Содержание

[ВВЕДЕНИЕ 5](#_Toc6822515)

[1 Обзор состояния вопроса 6](#_Toc6822516)

[1.1 Описание утилиты ipconfig 6](#_Toc6822517)

[1.2 Характеристика протокола TCP 7](#_Toc6822518)

[2 Постановка задачи 9](#_Toc6822519)

[3 Проектирование программного обеспечения 10](#_Toc6822520)

[3.1 Архитектура ПО и протокол взаимодействия клиента и сервера 10](#_Toc6822521)

[3.2 Алгоритмы работы модулей 11](#_Toc6822522)

[3.2.1 Проект Server 12](#_Toc6822523)

[3.2.2 Проект Client 13](#_Toc6822524)

[4 Программная реализация 14](#_Toc6822525)

[4.1 Физическая структура и разработанные классы сервера 14](#_Toc6822526)

[4.2 Физическая структура и разработанные классы клиента 16](#_Toc6822527)

[4.3 Библиотечные классы и методы 17](#_Toc6822528)

[5 Руководство пользователя 19](#_Toc6822529)

[5.1 Руководство пользователя сервера 19](#_Toc6822530)

[5.2 Руководство пользователя клиента 19](#_Toc6822531)

[ЗАКЛЮЧЕНИЕ 23](#_Toc6822532)

[СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ 24](#_Toc6822533)

Графическая часть  [25](#_Toc6822534)

[Приложение А Программный код сервера 28](#_Toc6822535)

[Приложение В Программный код клиента 35](#_Toc6822536)

# ВВЕДЕНИЕ

Вхождение Беларуси в мировое информационное пространство влечет за собой широчайшее использование новейших информационных технологий, и в первую очередь, компьютерных сетей. При этом резко возрастают и качественно видоизменяются возможности пользователя как в деле оказания услуг своим клиентам, так и при решении собственных организационно-экономических задач. Уместно отметить, что современные компьютерные сети являются системой, возможности и характеристики которой в целом существенно превышают соответствующие показатели простой суммы составляющих элементов сети персональных компьютеров при отсутствии взаимодействия между ними. Достоинства компьютерных сетей обусловили их широкое распространение в информационных системах кредитно-финансовой сферы, органов государственного управления и местного самоуправления, предприятий и организаций. Компьютерная сеть - объединение нескольких ЭВМ для совместного решения информационных, вычислительных, учебных и других задач. Компьютерные сети и сетевые технологии обработки информации стали основой для построения современных информационных систем. Компьютер ныне следует рассматривать не как отдельное устройство обработки, а как «окно» в компьютерные сети, средство коммуникаций с сетевыми ресурсами и другими пользователями сетей. За последние годы глобальная сеть Интернет превратилась в явление мирового масштаба. Сеть, которая до недавнего времени использовалась ограниченным кругом ученых, государственных служащих и работников образовательных учреждений в их профессиональной деятельности, стала доступной для больших и малых корпораций и даже для индивидуальных пользователей.

Программный продукт, разрабатываемый под ОС Windows является примером программы, позволяющей клиентам получать информацию о текущих соединениях TCP/IP удаленного узла.

Для реализации этой задачи будет использоваться язык программирования C#, встроенный в интегрированную среду разработки (IDE) Microsoft Visual Studio 2017. Он позволяет использовать API-функции для взаимодействия с ОС MS Windows. В частности, в работе были использованы API-функции системы для взаимодействия с таблицей сетевых соединений, а также для организации графического интерфейса пользователя с целью упрощения данного взаимодействия для обычных пользователей.

# 1 Обзор состояния вопроса

## **Описание утилиты ipconfig**

Утилита командной строки ipconfig.exe присутствует во всех версиях Windows и является одним из наиболее распространенных программных средств сетевой диагностики. Некоторые параметры командной строки не поддерживаются в версиях , предшествующих Windows Vista/Windows 7/8

Команда IPCONFIG используется для отображения текущих настроек протокола TCP/IP и для обновления некоторых параметров, задаваемых при автоматическом конфигурировании сетевых интерфейсов при использовании протокола Dynamic Host Configuration Protocol (DHCP)..

Формат командной строки:

ipconfig [/allcompartments] [/? | /all | /renew [адаптер] | /release [адаптер] | /renew6 [адаптер] | /release6 [адаптер] | /flushdns | /displaydns | /registerdns | /showclassid адаптер | /setclassid адаптер [идентификатор\_класса] ] /showclassid6 адаптер | /setclassid6 адаптер [идентификатор\_класса] ]

Параметры командной строки:

* /? - Вывод справочного сообщения
* /all - Вывод подробных сведений о конфигурации.
* /release - Освобождение адреса IPv4 для указанного адаптера.
* /release6 - Освобождение адреса IPv6 для указанного адаптера.
* /renew - Обновление адреса IPv4 для указанного адаптера.
* /renew6 - Обновление адреса IPv6 для указанного адаптера.
* /flushdns - Очистка кэша сопоставителя DNS.
* /registerdns - Обновление всех DHCP-аренд и перерегистрация DNS-имен
* /displaydns - Отображение содержимого кэша сопоставителя DNS.
* /showclassid - Отображение всех допустимых для этого адаптера идентификаторов классов DHCP.
* /setclassid - Изменение идентификатора класса DHCP.
* /showclassid6 - Отображение всех допустимых для этого адаптера идентификаторов классов DHCP IPv6.
* /setclassid6 - Изменение идентификатора класса DHCP IPv6.

