

Министерство цифрового развития связи и массовых коммуникаций РФ

Государственное бюджетное образовательное учреждение высшего
образования

Ордена Трудового Красного Знамени

«Московский технический университет связи и информатики»

Кафедра «Сетевые информационные технологии и сервисы»

Отчет по утилите ping

по дисциплине

«Распределенные операционные системы»

Подготовили: студенты группы

БВТ1902 Саввин Д.И., Долгих И.И.

Проверила: Беленькая М.Н.

Москва

2022

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ	3
ОСНОВНАЯ ЧАСТЬ.....	4
1. Основные функции pingg и средства их реализации.	4
2. Вспомогательные функции pingg.....	5
3. Обработка ошибок.	5
4. Блок-схемы, main() и основные функции для работы ping.	6
5. Код программы.	15
6. Сборка утилиты.....	36
7. Результаты тестирования.	38
ЗАКЛЮЧЕНИЕ	46

ВВЕДЕНИЕ

Ping – это промежуток времени, за который пакет, отосланный с вашего компьютера, доходит до другого конца сети, и возвращается обратно. С помощью разработанной утилиты “pingg”, можно узнать тот самый пинг, введя IP-адрес или доменное имя сервера, а также сколько раз мы хотим «пропинговать» удаленный сервер.

В основе реализации ping лежит обмен ICMP-пакетами. ICMP-пакет входит в стек протоколов TCP/IP, используется для передачи сигналов об ошибках и других исключительных ситуациях, а также для отправки эхо-запросов(Echo-Request) и эхо-ответов(Echo-Reply), последними двумя мы и воспользовались для реализации пинга. Структура ICMP-пакета представлена в таблице 1.

Таблица 1 – Структура ICMP-пакета.

Поле	Число байт	Значение
Type	1	8
Code	1	0
Control Sum	2	-
ID	2	-
Sequence	2	-
Data	4	“abcdefghijklmnopqrstuvwabcdefghi”

Для самой отправки пакетов были задействованы средства C++ ws2_32 библиотеки (Windows Socket). Утилита поддерживает запись каждой операции и ошибки в лог-файл, с указанием даты и времени, типа (операция или ошибка), кода (если ошибка) и словесного описания. При попытке ввода некорректных значений, будь то IP, domain name или количество «пингования». В первых двух случаях программа выдаст сообщение о неверном имени для подключения, в последнем же, при некорректном значении - количество для «пингования», по умолчанию утилита присвоит значение – 1. Также в конце работы на экран пользователя выводится статистика работы ping, количество отправленных, полученных и потерянных

пакетов. Также минимальный, максимальный и средний промежуток времени между отправкой пакета и получением ответа, в миллисекундах. Для сборки утилиты использовалась система Stake.

ОСНОВНАЯ ЧАСТЬ

1. Основные функции pingg и средства их реализации.

В начале работы утилита автоматически заполняет структуры описывающую ICMP-пакет стандартными значениями, тип – 8, код – 0 и т. д..

Для подсчета контрольной суммы используется следующий алгоритм: все данные пакеты собираются в 2-х байтовые пары в двоичном виде (тип + код = 00001000 + 00000000 = 0000100000000000), поля контрольной суммы(при подсчете суммы равно нулю), ID и очереди и так являются двухбайтовыми(считаем за пару), последними в пары мы собираем значения из поля данных(Data). Наконец мы складываем все пары, получаем значение превышающее 2 байта, корректируем значение контрольной суммы с помощью сдвига байтов на 16, пока сумма больше 1111111111111111(0xffff). Результатом контрольной суммы будет являться инвертированный по битам результат.

После заполнения ICMP-пакета, утилита запрашивает у пользователя IP-адрес или доменное имя удаленного сервера для подключения. Если введено доменное имя, программа узнает его IP, после чего заполняет необходимые структуры данных addrinfo введенным пользователем, обработанным или полученным IP-адресом. Начинается работа с сетью, инициализируется сокет и устанавливается соединение по нему.

В случае отсутствия проблем с подключением, утилита запрашивает у пользователя количество пакетов которое он хочет отправить, значение должно быть больше 0 и меньше 5000, в других случаях значение заменяется на 1. Далее утилита начинает отсылать ICMP-пакеты столько, сколько ввел пользователь, заменяя поля Sequence(очередь) и Control Sum, для каждого нового пакета, параллельно подсчитывая статистику. Тайм-аут для ответа от

удаленного сервера равен 5 секундам (5000 миллисекунд). В случае если сервер не отправляет пакеты в ответ, пользователь может преждевременно остановить отправку пакетов нажатием клавиши TAB (для реализации этого функционала используется отдельный поток, ожидающий нажатие той самой клавиши). После окончания всех вышеприведенных действий (в случае отсутствия ошибок), утилита выводит статистику и завершает свою работу.

2. Вспомогательные функции pingg.

В ходе разработки утилиты pingg появилась необходимость в вспомогательных функциях, выполняющие разного рода действия, начиная с перевода int в string в двоичном виде и перевода char = 254 в string = "254", заканчивая функциями записи в лог файл и ожидания нажатия клавиши для завершения работы программы (см. листинг 9).

3. Обработка ошибок.

В утилите pingg были предусмотрены возможные ошибки разного рода, программа способна обрабатывать почти любую ошибку ввода со стороны пользователя.

Для проверки корректности введенного пользователем IP, была реализована отдельная функция, которая проверяет строку IP на 3 следующих условия:

- 1) Количество точек должно равняться трем (пример 8.8.8.8);
- 2) Каждый символ строки, кроме точек, должен являться цифрой;
- 3) Каждое число, отделенное от другого точкой, должно быть не меньше 0 и не больше 255.

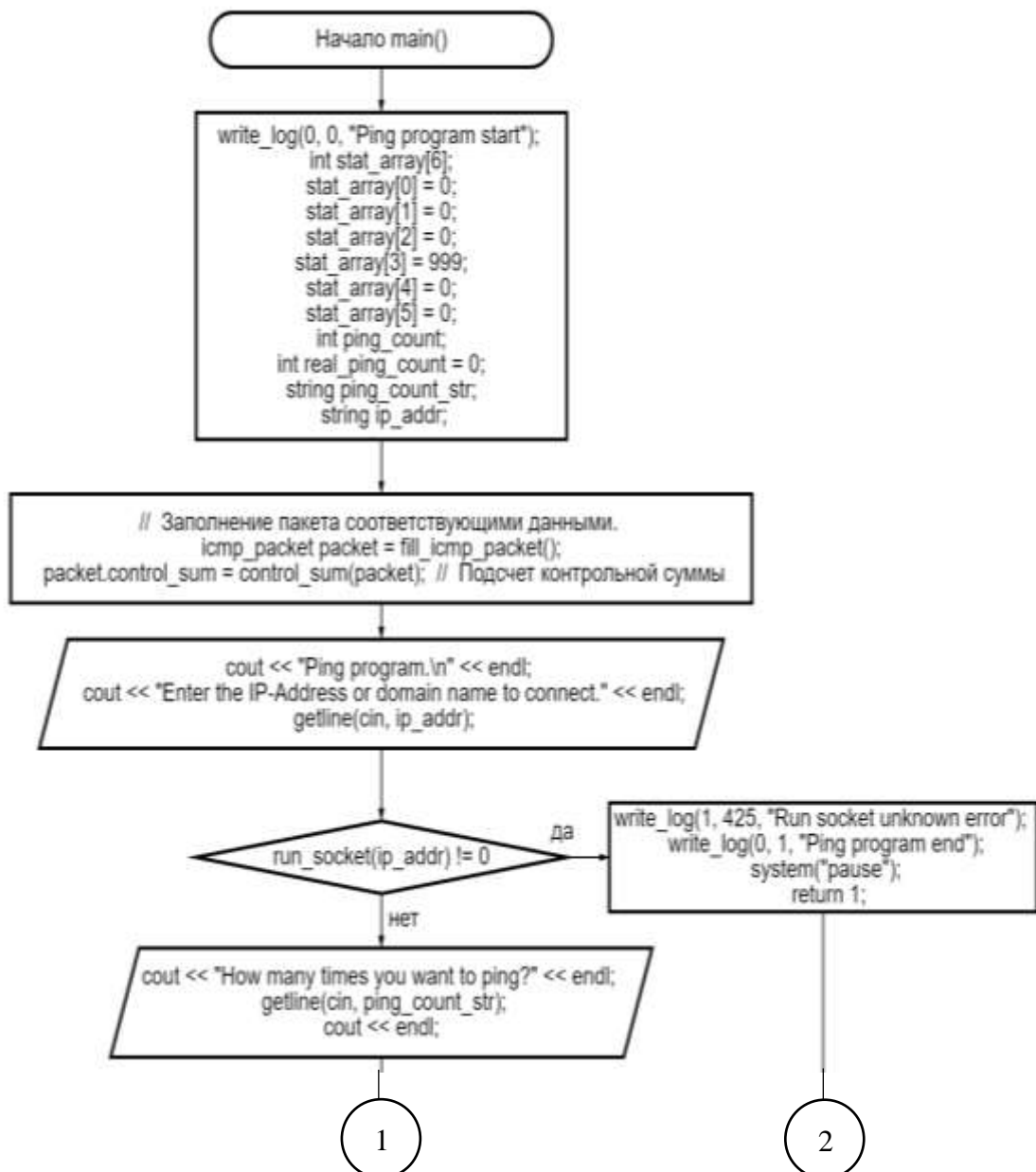
Для проверки корректности количества пакетов для отправки, которое ввел пользователь, используется несколько простых условий:

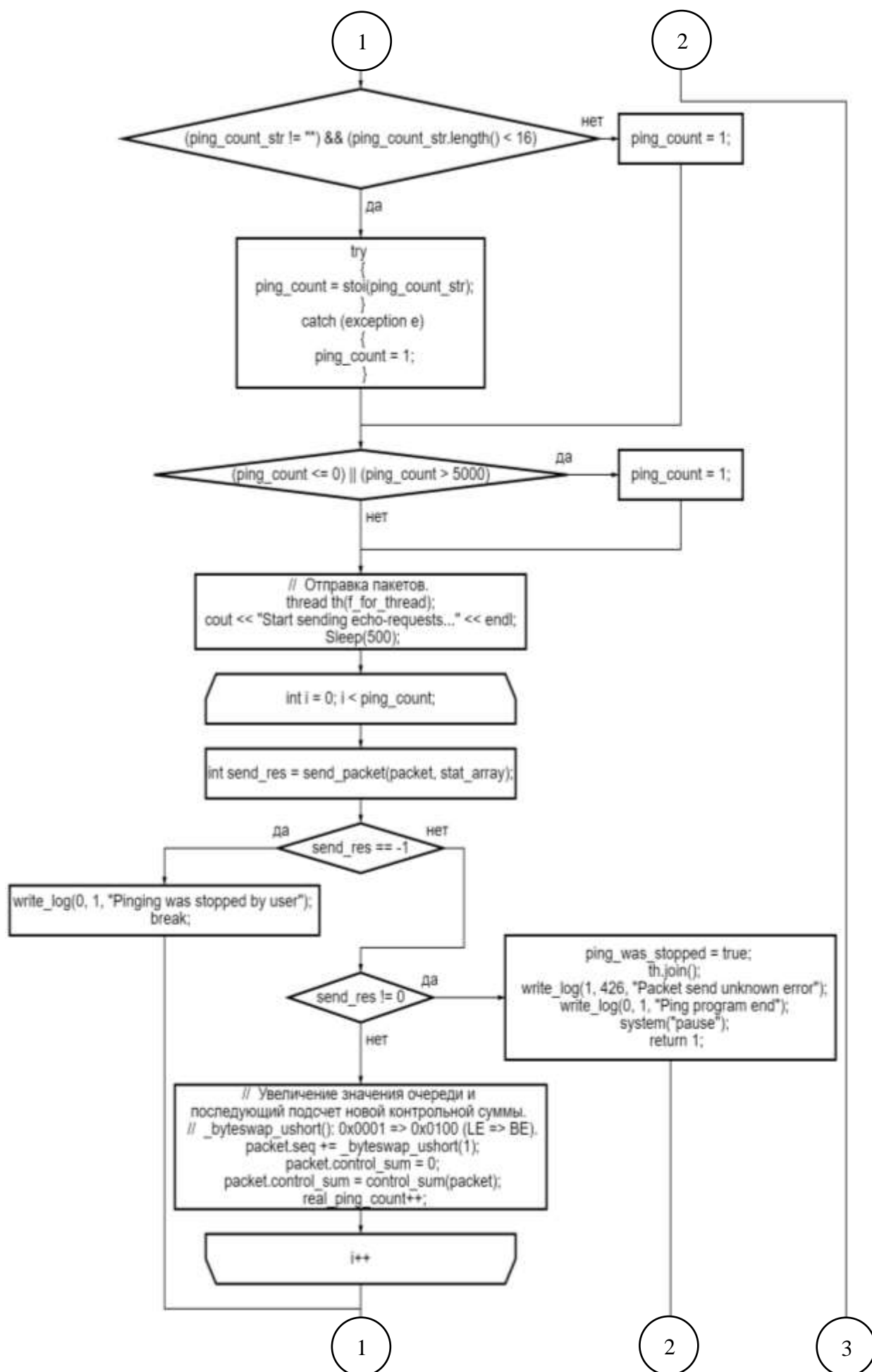
- 1) Если пользователь оставил поле ввода пустым, значение принимается за 1;

- 2) Если пользователь не оставил поле пустым, и если длина поля(строки) меньше 16, то утилита, с помощью try-catch, пробует перевести число из string в int. В случае неудачи значение принимается за 1;
- 3) Если введенное число меньше единицы или больше 5000, значение принимается за 1.

4. Блок-схемы, main() и основные функции для работы ping.

На рисунке ниже представлена блок-схема, описывающая алгоритм работы main функции pingg.





