == NO_ERROR) {
        for (int i = 0; i < (int) pUdpTable->dwNumEntries; i++) {
            std::string localIp = IpToString(pUdpTable-
>table[i].dwLocalAddr);
            unsigned short localPort = ntohs((u_short) pUdpTable-
>table[i].dwLocalPort);
            std::string localFull = localIp + ":" +
std::to_string(localPort);
            std::string remoteFull = "*:*";
            std::string state = "---";
            DWORD pid = pUdpTable->table[i].dwOwningPid;

            std::cout << std::left
                    << std::setw(7) << "UDP"
                    << std::setw(22) << localFull
                    << std::setw(22) << remoteFull
                    << std::setw(15) << state
                    << std::setw(10) << pid << "\n";
        }
    }
    free(pUdpTable);
}

int main() {
    SetConsoleOutputCP(65001);

    // Инициализация
    WSADATA wsaData;
    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
        std::cerr << "Winsock init failed.\n";
        return 1;
    }

    std::cout <<
"================================================================================\n";
    std::cout << "                              NETWORK MONITOR
\n";
    std::cout <<
"================================================================================\n";

    std::cout << std::left
            << std::setw(7) << "Proto"
            << std::setw(22) << "Local Address"
            << std::setw(22) << "Remote Address"
            << std::setw(15) << "State"
            << std::setw(10) << "PID" << "\n";
    std::cout <<
"--------------------------------------------------------------------------------
\n";
```

```
    // Сначала выводим TCP
    ShowTcpTable();

    // Затем выводим UDP
    ShowUdpTable();

    WSACleanup();
    system("pause");
    return 0;
}
```