|  |  |  |
| --- | --- | --- |
|  | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования  FPMI_ngtu_neti_rgb_polya«Новосибирский государственный технический университет» | | |
|  | | |
| Кафедра теоретической и прикладной информатики | | |
| Лабораторная работа № 4 | | |
| по дисциплине «Операционные системы и компьютерные сети» | | |
| Анализ структуры кадра технологии Ethernet | | |
|  | | |
|  | Бригада | ВесЕлый Денис |
| №1 | Ворончук Илья |
|  |  |
| Группа | ПМИ-32 |
|  |  |
|  |  |
| Преподаватели | кобылянский в.г. |
|  | сивак м.а. |
| Новосибирск, 2025 | | |

# Приложение

### **main.cpp**

#include <fstream>

#include <iomanip>

#include <iostream>

#include <vector>

// Для кроссплатформенной работы с сетевыми функциями

#ifdef \_WIN32

#include <winsock2.h>

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

#else

#include <arpa/inet.h>  // Для ntohs

#endif

// Выравнивание по 1 байту критически важно, чтобы компилятор не добавил "пустышки" между полями

#pragma pack(push, 1)

// Структура, описывающая заголовок Ethernet (первые 14 байт кадра)

struct EthernetHeader {

    unsigned char dest\_mac[6];

    unsigned char source\_mac[6];

    unsigned short type\_len;

};

// Структура, описывающая IP-заголовок (минимальный размер 20 байт)

// Нам нужны только адреса, поэтому опишем поля до них

struct IpHeader {

    unsigned char ihl\_version;      // Длина заголовка и версия

    unsigned char tos;              // Тип сервиса

    unsigned short total\_len;       // Общая длина пакета

    unsigned short identification;  // Идентификатор

    unsigned short flags\_offset;    // Флаги и смещение

    unsigned char ttl;              // Время жизни

    unsigned char protocol;         // Протокол

    unsigned short checksum;        // Контрольная сумма заголовка

    unsigned char source\_ip[4];     // IP-адрес отправителя

    unsigned char dest\_ip[4];       // IP-адрес получателя

};

#pragma pack(pop)

/\*\*

 \* @brief Выводит MAC-адрес в стандартном формате.

 \* @param mac Указатель на 6-байтовый массив MAC-адреса.

 \*/

void print\_mac\_address(const unsigned char\* mac) {

    for (int i = 0; i < 6; ++i) {

        std::cout << std::hex << std::setw(2) << std::setfill('0') << (int)mac[i];

        if (i < 5) std::cout << ":";

    }

    std::cout << std::dec;  // Возвращаем поток вывода в десятичный режим

}

/\*\*

 \* @brief Выводит IP-адрес в стандартном формате.

 \* @param ip Указатель на 4-байтовый массив IP-адреса.

 \*/

void print\_ip\_address(const unsigned char\* ip) {

    for (int i = 0; i < 4; ++i) {

        std::cout << (int)ip[i];

        if (i < 3) std::cout << ".";

    }

}

int main() {

    std::string filename;

    std::cout << "Enter the name of the .bin file to analyze: ";

    std::cin >> filename;

    std::ifstream file(filename, std::ios::binary);

    if (!file) {

        std::cerr << "Error: Could not open file " << filename << std::endl;

        return 1;

    }

    // Статистика

    int frame\_counter = 0;

    int ip\_frames = 0;

    int arp\_frames = 0;

    int novell\_raw\_frames = 0;

    int llc\_frames = 0;

    int snap\_frames = 0;

    while (!file.eof()) {

        EthernetHeader eth\_header;

        // Читаем заголовок Ethernet (14 байт)

        file.read((char\*)&eth\_header, sizeof(EthernetHeader));

        if (file.gcount() < sizeof(EthernetHeader)) {

            // Если прочитали меньше 14 байт, значит, достигли конца файла

            // или в файле "мусор" в конце

            break;

        }

        frame\_counter++;

        std::cout << "--------------------------------------\n";

        std::cout << "Frame #" << frame\_counter << std::endl;