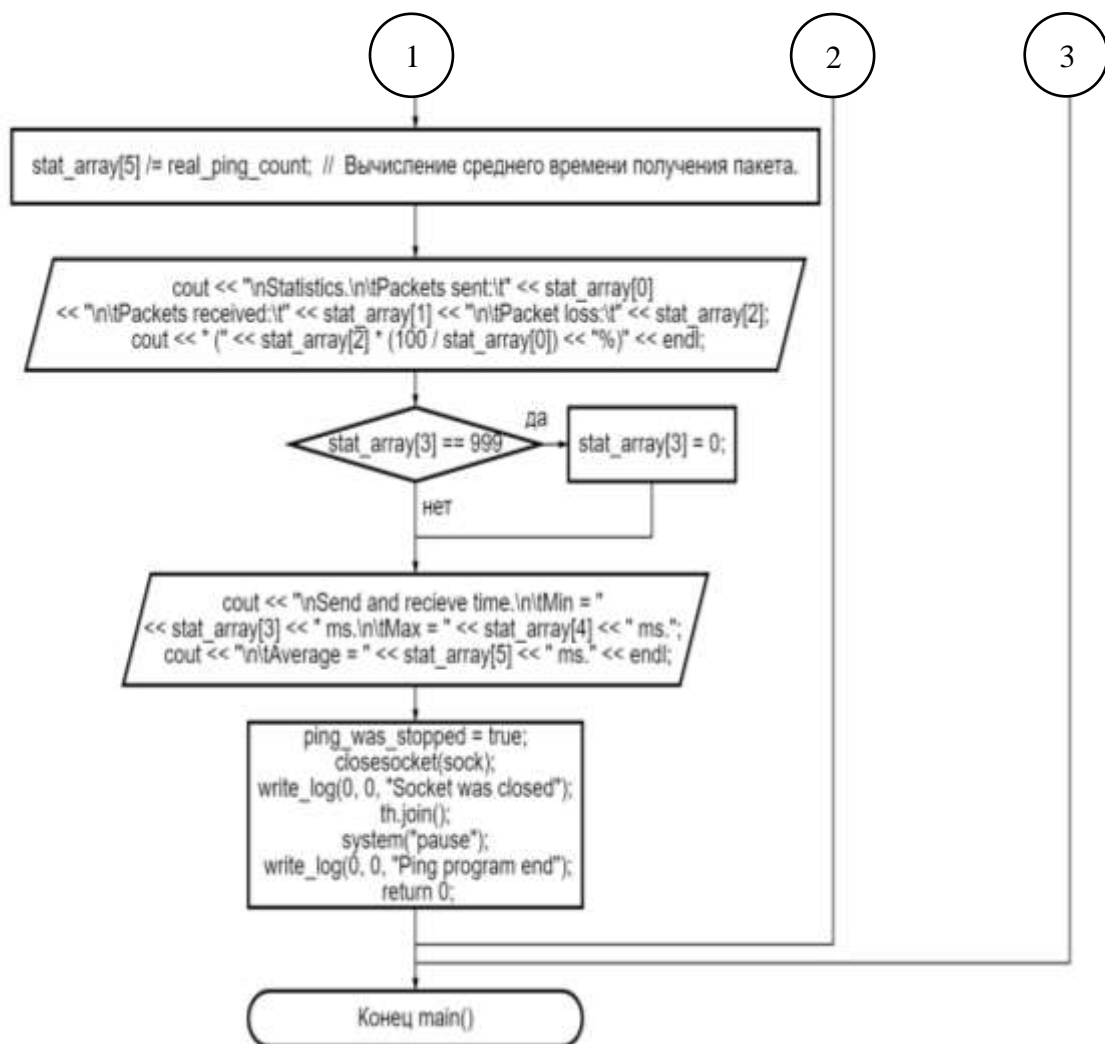
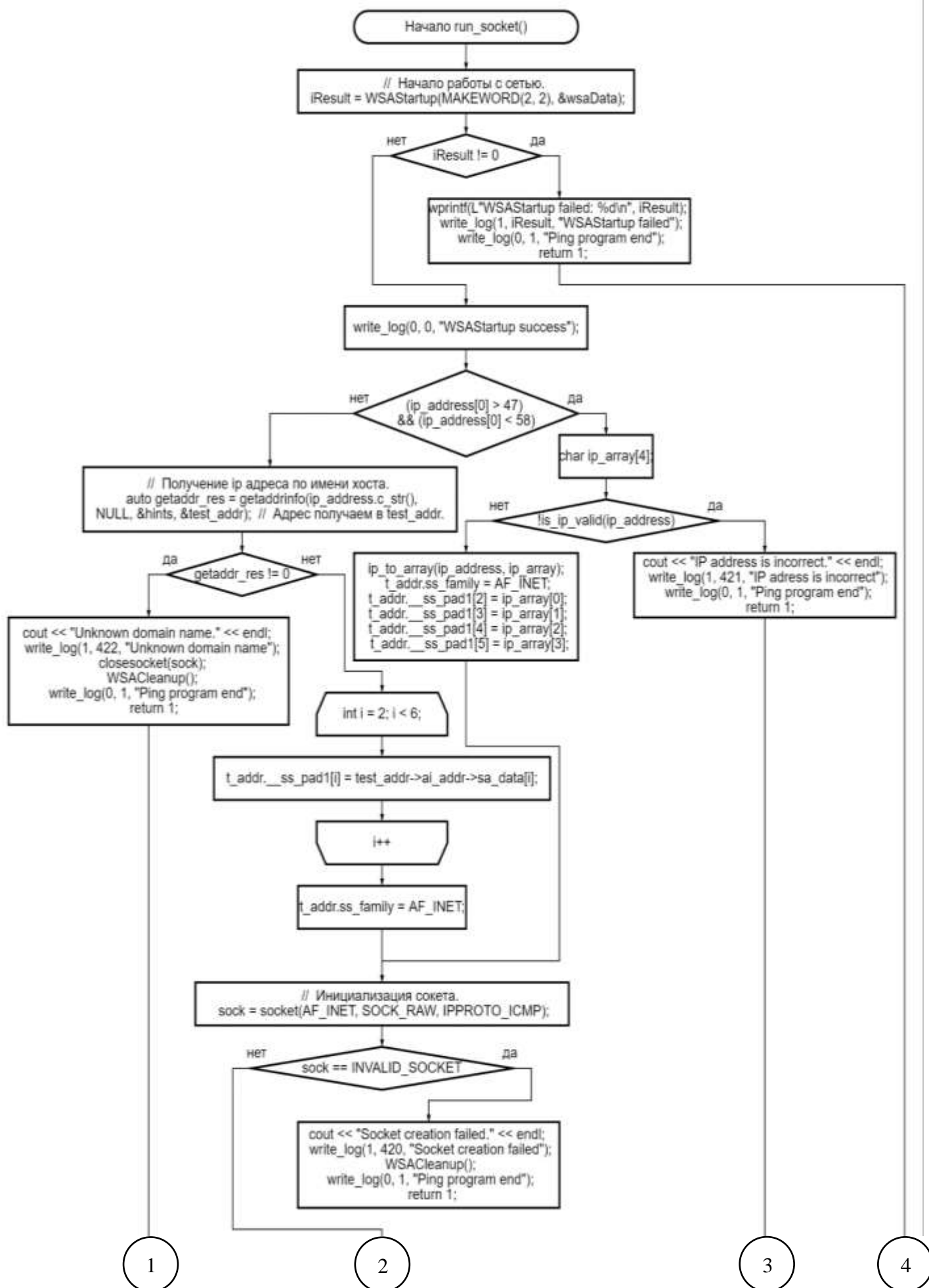


Рис. 1 – Схема алгоритма функции main().

На рисунке 2 представлена блок-схема алгоритма функции run_socket(string ip_address), отвечающая за начало работы с сокетом, его создание и установление соединения по нему.



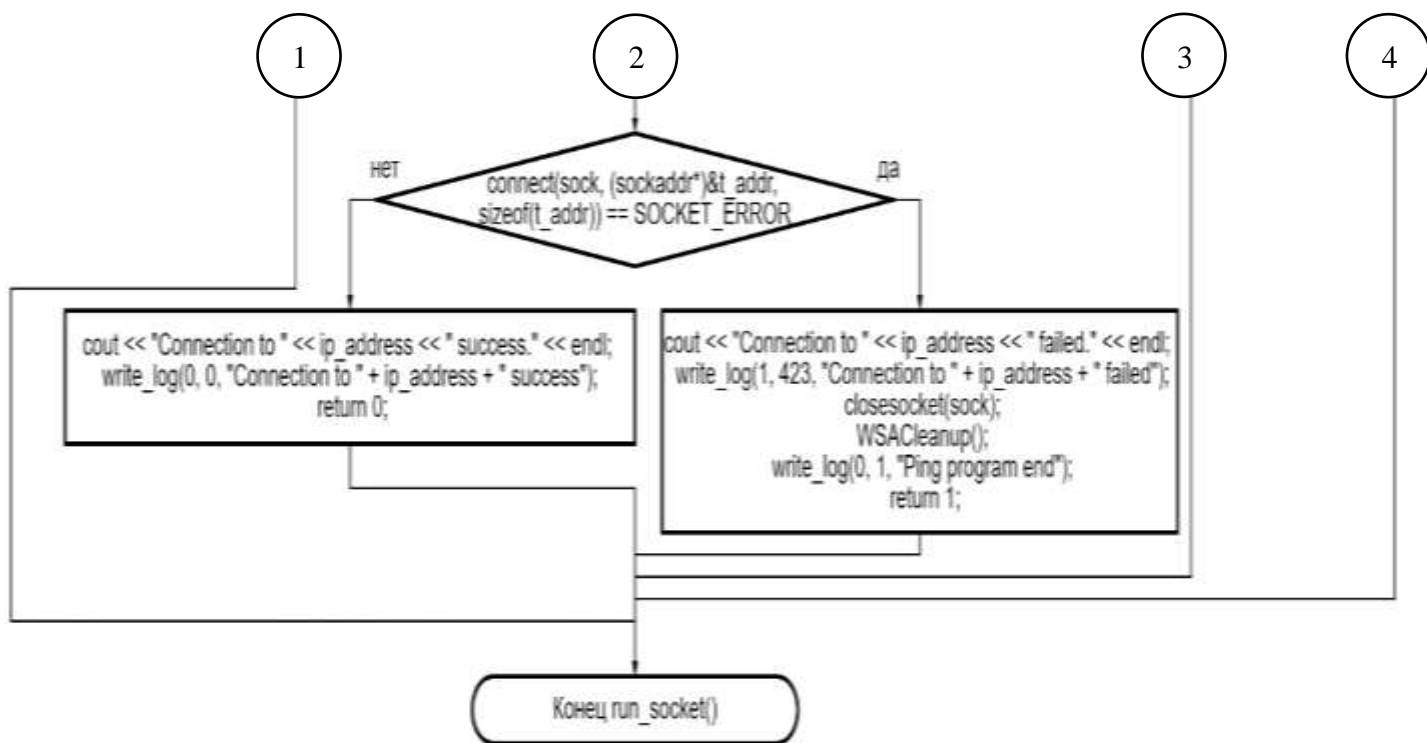
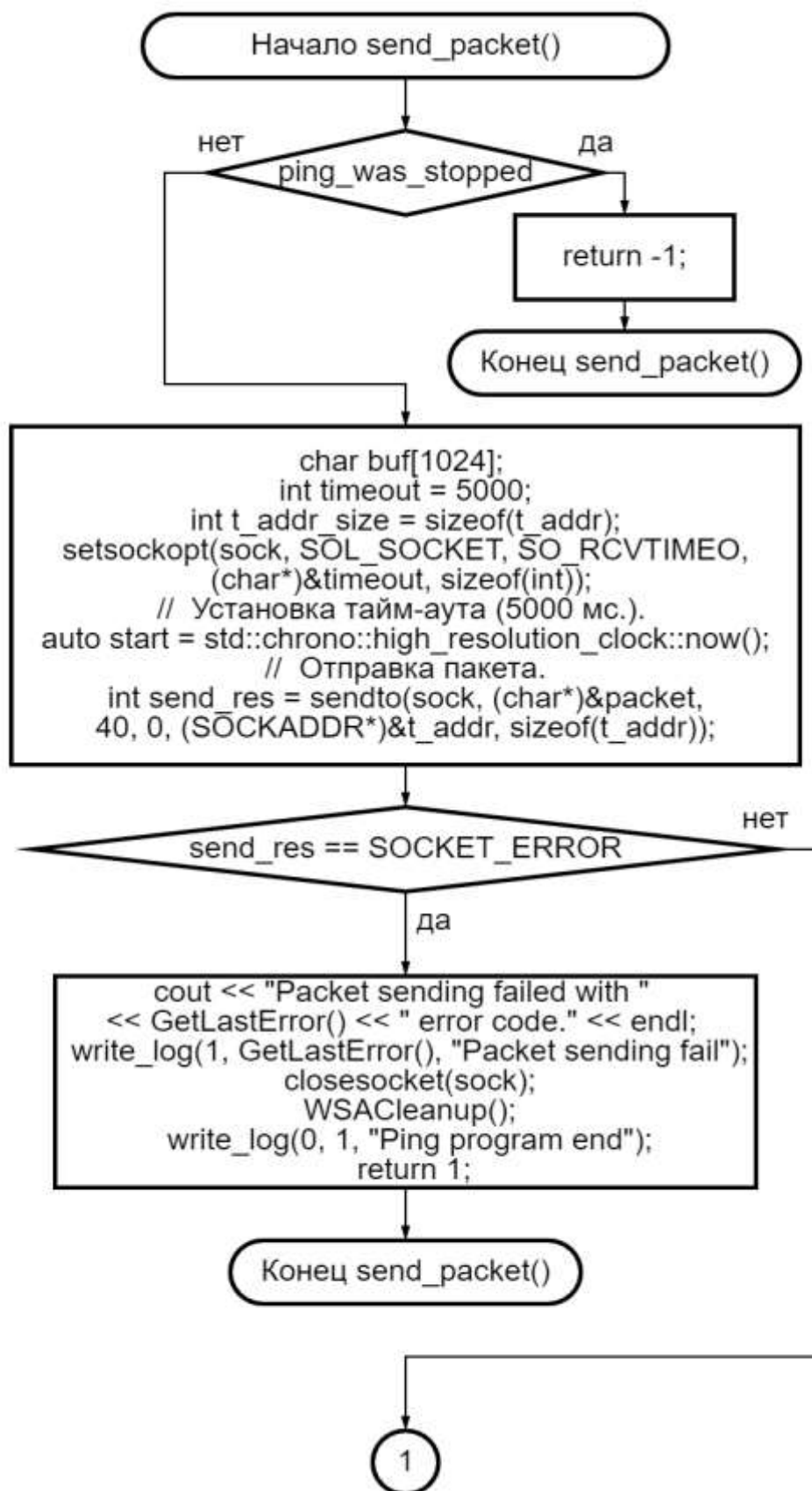
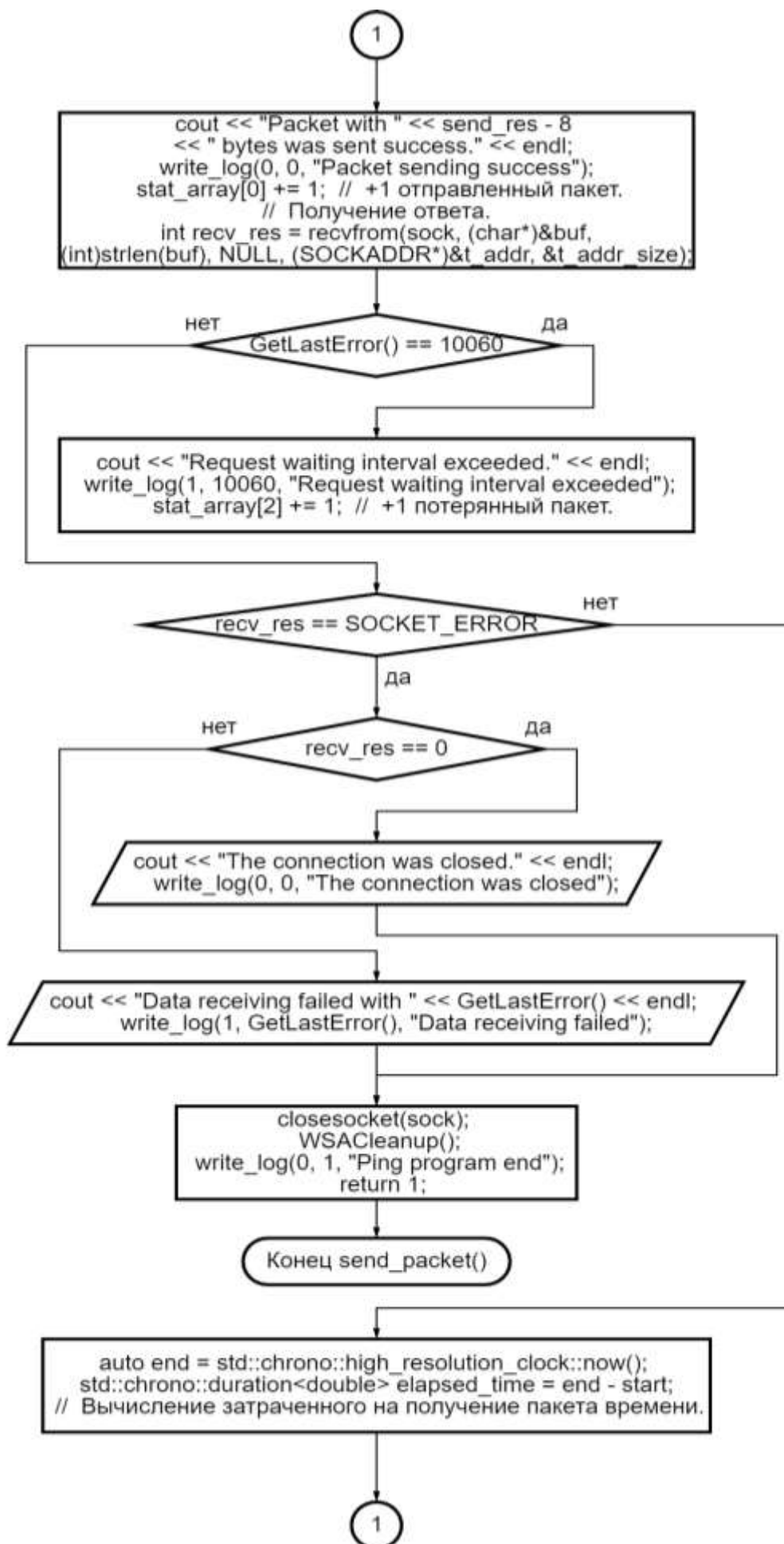


Рис. 2 – Схема алгоритма функции run_socket().

