**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ   
РОССИЙСКОЙ ФЕДЕРАЦИИ**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ**

**ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»**

**(БГТУ им. В.Г. Шухова)**



ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И УПРАВЛЯЮЩИХ СИСТЕМ

**Лабораторная работа №3**

по дисциплине: Компьютерные сети

тема: «Программирование протокола IP с использованием библиотеки Winsock»

Выполнил: ст. группы ПВ-223

Пахомов Владислав Андреевич

Проверили:

Рубцов Константин Анатольевич

Белгород 2025 г.

**Лабораторная работа №3  
Программирование протокола IP с использованием библиотеки Winsock  
Вариант 6**

**Цель работы:** изучить принципы и характеристику протокола IP и разработать программу для приема/передачи пакетов с использованием библиотеки Winsock.

**Краткие теоретические сведения**

Internet Protocol или IP (англ. internet protocol - межсетевой протокол) - маршрутизируемый сетевой протокол сетевого уровня семейства TCP/IP.

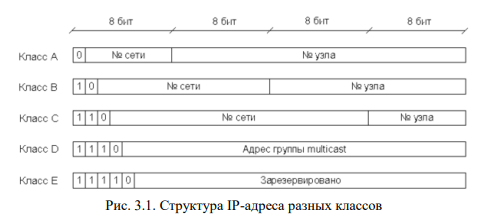
Протокол IP используется для негарантированной доставки данных, разделяемых на так называемые пакеты от одного узла сети к другому. Это означает, что на уровне этого протокола (третий уровень сетевой модели OSI) не даётся гарантий надёжной доставки пакета до адресата. В частности, пакеты могут прийти не в том порядке, в котором были отправлены, продублироваться (когда приходят две копии одного пакета - в реальности это бывает крайне редко), оказаться повреждёнными (обычно повреждённые пакеты уничтожаются) или не прибыть вовсе. Гарантии безошибочной доставки пакетов дают протоколы более высокого (транспортного) уровня сетевой модели OSI - например, TCP - который использует IP в качестве транспорта.

Обычно в сетях используется IP четвёртой версии, также известный как IPv4. В протоколе IP этой версии каждому узлу сети ставится в соответствие IP-адрес длиной 4 октета (1 октет состоит из 8 бит). При этом компьютеры в подсетях объединяются общими начальными битами адреса. Количество этих бит, общее для данной подсети, называется маской подсети (ранее использовалось деление пространства адресов по классам — A, B, C; класс сети определяется диапазоном значений старшего октета и определяет число адресуемых узлов в данной сети).

IP-пакет представляет собой форматированный блок информации, передаваемый по вычислительной сети. Соединения вычислительных сетей, которые не поддерживают пакеты, такие как традиционные соединения типа «точка-точка» в телекоммуникациях, просто передают данные в виде последовательности байтов, символов или битов. При использовании пакетного форматирования сеть может передавать длинные сообщения более надежно и эффективно.

IP-адрес имеет длину 4 байта и обычно записывается в виде четырех чисел, представляющих значения каждого байта в десятичной форме, и разделенных точками, например, 128.10.2.30 – традиционная десятично-точечная форма представления адреса, 10000000 00001010 00000010 00011110 - двоичная форма представления этого же адреса.

Классы сетей IP IP-адреса разделяются на 5 классов: A, B, C, D, E. Адреса классов A, B и C делятся на две логические части: номер сети и номер узла. На рис. 3.1 показана структура IP-адреса разных классов.



Идентификатор сети, также называемый адресом сети, обозначает один сетевой сегмент в более крупной объединенной сети, использующей протокол TCP/IP. IP-адреса всех систем, подключенных к одной сети, имеют один и тот же идентификатор сети. Этот идентификатор также используется для уникального обозначения каждой сети в более крупной объединенной сети.

Идентификатор узла, также называемый адресом узла, определяет узел TCP/IP (рабочую станцию, сервер, маршрутизатор или другое устройство) в пределах каждой сети. Идентификатор узла уникальным образом обозначает систему в том сегменте сети, к которой она подключена.

У адресов класса A старший бит установлен в 0. Длина сетевого префикса 8 бит. Для номера узла выделяется 3 байта (24 бита). Таким образом, в классе A может быть 126 сетей (27 - 2, два номера сети имеют специальное значение). Каждая сеть этого класса может поддерживать максимум 16777214 узлов (224 - 2). Адресный блок класса A может содержать максимум 231 уникальных адресов, в то время как в протоколе IP версии 4 возможно существование 232 адресов. Таким образом, адресное пространство класса A занимает 50% всего адресного пространства протокола IP версии 4. Адреса класса A предназначены для использования в больших сетях, с большим количеством узлов. На данный момент все адреса класса A распределены.

У адресов класса B два старших бита установлены в 1 и 0 соответственно. Длина сетевого префикса – 16 бит. Поле номера узла тоже имеет длину 16 бит. Таким образом, число сетей класса B равно 16384 (214); каждая сеть класса B может поддерживать до 65534 узлов (216 - 2). Адресный блок сетей класса Класс B предназначен для применения в сетях среднего размера. У адресов класса C три старших бита установлены в 1, 1 и 0 соответственно. Префикс сети имеет длину 24 бита, номер узла - 8 бит. Максимально возможное количество сетей класса C составляет 2097152 (221). Каждая сеть может поддерживать максимум 254 узла (28 - 2). Класс C предназначен для сетей с небольшим количеством узлов.

Адреса класса D представляют собой специальные адреса, не относящиеся к отдельным сетям. Первые 4 бита этих адресов равны 1110. Таким образом, значение первого октета этого диапазона адресов находится в пределах от 224 до 239. Адреса класса D используются для многоадресных пакетов, с помощью которых во многих разных протоколах данные передаются многочисленным группам узлов. Эти адреса можно рассматривать как заранее запрограммированные в логической структуре большинства сетевых устройств. Это означает, что при обнаружении в пакете адреса получателя такого типа устройство на него обязательно отвечает. Например, если один из хостов передает пакет с IP-адресом получателя 224.0.0.5, на него отвечают все маршрутизаторы (использующие протокол OSPF), которые находятся в сегменте сети с этим адресом Ethernet.

Адреса в диапазоне 240.0.0.0 - 255.255.255.255 называются адресами класса E. Первый октет этих адресов начинается с битов 1111. Эти адреса зарезервированы для будущих дополнений в схеме адресации IP. Но возможность того, что эти дополнения когда-либо будут приняты, находится под вопросом, поскольку уже появилась версия 6 протокола IP (IPv6).

Некоторые IP-адреса являются зарезервированными. Для таких адресов существуют следующие соглашения об их особой интерпретации:

1. Если все биты IP-адреса установлены в нуль, то он обозначает адрес данного устройства.

2. Если в поле номера сети стоят нули, то считается, что получатель принадлежит той же самой сети, что и отправитель.

3. Если все биты IP-адреса установлены в единицу, то пакет с таким адресом должен рассылаться всем узлам, находящимся в той же сети, что и отправитель. Такая рассылка называется ограниченным широковещательным сообщением.

4. Если все биты номера узла установлены в нуль, то пакет предназначен для данной сети.

5. Если все биты в поле номера узла установлены в единицу, то пакет рассылается всем узлам сети с данным номером сети. Такая рассылка называется широковещательным сообщением.

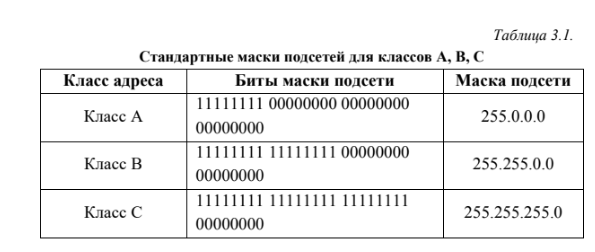
6. Если первый октет адреса равен 127, то адрес обозначает тот же самый узел. Такой адрес используется для взаимодействия процессов на одной и той же машине (например, для целей тестирования). Этот адрес имеет название возвратного.

