БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ
Кафедра программного обеспечения информационных технологий

Факультет ИНО
Специальность ПОИТ

Курсовой проект

по дисциплине «Компьютерные системы и сети»

на тему

«Прокси-сервер»

Выполнил студент: Костина Н.В.
Зачетная книжка № 89105909

Минск 2019

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Прокси-сервер является промежуточным звеном между пользователем и целевым сервером. К числу наиболее значимых преимуществ, которые дает использование прокси-серверов, можно отнести анонимизацию доступа к различным ресурсам, кэширование информации, ограничение доступа из локальной сети к внешней, защита локальной сети от внешнего доступа, обход ограничений доступа.

Целью курсового проекта является разработка программного средства, реализующего функции прокси-сервера. Данное программное средство позволит мониторить пользовательскую активность в сети Интернет и блокировать доступ к определенным сайтам.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Анализ литературы

Основным источником, раскрывающим теоретические основы работы прокси-серверов и компьютерных сетей в целом, послужила книга «Компьютерные сети. Принципы, технологии, протоколы» (5-е издание) авторов В. Олифер и Н. Олифер. Глава 28 данной книги описывает основные механизмы работы прокси-серверов и их назначение, а также приводит классификацию прокси-серверов. В части IV «Сети TCP/IP» данной книги подробно рассмотрены функции и схема работы TCP протокола.

Принцип работы протокола прикладного уровня HTTP подробно описан в спецификации RFC 2616 — HTTP/1.1 (Июнь 1999). Особое внимание было уделено разделам, описывающим структуру HTTP-запроса и HTTP-ответа, типы методов запроса и их назначение, а также коды состояния.

Тема туннелирования TCP наиболее емко отражена в статье Ари Луотонена «Tunneling TCP based protocols through Web proxy servers», на которую ссылается спецификация RFC 2616.

Программная реализация курсового проекта потребовала обращение к литературе по программированию на языке C#. В первую очередь речь идет о технической документации и учебных материалах Майкрософт, размещенных на сайте https://docs.microsoft.com. Также более глубокому пониманию основ программирования на языке C# способствовали книги Г. Шилдта «C# 4.0: полное руководство» и Алекса Дэвиса «Асинхронное программирование в C# 5.0».

1.2 Обзор аналогов программных средств

К аналогам ПС, разрабатываемого в рамках данного курсового проекта, можно отнести следующие программные средства:

1. HandyCache

HandyCache — кэширующий прокси-сервер для операционной системы Windows. Основная функция программы — экономия входящего трафика и ускорение загрузки веб-страниц, а также блокировка рекламы. Работает с любыми браузерами и программами, загружающими информацию по протоколу HTTP.

Возможности программы:

- настраиваемый список кэширования;
- чёрный список — фильтрация рекламы и прочего нежелательного контента;
- ограничение доступа к программе заданными ip-адресами, а также авторизация по имени пользователя;
- запрет загрузки файлов, превышающих заданный размер;
- очистка кэша по заданным параметрам;

- кэширование посещённых страниц в оперативной памяти компьютера;
- кэширование DNS;
- работа с FTP;
- SOCKS5-прокси
- интеграция в internet explorer;
- статистика экономии трафика.

Актуальная версия продукта – 098b1 (2 января 2007).

2. Kerio Control

Kerio Control – программный межсетевой экран, разработанный компаниями Kerio Technologies и Tiny Software. Данная программа позиционировалась на рынке в первую очередь как корпоративный брандмауэр. Тем не менее, в ней реализован полноценный прокси-сервер, позволяющий организовать групповую работу сотрудников компании в Интернете.

Основные характеристики продукта:

- прокси сервер с поддержкой SOCKS;
- контроль пропускной полосы канала;
- балансировка нагрузки на каналы;
- мониторинг и протоколирование пользовательской активности в Интернет;
- поддержка платформ Windows, Linux.

Актуальная версия продукта – 9.2.8 (27.11.2018).

3. 3proxy

3proxy — бесплатный кроссплатформенный прокси-сервер. Программа не имеет графического интерфейса, её настройка производится путём написания конфигурационного файла.

Основные возможности:

- поддержка протоколов HTTP, HTTPS, SOCKS, POP3, SMTP;
- переадресация TCP и UDP трафика;
- объединение прокси серверов в цепочку и функции обратного соединения;
- ограничение пропускной способности канала и трафика;
- контроль доступа (ACL);
- ведение журналов;
- поддержка IPv6;

Актуальная версия: 0.8.12 (18 апреля 2018).

1.3 Постановка задачи

Задачей курсового проекта является разработка и создание программы, реализующей функции прокси-сервера.

Решение данной задачи можно разбить на 5 этапов:

1. Разработка спецификации требований к ПС.
2. Разработка структурной схемы ПС.
3. Разработка схемы-алгоритма
4. Разработка программы на языке программирования C#.
5. Тестирование и отладка программы.

2 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

2.1 Разработка спецификации требований к ПС

2.1.1 Общее описание

Proxy Server – программное средство, реализующее работу прокси-сервера, который поддерживает протоколы HTTP, HTTPS, туннелирование TCP.

Основные особенности продукта:

1. Возможность мониторинга пользовательской активности в Интернет.
2. Возможность ограничения доступа из локальной сети к внешней.

Proxy Server предназначен для использования в организациях и учреждениях, где требуется контролировать деятельность пользователей в сети Интернет и при необходимости блокировать доступ к определенным веб-сайтам.

2.1.2 Классы и характеристики пользователей

Администратор	К данному классу могут быть отнесены руководители компаний, офис менеджеры. Администратор имеет возможность запускать сервер и завершать его работу, а также создавать и редактировать список заблокированных сайтов.
Пользователь	После запуска прокси сервера администратором, пользователь получает доступ к сети Интернет.

2.1.3 Операционная среда

Программа «Proxy Server» работает со следующими операционными системами:

- Windows 7
- Windows 8
- Windows 10.

2.1.4 Функции системы

1. Администратор должен иметь возможность указать порт, к которому будет привязан сервер.
2. Администратор должен иметь возможность запускать и останавливать работу сервера.
3. Администратор должен иметь возможность просматривать статус состояния сервера.

4. Администратор должен иметь возможность просматривать информацию о подключениях к прокси серверу:
    a. Администратор должен иметь возможность просматривать IP-адрес клиента.
    b. Администратор должен иметь возможность просматривать IP-адрес удаленного сервера, к которому подключается клиент.
    c. Администратор должен иметь возможность просматривать время подключения клиента к прокси серверу.
5. Администратор должен иметь возможность создавать и редактировать список заблокированных сайтов.
6. Пользователь должен иметь доступ в сеть Интернет.

2.2 Разработка структурной схемы ПС

На рисунке 1 изображена структурная схема программного средства.

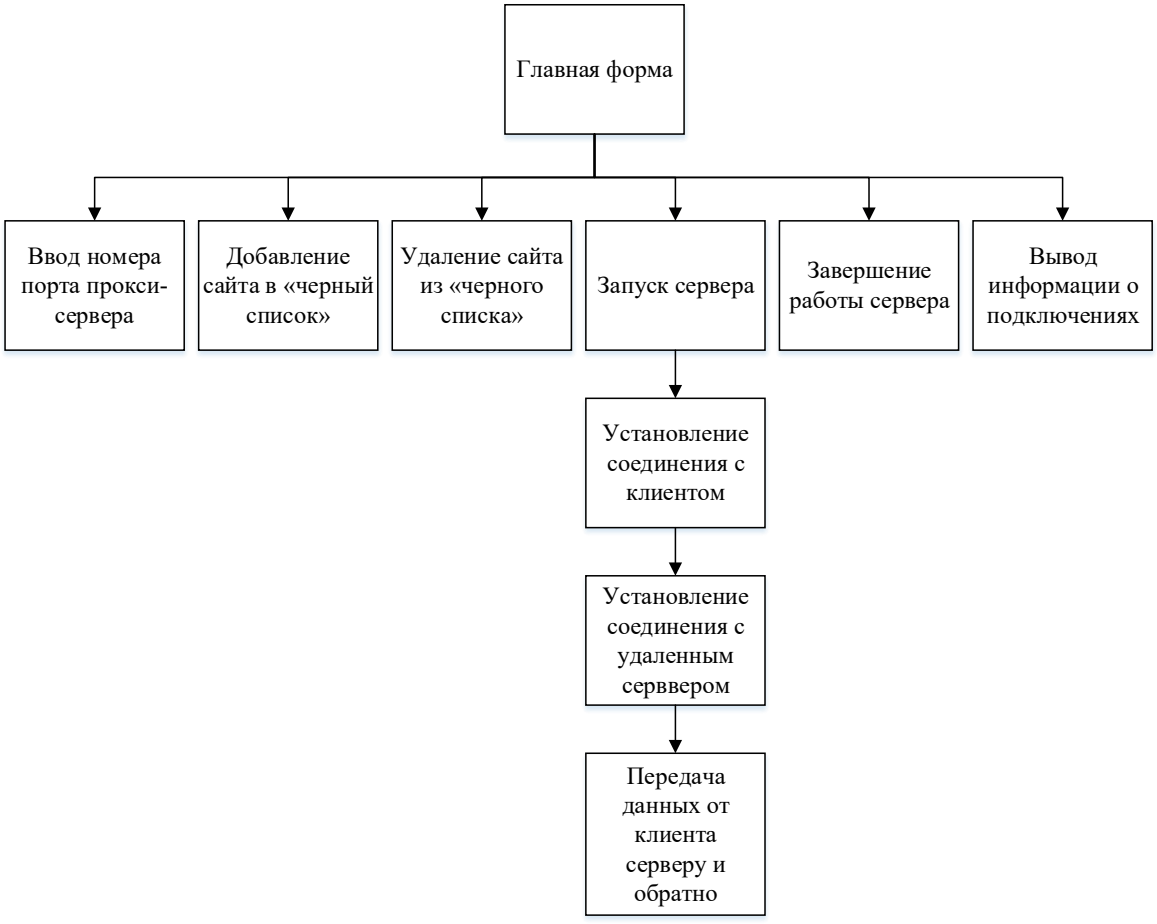


Рисунок 1

Как видно из рисунка 1 интерфейс программы позволяет вызывать следующие функции программы:

- ввод номера порта, к которому будет привязан прокси-сервер;

- добавление адреса в черный список;
- удаление адреса из черного списка;
- запуск сервера;
- остановка сервера;
- вывод информации об активности пользователей в сети.

2.3 Разработка схемы-алгоритма ПС

В плане функционирования программное средство должно повторять принцип работы прокси-сервера: программа должна принимать запросы клиента и отправлять их удаленному серверу, затем получать ответ от удаленного сервера и перенаправлять его клиенту. Таким образом, прокси-сервер должен действовать и как сервер, и как клиент.

В начале работы необходимо перевести прокси-сервер в состояние ожидания запросов на подключение на определенный порт.

После принятия запроса на подключение между клиентом и сервером должно установиться соединение. Далее прокси-сервер должен принять HTTP-запрос клиента. Для дальнейшей работы необходимо проанализировать HTTP-запрос, и выделить из него следующую информацию:

- метод запроса;
- имя сервера;
- номер порта сервера;
- версию HTTP протокола.

Запрос с методом CONNECT необходимо обработать особым образом. Клиент посылает запрос с данным методом для организации туннелирования TCP. Получив данный запрос, прокси-сервер должен послать клиенту утвердительный ответ, означающий, что прокси-сервер поддерживает безопасное соединение. После этого прокси-сервер может открыть безопасное соединение с указанными в запросе хостом и начать прозрачно передавать данные между сокетами со стороны клиента и со стороны сервера.

При поступлении запросов с методами, отличными от метода CONNECT, прокси-сервер должен сразу организовать передачу данных в обе стороны.

Имя и номер порта сервера извлекаются из первого запроса клиента.

Однако передача данных от клиента к удаленному серверу не должна состояться, если имя удаленного сервера находится в «черном списке», поэтому перед отправкой HTTP-запроса необходимо проверить, внесено ли имя удаленного сервера в список заблокированных сайтов. Если «черный список» содержит данный адрес, то прокси-серверу необходимо послать клиенту HTTP-ответ с кодом ошибки 403 (Forbidden). Иначе, данные должны быть доставлены удаленному серверу.

Цикл передачи данных от клиента серверу и обратно должен длиться до тех пор, пока активна одна из сторон. Как только клиент либо удаленный

сервер посылают запрос на завершение соединения, необходимо, чтобы прокси-сервер закрыл соединения с обеими сторонами.

Схема-алгоритм программного средства приведен в приложении А.

2.4 Обоснование языка и среды программирования.

В качестве программной среды была выбрана Visual Studio 2017, поставляемая вместе с платформой .NET, которая предоставляет необходимый инструментарий для эффективного и быстрого создания приложений с графическим интерфейсом.

Для реализации поставленной задачи был выбран язык программирования C#. C# — простой и современный язык программирования. Поддерживает такие парадигмы программирования как процедурное программирование, объектно-ориентированное программирование, обеспечивает модульность, раздельную компиляцию, обработку исключений, абстракцию данных, объявление типов (классов) объектов. Несмотря на то, что C# является самодостаточным языком программирования, у него имеется особая взаимосвязь со средой выполнения .NET Framework. Наличие такой взаимосвязи объясняется двумя причинами. Во-первых, C# первоначально предназначался для создания кода, который должен выполняться в среде .NET Framework. И во-вторых, используемые в C# библиотеки определены в среде .NET Framework. На практике это означает, что C# и .NET Framework тесно связаны друг с другом, хотя теоретически C# можно отделить от среды .NET Framework.

2.5 Программная реализация ПС

Программа состоит из трех классов.

Класс Program.cs содержит одну статическую функцию Main, которая создает объект основного окна программы.

Класс Form1.cs отображает основное окно программы и содержит обработчики событий, возникающих при работе с формой.

Класс Server.cs содержит функции, реализующие работу прокси-сервера. Ниже описаны функции данного класса.

1. **public async Task Start(int port, List<string> blacklist)**

Данная функция принимает в качестве аргументов номер порта, к которому пользователь желает привязать прокси сервер, и черный список сайтов. Функция создает слушающий сокет и привязывает его к заданному порту. Прокси сервер ожидает подключения на любом из своих адресов. Далее прокси-сервер переводится в режим прослушивания и вызывается функция StartAccepting для начала приема подключений.

2. **public async Task StartAccepting(List<string> blacklist)**

Функция приема запросов на подключение, построена на принципах асинхронной модели на основе задач (TAP). Функция запускает бесконечный

цикл для приема подключений. В теле цикла вызывается функция AcceptClientAsync, которая принимает нового клиента. Как только приходит новый запрос, вызывается функция для обработки нового подключения HandleClient.

3. **public async Task HandleClient(Socket Client, List<string> blacklist)**

Данная функция принимает в качестве параметров сокет клиента и черный список сайтов. Прокси-сервер принимает первый HTTP-запрос клиента. Далее вызывается функция HandleRequest для выделения из запроса имени сервера, номера порта, метода и версии запроса.

Затем проверяется, внесен ли запрашиваемый сайт в черный список. Если внесен, то клиенту отсылается ответ, содержащий текст 403 ошибки, и соединение с клиентом закрывается.

Если запрашиваемый сайт отсутствует в черном списке, то анализируется метод запроса.

Если поступил запрос с методом CONNECT, то вызывается функция HandleSequreRequest для обработки запроса на туннелирование TCP-соединения. Иначе, вызывается функция SendData для отправки серверу поступившего запроса.

Для передачи данных от сервера к клиенту и обратно организован цикл while. Сперва опрашивается клиентский сокет на доступность данных для чтения. Для этого используется метод Socket.Poll(Int32, SelectMode). Было принято решение установить время ожидания ответа от сокета равное 2 секунды. Однако следует учесть, что доступность данных для чтения не означает, что данные действительно пришли, поэтому необходимо дополнительно проверить количество пришедших данных. В таблице перечислены возможные ситуации, которые могут возникнуть при чтении данных.

№	Возможная ситуация	Программная реализация
1.	а) сокет имеет доступные для чтения данные; И б) количество считанных байт больше 0;	а) Socket.Poll(2000000, SelectMode.SelectRead)==true б) byte[] data = ReceiveData(Socket); data.Length > 0
2.	а) сокет имеет доступные для чтения данные; И б) количество считанных байт равно 0;	а) Socket.Poll(2000000, SelectMode.SelectRead)==true б) byte[] data = ReceiveData(Socket); data.Length == 0
3.	сокет не имеет доступные для чтения данные	Socket.Poll(2000000, SelectMode.SelectRead)==false

Данные ситуации справедливы как для клиентского, так и для серверного сокета.

Таким образом, клиентский и серверный сокеты поочередно опрашиваются. Когда приходят данные от одной из сторон, то они пересылаются другой стороне, и опрос продолжается.

Выход из цикла будет осуществлен, как только будут выполнены следующие три условия:

1) число событий, когда от клиента было получено 0 байт, превысило 10 (NoBytesFromClient > 10);

2) число событий, когда от сервера было получено 0 байт, превысило 10 (NoBytesFromServer > 10);

3) число событий, когда у сервера и у клиента не было доступных байт для чтения, превысило 10 (IterationNumber > 10).

После выхода из цикла соединения с сервером и клиентом закрываются.

4. **public byte[] ReceiveData(Socket s)**

Функция в качестве аргумента принимает сокет, у которого необходимо считать данные, и возвращает массив байтов.

5. **public void SendData(Socket s, byte[] data)**

Функция в качестве аргумента принимает сокет, которому необходимо отправить данные, и массив байтов, представляющий собой сами данные.

6. **public sHTTPRequest HandleRequest(string sHTTPReq)**

Функция принимает HTTP-запрос типа string и возвращает структуру sHTTPRequest:

```
public struct sHTTPRequest              // структура HTTP запроса
    {
      public string method;              // метод
      public string version;             // версия
      public string host;                // имя сервера
      public int port;                   // порт сервера
    }
```

Для парсинга HTTP-запроса был выбран формальный язык поиска – регулярные выражения. Данный выбор обусловлен тем, что HTTP-запросы имеют утвержденную стандартом структуру, поэтому есть возможность произвести поиск по строке, используя регулярное выражение. Таким образом, из запроса выделяется метод, имя удаленного узла, порт и версия HTTP-протокола. Если порт в запросе не указан, то используется стандартный 80-ый порт.