На рисунке 3 изображена блок-схема алгоритма функции `send_packet(icmp_packet packet, int* stat_array)`, отвечающую за отправку пакета.





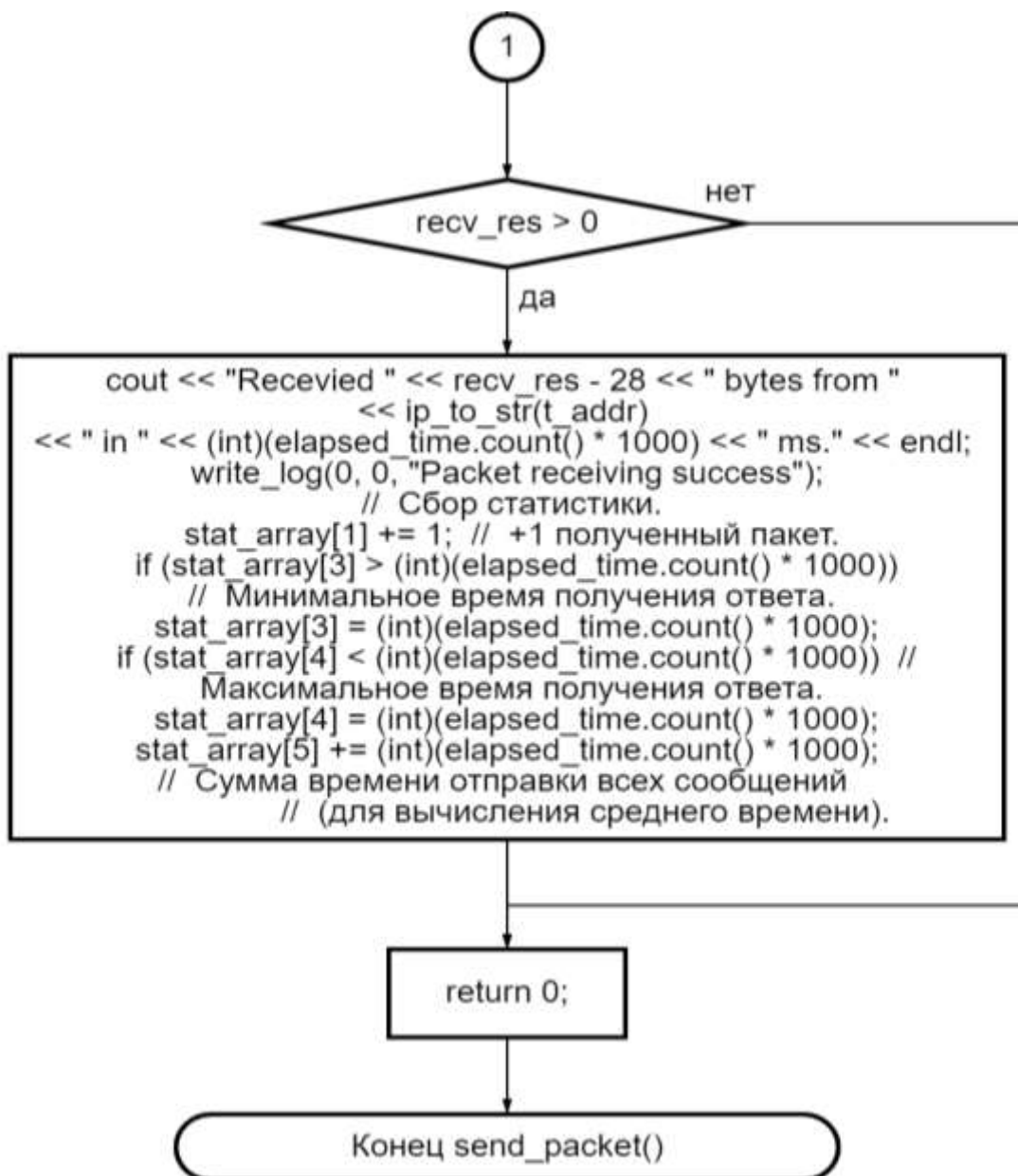


Рис. 3 – Схема алгоритма функции send_packet().

На рисунке 4 представлена блок-схема алгоритма работы функции write_log(bool type, int error_code, string description), отвечающую за запись событий и ошибок в лог-файл.

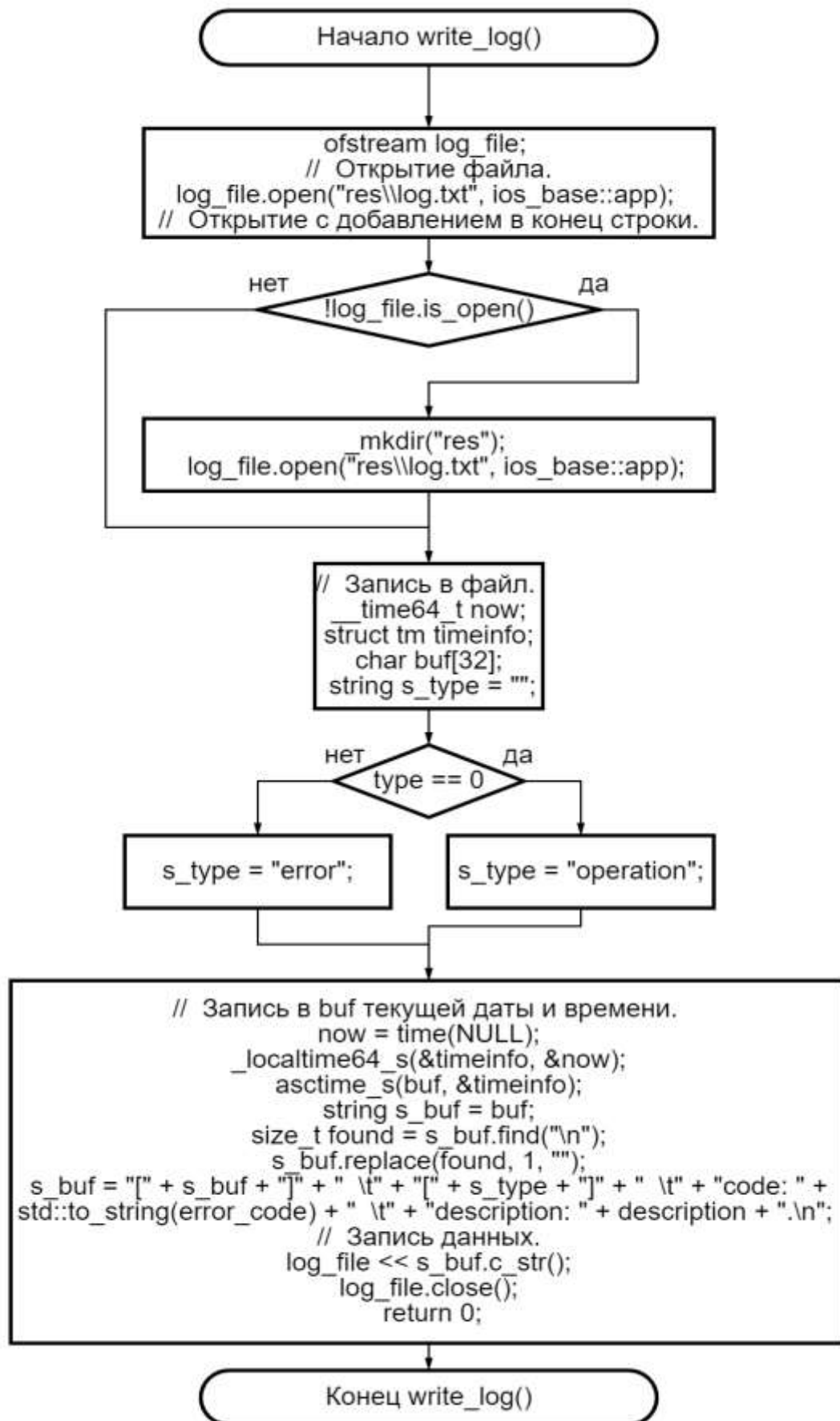


Рис. 4 – Схема алгоритма функции write_log().

5. Код программы.

В листинге 1 представлены библиотеки, необходимые для реализации работы утилиты.

Листинг 1 – Подключение библиотек.

```
/* Подключение библиотек. */
#include <WinSock2.h>
#include <WS2tcpip.h>
#include <iostream>
#include <string>
#include <intrin.h>
#include <chrono>
#include <time.h>
#include <conio.h>
#include <thread>
#include <fstream>
#include <direct.h>
```

В листинге 2 показано объявление глобальных переменных и констант, необходимых для работы утилиты pingg.

Листинг 2 – Глобальные переменные и константы.

```
/* Глобальные переменные и константы. */

SOCKET sock = INVALID_SOCKET;
struct sockaddr_storage t_addr;
struct addrinfo* test_addr = NULL;
struct addrinfo hints;
WSADATA wsaData = { 0 };
int iResult = 0;
bool ping_was_stopped = false;

struct sockaddr s_address;
```

В листинге 3 представлена структура, описывающая ICMP-пакет.

Листинг 3 – Структура описывающая ICMP-пакет.

```
// Структура описывающая ICMP-пакет.
struct icmp_packet
{
    // Заголовок пакета.
    char type; // Тип сообщения, 0 или 8.
    char code; // Код ошибки.
    short control_sum;
    short id;
    short seq; // Очередь.

    // Данные пакета.
    const char data[33] = "abcdefghijklmnopqrstuvwabcdefghi";
};
```

В листинге 4 представлен код основной функции main().

Листинг 4 – Функция main().

```
/* Основная функция main. */
int main()
{
    write_log(0, 0, "Ping program start");
    int stat_array[6];
    stat_array[0] = 0;
    stat_array[1] = 0;
    stat_array[2] = 0;
    stat_array[3] = 999;
    stat_array[4] = 0;
    stat_array[5] = 0;
    int ping_count;
    int real_ping_count = 0;
    string ping_count_str;
    string ip_addr;

    // Заполнение пакета соответствующими данными.
    icmp_packet packet = fill_icmp_packet();
    packet.control_sum = control_sum(packet); // Подсчет контрольной суммы

    cout << "Ping program.\n" << endl;
    cout << "Enter the IP-Address or domain name to connect." << endl;

    getline(cin, ip_addr);

    // Начало работы сокета.
    if (run_socket(ip_addr) != 0)
    {
        write_log(1, 425, "Run socket unknown error");
        write_log(0, 1, "Ping program end");
        system("pause");
        return 1;
    }

    cout << "How many times you want to ping?" << endl;
    getline(cin, ping_count_str);

    cout << endl;
    if ((ping_count_str != "") && (ping_count_str.length() < 10)) // Если
пользователь ввел сколько раз нужно отправить пакеты.
    {
        try
        {
            ping_count = stoi(ping_count_str);
        }
        catch (exception e)
        {
            ping_count = 1;
        }
    }
    else // Если оставил поле сколько раз нужно отправить пакеты пустым.
        ping_count = 1;

    if ((ping_count <= 0) || (ping_count > 5000))
        ping_count = 1;

    // Отправка пакетов.
```



```

thread th(f_for_thread);
cout << "Start sending echo-requests..." << endl;
Sleep(500);

for (int i = 0; i < ping_count; i++)
{
    int send_res = send_packet(packet, stat_array);
    if (send_res == -1) // Если пинг завершен досрочно, нажатием клавиши TAB.
    {
        write_log(0, 1, "Pinging was stopped by user");
        break;
    }
    else if (send_res != 0) // Остальные случаи ошибки отправки пакета.
    {
        ping_was_stopped = true;
        th.join();
        write_log(1, 426, "Packet send unknown error");
        write_log(0, 1, "Ping program end");
        system("pause");
        return 1;
    }

    // Увеличение значения очереди и последующий подсчет новой контрольной
    // суммы.
    // _byteswap_ushort(): 0x0001 => 0x0100 (LE => BE).
    packet.seq += _byteswap_ushort(1);
    packet.control_sum = 0;
    packet.control_sum = control_sum(packet);

    real_ping_count++;
}

stat_array[5] /= real_ping_count; // Вычисление среднего времени получения
пакета.

// Вывод статистики пинга.
cout << "\nStatistics.\n\tPackets sent:\t" << stat_array[0] << "\n\tPackets
received:\t" << stat_array[1] << "\n\tPacket loss:\t" << stat_array[2];
cout << " (" << stat_array[2] * (100 / stat_array[0]) << "%)" << endl;
if (stat_array[3] == 999)
    stat_array[3] = 0;
cout << "\nSend and recieve time.\n\tMin = " << stat_array[3] << " ms.\n\tMax = "
<< stat_array[4] << " ms.";
cout << "\n\tAverage = " << stat_array[5] << " ms." << endl;

ping_was_stopped = true;

closesocket(sock);
write_log(0, 0, "Socket was closed");
th.join();
system("pause");
write_log(0, 0, "Ping program end");

return 0;
}

```

В листинге 5 представлен код функции run_socket(string ip_address), отвечающей за корректное начало работы сокета и соединение по нему.

Листинг 5 – Код функции run_socket(string ip_address).

```
/* Функция создания и установки соединения сокета. */
int run_socket(string ip_address)
{
    // Начало работы с сетью.
    iResult = WSASStartup(MAKEWORD(2, 2), &wsaData);
    if (iResult != 0) {
        wprintf(L"WSAStartup failed: %d\n", iResult);
        write_log(1, iResult, "WSAStartup failed");
        write_log(0, 1, "Ping program end");
        return 1;
    }
    write_log(0, 0, "WSAStartup success");

    // Заполнение структуры хранящей информацию о адресе.
    if ((ip_address[0] > 47) && (ip_address[0] < 58)) // Если введен ip адрес.
    {
        char ip_array[4];

        if (!is_ip_valid(ip_address))
        {
            cout << "IP address is incorrect." << endl;
            write_log(1, 421, "IP address is incorrect");
            write_log(0, 1, "Ping program end");
            return 1;
        }

        ip_to_array(ip_address, ip_array);

        t_addr.ss_family = AF_INET;
        t_addr.__ss_pad1[2] = ip_array[0];
        t_addr.__ss_pad1[3] = ip_array[1];
        t_addr.__ss_pad1[4] = ip_array[2];
        t_addr.__ss_pad1[5] = ip_array[3];
    }
    else // Если введен домен.
    {
        // Получение ip адреса по имени хоста.
        auto getaddr_res = getaddrinfo(ip_address.c_str(), NULL, &hints,
&test_addr); // Адрес получаем в test_addr.
        if (getaddr_res != 0)
        {
            cout << "Unknown domain name." << endl;
            write_log(1, 422, "Unknown domain name");
            closesocket(sock);
            WSACleanup();
            write_log(0, 1, "Ping program end");
            return 1;
        }

        // Копирование полученного ip из структуры addrinfo* в sockaddr_storage.
        for (int i = 2; i < 6; i++)
            t_addr.__ss_pad1[i] = test_addr->ai_addr->sa_data[i];
        t_addr.ss_family = AF_INET;
    }

    // Инициализация сокета.
    sock = socket(AF_INET, SOCK_RAW, IPPROTO_ICMP);
    if (sock == INVALID_SOCKET)
    {
        cout << "Socket creation failed." << endl;
        write_log(1, 420, "Socket creation failed");
        WSACleanup();
        write_log(0, 1, "Ping program end");
    }
}
```

```

        return 1;
    }

    // Установка соединения по сокету.
    if (connect(sock, (sockaddr*)&t_addr, sizeof(t_addr)) == SOCKET_ERROR)
    {
        cout << "Connection to " << ip_address << " failed." << endl;
        write_log(1, 423, "Connection to " + ip_address + " failed");
        closesocket(sock);
        WSACleanup();
        write_log(0, 1, "Ping program end");
        return 1;
    }
    cout << "Connection to " << ip_address << " success." << endl;
    write_log(0, 0, "Connection to " + ip_address + " success");

    return 0;
}

```

В листинге 6 представлен код функции `send_packet(icmp_packet packet, int* stat_array)`, отвечающей за корректную отправку ICMP-пакета и подсчета статистики работы утилиты.