Поля номеров сети и подсети образуют расширенный сетевой префикс. Для выделения расширенного сетевого префикса используется маска подсети (subnet mask).

Маска подсети – это 32-разрядное двоичное число, в разрядах расширенного префикса содержащая единицу; в остальных разрядах находится ноль. Расширенный сетевой префикс получается побитным сложением по модулю два (операция XOR) IP-адреса и маски подсети.

При таком построении очевидно, что число подсетей представляет собой степень двойки – 2n, где n - длина поля номера подсети. Таким образом, характеристики IP-адреса полностью задаются собственно IP-адресом и маской подсети.

Стандартные маски подсетей для классов А, В, С приведены в табл. 3.1.



Для упрощения записи применяют следующую нотацию (так называемая CIDR-нотация): IP-адрес/длина расширенного сетевого префикса. Например, адрес 192.168.0.1 с маской 255.255.255.0 будет в данной нотации выглядеть как 192.168.0.1/24 (24 – это число единиц, содержащихся в маске подсети).

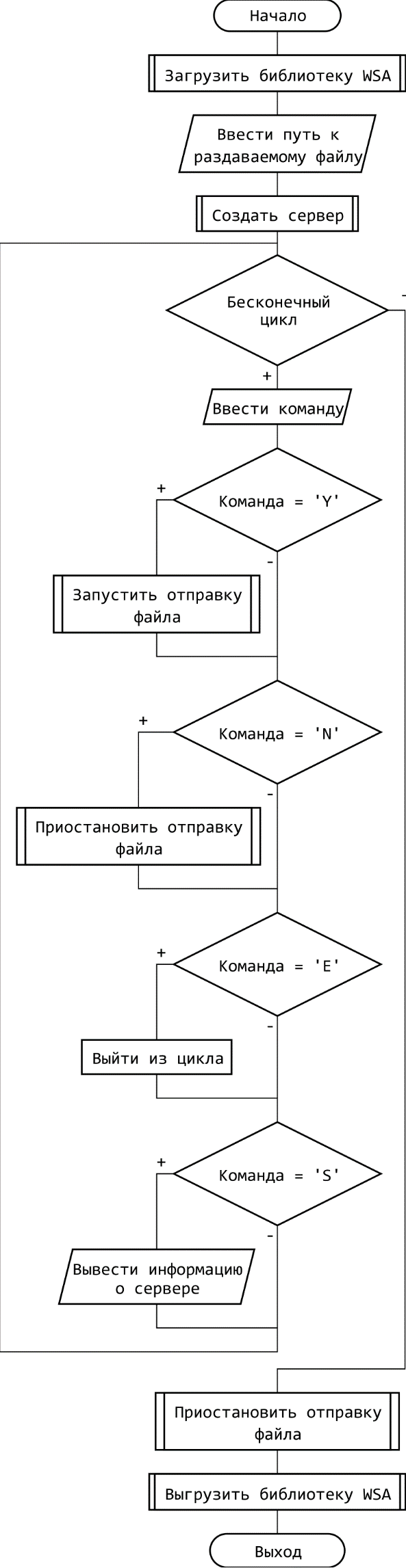
Для разбития сети на подсети необходимо найти минимальную степень двойки, большую или равную числу требуемых подсетей. Затем эту степень прибавить к префиксу сети. Количество IP-адресов в каждой подсети будет на 2 меньше теоретически возможного, потому что сеть должна будет вместить адрес сети и броадкастовый адрес.

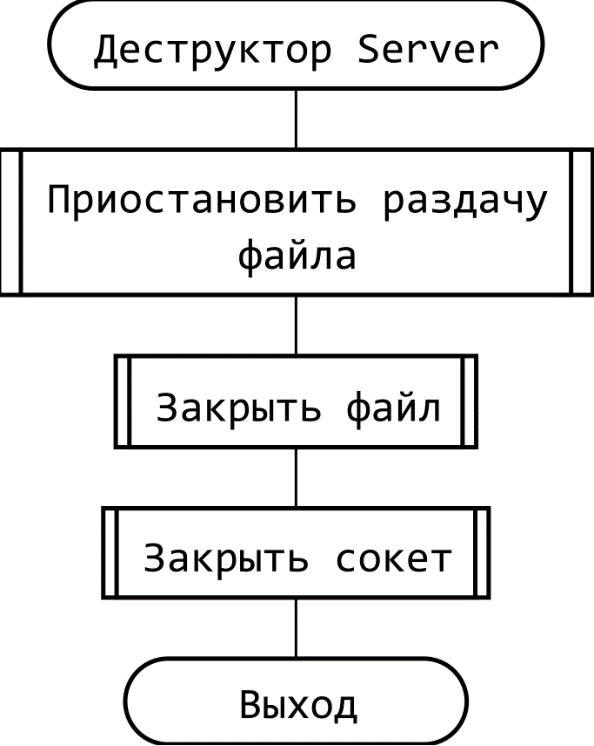
**Основные функции API для работы с протоколом IPX**