7. **public IPEndPoint GetServerEndPoint(sHTTPRequest argHTTPRequest)**

Функция в качестве аргумента принимает структуру sHTTPRequest и возвращает конечную точку удаленного сервера в виде IP-адреса и номера порта.

8. **public void HandleSequreRequest(Socket Client, String version)**

Функция в качестве аргумента принимает клиентский сокет и версию HTTP-протокола. Функция формирует стандартный утвердительный ответ, сообщающий о том, что прокси-сервер поддерживает метод CONNECT, и посылает его клиенту при помощи функции SendData(Socket s, byte[] data).

9. public async Task Stop()

Данная функция завершает работу сервера.

3 ТЕСТИРОВАНИЕ

Результаты тестирования программного средства приведены в таблице 1.

Таблица 1 – Результаты тестирования

№	Тест	Ожидаемый результат	Фактический результат	Вывод
1.	Привязка прокси-сервера к занятому порту.	Вывод сообщения об ошибке.	Вывод сообщения об ошибке.	Программа работает правильно.
2.	1. Добавление сайта в черный список. 2. Запуск сервера.	Сайт недоступен пользователю.	Сайт недоступен пользователю.	Программа работает правильно.
3.	Подключение нескольких клиентов к прокси-серверу.	Прокси-сервер обслуживает несколько клиентов одновременно.	Прокси-сервер обслуживает несколько клиентов одновременно.	Программа работает правильно.
4.	Ввод буквенных символов в поле для ввода номера порта.	Вывод сообщения о некорректном вводе.	Вывод сообщения о некорректном вводе.	Программа работает правильно.

4 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

4.1 Руководство администратора

Для запуска программы необходимо запустить файл ProxyServer.exe. При запуске программа отображает окно, представленное на рисунке 2.

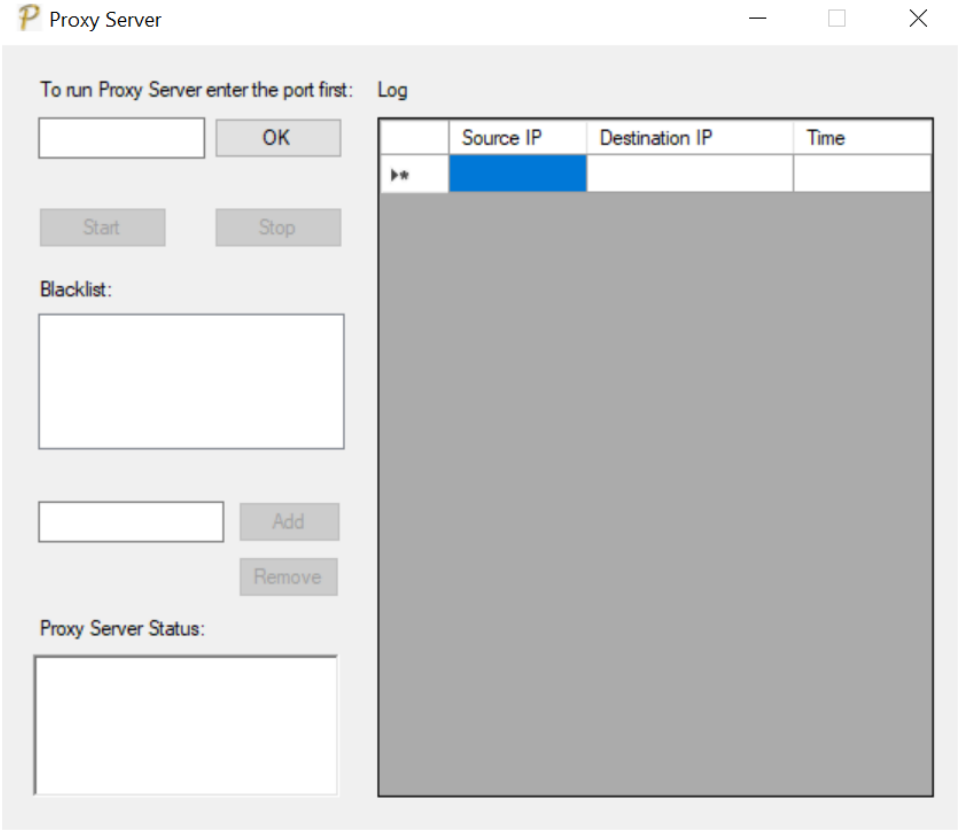


Рисунок 2

Перед запуском сервера необходимо сперва вписать в текстовое поле номер порта и нажать на кнопку OK (см. Рисунок 3).

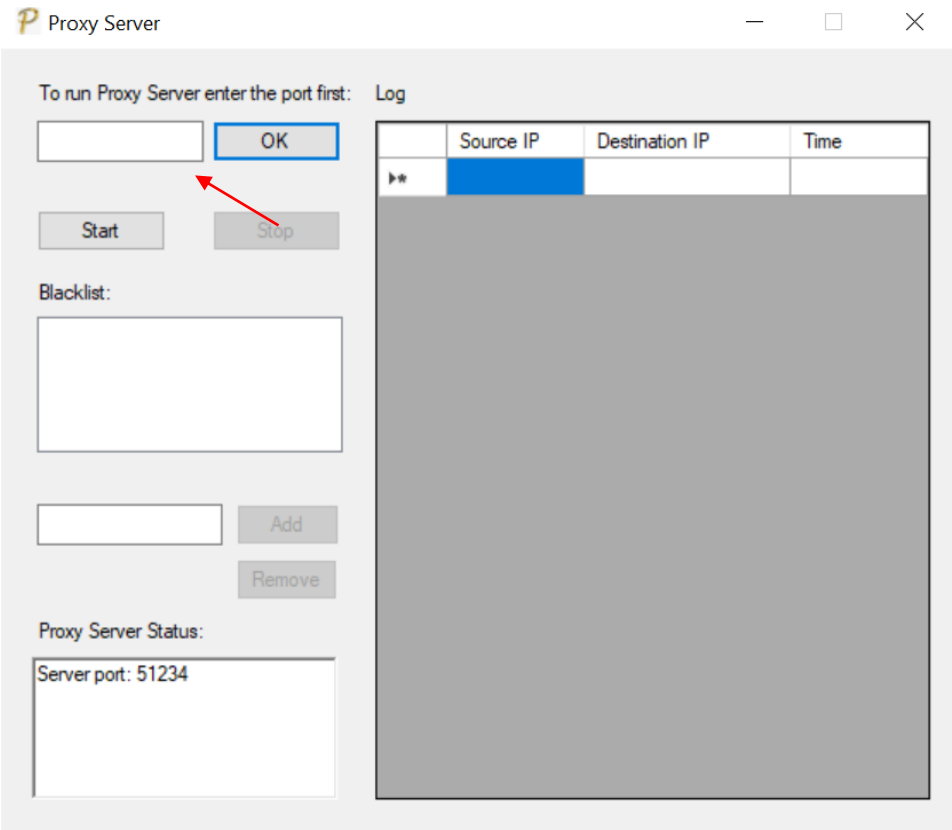


Рисунок 3

Далее при необходимости можно добавить сайт в черный список. Для этого необходимо в текстовое поле вписать сайт и нажать на кнопку Add (см. Рисунок 4).

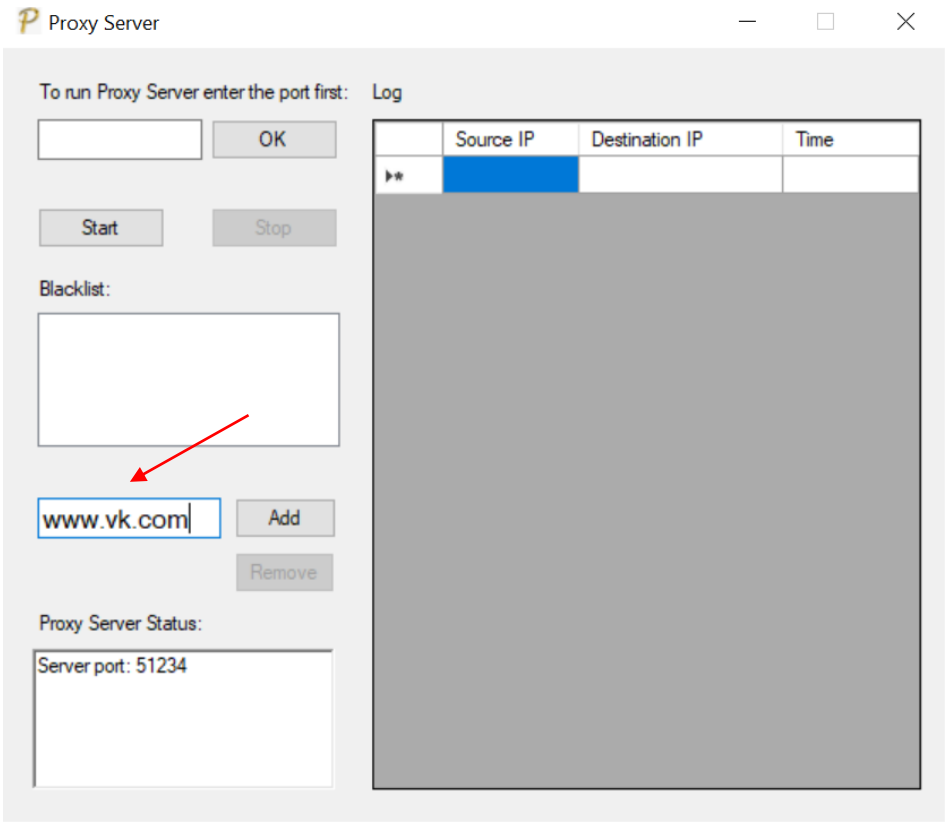


Рисунок 4

Для удаления сайта из списка необходимо его выделить и нажать на кнопку Remove (см. Рисунок 5).