Листинг 6 – Код функции `send_packet(icmp_packet packet, int* stat_array)`.

```

/* Функция отправки ICMP пакета. */
int send_packet(icmp_packet packet, int* stat_array)
{
    if (ping_was_stopped)
        return -1;

    char buf[1024];
    int timeout = 5000;
    int t_addr_size = sizeof(t_addr);
    setsockopt(sock, SOL_SOCKET, SO_RCVTIMEO, (char*)&timeout, sizeof(int)); //
    Установка тайм-аута (5000 мс.).

    auto start = std::chrono::high_resolution_clock::now();

    // Отправка пакета.
    int send_res = sendto(sock, (char*)&packet, 40, 0, (SOCKADDR*)&t_addr,
sizeof(t_addr));
    if (send_res == SOCKET_ERROR)
    {
        cout << "Packet sending failed with " << GetLastError() << " error code."
<< endl;
        write_log(1, GetLastError(), "Packet sending fail");
        closesocket(sock);
        WSACleanup();
        write_log(0, 1, "Ping program end");
        return 1;
    }
    cout << "Packet with " << send_res - 8 << " bytes was sent success." << endl;
    write_log(0, 0, "Packet sending success");
    stat_array[0] += 1; // +1 отправленный пакет.

    // Получение ответа.
}

```

```

    int recv_res = recvfrom(sock, (char*)&buf, (int)strlen(buf), NULL,
(SOCKADDR*)&t_addr, &t_addr_size);
    if (GetLastError() == 10060)
    {
        cout << "Request waiting interval exceeded." << endl;
        write_log(1, 10060, "Request waiting interval exceeded");
        stat_array[2] += 1; // +1 потерянный пакет.
    }
    else if (recv_res == SOCKET_ERROR)
    {
        if (recv_res == 0)
        {
            cout << "The connection was closed." << endl;
            write_log(0, 0, "The connection was closed");
        }
        else
        {
            cout << "Data receiving failed with " << GetLastError() << endl;
            write_log(1, GetLastError(), "Data receiving failed");
        }
        closesocket(sock);
        WSACleanup();
        write_log(0, 1, "Ping program end");
        return 1;
    }

    auto end = std::chrono::high_resolution_clock::now();
    std::chrono::duration<double> elapsed_time = end - start; // Вычисление
затраченного на получение пакета времени.
    if (recv_res > 0)
    {
        cout << "Receieved " << recv_res - 28 << " bytes from " << ip_to_str(t_addr)
<< " in " << (int)(elapsed_time.count() * 1000) << " ms." << endl;
        write_log(0, 0, "Packet receiving success");

        // Сбор статистики.
        stat_array[1] += 1; // +1 полученный пакет.
        if (stat_array[3] > (int)(elapsed_time.count() * 1000)) // Минимальное
время получения ответа.
            stat_array[3] = (int)(elapsed_time.count() * 1000);
        if (stat_array[4] < (int)(elapsed_time.count() * 1000)) // Максимальное
время получения ответа.
            stat_array[4] = (int)(elapsed_time.count() * 1000);
        stat_array[5] += (int)(elapsed_time.count() * 1000); // Сумма времени
отправки всех сообщений

        // (для вычисления среднего времени).
    }

    return 0;
}

```

В листинге 7 представлен код функции `write_log(bool type, int error_code, string description)`, отвечающей за запись событий в лог-файл.

Листинг 7 – Код функции write_log(bool type, int error_code, string description).

```
/* Функция записи событий в лог-файл. */
int write_log(bool type, int error_code, string description)
{
    ofstream log_file;

    // Открытие файла.
    log_file.open("res\\log.txt", ios_base::app); // Открытие с добавлением в конец
    строки.
    if (!log_file.is_open())
    {
        _mkdir("res");
        log_file.open("res\\log.txt", ios_base::app);
    }

    // Запись в файл.
    __time64_t now;
    struct tm timeinfo;
    char buf[32];
    string s_type = "";
    if (type == 0)
        s_type = "operation";
    else
        s_type = "error";

    // Запись в buf текущей даты и времени.
    now = time(NULL);
    _localtime64_s(&timeinfo, &now);
    asctime_s(buf, &timeinfo);
    string s_buf = buf;

    size_t found = s_buf.find("\n");
    s_buf.replace(found, 1, "");
    s_buf = "[" + s_buf + "]" + " \t" + "[" + s_type + "]" + " \t" + "code: " +
    std::to_string(error_code) + " \t" + "description: " + description + ".\n";

    // Запись данных.
    log_file << s_buf.c_str();

    log_file.close();

    return 0;
}
```

В листинге 8 представлен код функции control_sum(icmp_packet packet), которая вычисляет контрольную сумму для входящего ICMP-пакета.

Листинг 8 – Код функции control_sum(icmp_packet packet).

```
/* Функция подсчета контрольной суммы. */
int control_sum(icmp_packet packet)
{
    int sum = 0;

    // Составление пары type&code в 16 бит.
    string pair = to_binary_string(packet.type) + to_binary_string(packet.code);
    sum += bin_to_short(pair);
    sum += _byteswap_ushort(packet.id); // 0x0100 -> 0x0001
    sum += _byteswap_ushort(packet.seq);

    pair = "";
    // Составление пар из значений поля данных.
    for (int i = 0; i < 32; i++)
    {
        pair += to_binary_string(packet.data[i]);

        if ((i % 2) != 0)
        {
            sum += bin_to_short(pair);
            pair = "";
        }
    }

    // Корректировка результата под 16 бит.
    while (sum > 0xffff)
        sum = (sum & 0xffff) + (sum >> 16);

    return _byteswap_ushort(~sum);
}
```

В листинге 9 представлены остальные вспомогательные функции, необходимые для формирования ICMP-пакета, остановки программы, преобразований из одной структуры данных в другую и т. д.

Листинг 9 – Код остальных вспомогательных функций.

```
/* Вспомогательные функции. */

/* Функция для потока, для выхода из программы. */
void f_for_thread()
{
    while (!ping_was_stopped)
    {
        Sleep(500);
        if (_getch() == VK_TAB) // Условие нажатия TAB.
        {
            ping_was_stopped = true;
            system("pause>0"); // Пауза без надписи "для продолжения нажмите
любую клавишу".
            exit(0);
        }
    }
}
```

```

    }
}

/* Функция перевода int в двоичный string. */
string to_binary_string(unsigned int n)
{
    string buffer; // символы ответа в обратном порядке
    // выделим память заранее по максимуму
    buffer.reserve(numeric_limits<unsigned int>::digits);
    do
    {
        buffer += char('0' + n % 2); // добавляем в конец
        n = n / 2;
    } while (n > 0);
    buffer = string(buffer.crbegin(), buffer.crend());
    while (buffer.length() < 8)
        buffer = '0' + buffer;
    return buffer; // разворачиваем результат
}

/* Функция перевода из двоичного значения string в десятичное short. */
short bin_to_short(string value)
{
    short result = 0;
    short mnozh = 1;
    for (size_t i = value.length() - 1; i > 0; i--)
    {
        result += (value[i] - 48) * mnozh;
        mnozh *= 2;
    }
    return result;
}

/* Функция формирования ICMP пакета. */
icmp_packet fill_icmp_packet()
{
    icmp_packet packet;
    packet.type = 8;
    packet.code = 0;
    packet.control_sum = 0;
    packet.id = 0x0100;
    packet.seq = 0x5407;

    return packet;
}

/* Функция преобразования char = 8 => string = "8". */
string ch_to_str(unsigned char val)
{
    string res = "";
    char chislo;

    // Обработка единиц.
    chislo = val % 10;
    val = val - chislo;
    chislo += 48;
    if (chislo != '0')
        res = res + chislo;

    // Обработка десятков.
    chislo = (val % 100);
    val = val - chislo;
    chislo /= 10;
    chislo += 48;
    if (chislo != '0')
        res = chislo + res;
}

```

```

        // Обработка сотен.
        chislo = (val % 1000);
        chislo /= 100;
        chislo += 48;
        if (chislo != '0')
            res = chislo + res;

        return res;
    }

    /* Функция представляющая строку ip в виде массива из 4-х элементов */
    char* ip_to_array(string ip, char* arr)
    {
        char val = 0;
        short cntr = 0;

        for (int i = 0; i < ip.length(); i++)
        {
            if ((ip[i] > 47) && (ip[i] < 58)) // Собираем символы цифр в число.
                val = (val * 10) + ((int)ip[i]) - 48;
            else
            {
                arr[cntr] = val;
                val = 0;
                cntr++;
            }
        }
        arr[cntr] = val;

        return arr;
    }

    /* Функция преобразования ip из структуры SOCKADDR_STORAGE в string. */
    string ip_to_str(SOCKADDR_STORAGE ip)
    {
        string res = "";

        for (int i = 2; i < 6; i++)
        {
            res += ch_to_str((unsigned char)ip.__ss_pad1[i]);
            if ((i + 1) < 6)
                res += '.';
        }

        return res;
    }

    /* Функция проверки ip адреса в виде строки на корректность. */
    bool is_ip_valid(string ip)
    {
        byte point_count = 0; // Переменная для хранения количества точек в адресе.
        char arr[4];
        string source_ip = ip;

        // Проверка на количество точек в адресе (должно быть 3).
        size_t found = ip.find(".");
        while (found != string::npos)
        {
            ip.replace(found, 1, "");
            point_count++;
            found = ip.find(".");
        }
    }

```



```

    if (point_count != 3)
        return false;

    // Проверка, являются ли все символы строки цифрами.
    for (int i = 0; i < ip.length(); i++)
        if (!((ip[i] > 47) && (ip[i] < 58)))
            return false;

    // Проверка каждого числа в ip адресе (0 >= chislo < 256).
    ip_to_array(source_ip, arr);
    for (int i = 0; i < 4; i++)
        if ((arr[i] < 0) || (arr[i] > 255))
            return false;

    return true;
}

```

В листинге 10 представлен полный код программы в одном файле ping.cpp.

Листинг 10 – Код файла ping.cpp.

```

/* Ping Project. */

/* Подключение библиотек. */
#include <WinSock2.h>
#include <WS2tcpip.h>
#include <iostream>
#include <string>
#include <intrin.h>
#include <chrono>
#include <time.h>
#include <conio.h>
#include <thread>
#include <fstream>
#include <direct.h>

/* Прочие объявления. */
#pragma comment(lib, "ws2_32.lib")

using namespace std;

/* Глобальные переменные и константы. */

SOCKET sock = INVALID_SOCKET;
struct sockaddr_storage t_addr;
struct addrinfo* test_addr = NULL;
struct addrinfo hints;
WSADATA wsaData = { 0 };
int iResult = 0;
bool ping_was_stopped = false;

struct sockaddr s_address;

// Структура описывающая ICMP-пакет.

```

```

struct icmp_packet
{
    // Заголовок пакета.
    char type; // Тип сообщения, 0 или 8.
    char code; // Код ошибки.
    short control_sum;
    short id;
    short seq; // Очередь.

    // Данные пакета.
    const char data[33] = "abcdefghijklmnopqrstuvwabcdefghi";
};

/* Вспомогательные функции. */

/* Функция записи событий в лог-файл. */
int write_log(bool type, int error_code, string description)
{
    ofstream log_file;

    // Открытие файла.
    log_file.open("res\\log.txt", ios_base::app); // Открытие с добавлением в
конец строки.
    if (!log_file.is_open())
    {
        _mkdir("res");
        log_file.open("res\\log.txt", ios_base::app);
    }

    // Запись в файл.
    __time64_t now;
    struct tm timeinfo;
    char buf[32];
    string s_type = "";
    if (type == 0)
        s_type = "operation";
    else
        s_type = "error";

    // Запись в buf текущей даты и времени.
    now = time(NULL);
    _localtime64_s(&timeinfo, &now);
    asctime_s(buf, &timeinfo);
    string s_buf = buf;

    size_t found = s_buf.find("\n");
    s_buf.replace(found, 1, "");
    s_buf = "[" + s_buf + "]" + " \t" + "[" + s_type + "]" + " \t" + "code: " +
std::to_string(error_code) + " \t" + "description: " + description + ".\n";

    // Запись данных.
    log_file << s_buf.c_str();

    log_file.close();

    return 0;
}

/* Функция для потока, для выхода из программы. */

```

```

void f_for_thread()
{
    while (!ping_was_stopped)
    {
        Sleep(500);
        if (_getch() == VK_TAB) // Условие нажатия TAB.
        {
            ping_was_stopped = true;
            system("pause>0"); // Пауза без надписи "для продолжения
нажмите любую клавишу".
            exit(0);
        }
    }
}

/* Функция перевода int в двоичный string. */
string to_binary_string(unsigned int n)
{
    string buffer; // символы ответа в обратном порядке
    // выделим память заранее по максимуму
    buffer.reserve(numeric_limits<unsigned int>::digits);
    do
    {
        buffer += char('0' + n % 2); // добавляем в конец
        n = n / 2;
    } while (n > 0);
    buffer = string(buffer.crbegin(), buffer.crend());
    while (buffer.length() < 8)
        buffer = '0' + buffer;
    return buffer; // разворачиваем результат
}

/* Функция перевода из двоичного значения string в десятичное short. */
short bin_to_short(string value)
{
    short result = 0;
    short mnozh = 1;
    for (size_t i = value.length() - 1; i > 0; i--)
    {
        result += (value[i] - 48) * mnozh;
        mnozh *= 2;
    }
    return result;
}

/* Функция подсчета контрольной суммы. */
int control_sum(ICMP_Packet packet)
{
    int sum = 0;

    // Составление пары type&code в 16 бит.
    string pair = to_binary_string(packet.type) + to_binary_string(packet.code);
    sum += bin_to_short(pair);
    sum += _byteswap_ushort(packet.id); // 0x0100 -> 0x0001
    sum += _byteswap_ushort(packet.seq);

    pair = "";
    // Составление пар из значений поля данных.
    for (int i = 0; i < 32; i++)
    {

```

```

        pair += to_binary_string(packet.data[i]);

        if ((i % 2) != 0)
        {
            sum += bin_to_short(pair);
            pair = "";
        }
    }

    // Корректировка результата под 16 бит.
    while (sum > 0xffff)
        sum = (sum & 0xffff) + (sum >> 16);

    return _byteswap_ushort(~sum);
}

/* Функция формирования ICMP пакета. */
icmp_packet fill_icmp_packet()
{
    icmp_packet packet;
    packet.type = 8;
    packet.code = 0;
    packet.control_sum = 0;
    packet.id = 0x0100;
    packet.seq = 0x5407;

    return packet;
}

/* Функция преобразования char = 8 => string = "8". */
string ch_to_str(unsigned char val)
{
    string res = "";
    char chislo;

    // Обработка едениц.
    chislo = val % 10;
    val = val - chislo;
    chislo += 48;
    if (chislo != '0')
        res = res + chislo;

    // Обработка десятков.
    chislo = (val % 100);
    val = val - chislo;
    chislo /= 10;
    chislo += 48;
    if (chislo != '0')
        res = chislo + res;

    // Обработка сотен.
    chislo = (val % 1000);
    chislo /= 100;
    chislo += 48;
    if (chislo != '0')
        res = chislo + res;

    return res;
}