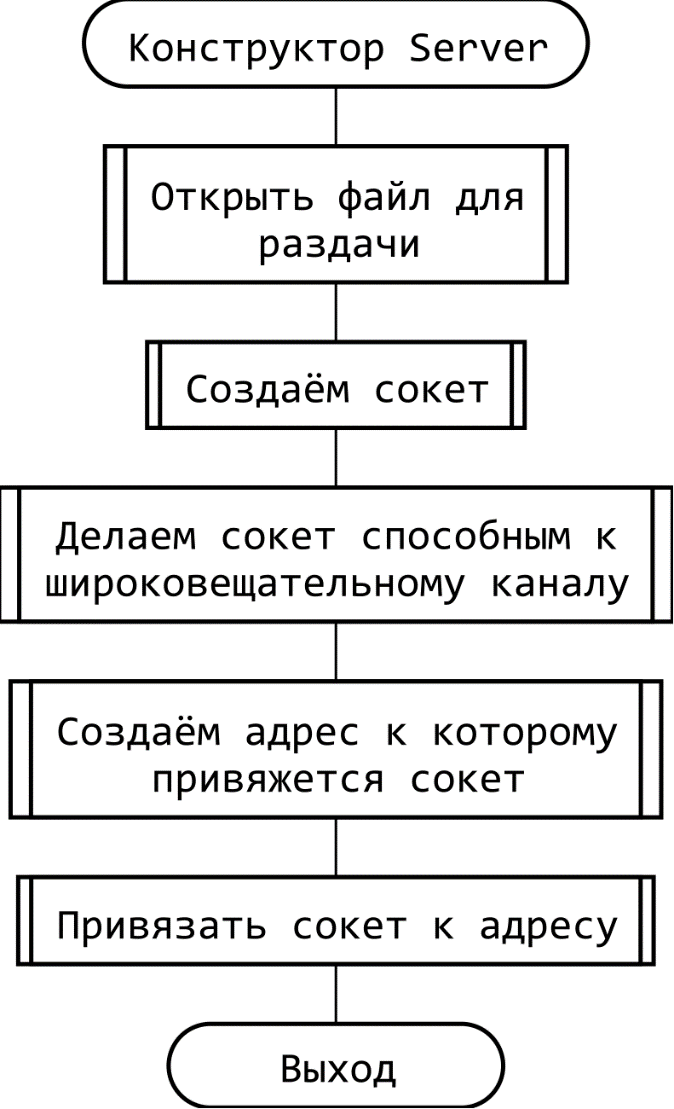
* Функция WSAStartup (WORD wVersionRequested, LPWSADATA lpWSAData) необходима для инициализации библиотеки Winsock. Здесь wVersionRequested – запрашиваемая версия winsock; lpWSAData – структура, в которую возвращается информация по инициализированной библиотеке (статус, версия и пр.). В случае успеха возвращает 0, иначе возвращает код возникшей ошибки.
* Функция WSAGetLastError (void) возвращает код ошибки возникшей при выполнении последней операции.
* Функция WSACleanup (void) очищает память, занимаемую библиотекой Winsock. Возвращает 0, если операция была выполнена успешно, иначе возвращает SOCKET\_ERROR и номер ошибки можно получить при помощи функции WSAGetLastError.
* Функция u\_short htons (u\_short hostshort) осуществляет перевод целого короткого числа из порядка байт, принятого на компьютере, в сетевой порядок байт. hostshort – число, которое необходимо преобразовать. Возвращает преобразованное число.
* Функция socket (int af, int type, int protocol) нужна для создания и инициализации сокета. Здесь af – сведения о семействе адресов. Для интернет-протоколов указывается константа AF\_INET; type – тип передаваемых данных (поток или дейтаграммы). В данной лабораторной работе используется константа SOCK\_DGRAM; protocol – протокол передачи данных. Для протокола IP используется константа IPPROTO\_IP. Функция возвращает дескриптор созданного сокета.
* Функция bind (SOCKET s, const struct sockaddr FAR\* name, int namelen) привязывает адрес и порт к ранее созданному сокету Здесь s – дескриптор сокета; name – структура, содержащая нужный адрес и порт; namelen – размер, в байтах, структуры name.
* Функция unsigned long inet\_addr (const char FAR \*cp) конвертирует строку в значение, которое можно использовать в структуре sockaddr\_in. Здесь cp – строка, которая содержит IP адрес в десятично-точечном формате (например, 123.23.45.89). Возвращает IP адрес в виде целого числа, либо если произошла ошибка возвращает константу INADDR\_NONE. Для конвертации адреса в стандартный формат используется функция char (FAR \* inet\_ntoa(struct in\_addr in); in – IP-адрес, заданный в сетевом порядке расположения байт. Она возвращает строку, содержащую IP-адрес в стандартном строчном виде, с числами и точками.
* Для определения IP адреса по имени используется функция struct hostent FAR \* gethostbyname (const char FAR \* name). В качестве результата, функция возвращает структуру hostent.
* В случае автоматического распределения адресов и портов узнать какой адрес и порт присвоен сокету можно при помощи функции getsockname (SOCKET s, struct sockaddr FAR\* name, int FAR\* namelen). Если операция выполнена успешно, возвращает 0, иначе 35 возвращает SOCKET\_ERROR и номер ошибки можно получить при помощи функции WSAGetLastError.
* Функция gethostname (char FAR \* name, int namelen) записывает в буфер name длинной namelen стандартное имя хоста для данного компьютера. В случае успеха возвращает 0, иначе возвращает SOCKET\_ERROR.
* Функция sendto (SOCKET s, const char FAR \* buf, int len, int flags, const struct sockaddr FAR \* to, int tolen) служит для передачи данных. Здесь s - дескриптор сокета; buf - указатель на буфер с данными, которые необходимо переслать; len - размер (в байтах) данных, которые содержатся по указателю buf; flags - совокупность флагов, определяющих, каким образом будет произведена передача данных; to - указатель на структуру sockaddr, которая содержит адрес сокета-приёмника; tolen - размер структуры to. Если операция выполнена успешно, возвращает количество переданных байт, иначе возвращает SOCKET\_ERROR и номер ошибки можно получить при помощи функции WSAGetLastError.
* Функция recvfrom (SOCKET s, char FAR\* buf, int len, int flags, struct sockaddr FAR\* from, int FAR\* fromlen) служит для приема данных. Если операция выполнена успешно, возвращает количество полученных байт, иначе возвращает SOCKET\_ERROR и номер ошибки можно получить при помощи функции WSAGetLastError.
* Функция closesocket (SOCKET s) нужна для закрытия сокета. Она возвращает 0, если операция была выполнена успешно, иначе возвращает SOCKET\_ERROR и номер ошибки можно получить при помощи функции WSAGetLastError.

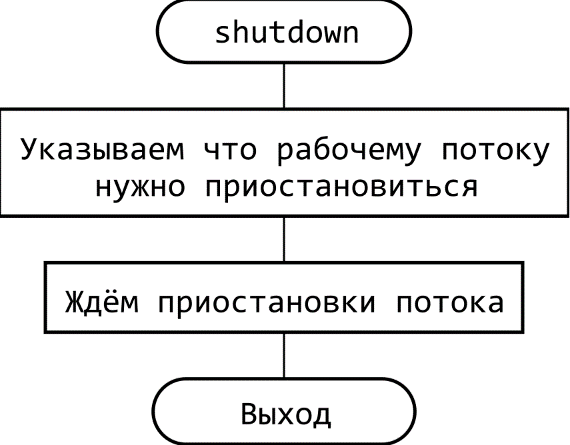
**Разработка программы. Блок-схемы программы.**