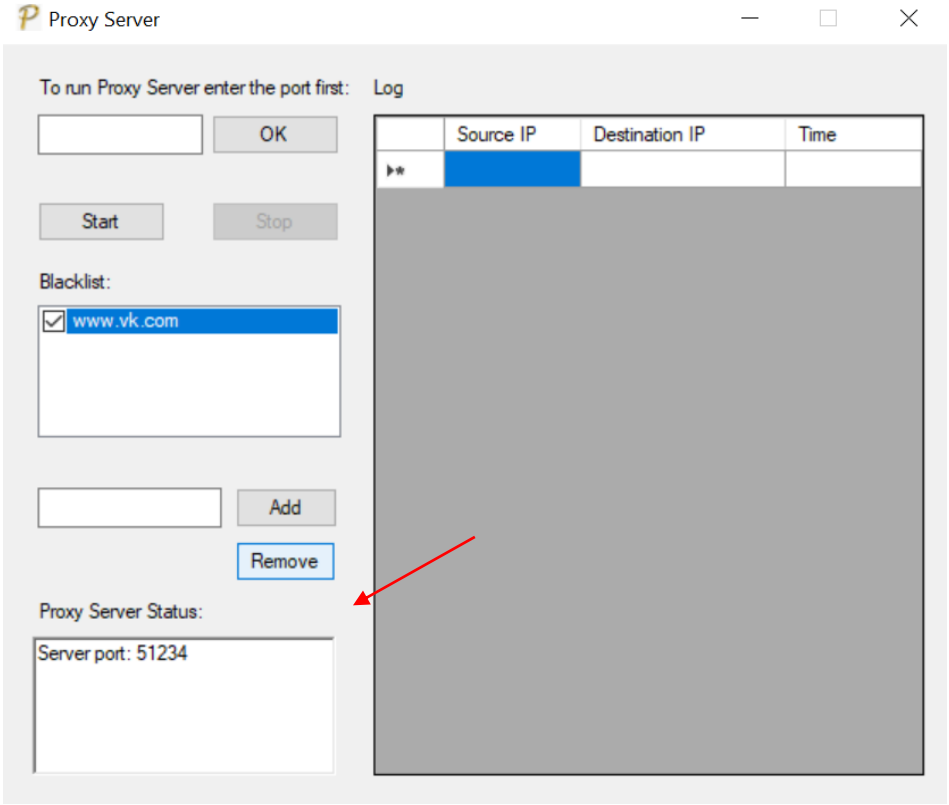


Рисунок 5

Для запуска сервера необходимо нажать на кнопку Start.

Чтобы завершить работу сервера, необходимо нажать на кнопку Stop.

Информация о подключениях новых клиентов отображается в виде таблицы (см. Рисунок 6).

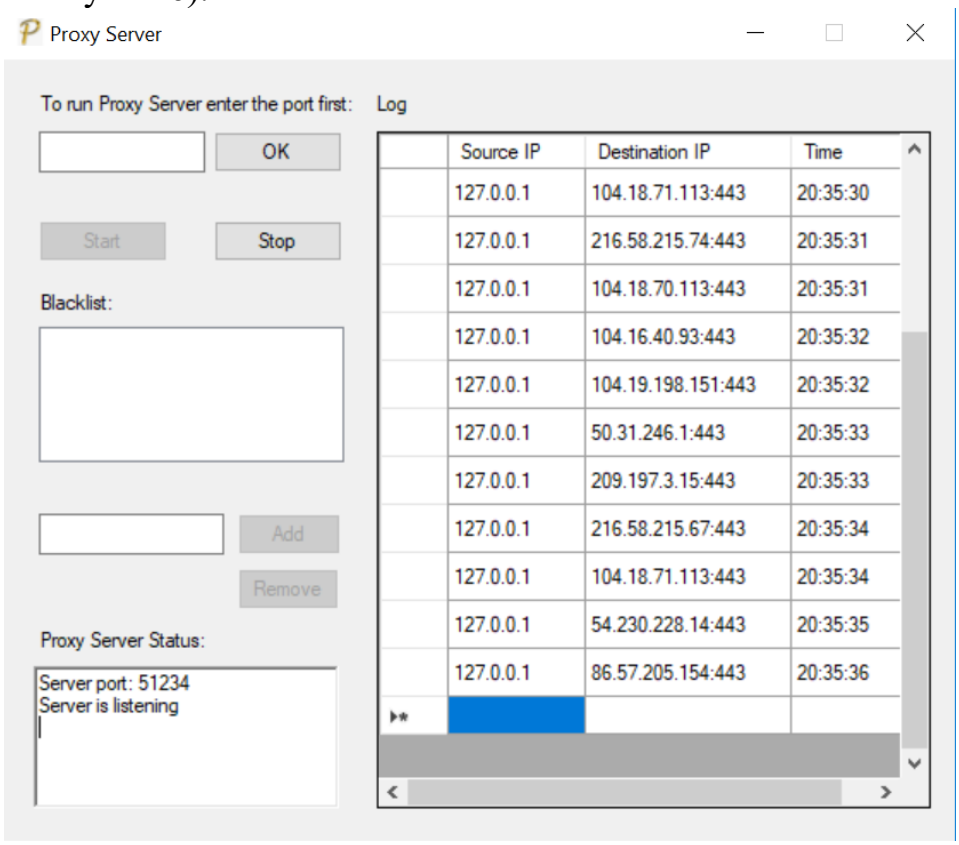


Рисунок 6

В текстовом поле выводится информация о состоянии сервера (см. Рисунок 7).

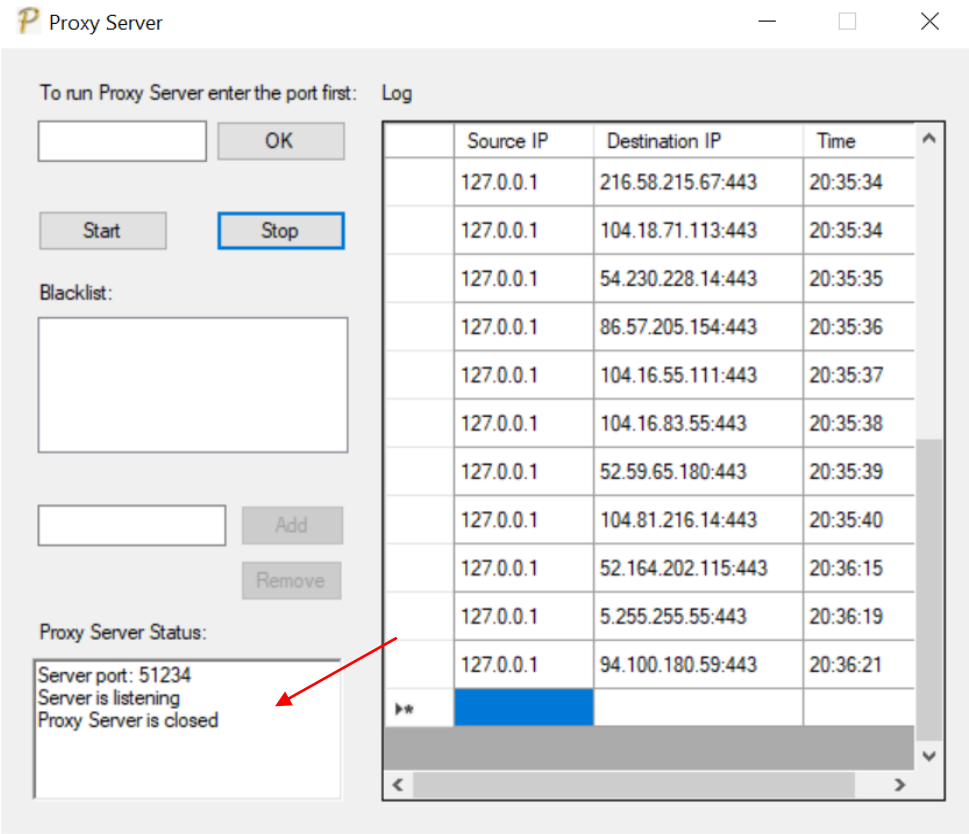


Рисунок 7

4.2 Конфигурирование прокси-сервера на клиенте

Откройте панель управления. Далее Сеть и Интернет → Свойства браузера. Перейдите на вкладку «Подключения» в верхней части окна «Свойства браузера». Нажмите кнопку «Настройки сети» в нижней части окна.