```

```

/* Функция представляющая строку ip в виде массива из 4-х элементов */
char* ip_to_array(string ip, char* arr)
{
    char val = 0;
    short cnt = 0;

    for (int i = 0; i < ip.length(); i++)
    {
        if ((ip[i] > 47) && (ip[i] < 58)) // Собираем символы цифр в число.
            val = (val * 10) + ((int)ip[i]) - 48;
        else
        {
            arr[cnt] = val;
            val = 0;
            cnt++;
        }
    }
    arr[cnt] = val;

    return arr;
}

/* Функция преобразования ip из структуры SOCKADDR_STORAGE в string. */
string ip_to_str(SOCKADDR_STORAGE ip)
{
    string res = "";

    for (int i = 2; i < 6; i++)
    {
        res += ch_to_str((unsigned char)ip.__ss_pad1[i]);
        if ((i + 1) < 6)
            res += '.';
    }

    return res;
}

/* Функция отправки ICMP пакета. */
int send_packet(ICMP_PACKET packet, int* stat_array)
{
    if (ping_was_stopped)
        return -1;

    char buf[1024];
    int timeout = 5000;
    int t_addr_size = sizeof(t_addr);
    setsockopt(sock, SOL_SOCKET, SO_RCVTIMEO, (char*)&timeout, sizeof(int)); //
    Установка тайм-аута (5000 мс.).

    auto start = std::chrono::high_resolution_clock::now();

    // Отправка пакета.
    int send_res = sendto(sock, (char*)&packet, 40, 0, (SOCKADDR*)&t_addr,
sizeof(t_addr));
    if (send_res == SOCKET_ERROR)
    {
        cout << "Packet sending failed with " << GetLastError() << " error
code." << endl;
        write_log(1, GetLastError(), "Packet sending fail");
        closesocket(sock);
    }
}

```

```

        WSACleanup();
        write_log(0, 1, "Ping program end");
        return 1;
    }
    cout << "Packet with " << send_res - 8 << " bytes was sent success." << endl;
    write_log(0, 0, "Packet sending success");
    stat_array[0] += 1; // +1 отправленный пакет.

    // Получение ответа.
    int recv_res = recvfrom(sock, (char*)&buf, (int)strlen(buf), NULL,
(SOCKADDR*)&t_addr, &t_addr_size);
    if (GetLastError() == 10060)
    {
        cout << "Request waiting interval exceeded." << endl;
        write_log(1, 10060, "Request waiting interval exceeded");
        stat_array[2] += 1; // +1 потерянный пакет.
    }
    else if (recv_res == SOCKET_ERROR)
    {
        if (recv_res == 0)
        {
            cout << "The connection was closed." << endl;
            write_log(0, 0, "The connection was closed");
        }
        else
        {
            cout << "Data receiving failed with " << GetLastError() << endl;
            write_log(1, GetLastError(), "Data receiving failed");
        }
        closesocket(sock);
        WSACleanup();
        write_log(0, 1, "Ping program end");
        return 1;
    }

    auto end = std::chrono::high_resolution_clock::now();
    std::chrono::duration<double> elapsed_time = end - start; // Вычисление
затраченного на получение пакета времени.
    if (recv_res > 0)
    {
        cout << "Receved " << recv_res - 28 << " bytes from " <<
ip_to_str(t_addr) << " in " << (int)(elapsed_time.count() * 1000) << " ms." << endl;
        write_log(0, 0, "Packet receiving success");

        // Сбор статистики.
        stat_array[1] += 1; // +1 полученный пакет.
        if (stat_array[3] > (int)(elapsed_time.count() * 1000)) // Минимальное
время получения ответа.
            stat_array[3] = (int)(elapsed_time.count() * 1000);
        if (stat_array[4] < (int)(elapsed_time.count() * 1000)) //
Максимальное время получения ответа.
            stat_array[4] = (int)(elapsed_time.count() * 1000);
        stat_array[5] += (int)(elapsed_time.count() * 1000); // Сумма времени
отправки всех сообщений

        // (для вычисления среднего времени).
    }

    return 0;
}

```

```

/* Функция проверки ip адреса в виде строки на корректность. */
bool is_ip_valid(string ip)
{
    byte point_count = 0; // Переменная для хранения количества точек в адресе.
    char arr[4];
    string source_ip = ip;

    // Проверка на количество точек в адресе (должно быть 3).
    size_t found = ip.find(".");
    while (found != string::npos)
    {
        ip.replace(found, 1, "");
        point_count++;
        found = ip.find(".");
    }

    if (point_count != 3)
        return false;

    // Проверка, являются ли все символы строки цифрами.
    for (int i = 0; i < ip.length(); i++)
        if (!((ip[i] > 47) && (ip[i] < 58)))
            return false;

    // Проверка каждого числа в ip адресе (0 >= chislo < 256).
    ip_to_array(source_ip, arr);
    for (int i = 0; i < 4; i++)
        if ((arr[i] < 0) || (arr[i] > 255))
            return false;

    return true;
}

/* Функция создания и установки соединения сокета. */
int run_socket(string ip_address)
{
    // Начало работы с сетью.
    iResult = WSASStartup(MAKEWORD(2, 2), &wsaData);
    if (iResult != 0) {
        wprintf(L"WSAStartup failed: %d\n", iResult);
        write_log(1, iResult, "WSAStartup failed");
        write_log(0, 1, "Ping program end");
        return 1;
    }
    write_log(0, 0, "WSAStartup success");

    // Заполнение структуры хранящей информацию о адресе.
    if ((ip_address[0] > 47) && (ip_address[0] < 58)) // Если введен ip адрес.
    {
        char ip_array[4];

        if (!is_ip_valid(ip_address))
        {
            cout << "IP address is incorrect." << endl;
            write_log(1, 421, "IP adress is incorrect");
            write_log(0, 1, "Ping program end");
            return 1;
        }
    }
}

```

```

        ip_to_array(ip_address, ip_array);

        t_addr.ss_family = AF_INET;
        t_addr.__ss_pad1[2] = ip_array[0];
        t_addr.__ss_pad1[3] = ip_array[1];
        t_addr.__ss_pad1[4] = ip_array[2];
        t_addr.__ss_pad1[5] = ip_array[3];
    }
    else // Если введен домен.
    {
        // Получение ip адреса по имени хоста.
        auto getaddr_res = getaddrinfo(ip_address.c_str(), NULL, &hints,
&test_addr); // Адрес получаем в test_addr.
        if (getaddr_res != 0)
        {
            cout << "Unknown domain name." << endl;
            write_log(1, 422, "Unknown domain name");
            closesocket(sock);
            WSACleanup();
            write_log(0, 1, "Ping program end");
            return 1;
        }

        // Копирование полученного ip из структуры addrinfo* в
sockaddr_storage.
        for (int i = 2; i < 6; i++)
            t_addr.__ss_pad1[i] = test_addr->ai_addr->sa_data[i];
        t_addr.ss_family = AF_INET;
    }

    // Инициализация сокета.
    sock = socket(AF_INET, SOCK_RAW, IPPROTO_ICMP);
    if (sock == INVALID_SOCKET)
    {
        cout << "Socket creation failed." << endl;
        write_log(1, 420, "Socket creation failed");
        WSACleanup();
        write_log(0, 1, "Ping program end");
        return 1;
    }

    // Установка соединения по сокету.
    if (connect(sock, (sockaddr*)&t_addr, sizeof(t_addr)) == SOCKET_ERROR)
    {
        cout << "Connection to " << ip_address << " failed." << endl;
        write_log(1, 423, "Connection to " + ip_address + " failed");
        closesocket(sock);
        WSACleanup();
        write_log(0, 1, "Ping program end");
        return 1;
    }
    cout << "Connection to " << ip_address << " success." << endl;
    write_log(0, 0, "Connection to " + ip_address + " success");

    return 0;
}

/* Основная функция main. */
int main()

```



```

{
    write_log(0, 0, "Ping program start");
    int stat_array[6];
    stat_array[0] = 0;
    stat_array[1] = 0;
    stat_array[2] = 0;
    stat_array[3] = 999;
    stat_array[4] = 0;
    stat_array[5] = 0;
    int ping_count;
    int real_ping_count = 0;
    string ping_count_str;
    string ip_addr;

    // Заполнение пакета соответствующими данными.
    icmp_packet packet = fill_icmp_packet();
    packet.control_sum = control_sum(packet); // Подсчет контрольной суммы

    cout << "Ping program.\n" << endl;
    cout << "Enter the IP-Address or domain name to connect." << endl;

    getline(cin, ip_addr);

    // Начало работы сокета.
    if (run_socket(ip_addr) != 0)
    {
        write_log(1, 425, "Run socket unknown error");
        write_log(0, 1, "Ping program end");
        system("pause");
        return 1;
    }

    cout << "How many times you want to ping?" << endl;
    getline(cin, ping_count_str);

    cout << endl;
    if ((ping_count_str != "") && (ping_count_str.length() < 10)) // Если
пользователь ввел сколько раз нужно отправить пакеты.
    {
        try
        {
            ping_count = stoi(ping_count_str);
        }
        catch (exception e)
        {
            ping_count = 1;
        }
    }
    else // Если оставил поле сколько раз нужно отправить пакеты пустым.
        ping_count = 1;

    if ((ping_count <= 0) || (ping_count > 5000))
        ping_count = 1;

    // Отправка пакетов.
    thread th(f_for_thread);
    cout << "Start sending echo-requests..." << endl;
    Sleep(500);
}

```

```

for (int i = 0; i < ping_count; i++)
{
    int send_res = send_packet(packet, stat_array);
    if (send_res == -1) // Если пинг завершен досрочно, нажатием клавиши
TAB.
    {
        write_log(0, 1, "Pinging was stopped by user");
        break;
    }
    else if (send_res != 0) // Остальные случаи ошибки отправки пакета.
    {
        ping_was_stopped = true;
        th.join();
        write_log(1, 426, "Packet send unknown error");
        write_log(0, 1, "Ping program end");
        system("pause");
        return 1;
    }

    // Увеличение значения очереди и последующий подсчет новой контрольной
суммы.
    // _byteswap_ushort(): 0x0001 => 0x0100 (LE => BE).
    packet.seq += _byteswap_ushort(1);
    packet.control_sum = 0;
    packet.control_sum = control_sum(packet);

    real_ping_count++;
}

stat_array[5] /= real_ping_count; // Вычисление среднего времени получения
пакета.

// Вывод статистики пинга.
cout << "\nStatistics.\n\tPackets sent:\t" << stat_array[0] << "\n\tPackets
received:\t" << stat_array[1] << "\n\tPacket loss:\t" << stat_array[2];
cout << " (" << stat_array[2] * (100 / stat_array[0]) << "%)" << endl;
if (stat_array[3] == 999)
    stat_array[3] = 0;
cout << "\nSend and recieve time.\n\tMin = " << stat_array[3] << " ms.\n\tMax
= " << stat_array[4] << " ms.";
cout << "\n\tAverage = " << stat_array[5] << " ms." << endl;

ping_was_stopped = true;

closesocket(sock);
write_log(0, 0, "Socket was closed");
th.join();
system("pause");
write_log(0, 0, "Ping program end");

return 0;
}

```

В листинге 11 представлен код файла CmakeLists.txt, файл необходим для сборки утилиты системой CMake.

Листинг 11 – Код файла CmakeLists.txt.

```
cmake_minimum_required(VERSION 3.5)

project(pingg)

set(CMAKE_CXX_STANDARD 11)

file(GLOB
      CPPS "*.cpp")

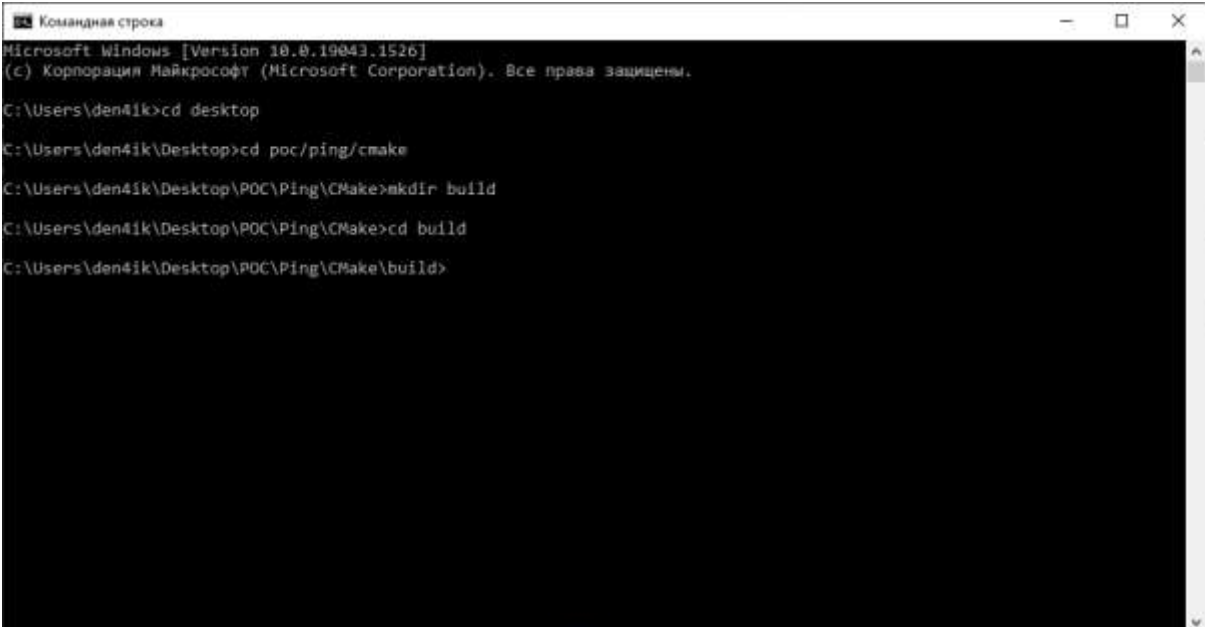
add_executable(${PROJECT_NAME} ${CPPS})

target_link_libraries(${PROJECT_NAME} wsock32 ws2_32)
```

6. Сборка утилиты.

Для сборки разработанного продукта воспользуемся системой CMake. Инструкции для сборки представлены в листинге 11. На рисунках 5, 6 и 7 последовательно изображены этапы сборки.

В директории ...\\Ping\\CMake расположим файлы ping.cpp и CmakeLists.txt, а также создадим папку, где будет расположена собранная программа, назовем ее build. Вышеописанные действия представлены на рисунке 5.

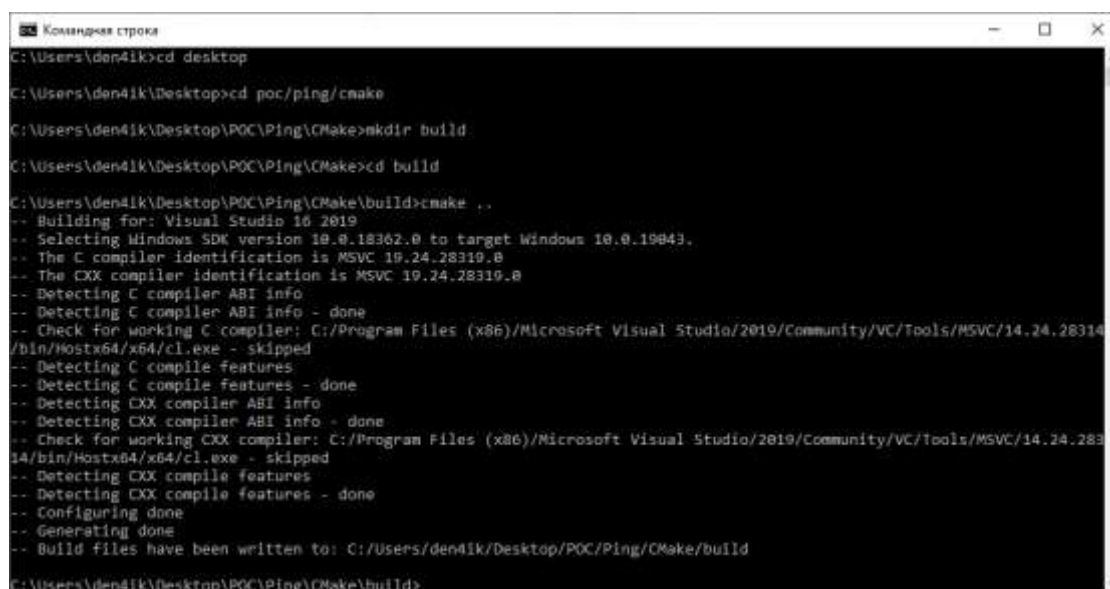


```
Командная строка
Microsoft Windows [Version 10.0.19043.1526]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

C:\Users\den4ik>cd desktop
C:\Users\den4ik\Desktop>cd poc/ping/cmake
C:\Users\den4ik\Desktop\POC\Ping\CMake>mkdir build
C:\Users\den4ik\Desktop\POC\Ping\CMake>cd build
C:\Users\den4ik\Desktop\POC\Ping\CMake\build>
```

Рис. 5 – Создание директории в которой будет собрана программа.