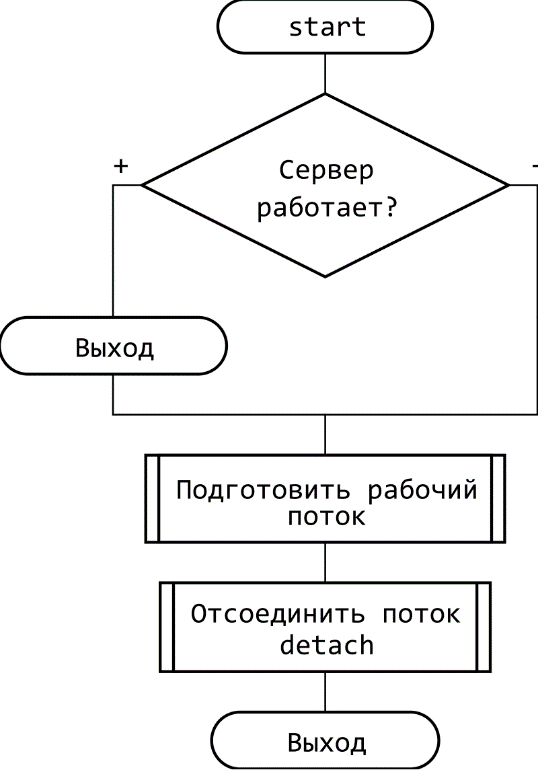
**Сервер**

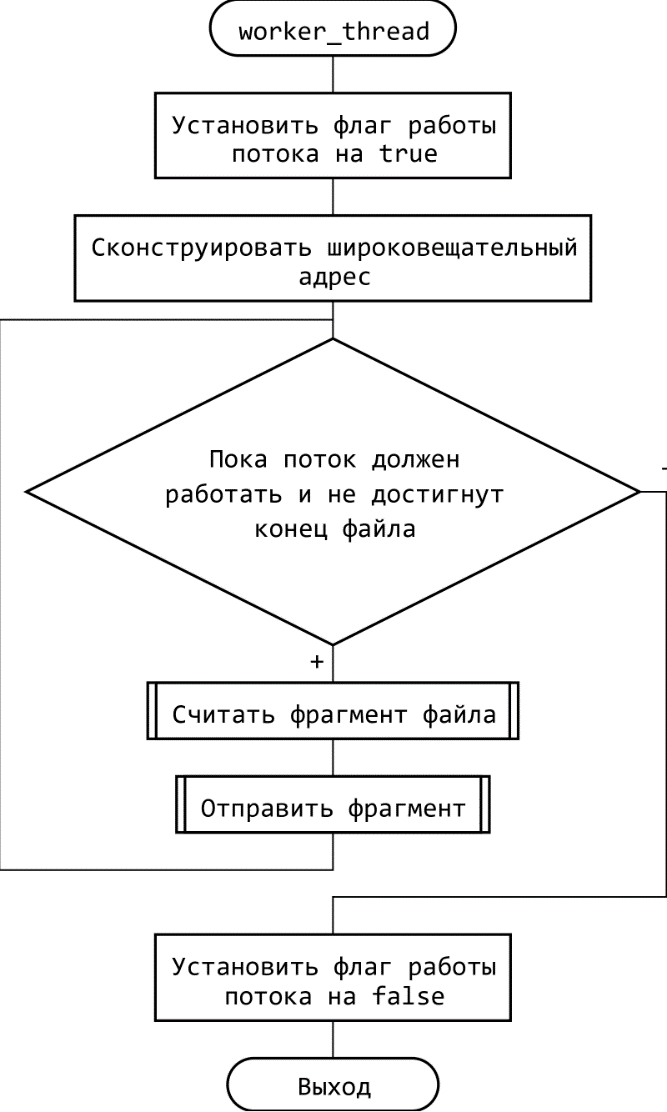
****

****

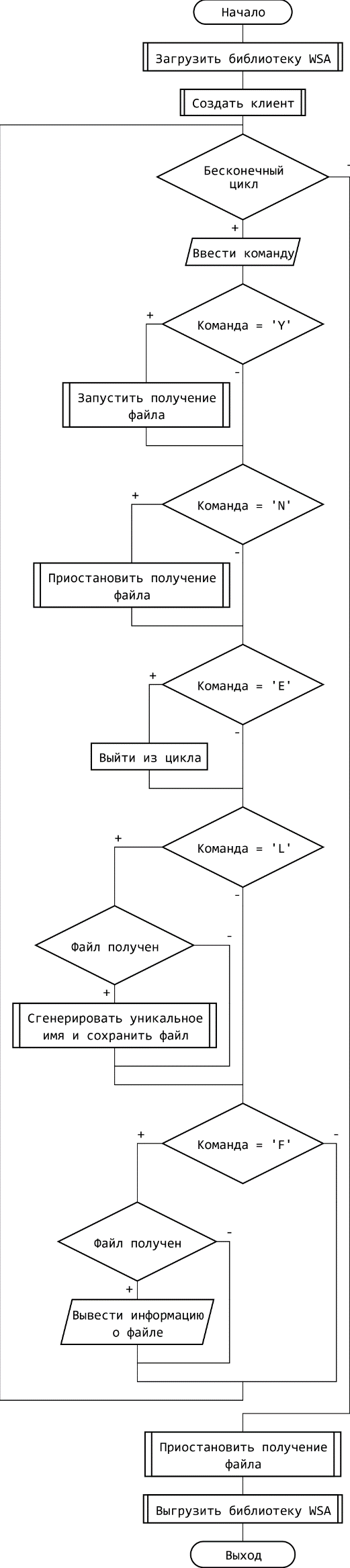
****

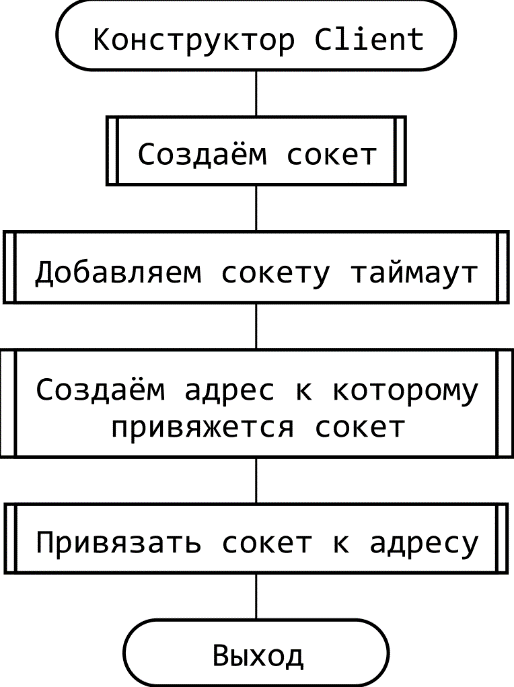
****

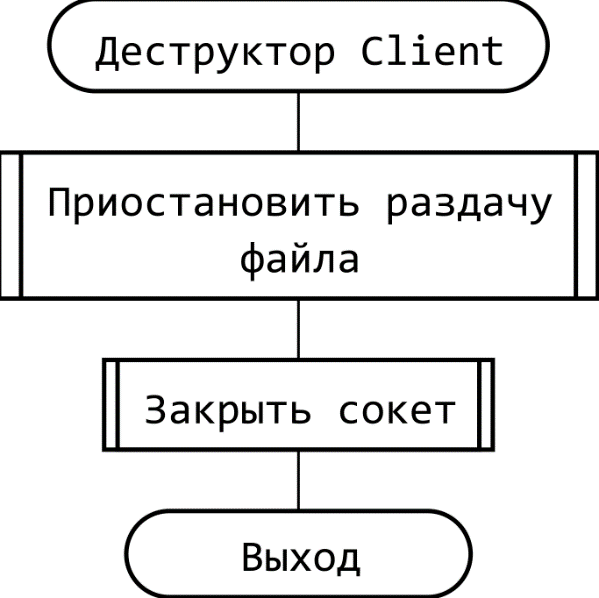
****

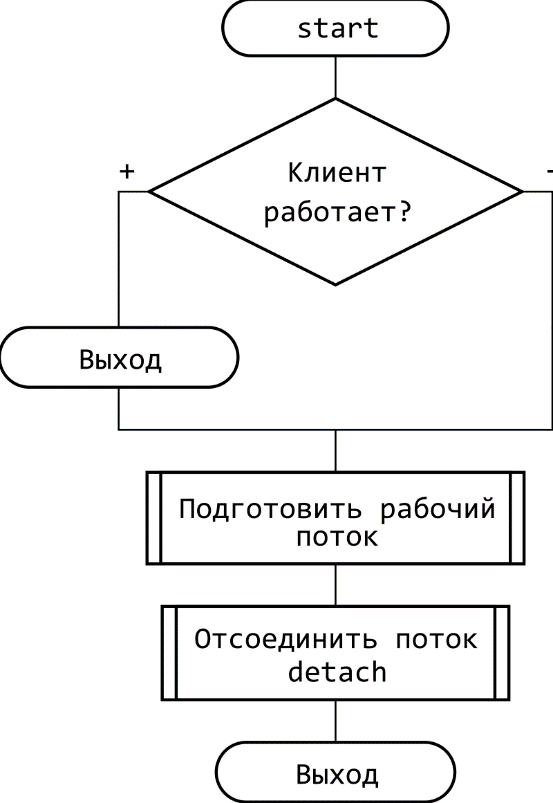
****

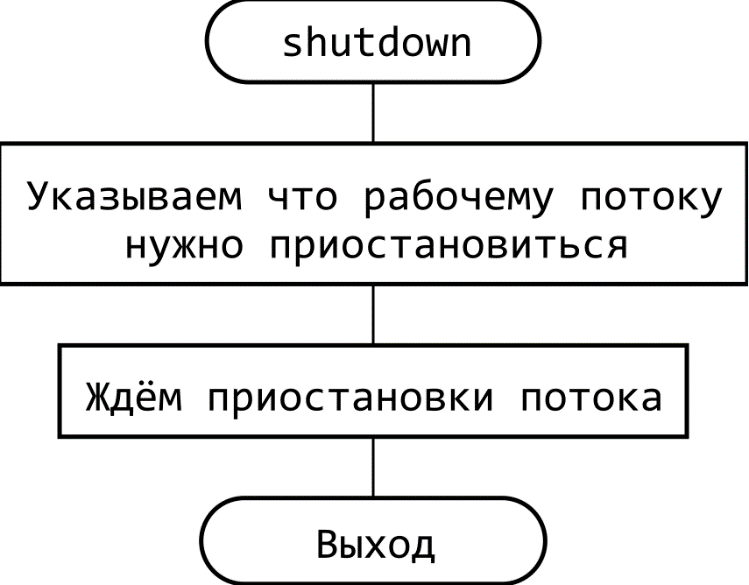
**Клиент**

****

****

****

****

****

**Анализ функционирования программ**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Время, сек.** |  | **Размер файла, МиБ** |  | **Скорость передачи, Мбит/с** |
| №1 | 9,860 |  | 593,04101562500000 |  | 478,66420364379800 |
| №2 | 9,776 |  |  |  |  |
| №3 | 10,006 |  |  |  |  |
| №4 | 10,114 |  |  |  |  |
| №5 | 9,802 |  |  |  |  |
| **Среднее** | 9,912 |  |  |  |  |
| **Дисперсия** | 0,021 |  |  |  |  |
|  |  |  |  |  |  |
|  | **Объём (Virtual Box), МиБ** |  | **Объём (ноутбук), МиБ** |  |  |
| №1 | 144,77943801879800 |  | 6,33197784423828 |  |  |
| №2 | 124,54181671142500 |  | 5,26857376098632 |  |  |
| №3 | 148,39307403564400 |  | 5,85187149047851 |  |  |
| №4 | 123,90086746215800 |  | 6,45842361450195 |  |  |
| №5 | 153,90678787231400 |  | 6,29322052001953 |  |  |
| №6 | 139,83787918090800 |  |  |  |  |
| №7 | 141,66141128540000 |  |  |  |  |
| №8 | 150,49905014038000 |  |  |  |  |
| **Среднее** | 140,94004058837800 |  | 6,04081344604492 |  |  |
| **Средние потери, %** | 76% |  | 99% |  |  |

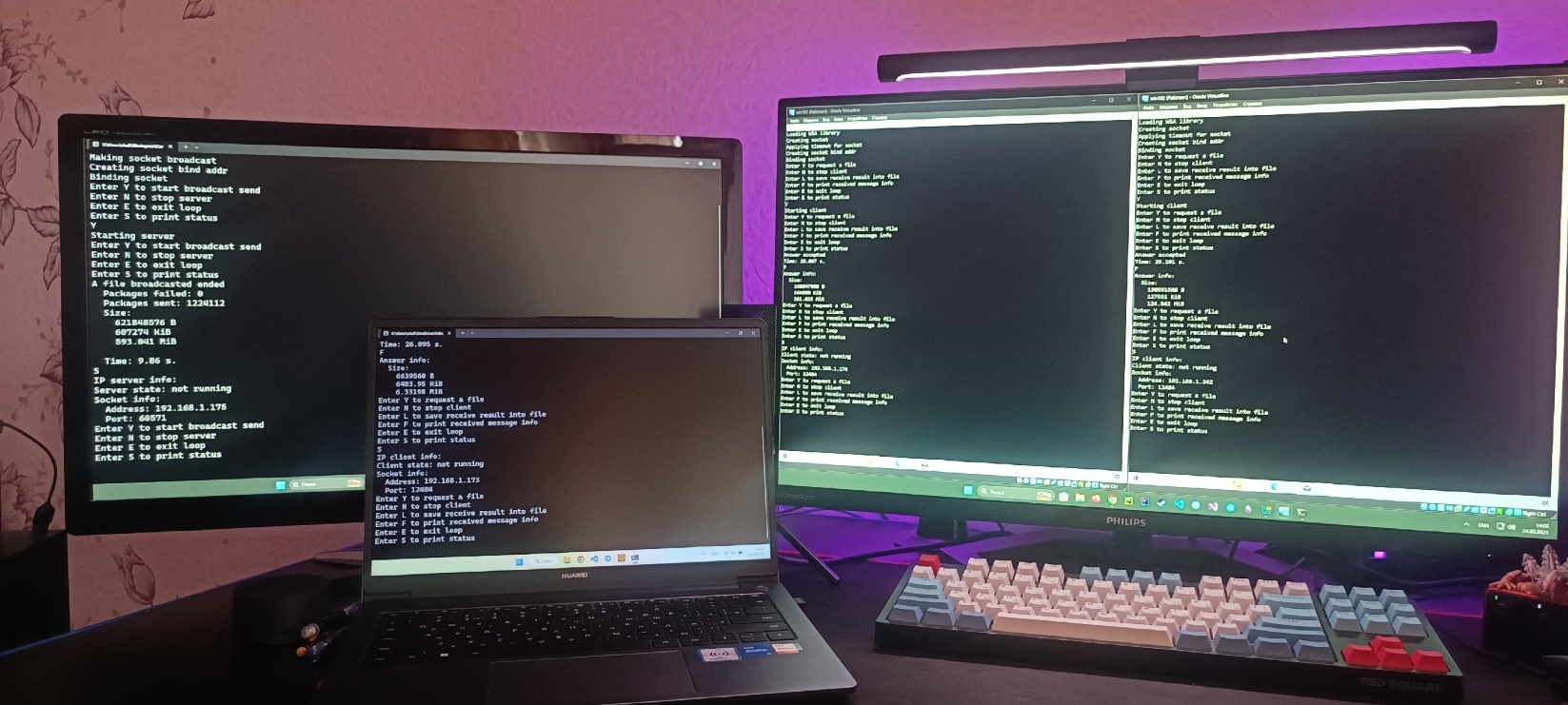
Скорость передачи информации многократно выросла по сравнению с прошлой лабораторной работой и приблизилась к возможностями компьютеров (для ПК Intel® WiFi 6, Realtek 2.5Gb Ethernet, TUF LANGuard, для ноутбука IEEE 802.11a/b/g/n/ac/ax,160 MHz) и роутера (TP-Link Archer AX1800 с максимальной скоростью 1000 Мбит/с).