Установите флажок «Использовать прокси-сервер для локальных подключений». Ниже введите сетевой адрес и порт прокси-сервера, которые предоставит администратор.

ЗАКЛЮЧЕНИЕ

В результате курсового проектирования была разработана программа, реализующая функции прокси-сервера. Программа имеет ясный и понятный интерфейс, обеспечивающий удобство в работе.

В ходе курсового проектирования были закреплены теоретические и практические навыки по дисциплине «Компьютерные системы и сети».

СПИСОК ЛИТЕРАТУРЫ

1. Дэвис, А. Асинхронное программирование в C# 5.0. / Пер. с англ. Слинкин А. А. – М.: ДМК Пресс, 2013. – 120 с.
2. Олифер, В. Компьютерные сети. Принципы, технологии, протоколы: учебник для вузов. 5-е изд. / В. Олифер, Н. Олифер – Спб.: Питер, 2016. – 992 с.
3. Шилдт, Г. C# 4.0: полное руководство / Ш. Герберт. Пер. с англ. — М.: ООО «И.Д. Вильямс», 2011. — 1056 с.
4. Hypertext Transfer Protocol – HTTP/1.1: RFC 2616. – 1999.
5. Luotonen, A. Tunneling TCP based protocols through Web proxy servers / Ari Luotonen. – 1998.

ПРИЛОЖЕНИЕ А
Листинг программы

Класс Server.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.Text.RegularExpressions;
using System.Threading.Tasks;

namespace ProxyServer
{
    public class Server
    {
        public delegate void AddRecordEventHandler(string srcIP, string dstIP, string
date, string comment);//делегат
        public event AddRecordEventHandler AddNewRecord; // событие: добавление записи о
новом клиенте
        public delegate void AddStatusEventHandler(string s);// делегат
        public event AddStatusEventHandler AddStatus; // событие: вывод статуса состояния
сервера
        public Socket Listener;                     // слушающий сокет
        public struct sHTTPRequest                  // структура HTTP запроса
        {
            public string method;                   // метод
            public string version;                  // версия
            public string host;                     // имя сервера
            public int port;                        // порт сервера
        }

        // Запуск сервера
        public async Task Start(int port, List<string> blacklist)
        {
            try
            {
                await Task.Run(() =>
                {
                    Listener = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp);//создание слушающего сокета сервера
                    Listener.Bind(new IPEndPoint(IPAddress.Any, port));//привязка к
локальному адресу
                    Listener.Listen(5); //перевод в режим прослушивания
                    AddStatus("Server is listening\r\n");
                    StartAccepting(blacklist);   //вызов асинхронного метода приема
подключений

                });
            }
            catch (Exception e)
            {
                AddStatus("Error! Server is closed" + e.Message);
                Listener.Close();
            }
        } // Завершение работы сервера
        public async Task Stop()
        {
            await Task.Run(() =>
            {
```

```
                Listener.Close();       // закрытие сокета
            });
        }

        public async Task StartAccepting(List<string> blacklist)
        {
            try
            {
                while (true)         // пока сервер запущен
                {
                    Socket Client = await AcceptClientAsync(Listener);
                    HandleClient(Client, blacklist);
                }
            }
            catch (SocketException e) when (e.ErrorCode == 10004)
            {
                AddStatus("Proxy Server is closed\r\n");
            }
            catch (Exception e)
            {
                AddStatus(e.Message);
                Listener.Close();
            }
        }

        // Прием нового подключения
        public async Task<Socket> AcceptClientAsync(Socket Listener)
        {
            Socket Client = Listener.Accept();      // установка соединения с клиентом
            return Client;
        }

        // Обработка нового подключения
        public async Task HandleClient(Socket Client, List<string> blacklist)
        {
            NetworkStream myNetworkStream = new NetworkStream(Client);
            byte[] buffer = new byte[4096];
            bool boolBlacklist = false;
            try
            {
                var FirstRequest = await myNetworkStream.ReadAsync(buffer, 0,
buffer.Length);    // асинхронное чтение запроса клиента
                if (FirstRequest != 0)
                {
                    string sFirstRequest = Encoding.UTF8.GetString(buffer, 0,
FirstRequest);  // преобразование запроса в тип string
                    sHTTPRequest HTTPRequest = HandleRequest(sFirstRequest);
// обработка запроса
                    foreach (string blackAddress in blacklist)
                        if (HTTPRequest.host == blackAddress)
                        {
                            boolBlacklist = true;
                            break;
                        }
                    if (!boolBlacklist)
                    {

                        IPEndPoint ServerEndPoint = GetServerEndPoint(HTTPRequest);
// получение адреса и порта сервера
                        Socket ConectionWithServer = new
Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);   // создание
сокета для соединения с сервером
                        ConectionWithServer.Connect(ServerEndPoint);    // установка
соединения с сервером
```

```
                        if (ConectionWithServer.Connected == true)
                        {
                            string srcIP =
Convert.ToString(((IPEndPoint)Client.RemoteEndPoint).Address);   // адрес клиента
                            string dstIP = Convert.ToString(ServerEndPoint);        //
адрес сервера
                            AddNewRecord(srcIP, dstIP, DateTime.Now.ToString("HH:mm:ss"),
"");  // добавление записи о новом подключении
                            if (HTTPRequest.method == "CONNECT")        // если поступил
запрос с методом CONNECT
                                HandleSequreRequest(Client, HTTPRequest.version);
                            else    // если поступил запрос с методом, отличным от
CONNECT
                                SendData(ConectionWithServer, buffer);
                            int IterationNumber = 0;    // количество проходов цикла
                            int NoBytesFromClient = 0;
                            int NoBytesFromServer = 0;
                            while ((IterationNumber < 10) && (NoBytesFromServer < 10) &&
(NoBytesFromClient < 10))
                            {
                                if (Client.Poll(2000000, SelectMode.SelectRead))
                                {
                                    IterationNumber = 0;        // сброс счетчика
итераций цикла
                                    byte[] data = ReceiveData(Client);   // получить
данные от клиента
                                    if (data.Length == 0)      // если было получено 0
байт
                                    {
                                        NoBytesFromClient++;        // инкрементировать
счетчик
                                    }
                                    else
                                    {
                                        NoBytesFromClient = 0;      // сброс счетчика
                                        SendData(ConectionWithServer, data);   //
отправить данные серверу
                                    }
                                }
                                if (ConectionWithServer.Poll(2000000,
SelectMode.SelectRead))
                                {
                                    IterationNumber = 0;        // сброс счетчика
итераций цикла
                                    byte[] data = ReceiveData(ConectionWithServer);   //
получить данные от сервера
                                    if (data.Length == 0)
                                    {
                                        NoBytesFromServer++;
                                    }
                                    else
                                    {
                                        NoBytesFromServer = 0;      // сброс счетчика
                                        SendData(Client, data);     // отправить данные
клиенту
                                    }
                                }
                                IterationNumber++;  // инкрементировать счетчик итераций
цикла
                            }
                            ConectionWithServer.Close();    // закрыть соединение с
сервером
                        }
                    } else // если сайт находится в черном списке
```

```
                    {
                        string sContent = "<html><head><meta http-equiv=\"Content-Type\"
content=\"text/html; charset=utf-8\"></head><body><h2>Ooops!</h2><div>403 -
Forbidden</div></body></html>";
                        byte[] bContent = Encoding.Default.GetBytes(sContent);
                        byte[] header = Encoding.Default.GetBytes(
                            "HTTP/" + HTTPRequest.version + " 403 Forbidden\r\nContent-
Length: " + bContent.Length.ToString() +
                            "Connection: close\r\nContent-Type: text/html\r\n\r\n"
                            );
                        SendData(Client, header);
                        SendData(Client, bContent);
                    }
                }
            }
            catch
            {
            }
            finally
            {
                Client.Close();     // закрыть соединение с клиентом
            }
        }

        // Обработка запроса
        public sHTTPRequest HandleRequest(string sHTTPReq)
        {
            sHTTPRequest HTTPRequest;
            Regex Reg1 = new Regex(@"(?<method>.+)\s+.+HTTP\/(?<version>[\d\.]+)",
RegexOptions.Multiline);
            Match Match1 = Reg1.Match(sHTTPReq); // сопоставление запроса с регулярным
выражением
            HTTPRequest.method = Match1.Groups["method"].Value;     // извлечение метода
из запроса
            HTTPRequest.version = Match1.Groups["version"].Value;   // извлечение версии
HTTP
            Regex Reg2 = new
Regex(@"Host:\s+(?<http>http(\w)?:)?(?<host>[^\r\n:]+)(?<port>:.+)?",
RegexOptions.Multiline);
            Match Match2 = Reg2.Match(sHTTPReq);
            HTTPRequest.host = Match2.Groups["host"].Value;     // извлечение имени
сервера
            string port = Match2.Groups["port"].Value;     // извлечение порта сервера
            if (port.Length > 0)
            {
                port = port.Substring(1);
                HTTPRequest.port = Convert.ToInt32(port);
            }
            else     // если порт сервера не указан в запросе, то используется 80-ый порт
по умолчанию
                HTTPRequest.port = 80;
            return HTTPRequest;
        }