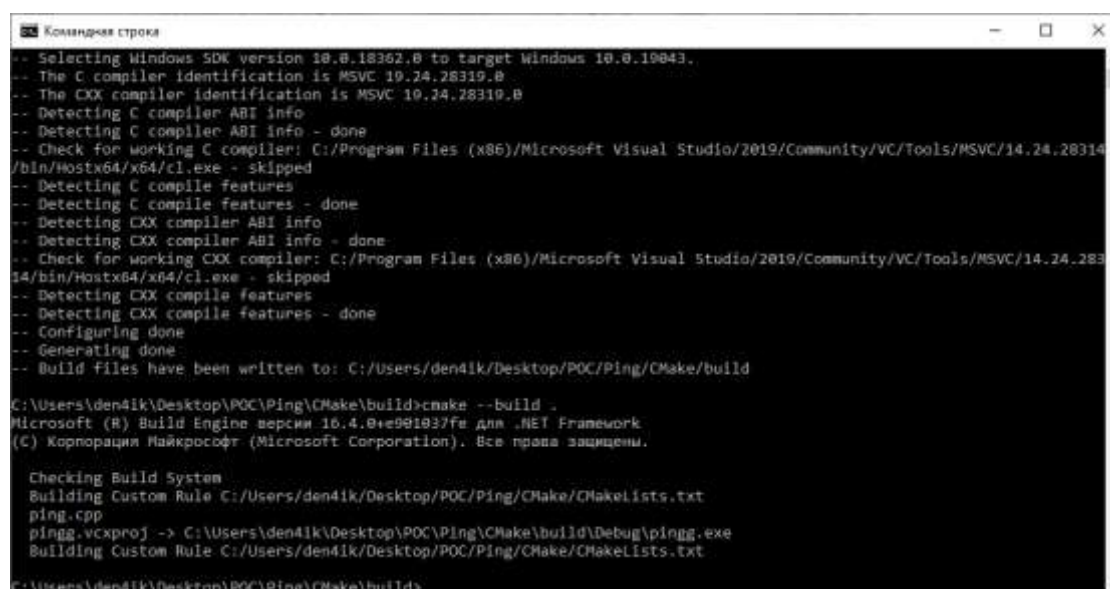
Из директории `..\Ping\CMake\build` вводим команду “`cmake ..`”, после этого в этом же каталоге появится решение для Windows. На рисунке 6 изображены вышеописанные действия.



```
Командная строка
C:\Users\den4ik>cd desktop
C:\Users\den4ik\Desktop>cd poc\ping\cmake
C:\Users\den4ik\Desktop\POC\Ping\CMake>mkdir build
C:\Users\den4ik\Desktop\POC\Ping\CMake>cd build
C:\Users\den4ik\Desktop\POC\Ping\CMake\build>cmake ..
-- Building for: Visual Studio 16 2019
-- Selecting Windows SDK version 10.0.18362.0 to target Windows 10.0.19043.
-- The C compiler identification is MSVC 19.24.28319.0
-- The CXX compiler identification is MSVC 19.24.28319.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: C:/Program Files (x86)/Microsoft Visual Studio/2019/Community/VC/Tools/MSVC/14.24.28314/bin/Hostx64/x64/cl.exe - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: C:/Program Files (x86)/Microsoft Visual Studio/2019/Community/VC/Tools/MSVC/14.24.28314/bin/Hostx64/x64/cl.exe - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: C:/Users/den4ik/Desktop/POC/Ping/CMake/build
C:\Users\den4ik\Desktop\POC\Ping\CMake\build>
```

Рис. 6 – Создание решения для Windows.

Для сборки исполняемого файла из той же директории `build` введем команду “`cmake --build .`”, после чего в каталоге `build\Debug` появится exe файл утилиты. Вышеописанные действия представлены на рисунке 7.



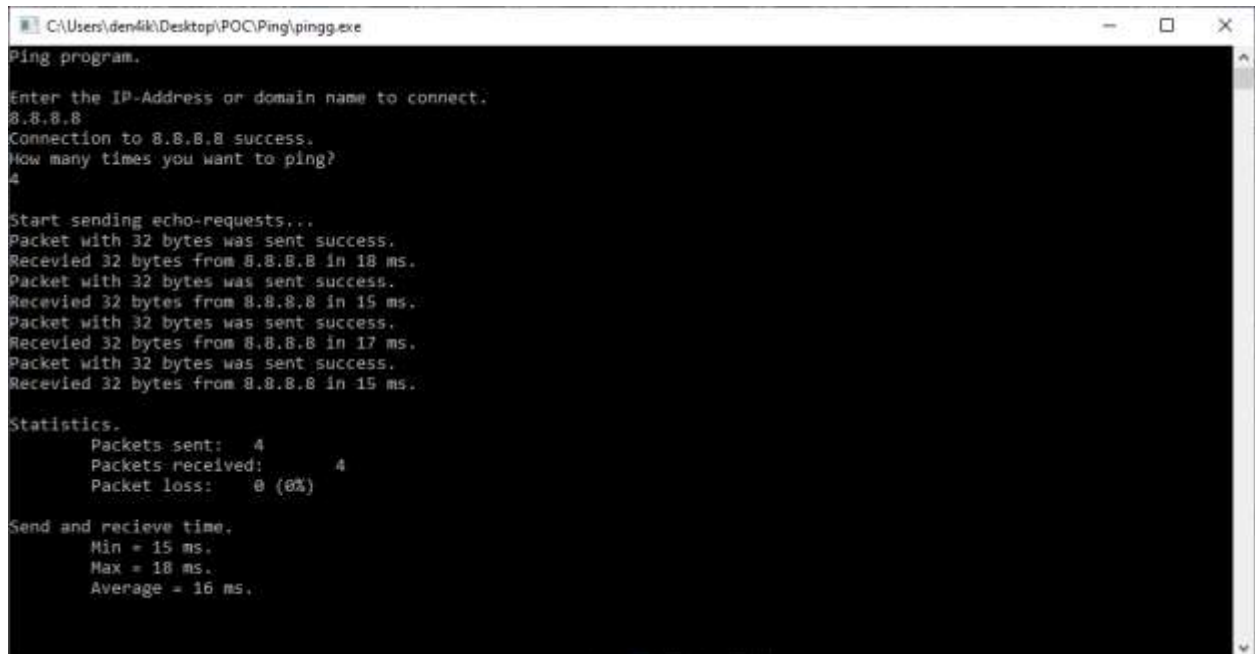
```
Командная строка
-- Selecting Windows SDK version 10.0.18362.0 to target Windows 10.0.19043.
-- The C compiler identification is MSVC 19.24.28319.0
-- The CXX compiler identification is MSVC 19.24.28319.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: C:/Program Files (x86)/Microsoft Visual Studio/2019/Community/VC/Tools/MSVC/14.24.28314/bin/Hostx64/x64/cl.exe - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: C:/Program Files (x86)/Microsoft Visual Studio/2019/Community/VC/Tools/MSVC/14.24.28314/bin/Hostx64/x64/cl.exe - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: C:/Users/den4ik/Desktop/POC/Ping/CMake/build
C:\Users\den4ik\Desktop\POC\Ping\CMake\build>cmake --build .
Microsoft (R) Build Engine версии 16.4.0+e991937fe для .NET Framework
(C) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

Checking Build System
Building Custom Rule C:/Users/den4ik/Desktop/POC/Ping/CMake/CMakeLists.txt
ping.cpp
pingg.vcxproj -> C:/Users/den4ik/Desktop/POC/Ping/CMake/build/Debug/pingg.exe
Building Custom Rule C:/Users/den4ik/Desktop/POC/Ping/CMake/CMakeLists.txt
C:\Users\den4ik\Desktop\POC\Ping\CMake\build>
```

Рис. 7 – Сборка исполняемого файла.

7. Результаты тестирования.

На рисунках ниже представлены скриншоты работы утилиты, а также ее реакции на ввод некорректных данных.



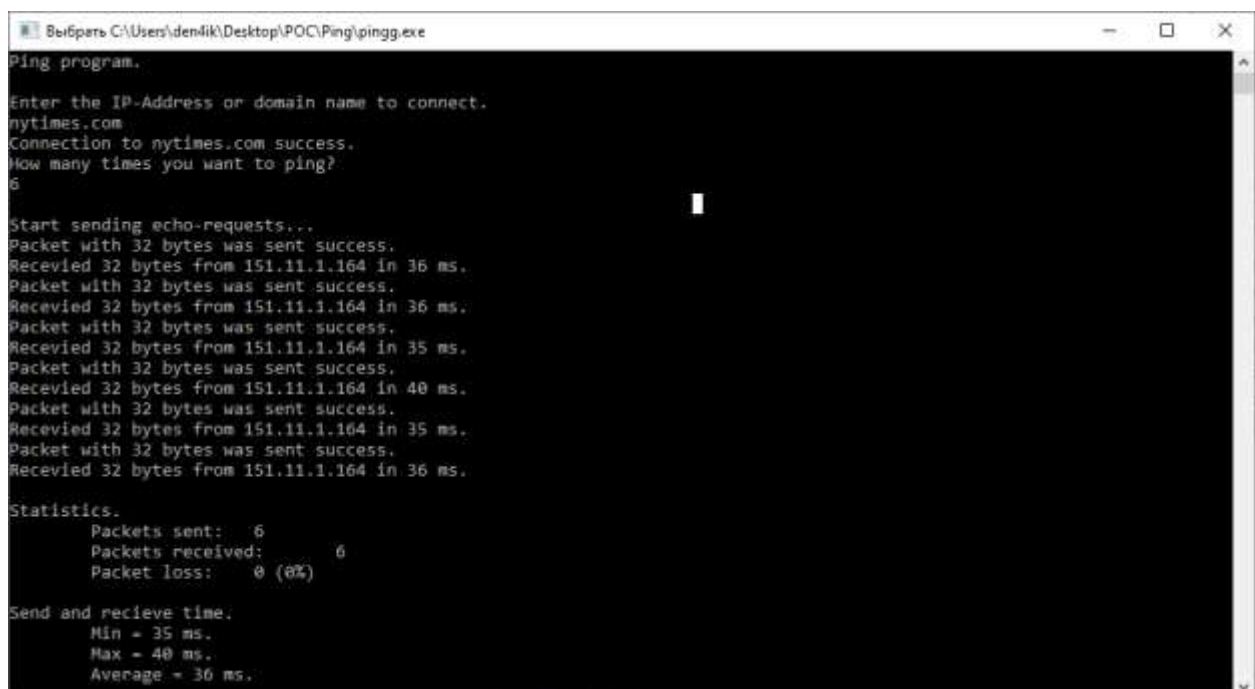
```
C:\Users\denlik\Desktop\POC\Ping\pingg.exe
Ping program.
Enter the IP-Address or domain name to connect:
8.8.8.8
Connection to 8.8.8.8 success.
How many times you want to ping?
4

Start sending echo-requests...
Packet with 32 bytes was sent success.
Receivied 32 bytes from 8.8.8.8 in 18 ms.
Packet with 32 bytes was sent success.
Receivied 32 bytes from 8.8.8.8 in 15 ms.
Packet with 32 bytes was sent success.
Receivied 32 bytes from 8.8.8.8 in 17 ms.
Packet with 32 bytes was sent success.
Receivied 32 bytes from 8.8.8.8 in 15 ms.

Statistics.
Packets sent: 4
Packets received: 4
Packet loss: 0 (0%)

Send and recieve time.
Min = 15 ms.
Max = 18 ms.
Average = 16 ms.
```

Рис. 8 – Результат работы утилиты.



```
Былбрат C:\Users\denlik\Desktop\POC\Ping\pingg.exe
Ping program.
Enter the IP-Address or domain name to connect.
nytimes.com
Connection to nytimes.com success.
How many times you want to ping?
6

Start sending echo-requests...
Packet with 32 bytes was sent success.
Receivied 32 bytes from 151.11.1.164 in 36 ms.
Packet with 32 bytes was sent success.
Receivied 32 bytes from 151.11.1.164 in 36 ms.
Packet with 32 bytes was sent success.
Receivied 32 bytes from 151.11.1.164 in 35 ms.
Packet with 32 bytes was sent success.
Receivied 32 bytes from 151.11.1.164 in 40 ms.
Packet with 32 bytes was sent success.
Receivied 32 bytes from 151.11.1.164 in 35 ms.
Packet with 32 bytes was sent success.
Receivied 32 bytes from 151.11.1.164 in 36 ms.

Statistics.
Packets sent: 6
Packets received: 6
Packet loss: 0 (0%)

Send and recieve time.
Min = 35 ms.
Max = 40 ms.
Average = 36 ms.
```

Рис. 9 – Результат работы утилиты.

```
C:\Users\denlik\Desktop\POC\Ping\CMake\build\Debug\pingg.exe
Ping program.
Enter the IP-Address or domain name to connect:
156.24.70.219
Connection to 156.24.70.219 success.
How many times you want to ping?
4

Start sending echo-requests...
Packet with 32 bytes was sent success.
Request waiting interval exceeded.
Packet with 32 bytes was sent success.
Request waiting interval exceeded.
Packet with 32 bytes was sent success.
Request waiting interval exceeded.
Packet with 32 bytes was sent success.
Request waiting interval exceeded.

Statistics.
Packets sent: 4
Packets received: 0
Packet loss: 4 (100%)

Send and recieve time.
Min = 0 ms.
Max = 0 ms.
Average = 0 ms.
```