Производительность сильно выросла, что позволило передавать гораздо большие по объёму данные, однако потери также выросли. Для передачи по сети «по проводу» процент потерь составил 76%, а для передачи по Wi-Fi - 99%. Такие потери связаны с тем, что скорость передачи данных с сервера (запускался с ПК с хоста) гораздо выше чем скорость приёма и на виртуальной машине и на ноутбуке.

**Вывод:** в ходе лабораторной изучили принципы и характеристику протокола IP и разработать программу для приема/передачи пакетов с использованием библиотеки Winsock.

**Текст программ. Скриншоты программ.**

Ссылка на репозиторий с кодом: <https://github.com/IAmProgrammist/comp_net/tree/lab3>



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Исходное изображение | Получено на виртуальной машине (локальное подключение по Ethernet-кабелю) | | Получено на ноутбуке (Wi-Fi подключение) | |
|  |  |  |  |  |
|  |  |  |  |

#include <iostream>

#include <WinSock2.h>

#include <cstdio>

#include <fstream>

#include "client.h"

#include "../shared.h"

Client::Client() {

    std::clog << "Creating socket" << std::endl;

    // Создаём сокет

    this->socket\_descriptor = socket(

        AF\_INET,

        SOCK\_DGRAM,

        IPPROTO\_IP

    );

    std::clog << "Applying timeout for socket" << std::endl;

    // Добавляем сокету таймаут

    int timeout\_ms = 10000;

    if (setsockopt(this->socket\_descriptor, SOL\_SOCKET, SO\_RCVTIMEO, (char\*)&timeout\_ms, sizeof(timeout\_ms)) == SOCKET\_ERROR)

        throw std::runtime\_error(getErrorTextWithWSAErrorCode("Unable to make socket broadcast"));

    std::clog << "Creating socket bind addr" << std::endl;