        // Получение данных
        public byte[] ReceiveData(Socket s)
        {
            byte[] bufReceived = new byte[4096];
            int bytesReceived;
            MemoryStream m = new MemoryStream(); // создание потока для хранения
считанных данных
            // пока данные доступны для чтения и пока количество считанных данных > 0
            while ((s.Poll(100000, SelectMode.SelectRead)) && (bytesReceived =
s.Receive(bufReceived)) > 0)
```

```
            {
                m.Write(bufReceived, 0, bytesReceived);
            }
            byte[] Request = m.ToArray();       // преобразование данных в массив байтов
            return Request;
        }
        // Отправка данных
        public void SendData(Socket s, byte[] data)
        {
            s.Send(data, data.Length, SocketFlags.None);
        }

        // Получение конечной точки сервера
        public IPEndPoint GetServerEndPoint(sHTTPRequest argHTTPRequest)
        {
            IPHostEntry host = Dns.GetHostEntry(argHTTPRequest.host);    // получение
массива IP-адресов сервера
            IPAddress IPHost = host.AddressList[0];     // извлечение из массива первого
IP-адреса
            IPEndPoint ServerEndPoint = new IPEndPoint(IPHost, argHTTPRequest.port);
// формирование конечной точки сервера
            return ServerEndPoint;
        }

        // Обработка запроса на туннелирование TCP-соединения
        public void HandleSequreRequest(Socket Client, String version)
        {
            string Response = "HTTP/" + version + " 200 Connection established\r\nProxy -
agent: Proxy Server 1.0\r\n\r\n"; // ответ прокси сервера
            byte[] bufferResponse = Encoding.Default.GetBytes(Response);
            SendData(Client, bufferResponse);   // отправка ответа клиенту
        }
    }
}
```

Класс Form1.cs

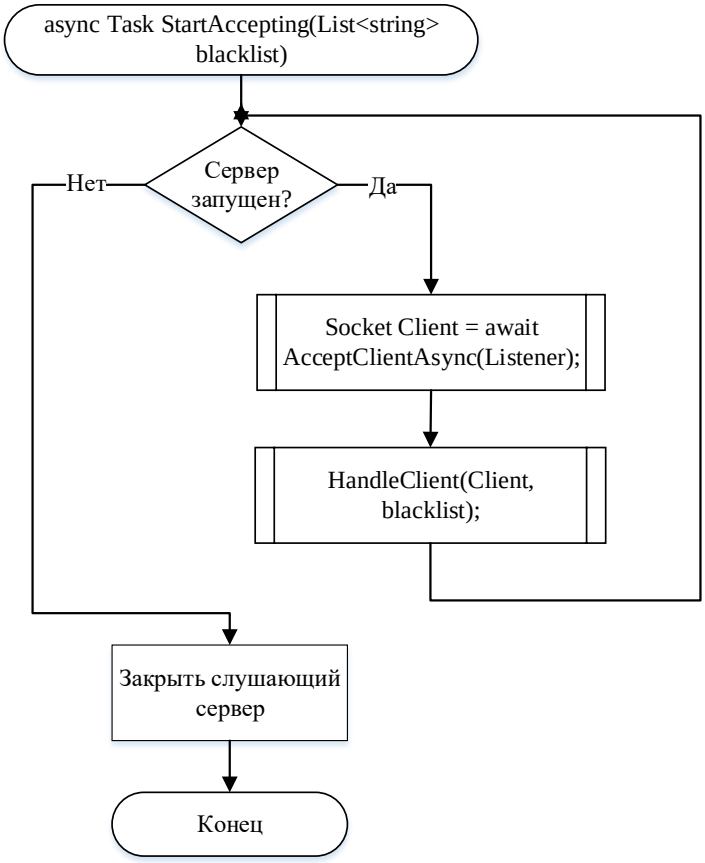
```
using System;
using System.Collections.Generic;
using System.Windows.Forms;

namespace ProxyServer
{
    public partial class Form1 : Form
    {
        Server Server = new Server();       //создание экземпляра класса Server
        int port;
        List<string> blacklist = new List<string>();

        public Form1()
        {
            InitializeComponent();
            Server.AddNewRecord += new
Server.AddRecordEventHandler(dgv_AddNewRecord);//подписка на событие добавления строки в
таблицу
            Server.AddStatus += new
Server.AddStatusEventHandler(richTextBox_Add);//подписка на событие вывода статуса сервер
            FormBorderStyle = FormBorderStyle.FixedDialog;      //запрет на изменение
размеров окна
            btnStop.Enabled = false;        //начальное состояние кнопки Stop
            btnRemove.Enabled = false;
            btnAdd.Enabled = false;
            btnStart.Enabled = false;
```

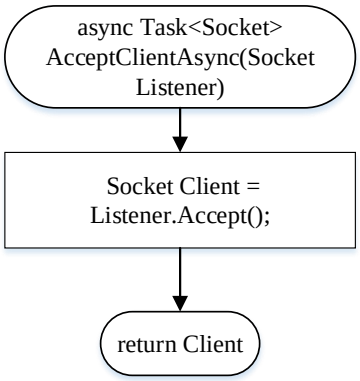
```
        }

        private void dgv_AddNewRecord(string srcIP, string dstIP, string date, string
comment)
        {
            if (dgv.InvokeRequired)       //если вызывающий оператор находится в другом
потоке
            {
                dgv.Invoke(new Action(() => dgv.Rows.Add(srcIP, dstIP, date,
comment)));//выполнение делегата в том потоке,в котором был создан контрол
            }
            else
            {
                dgv.Rows.Add(srcIP, dstIP, date, comment);
            }
            dgv.FirstDisplayedScrollingRowIndex = dgv.RowCount - 1; //опустить скролл
вниз
        }

        private async void btnStart_Click(object sender, EventArgs e)
        {
            btnStart.Enabled = false;
            btnStop.Enabled = true;
            btnRemove.Enabled = false;
            btnAdd.Enabled = false;
            await Server.Start(port, blacklist);       //запуск сервера
        }
        private void richTextBox_Add(string s)
        {
            richTextBox1.AppendText(s);   //вывод статуса сервера
        }

        private async void btnStop_Click(object sender, EventArgs e)
        {

            await Server.Stop();        //завершить работу сервера
            btnStart.Enabled = true;
        }

        private void btnOk_Click(object sender, EventArgs e)
        {
            string input = txtbPort.Text;
            try
            {
                port = Convert.ToInt32(txtbPort.Text);
                // MessageBox.Show("Номер порта: "+port.ToString());
                richTextBox1.AppendText("Server port: "+port.ToString()+"\r\n");
                txtbPort.Clear();
                btnStart.Enabled = true;
            }
            catch
            {
                MessageBox.Show("Wrong input! Try again.");
            }