Рис. 10 – Результат рабо ты утилиты.

```
C:\Users\denlik\Desktop\POC\Ping\CMake\build\Debug\pingg.exe
Ping program.
Enter the IP-Address or domain name to connect:
rambler.ru
Connection to rambler.ru success.
How many times you want to ping?
3

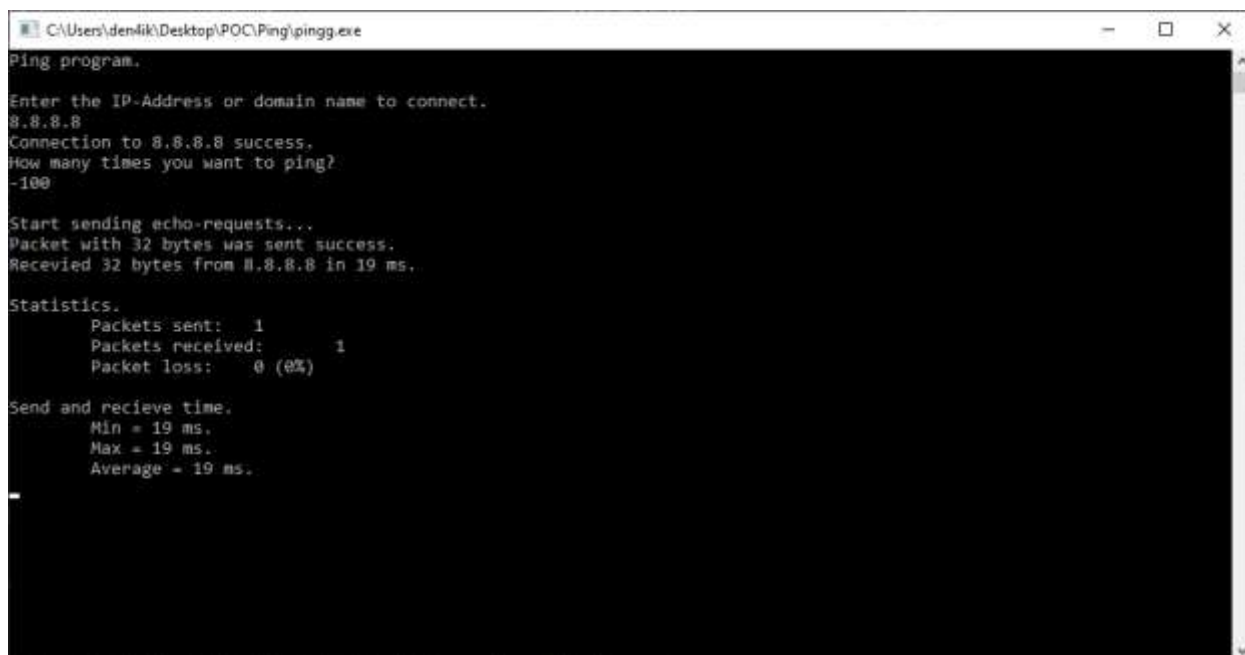
Start sending echo-requests...
Packet with 32 bytes was sent success.
Receivied 32 bytes from 81.19.82.98 in 3 ms.
Packet with 32 bytes was sent success.
Receivied 32 bytes from 81.19.82.98 in 2 ms.
Packet with 32 bytes was sent success.
Receivied 32 bytes from 81.19.82.98 in 2 ms.

Statistics.
Packets sent: 3
Packets received: 3
Packet loss: 0 (0%)

Send and recieve time.
Min = 2 ms.
Max = 3 ms.
Average = 2 ms.
Для продолжения нажмите любую клавишу . . .
```

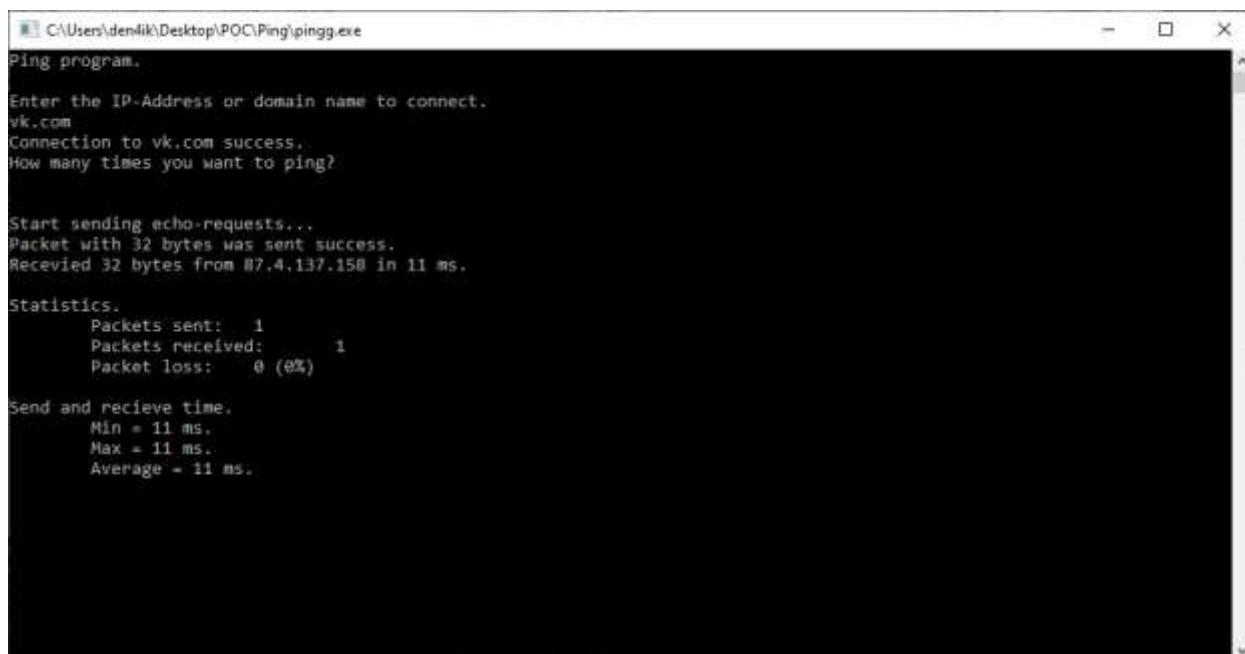
Рис. 11 – Результат работы утилиты.

На рисунках ниже изображены результаты обработки ошибок утилитой, при некорректно введенных значениях.



```
Ping program.  
Enter the IP-Address or domain name to connect.  
8.8.8.8  
Connection to 8.8.8.8 success.  
How many times you want to ping?  
-100  
  
Start sending echo-requests...  
Packet with 32 bytes was sent success.  
Receieved 32 bytes from 8.8.8.8 in 19 ms.  
  
Statistics.  
Packets sent: 1  
Packets received: 1  
Packet loss: 0 (0%)  
  
Send and recieve time.  
Min = 19 ms.  
Max = 19 ms.  
Average = 19 ms.  
-
```

Рис. 12 – Обработка ошибок (некорректное кол-во пакетов для отправки).



```
Ping program.  
Enter the IP-Address or domain name to connect.  
vk.com  
Connection to vk.com success.  
How many times you want to ping?  
  
Start sending echo-requests...  
Packet with 32 bytes was sent success.  
Receieved 32 bytes from 74.137.158 in 11 ms.  
  
Statistics.  
Packets sent: 1  
Packets received: 1  
Packet loss: 0 (0%)  
  
Send and recieve time.  
Min = 11 ms.  
Max = 11 ms.  
Average = 11 ms.
```

Рис. 13 – Обработка ошибок (некорректное кол-во пакетов для отправки).

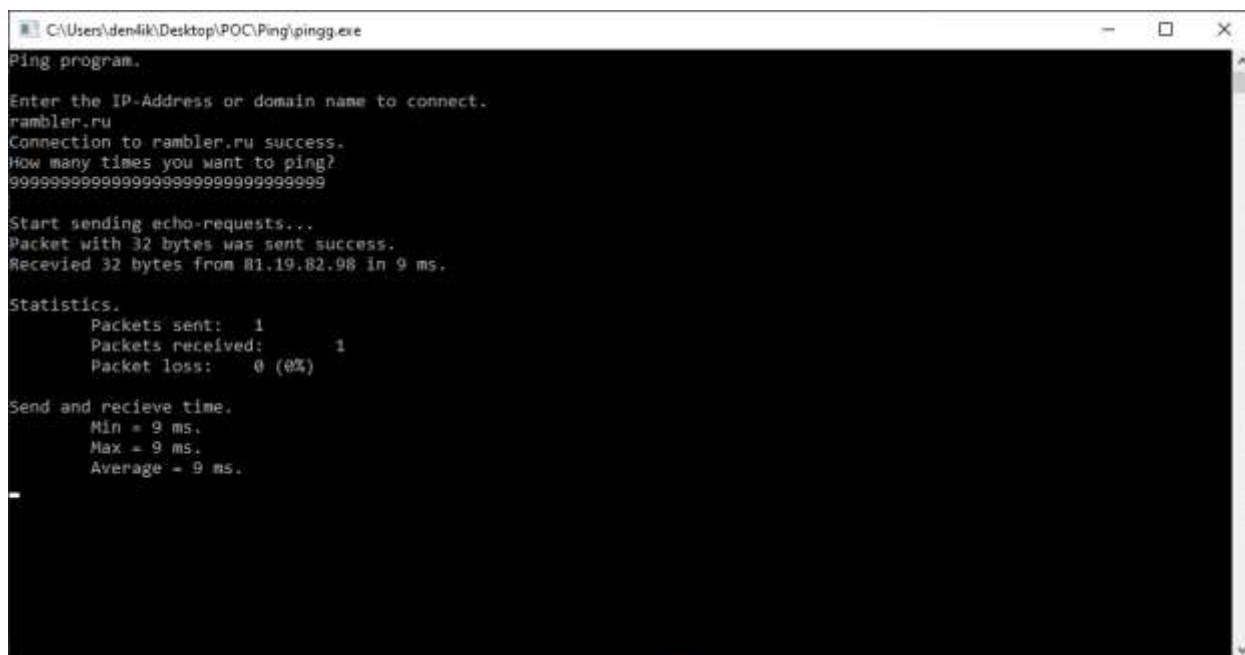


Рис. 14 – Обработка ошибок (некорректное кол-во пакетов для отправки).

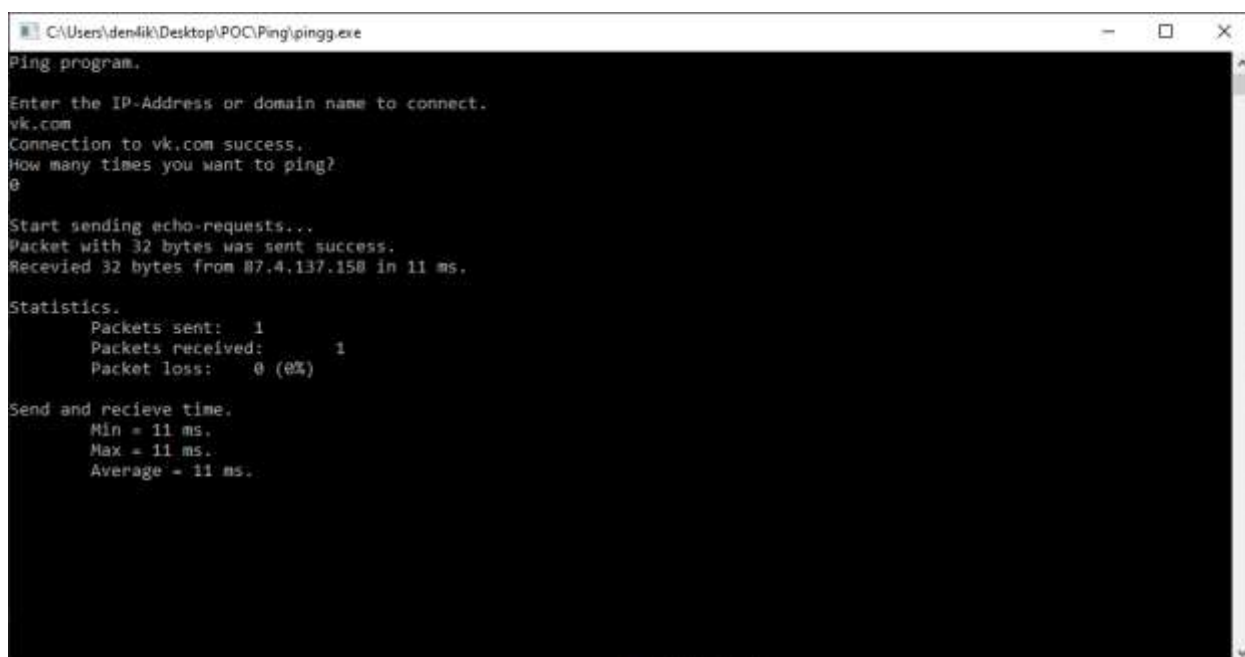


Рис. 15 – Обработка ошибок (некорректное кол-во пакетов для отправки).

```
C:\Users\denlik\Desktop\POC\Ping\pingg.exe
Ping program.
Enter the IP-Address or domain name to connect.
vk.com
Connection to vk.com success.
How many times you want to ping?
ten

Start sending echo-requests...
Packet with 32 bytes was sent success.
Received 32 bytes from 87.4.137.158 in 49 ms.

Statistics.
Packets sent: 1
Packets received: 1
Packet loss: 0 (0%)

Send and receive time.
Min = 49 ms.
Max = 49 ms.
Average = 49 ms.
```

Рис. 16 – Обработка ошибок (некорректное кол-во пакетов для отправки).

```
C:\Users\denlik\Desktop\POC\Ping\pingg.exe
Ping program.
Enter the IP-Address or domain name to connect.
8888
IP address is incorrect.
Для продолжения нажмите любую клавишу . . .
```

Рис. 17 – Обработка ошибок (некорректный IP-адрес).

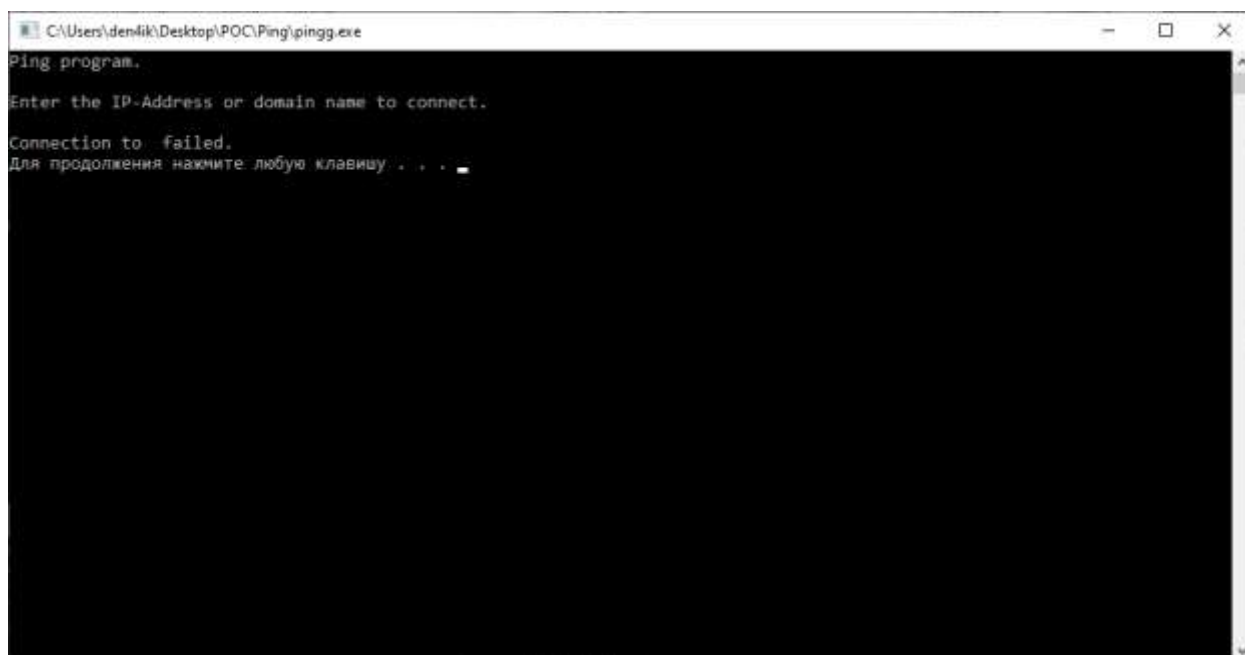


Рис. 18 – Обработка ошибок (некорректный IP-адрес).