    // Создаём адрес к которому привяжется сокет

    sockaddr\_in bind\_addr;

    bind\_addr.sin\_family = AF\_INET;

    bind\_addr.sin\_addr = getDeviceAddrInfo().sin\_addr;

    bind\_addr.sin\_port = htons(CLIENT\_DEFAULT\_PORT);

    std::clog << "Binding socket" << std::endl;

    // Привязать сокет к адресу

    if (bind(this->socket\_descriptor, (sockaddr\*)&bind\_addr, sizeof(bind\_addr)) == SOCKET\_ERROR)

        throw std::runtime\_error(getErrorTextWithWSAErrorCode("Unable to bind socket descriptor"));

}

Client::~Client() {

    std::clog << "Stopping worker thread" << std::endl;

    // Приостанавливаем рабочий поток

    this->shutdown();

    std::clog << "Closing socket" << std::endl;

    // Закрываем сокет

    if (closesocket(this->socket\_descriptor) == SOCKET\_ERROR)

        throw std::runtime\_error(getErrorTextWithWSAErrorCode("Unable to close socket"));

}

void Client::request(char\* payload, int payload\_size) {

    // Если клиент уже работает, выходим из него

    if (this->running) {

        std::clog << "Client is already running" << std::endl;

        return;

    }

    std::clog << "Starting client" << std::endl;

    // Подготавливаем рабочий поток

    delete this->current\_runner;

    this->should\_run = true;

    this->temporary\_data.clear();

    this->current\_runner = new std::thread([this]() {

        // Устанавливаем флаг работы потока на true

        this->running = true;

        char buffer[IMAGE\_FRAGMENT\_SIZE];

        int bytes\_received;

        auto a = std::chrono::high\_resolution\_clock::now();

        // Получаем сегменты и сохраняем их в буфер

        while (should\_run && (bytes\_received = recvfrom(

            this->socket\_descriptor,

            buffer,

            sizeof(buffer),

            0,

            nullptr,

            nullptr)) != SOCKET\_ERROR) {

            this->temporary\_data.insert(this->temporary\_data.end(), buffer, buffer + sizeof(buffer));

        }

        auto b = std::chrono::high\_resolution\_clock::now();

        std::clog << "Answer accepted\n" <<

            "Time: " << std::chrono::duration\_cast<std::chrono::milliseconds>(b - a).count() / 1000.0 << " s." << std::endl;

        // Устанавливаем флаг работы потока на false

        this->running = false;

    });

    this->current\_runner->detach();

}

bool Client::isReady() {

    return !this->running;

}

const std::vector<char>& Client::getAnswer() {

    return this->temporary\_data;

}

void Client::wait\_for\_client\_stop() {

    while (this->running) {

    }

}

void Client::shutdown() {

    if (!this->running) {

        std::clog << "Client is already stopped" << std::endl;

        return;

    }

    std::clog << "Stopping client" << std::endl;

    // Указываем что клиенту нужно приостановиться

    // и ждём пока он остановится

    this->should\_run = false;

    wait\_for\_client\_stop();

    delete this->current\_runner;

}

std::ostream& Client::printClientInfo(std::ostream& out) {

    sockaddr\_in client\_address;

    int client\_address\_size = sizeof(client\_address);

    int get\_sock\_name\_res = getsockname(this->socket\_descriptor, (sockaddr\*)&client\_address, &client\_address\_size);

    out << "IP client info:\n" <<

        "Client state: " << (this->running ? "running" : "not running") << "\n";

    if (get\_sock\_name\_res == SOCKET\_ERROR) {

        out << "Unable to get socket info\n";

    }

    else {

        out << "Socket info:\n";

        printSockaddrInfo(out, client\_address);

        out << "\n";

    }

    out.flush();

    return out;

}

std::ostream& Client::printAnswerInfo(std::ostream& out) {

    out << "Answer info:\n" << "  Size:\n";

    out << "    " << this->temporary\_data.size() << " B\n";

    out << "    " << this->temporary\_data.size() / 1024.0 << " KiB\n";

    out << "    " << this->temporary\_data.size() / 1024.0 / 1024.0 << " MiB\n";

    out.flush();

    return out;

}

#pragma once

#include <vector>

#include <thread>

#include "iclient.h"

class Client : public IClient {

    void wait\_for\_client\_stop();

protected:

    std::atomic<bool> running = false;

    std::atomic<bool> should\_run = false;

    SOCKET socket\_descriptor;

    std::vector<char> temporary\_data;

    std::thread\* current\_runner = nullptr;

public:

    // Создаёт клиента

    Client();

    // Освобождает ресурсы клиента

    virtual ~Client();

    // Приостановить работу клиента

    void shutdown();

    // Запрашивает данные с клиента с дополнительной отправкой данных.

    // payload и payload\_size игнорируется в данной реализации

    void request(char\* payload, int payload\_size);

    // Возвращает true если ответ принят

    bool isReady();

    // Возвращает true если ответ принят

    const std::vector<char>& getAnswer();

    // Отобразить информацию о клиенте

    std::ostream& printClientInfo(std::ostream& out);

    // Отобразить информацию о файле

    std::ostream& printAnswerInfo(std::ostream& out);

};

#pragma once

#include <sstream>

#include <WinSock2.h>

#include <iostream>

#include "iclient.h"

#include "../shared.h"

void IClient::init() {

    std::clog << "Loading WSA library" << std::endl;

    loadWSA();

}

void IClient::detach() {

    std::clog << "Unloading WSA library" << std::endl;

    unloadWSA();

}

IClient::IClient() {

}

IClient::~IClient() {

}

void IClient::request() {

    return this->request(nullptr, 0);

}

#pragma once

#include <string>

#include <vector>

class IClient {

public:

    // Создаёт клиента

    IClient();

    // Освобождает ресурсы клиента

    virtual ~IClient();

    // Приостановить работу клиента

    virtual void shutdown() = 0;

    // Запрашивает данные с клиента

    void request();

    // Запрашивает данные с клиента с дополнительной отправкой данных

    virtual void request(char\* payload, int payload\_size) = 0;

    // Возвращает true если ответ принят

    virtual bool isReady() = 0;

    // Возвращает true если ответ принят

    virtual const std::vector<char>& getAnswer() = 0;

    // Отобразить информацию о клиенте

    virtual std::ostream& printClientInfo(std::ostream& out) = 0;

    // Отобразить информацию о файле

    virtual std::ostream& printAnswerInfo(std::ostream& out) = 0;

    // Загружает библиотеку WinSock

    static void init();

    // Выгружает библиотеку WinSock

    static void detach();

};

#include <iostream>

#include <WinSock2.h>

#include <algorithm>

#include "../shared.h"

#include "client.h"

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

int main() {

    try {

        // Инициализация библиотеки WSA

        IClient::init();

        // Создать клиент

        IClient\* c = new Client();

        // Оставляем клиент работать, пока пользователь не решит его приостановить

        std::string input;

        while (true)

        {

            std::cout << "Enter Y to request a file\n"

                << "Enter N to stop client\n"

                << "Enter L to save receive result into file\n"

                << "Enter F to print received message info\n"

                << "Enter E to exit loop\n"

                << "Enter S to print status" << std::endl;

            std::cin >> input;

            std::transform(input.begin(), input.end(), input.begin(), toupper);

            if (input == "Y") {

                // Запустить клиент

                c->request();

            }

            else if (input == "N") {

                // Приостановить клиент

                c->shutdown();

            }

            else if (input == "E") {

                // Выход из цикла

                break;

            }

            else if (input == "L") {

                // Сохранить файл если он был успешно получен

                if (!c->isReady())

                    std::cout << "Client response is not ready yet" << std::endl;

                std::string save\_path = getUniqueFilepath();

                saveByteArray(c->getAnswer(), save\_path);

                std::cout << "A response was saved at '" << save\_path << "'" << std::endl;