        // Преобразуем поле Type/Length из сетевого порядка байт (Big-Endian) в хостовый

        unsigned short type\_len = ntohs(eth\_header.type\_len);

        std::cout << "  Source MAC:      ";

        print\_mac\_address(eth\_header.source\_mac);

        std::cout << std::endl;

        std::cout << "  Destination MAC: ";

        print\_mac\_address(eth\_header.dest\_mac);

        std::cout << std::endl;

        long long frame\_data\_len = 0;

        // ------ Логика определения типа кадра ------

        if (type\_len > 1500) {  // Это кадр Ethernet II (DIX)

            std::cout << "  Frame Type:      Ethernet II (DIX)" << std::endl;

            if (type\_len == 0x0800) {  // IP-пакет

                ip\_frames++;

                std::cout << "  Upper Protocol:  IP (0x0800)" << std::endl;

                // Читаем IP заголовок

                IpHeader ip\_header;

                file.read((char\*)&ip\_header, sizeof(IpHeader));

                frame\_data\_len = file.gcount();

                std::cout << "  Source IP:       ";

                print\_ip\_address(ip\_header.source\_ip);

                std::cout << std::endl;

                std::cout << "  Destination IP:  ";

                print\_ip\_address(ip\_header.dest\_ip);

                std::cout << std::endl;

                // Пропускаем оставшуюся часть IP-пакета, чтобы перейти к следующему кадру

                // Общая длина IP-пакета хранится в ip\_header.total\_len

                long remaining\_bytes = ntohs(ip\_header.total\_len) - sizeof(IpHeader);

                if (remaining\_bytes > 0) {

                    file.seekg(remaining\_bytes, std::ios::cur);

                    frame\_data\_len += remaining\_bytes;

                }

            } else if (type\_len == 0x0806) {  // ARP-пакет

                arp\_frames++;

                std::cout << "  Upper Protocol:  ARP (0x0806)" << std::endl;

                // Размер ARP-пакета стандартный - 28 байт

                file.seekg(28, std::ios::cur);

                frame\_data\_len = 28;

            } else {

                std::cout << "  Upper Protocol:  Unknown (0x" << std::hex << type\_len << std::dec << ")" << std::endl;

                // Неизвестный тип Ethernet II. Пропускаем остаток кадра.

                // В реальном анализаторе нужно было бы смотреть на размер пакета,

                // но здесь для простоты считаем его минимальным.

                // Длина поля данных должна быть минимум 46 байт

                file.seekg(46, std::ios::cur);

                frame\_data\_len = 46;

            }

        } else {  // Это кадр IEEE 802.3

            std::cout << "  Frame Length:    " << type\_len << " bytes" << std::endl;

            // Читаем первые два байта поля данных, чтобы определить подтип

            unsigned short data\_header;

            file.read((char\*)&data\_header, sizeof(data\_header));

            if (data\_header == 0xFFFF) {

                novell\_raw\_frames++;

                std::cout << "  Frame Type:      IEEE 802.3 Novell RAW" << std::endl;

            } else if (data\_header == 0xAAAA) {

                snap\_frames++;

                std::cout << "  Frame Type:      IEEE 802.3 SNAP" << std::endl;

            } else {

                llc\_frames++;

                std::cout << "  Frame Type:      IEEE 802.3 LLC" << std::endl;

            }

            // Пропускаем оставшуюся часть поля данных

            // Мы уже прочитали 2 байта из type\_len, так что пропускаем остаток

            if (type\_len > 2) {

                file.seekg(type\_len - 2, std::ios::cur);

            }

            frame\_data\_len = type\_len;

        }

        std::cout << "  Frame Size:      " << sizeof(EthernetHeader) + frame\_data\_len << " bytes" << std::endl;

    }

    // Вывод итоговой статистики

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

    std::cout << "           Analysis Summary\n";

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

    std::cout << "Total frames processed: " << frame\_counter << std::endl;

    std::cout << "\n--- Ethernet II (DIX) Frames ---\n";

    std::cout << "  IP Frames:            " << ip\_frames << std::endl;

    std::cout << "  ARP Frames:           " << arp\_frames << std::endl;

    std::cout << "\n--- IEEE 802.3 Frames ---\n";

    std::cout << "  LLC Frames:           " << llc\_frames << std::endl;

    std::cout << "  Novell RAW Frames:    " << novell\_raw\_frames << std::endl;

    std::cout << "  SNAP Frames:          " << snap\_frames << std::endl;

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

    file.close();

    return 0;

}