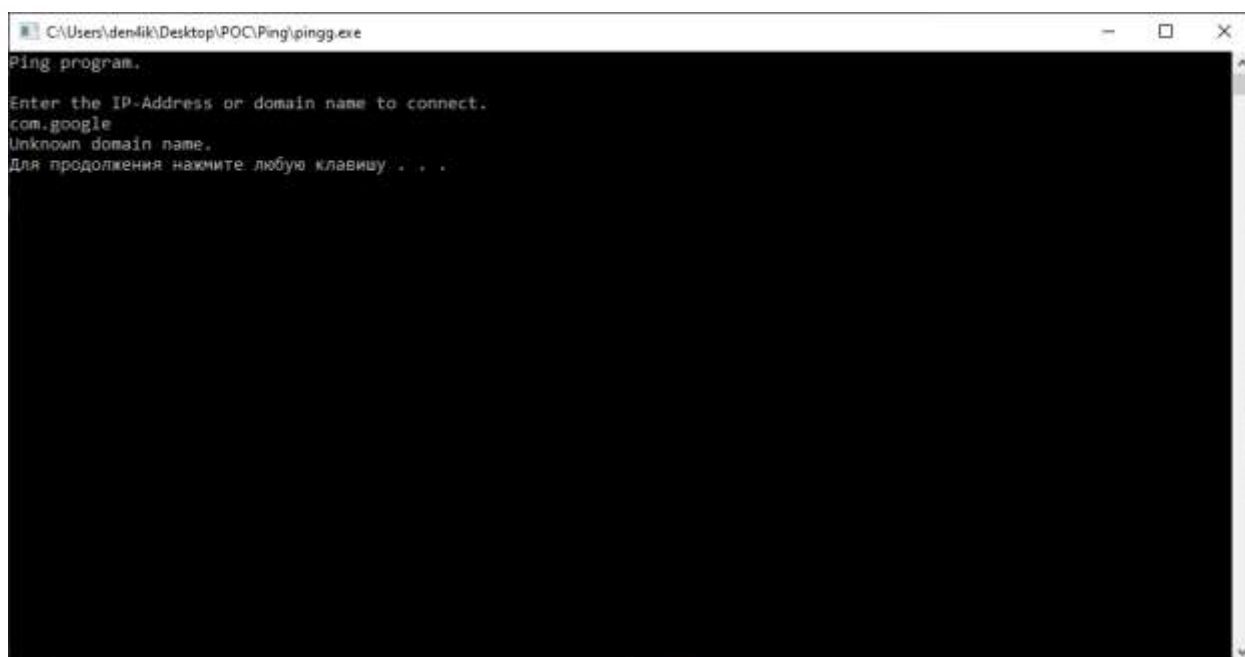


Рис. 19 – Обработка ошибок (некорректное имя домена).

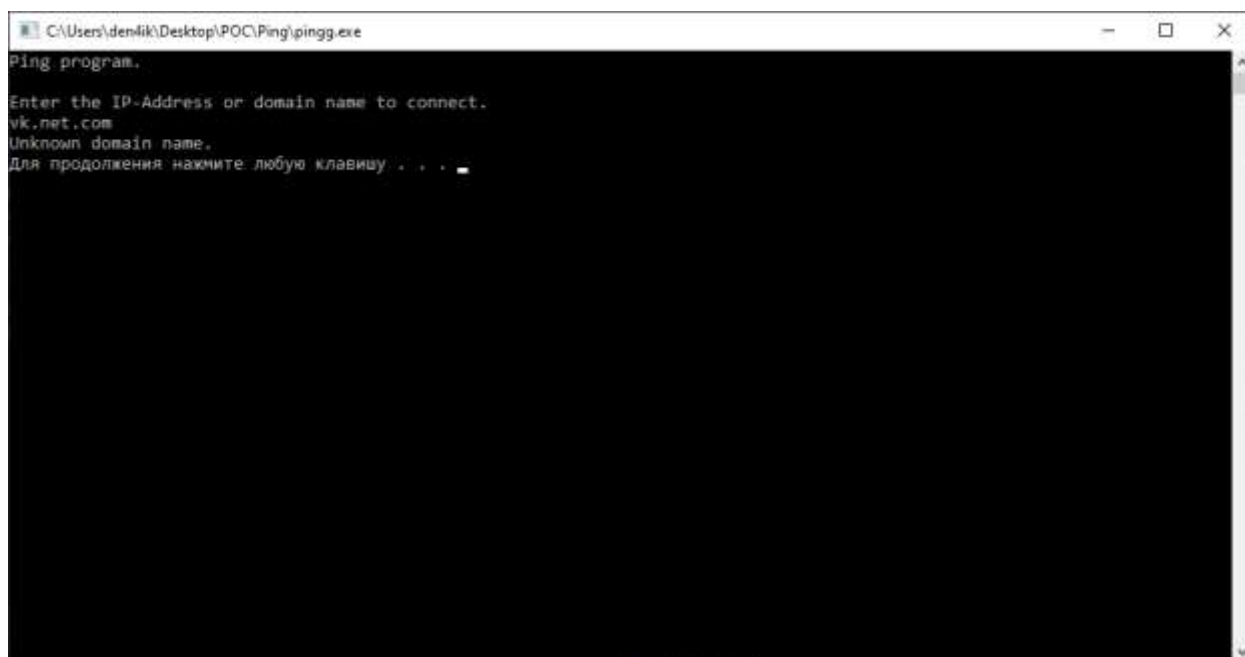


Рис. 20 – Обработка ошибок (некорректное имя домена).

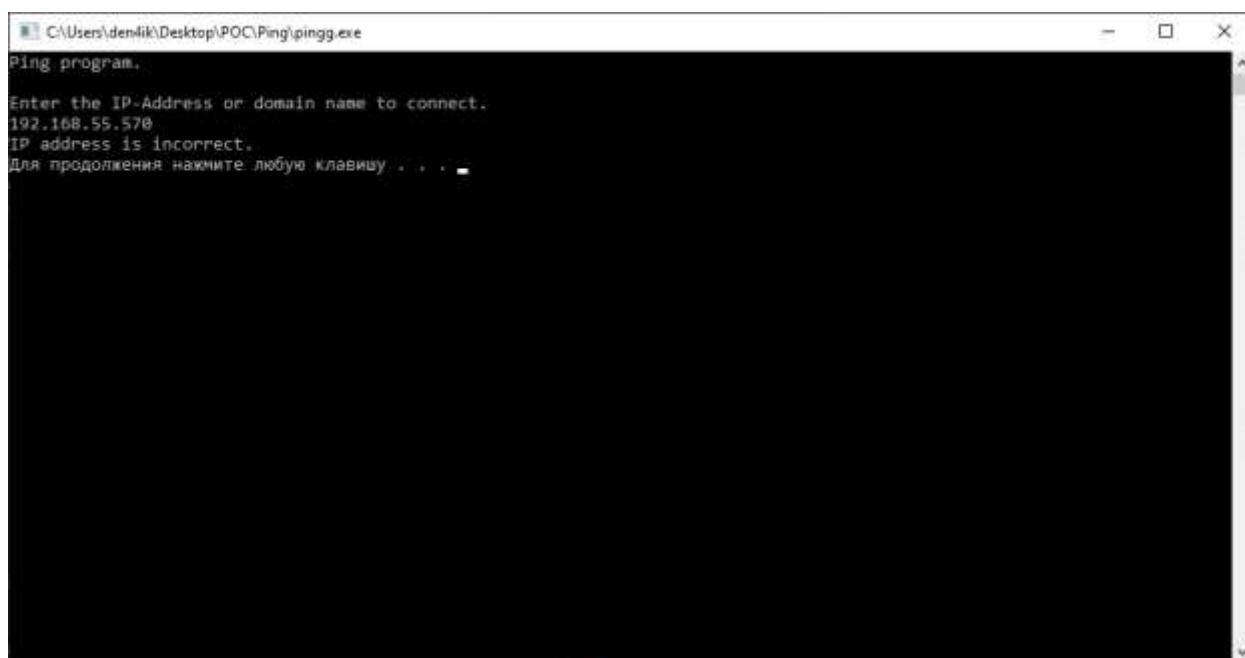


Рис. 21 – Обработка ошибок (некорректный IP-адрес).

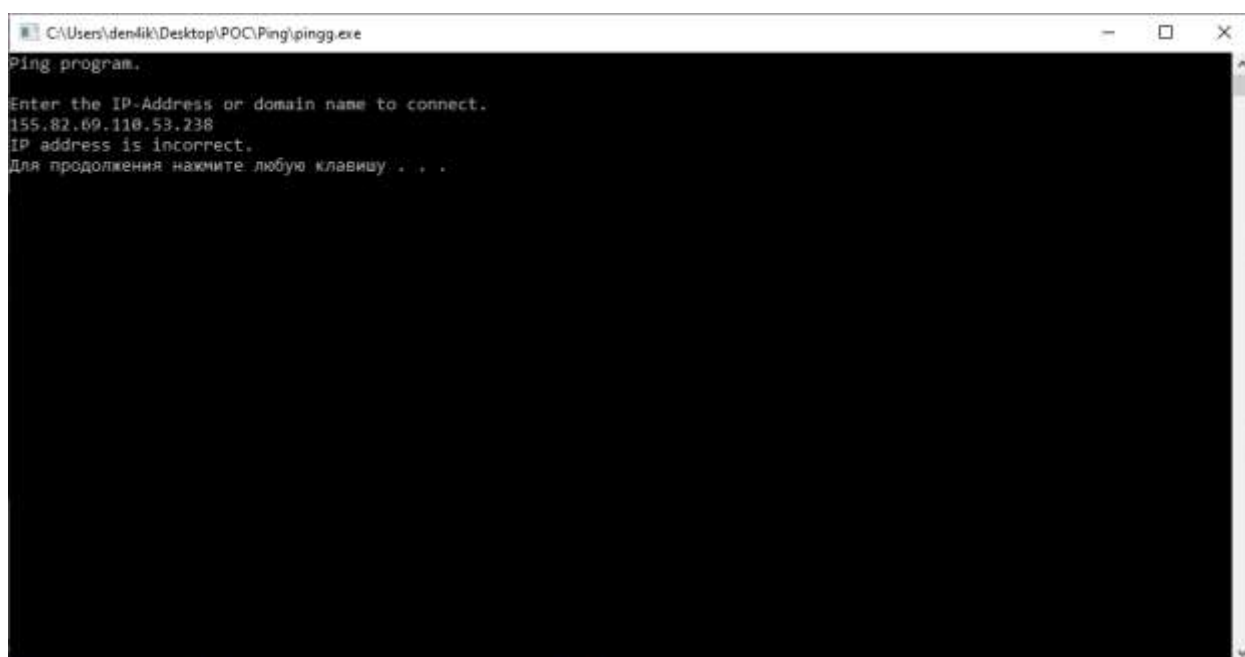


Рис. 22 – Обработка ошибок (некорректный IP-адрес).

На рисунках 23 и 24 представлены скриншоты лог-файла утилиты.

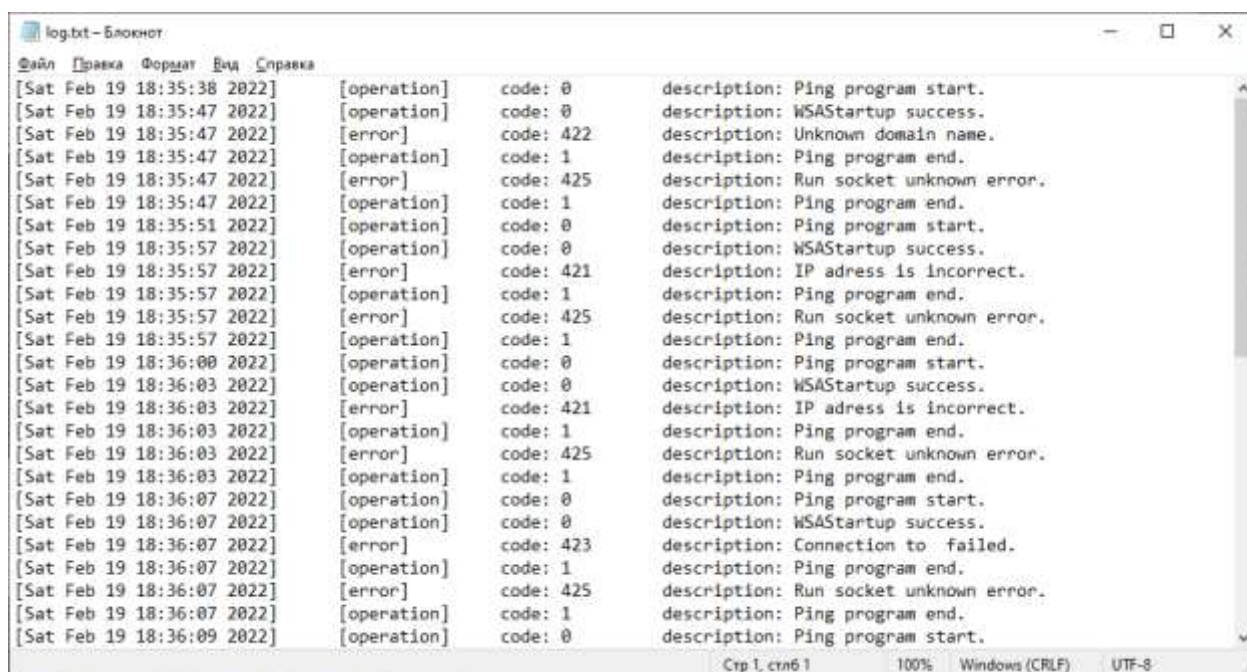


Рис. 23 – Лог-файл.

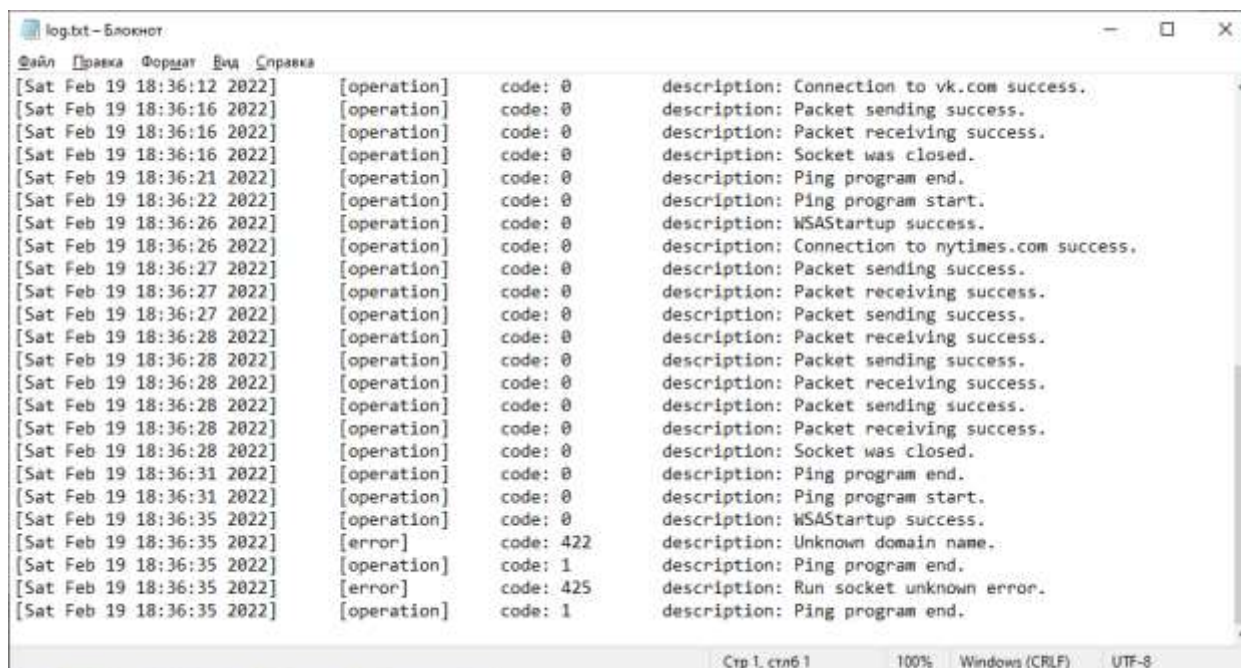


Рис. 24 – Лог-файл.

ЗАКЛЮЧЕНИЕ

По итогу разработки, сборки, и тестирования, мы получили утилиту выполняющие основные требования:

1. Утилита проверяет соединение с удаленным сервером, указанного в виде IP-адреса либо же доменного имени;
2. Утилита записывает все основные действия в лог-файл;
3. Утилита корректно обрабатывает любые ошибки, будь то некорректно введенные данные пользователем, или же другая внутренняя ошибка (инициализация сокета, установка соединения, отправка пакета и т.д.).