        }

        private void btnAdd_Click(object sender, EventArgs e)
        {
            string blackAddress = txtbBlackAddress.Text;
            txtbBlackAddress.Clear();
            if (blackAddress.Length > 0)
            {
```

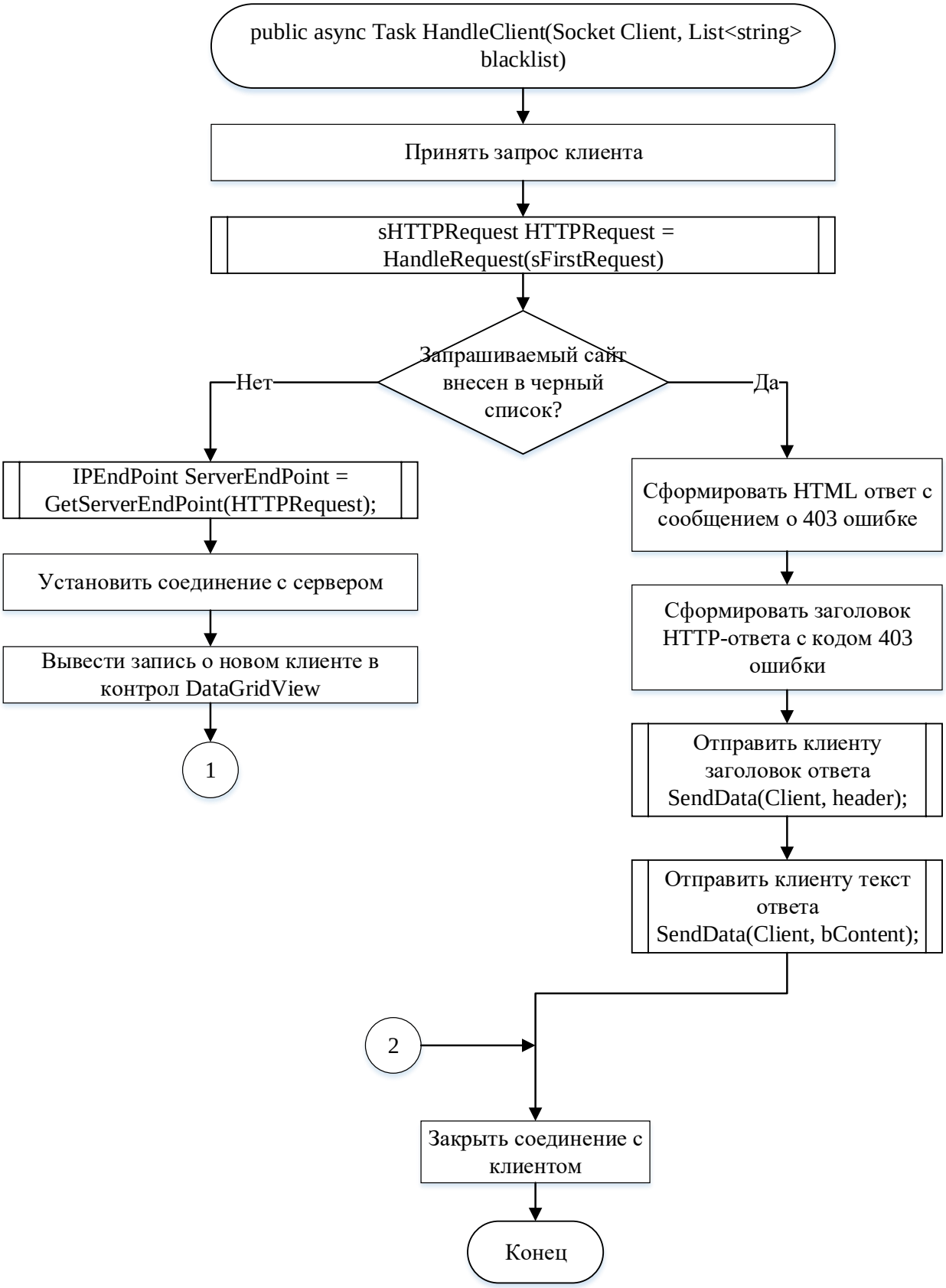
```
                chlb.Items.Add(blackAddress);
                blacklist.Add(blackAddress);
                btnRemove.Enabled = true;
            }
            else
                MessageBox.Show("Enter address.");
        }

        private void btnRemove_Click(object sender, EventArgs e)
        {
            foreach (int indexChecked in chlb.CheckedIndices)
            {
                chlb.Items.RemoveAt(indexChecked);
                blacklist.RemoveAt(indexChecked);
            }
        }

        private void txtbPort_TextChanged(object sender, EventArgs e)
        {
            btnOk.Enabled = true;
        }

        private void txtbBlackAddress_TextChanged(object sender, EventArgs e)
        {
            btnAdd.Enabled = true;
        }