            }

            else if (input == "F") {

                // Вывести информацию о файле если он был успешно получен

                if (!c->isReady())

                    std::cout << "Client response is not ready yet" << std::endl;

                c->printAnswerInfo(std::cout);

            }

            else if (input == "S") {

                // Вывести информацию о клиенте

                c->printClientInfo(std::cout);

            }

        }

        // Приостановить сервер

        c->shutdown();

        delete c;

    }

    catch (const std::runtime\_error& error) {

        std::cerr << "Failed while running server. Caused by: '" << error.what() << "'" << std::endl;

        return -1;

    }

    // Выгрузка библиотеки WSA

    IClient::detach();

    return 0;

}

#include <sstream>

#include <WinSock2.h>

#include <WS2tcpip.h>

#include <filesystem>

#include <random>

#include <fstream>

#include "shared.h"

namespace uuid {

    static std::random\_device              rd;

    static std::mt19937                    gen(rd());

    static std::uniform\_int\_distribution<> dis(0, 15);

    static std::uniform\_int\_distribution<> dis2(8, 11);

    std::string generate\_uuid\_v4() {

        std::stringstream ss;

        int i;

        ss << std::hex;

        for (i = 0; i < 8; i++) {

            ss << dis(gen);

        }

        ss << "-";

        for (i = 0; i < 4; i++) {

            ss << dis(gen);

        }

        ss << "-4";

        for (i = 0; i < 3; i++) {

            ss << dis(gen);

        }

        ss << "-";

        ss << dis2(gen);

        for (i = 0; i < 3; i++) {

            ss << dis(gen);

        }

        ss << "-";

        for (i = 0; i < 12; i++) {

            ss << dis(gen);

        };

        return ss.str();

    }

}

std::string getErrorTextWithWSAErrorCode(std::string errorText) {

    std::ostringstream resultError;

    resultError << errorText << " " << WSAGetLastError();

    return resultError.str();

}

std::ostream& printSockaddrInfo(std::ostream& out, sockaddr\_in& sock) {

    char address[64] = {};

    inet\_ntop(AF\_INET, &sock.sin\_addr, address, sizeof(address));

    out << "  Address: " << address << "\n" <<

        "  Port: " << htons(sock.sin\_port);

    return out;

}

sockaddr\_in getDeviceAddrInfo() {

    // Получить имя текущего устройства

    char host\_name[256] = {};

    if (gethostname(host\_name, sizeof(host\_name)) == SOCKET\_ERROR)

        throw std::runtime\_error(getErrorTextWithWSAErrorCode("Unable to get host name"));

    addrinfo hints = {};

    hints.ai\_family = AF\_UNSPEC;

    hints.ai\_socktype = SOCK\_STREAM;

    hints.ai\_protocol = IPPROTO\_TCP;

    struct addrinfo\* result = NULL;

    // Получить информацию об адресах на устройстве в сети

    DWORD dwRetval = getaddrinfo(host\_name, "", &hints, &result);

    if (dwRetval != 0) {

        throw std::runtime\_error(getErrorTextWithWSAErrorCode("Getaddrinfo failed for device host name"));

    }

    for (addrinfo\* ptr = result; ptr != NULL; ptr = ptr->ai\_next) {

        // Если адрес для устройства является IPv4 адресом, мы нашли нужный адрес, возвращаем его

        if (ptr->ai\_family == AF\_INET) {

            sockaddr\_in device\_sockaddr = {};

            memcpy(&device\_sockaddr, ptr->ai\_addr, sizeof(device\_sockaddr));

            // Плохой костыль, нужно было использовать GetAdaptersInfo

            // Иначе не получится найти нужный айпишник с гейтвеем на роутер

            if (device\_sockaddr.sin\_addr.S\_un.S\_un\_b.s\_b3 > 10) {

                continue;

            }

            return device\_sockaddr;

        }

    }

    throw std::runtime\_error(getErrorTextWithWSAErrorCode("Getaddrinfo failed for device host name"));

}

void loadWSA() {

    WORD wVersionRequested;

    WSADATA wsaData;

    wVersionRequested = MAKEWORD(2, 0);

    // Загрузить библиотеку WinSock

    if (WSAStartup(wVersionRequested, &wsaData) == SOCKET\_ERROR)

        throw std::runtime\_error(getErrorTextWithWSAErrorCode("Unable to load WSA library"));

}

void unloadWSA() {

    // Выгрузить библиотеку WinSock

    if (WSACleanup() == SOCKET\_ERROR)

        throw std::runtime\_error(getErrorTextWithWSAErrorCode("Unable to unload WSA library"));

}

std::string getUniqueFilepath() {

    return "./" + uuid::generate\_uuid\_v4() + ".jpg";

}

void saveByteArray(const std::vector<char>& data, std::string path) {

    // Открываем файл

    std::ofstream save\_file(path, std::ios::binary);

    if (!save\_file.good())

        throw std::runtime\_error("Unable to open file '" + path + "' for reading");

    // Пишем массив в файл

    for (int i = 0; i < data.size() / IMAGE\_FRAGMENT\_FILE\_SIZE; i++)

        save\_file.write(&data[i \* IMAGE\_FRAGMENT\_FILE\_SIZE], sizeof(char) \* IMAGE\_FRAGMENT\_FILE\_SIZE);

    // Сохраняем и закрываем файл

    save\_file.flush();

    save\_file.close();

}

#pragma once

#include <string>

#include <WinSock2.h>

#include <vector>

#define SERVER\_DEFAULT\_PORT      0

#define CLIENT\_DEFAULT\_PORT      12484

#define IMAGE\_FRAGMENT\_SIZE      508

#define IMAGE\_FRAGMENT\_FILE\_SIZE 512

// Формирует текст ошибки вместе с WSA кодом

std::string getErrorTextWithWSAErrorCode(std::string errorText);

// Выводит информацию о структуре sockaddr

std::ostream& printSockaddrInfo(std::ostream& out, sockaddr\_in& sock);

// Возвращает IP адрес для текущего ПК

sockaddr\_in getDeviceAddrInfo();

// Загружает WSA

void loadWSA();

// Выгружает WSA

void unloadWSA();

// Возвращает неконфликтующее имя для текущего файла

std::string getUniqueFilepath();

// Сохраняет массив байтов в файл

void saveByteArray(const std::vector<char>& data, std::string path);

#pragma once

#include <sstream>

#include <WinSock2.h>

#include <iostream>

#include "iserver.h"

#include "../shared.h"

void IServer::init() {

    std::clog << "Loading WSA library" << std::endl;

    loadWSA();

}

void IServer::detach() {

    std::clog << "Unloading WSA library" << std::endl;

    unloadWSA();

}

IServer::IServer() {

}

IServer::~IServer() {

}

#pragma once

#include <string>

// Абстрактный класс для работы с WinSock библиотекой

class IServer {

public:

    // Создаёт сервер

    IServer();

    // Освобождает ресурсы сервера

    virtual ~IServer() = 0;

    // Возобновить работу сервера

    virtual void start() = 0;

    // Приостановить работу сервера

    virtual void shutdown() = 0;

    // Отобразить информацию о сервере

    virtual std::ostream& printServerInfo(std::ostream& out) = 0;

    // Загружает библиотеку WinSock

    static void init();

    // Выгружает библиотеку WinSock

    static void detach();

};

#include <iostream>

#include <algorithm>

#include "server.h"

int main() {

    try {

        // Инициализация библиотеки WSA

        IServer::init();

        // Ввести путь к раздаваемому файлу

        std::cout << "Input path to file to send: ";

        std::cout.flush();

        std::string input;

        std::cin >> input;

        // Создать сервер

        IServer\* s = new Server(input);

        // Оставляем сервер работать, пока пользователь не решит его приостановить

        while (true)

        {

            std::cout << "Enter Y to start broadcast send\n"

                << "Enter N to stop server\n"

                << "Enter E to exit loop\n"

                << "Enter S to print status" << std::endl;

            std::cin >> input;

            std::transform(input.begin(), input.end(), input.begin(), toupper);

            if (input == "Y") {

                // Запустить сервер

                s->start();

            }

            else if (input == "N") {

                // Приостановить сервер

                s->shutdown();

            }

            else if (input == "E") {

                // Выход из цикла

                break;

            }

            else if (input == "S") {

                // Вывести информацию о сервере

                s->printServerInfo(std::cout);

            }

        }

        // Приостановить сервер

        s->shutdown();

        delete s;

    }

    catch (const std::runtime\_error& error) {

        std::cerr << "Failed while running server. Caused by: '" << error.what() << "'" << std::endl;

        return -1;

    }

    // Выгрузка библиотеки WSA

    IServer::detach();

    return 0;

}

#include <exception>

#include <WinSock2.h>

#include <sstream>

#include <iostream>

#include <chrono>

#include "server.h"

#include "../shared.h"

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

Server::Server(std::string file\_path, int port) {

    std::clog << "Opening file " << file\_path << std::endl;

    // Открываем файл

    this->file = new std::ifstream(file\_path, std::ios::binary);

    if (!this->file->is\_open())

        throw std::runtime\_error("Unable to open file for read " + file\_path);

    std::clog << "Creating socket" << std::endl;

    // Создаём сокет

    this->socket\_descriptor = socket(

        AF\_INET,

        SOCK\_DGRAM,

        IPPROTO\_IP

    );

    std::clog << "Making socket broadcast" << std::endl;

    // Делаем сокет способным к широковещательному каналу

    bool broadcast = true;

    if (setsockopt(this->socket\_descriptor, SOL\_SOCKET, SO\_BROADCAST, (char\*)&broadcast, sizeof(broadcast)) == SOCKET\_ERROR)

        throw std::runtime\_error(getErrorTextWithWSAErrorCode("Unable to make socket broadcast"));

    std::clog << "Creating socket bind addr" << std::endl;

    // Создаём адрес к которому привяжется сокет

    sockaddr\_in bind\_addr;

    bind\_addr.sin\_family = AF\_INET;

    bind\_addr.sin\_addr = getDeviceAddrInfo().sin\_addr;

    bind\_addr.sin\_port = htons(port);

    std::clog << "Binding socket" << std::endl;

    // Привязать сокет к адресу

    if (bind(this->socket\_descriptor, (sockaddr\*)&bind\_addr, sizeof(bind\_addr)) == SOCKET\_ERROR)

        throw std::runtime\_error(getErrorTextWithWSAErrorCode("Unable to bind socket descriptor"));

}

Server::~Server() {

    std::clog << "Stopping worker thread" << std::endl;

    // Приостанавливаем рабочий поток

    this->shutdown();

    std::clog << "Closing file" << std::endl;

    // Закрываем файл

    this->file->close();

    delete this->file;

    std::clog << "Closing socket" << std::endl;

    // Закрываем сокет

    if (closesocket(this->socket\_descriptor) == SOCKET\_ERROR)

        throw std::runtime\_error(getErrorTextWithWSAErrorCode("Unable to close socket"));

}

std::ostream& Server::printServerInfo(std::ostream& out) {

    sockaddr\_in server\_address;

    int server\_address\_size = sizeof(server\_address);

    int get\_sock\_name\_res = getsockname(this->socket\_descriptor, (sockaddr\*)&server\_address, &server\_address\_size);

    std::cout << "IP server info:\n" <<

        "Server state: " << (this->running ? "running" : "not running") << "\n";

    if (get\_sock\_name\_res == SOCKET\_ERROR) {

        std::cout << "Unable to get socket info\n";

    }

    else {

        std::cout << "Socket info:\n";

        printSockaddrInfo(std::cout, server\_address);

        std::cout << "\n";

    }

    std::cout.flush();

    return out;

}

void Server::start() {

    // Если сервер уже работает, выходим из него

    if (this->running) {

        std::clog << "Server is already running" << std::endl;

        return;

    }

    // Подготавливаем рабочий поток

    delete this->current\_runner;

    std::clog << "Starting server" << std::endl;

    this->should\_run = true;

    this->current\_runner = new std::thread([this]() {

        // Устанавливаем флаг работы потока на true

        this->running = true;

        this->file->clear();

        this->file->seekg(0, std::ios::beg);

        char buffer[IMAGE\_FRAGMENT\_SIZE];

        int packages\_success = 0, packages\_failed = 0;

        // Конструируем широковещательный sockaddr\_in

        sockaddr\_in client\_sockaddr;

        client\_sockaddr.sin\_family = AF\_INET;

        client\_sockaddr.sin\_port = htons(CLIENT\_DEFAULT\_PORT);

        client\_sockaddr.sin\_addr = getDeviceAddrInfo().sin\_addr;

        memset(((char\*)&client\_sockaddr.sin\_addr) + 3, 0xff, 1);

        int total\_bytes = 0;

        auto a = std::chrono::high\_resolution\_clock::now();

        // Пока поток должен работать и не достигнут конец файла

        while (this->should\_run && !this->file->eof()) {

            // Считать файл

            this->file->read(buffer, sizeof(buffer));

            int bytes\_read = this->file->gcount();

            total\_bytes += bytes\_read;

            // Отправить фрагмент

            if (sendto(

                this->socket\_descriptor,

                buffer,

                bytes\_read,

                0,

                (sockaddr\*)&client\_sockaddr,

                sizeof(client\_sockaddr)) == SOCKET\_ERROR) {

                packages\_failed++;

            }

            else {

                packages\_success++;

            }

        }

        auto b = std::chrono::high\_resolution\_clock::now();

        std::clog << "A file broadcasted ended\n  Packages failed: "

            << packages\_failed << "\n  Packages sent: " << packages\_success << "\n" <<

            "  Size:" << "\n" <<

            "    " << total\_bytes << " B\n" <<

            "    " << total\_bytes / 1024.0 << " KiB\n" <<

            "    " << total\_bytes / 1024.0 / 1024.0 << " MiB\n" <<

            "\n  Time: " << std::chrono::duration\_cast<std::chrono::milliseconds>(b - a).count() / 1000.0 << " s." << std::endl;

        // Устанавливаем флаг работы потока на false

        this->running = false;

    });

    this->current\_runner->detach();

}

void Server::shutdown() {

    if (!this->running) {

        std::clog << "Server is already stopped" << std::endl;

        return;

    }

    std::clog << "Stopping server" << std::endl;

    // Указываем что серверу нужно приостановиться

    // и ждём пока он остановится

    this->should\_run = false;

    wait\_for\_server\_stop();

    delete this->current\_runner;

}

void Server::wait\_for\_server\_stop() {

    // Используем спинлок, так как пакеты относительно небольшие и сервер должен

    // быстро увидеть что пора заканчивать работу

    while (this->running) {

    }

}

#pragma once

#include <WinSock2.h>

#include <fstream>

#include <thread>

#include <mutex>

#include <atomic>

#include "iserver.h"

#include "../shared.h"

class Server : public IServer {

    void wait\_for\_server\_stop();

protected:

    std::atomic<bool> running = false;

    std::atomic<bool> should\_run = false;

    std::ifstream\* file = nullptr;

    SOCKET socket\_descriptor;

    std::thread\* current\_runner = nullptr;

public:

    // Создаёт сервер

    Server(std::string file\_path, int port = SERVER\_DEFAULT\_PORT);

    // Освобождает ресурсы сервера

    ~Server();

    // Возобновить работу сервера

    void start();

    // Приостановить работу сервера

    void shutdown();

    // Отобразить информацию о сервере

    std::ostream& printServerInfo(std::ostream& out);

};