    }
}
```

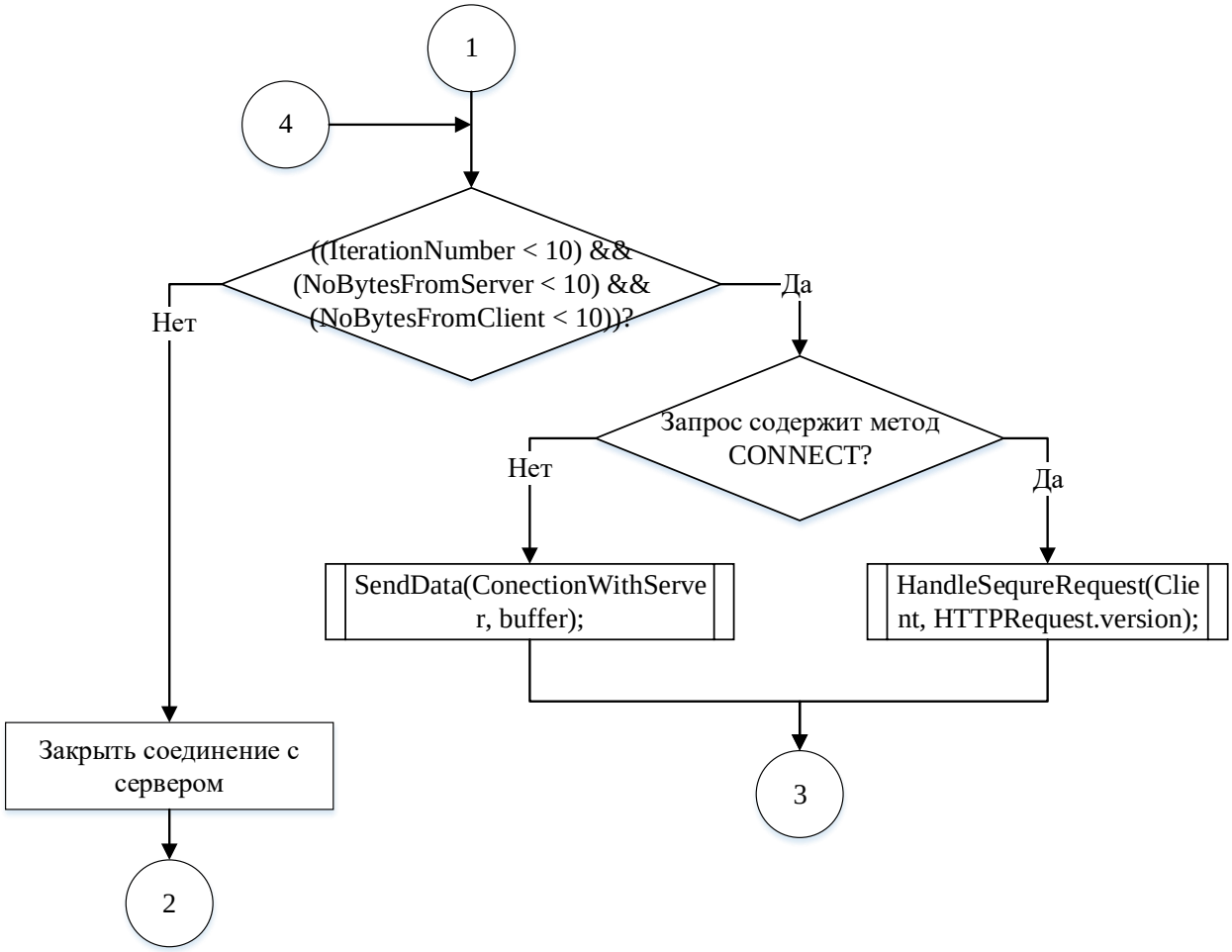
СХЕМА АЛГОРИТМА ПС

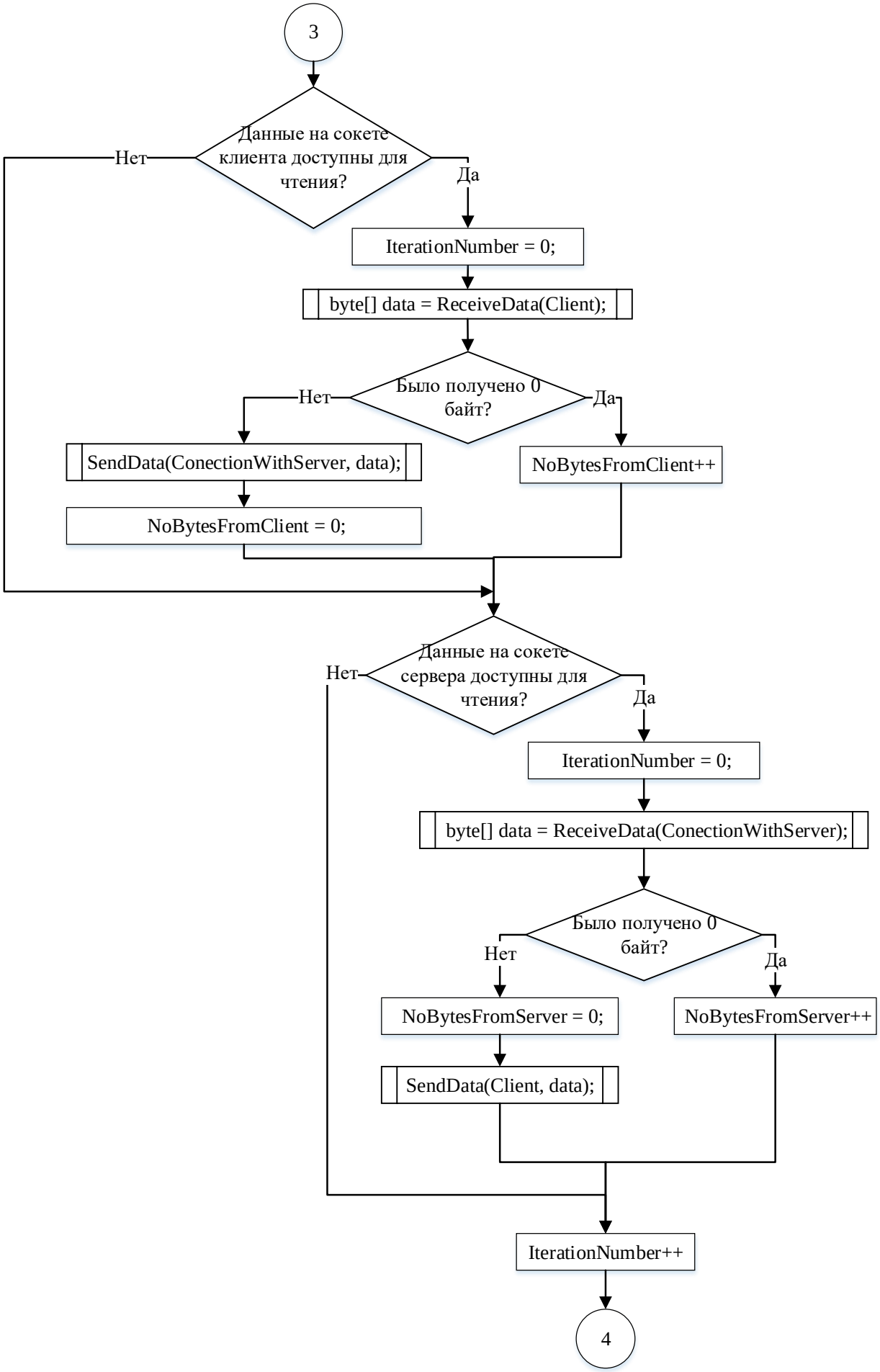
Ниже приведены алгоритмы функций класса Server.cs.

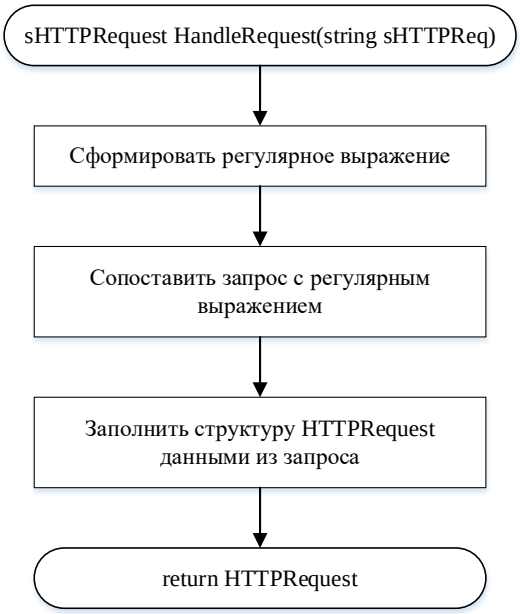


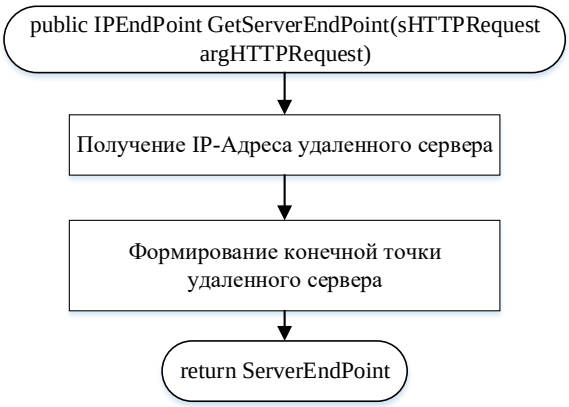


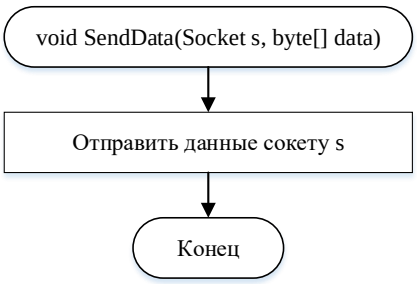

public async Task HandleClient(Socket Client, List<string> blacklist)
Принять запрос клиента
sHTTPRequest HTTPRequest = HandleRequest(sFirstRequest)
Запрашиваемый сайт внесен в черный список?
Нет
Да
IPEndPoint ServerEndPoint = GetServerEndPoint(HTTPRequest);
Установить соединение с сервером
Вывести запись о новом клиенте в контрол DataGridView
1
Сформировать HTML ответ с сообщением о 403 ошибке
Сформировать заголовок HTTP-ответа с кодом 403 ошибки
Отправить клиенту заголовок ответа SendData(Client, header);
Отправить клиенту текст ответа SendData(Client, bContent);
2
Закрыть соединение с клиентом
Конец

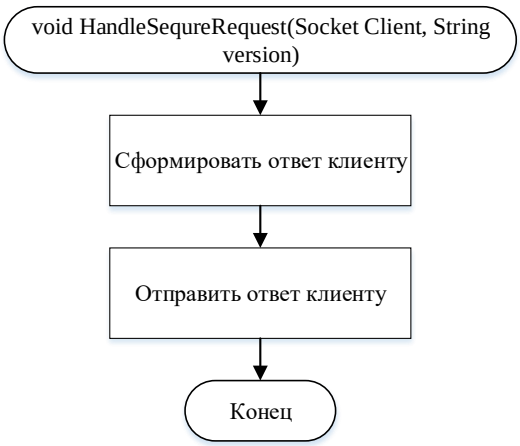

1
4
((IterationNumber < 10) &&
(NoBytesFromServer < 10) &&
(NoBytesFromClient < 10))?
Нет
Да
Запрос содержит метод
CONNECT?
Нет
Да
SendData(ConectionWithServe
r, buffer);
HandleSequreRequest(Clie
nt, HTTPRequest.version);
Закрыть соединение с
сервером
3
2

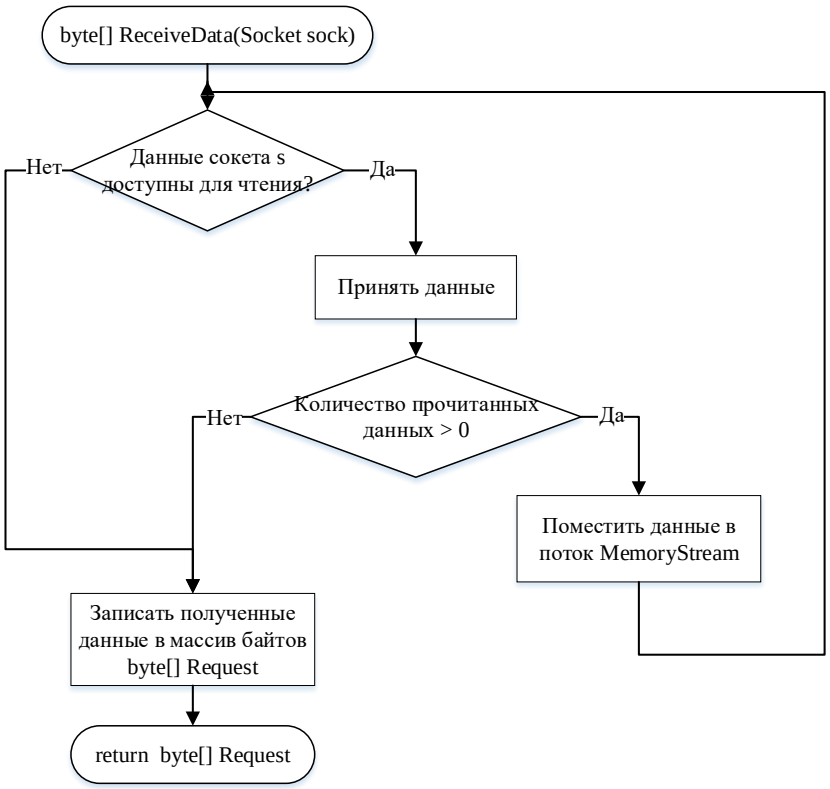

3
Данные на сокете клиента доступны для чтения?
Нет
Да
IterationNumber = 0;
byte[] data = ReceiveData(Client);
Было получено 0 байт?
Нет
Да
SendData(ConectionWithServer, data);
NoBytesFromClient = 0;
NoBytesFromClient++
Данные на сокете сервера доступны для чтения?
Нет
Да
IterationNumber = 0;
byte[] data = ReceiveData(ConectionWithServer);
Было получено 0 байт?
Нет
Да
NoBytesFromServer = 0;
SendData(Client, data);
NoBytesFromServer++
IterationNumber++
4


sHTTPRequest HandleRequest(string sHTTPReq)
Сформировать регулярное выражение
Сопоставить запрос с регулярным выражением
Заполнить структуру HTTPRequest данными из запроса
return HTTPRequest


public IPEndPoint GetServerEndPoint(sHTTPRequest argHTTPRequest)
Получение IP-Адреса удаленного сервера
Формирование конечной точки удаленного сервера
return ServerEndPoint


void SendData(Socket s, byte[] data)
Отправить данные сокету s
Конец


void HandleSequreRequest(Socket Client, String version)
Сформировать ответ клиенту
Отправить ответ клиенту
Конец


byte[] ReceiveData(Socket sock)
Данные сокета s доступны для чтения?
Нет
Да
Принять данные
Количество прочитанных данных > 0
Нет
Да
Поместить данные в поток MemoryStream
Записать полученные данные в массив байтов byte[] Request
return byte[] Request