Изменение сетевых настроек с помощью команды IPCONFIG, в основном, применимо к тем сетевым адаптерам, которые настроены на автоматическое конфигурирование с использованием службы динамической настройки основных параметров на сетевом уровне DHCP (Dynamic Host Configuration Protocol) или службы автоматической настройки приватных IP - адресов APIPA (Automatic Private IP Addressing).

Если в параметрах командной строки IPCONFIG используется имя адаптера, содержащее пробелы, то оно должно заключаться в двойные кавычки. Если имя содержит символы русского алфавита, то оно должно быть представлено в DOS-кодировке.

## 1.2 Характеристика протокола TCP

TCP — это транспортный механизм, предоставляющий поток данных, с предварительной установкой соединения, за счёт этого дающий уверенность в достоверности получаемых данных, осуществляет повторный запрос данных в случае потери данных и устраняет дублирование при получении двух копий одного пакета. В отличие от UDP гарантирует целостность передаваемых данных и уведомление отправителя о результатах передачи.

Когда осуществляется передача от компьютера к компьютеру через Интернет, TCP работает на верхнем уровне между двумя конечными системами, например, браузером и веб-сервером. Также TCP осуществляет надежную передачу потока байтов от одной программы на некотором компьютере к другой программе на другом компьютере. Программы для электронной почты и обмена файлами используют TCP.

В отличие от традиционной альтернативы — UDP, который может сразу же начать передачу пакетов, TCP устанавливает соединения, которые должны быть созданы перед передачей данных. TCP соединение можно разделить на 3 стадии:

* + Установка соединения
  + Передача данных
  + Завершение соединения

В связи с чем можно говорить о состояниях TCP сеанса (рисунок 1):

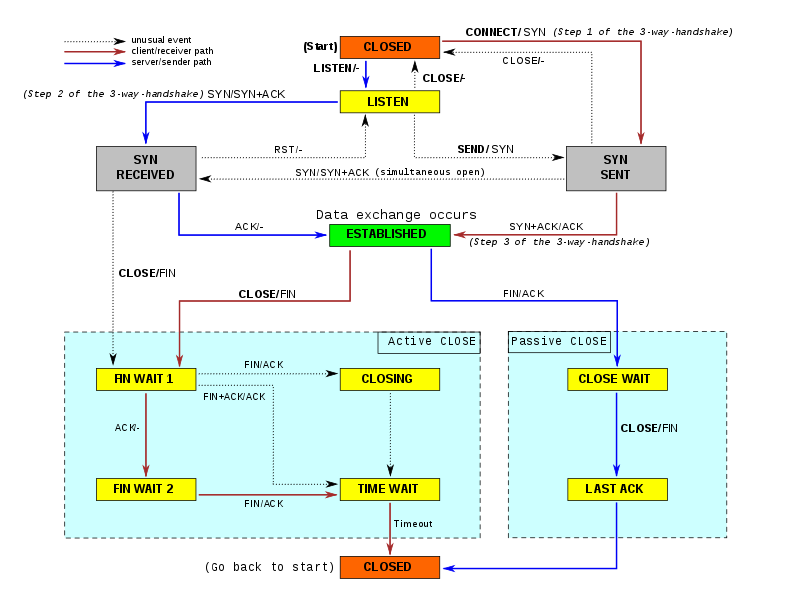


Рисунок 1 – Состояние TCP сеанса

Основные особенности протокола TCP:

* + Устанавливается соединение.
  + Данные передаются сегментами. Модуль TCP нарезает большие блоки данных на пакеты, каждый из которых передается отдельно, а на приемнике наоборот пакеты собираются в исходный блок. Для этого нужен порядковый номер (Sequence Number – SN) пакета.
  + Посылает запрос на следующий пакет, указывая его номер в поле "Номер подтверждения" (AS). Тем самым, подтверждая получение предыдущего пакета.
  + Делает проверку целостности данных, если пакет битый (контрольная сумма не совпала) – посылает повторный запрос.

# 2 Постановка задачи

В данной курсовой работе нужно написать многопоточный клиент-серверное приложение на основе протокола передачи данных TCP.

Для этого были сформированы следующие задачи:

* + Изучить работу протокола TCP;
  + Изучить работу утилиты ipconfig;
  + Написать приложение многопоточного сервера;
  + Написать приложение для работы клиента.

# 3 Проектирование программного обеспечения

## 3.1 Архитектура ПО и протокол взаимодействия клиента и сервера

Технология "клиент-сервер" описывает взаимодействие между двумя компьютерами, согласно которому клиент запрашивает у сервера некоторые услуги, а сервер обслуживает запрос.

Общая схема взаимодействий серверного и клиентского приложений представлена рисунке 2.

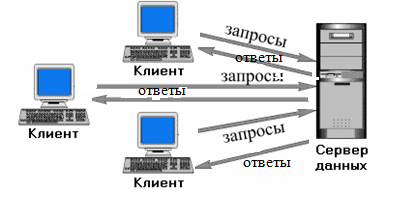


Рисунок 2 – Общая схема взаимодействия клиент-серверного приложения

Данная архитектура имеет рад преимуществ, а также ряд и недостатков. Из преимуществ можно выделить то, что в данной архитектуре отсутствие дублирования кода программы-сервера программами-клиентами, все вычисления выполняются на сервере, то требования к компьютерам, на которых установлен клиент, снижаются и все данные хранятся на сервере, который, как правило, защищён гораздо лучше большинства клиентов. На сервере проще организовать контроль полномочий, чтобы разрешать доступ к данным только клиентам с соответствующими правами доступа.

Сетевой протокол — набор правил и действий (очерёдности действий), позволяющий осуществлять соединение и обмен данными между двумя и более включёнными в сеть устройствами.

Разные протоколы зачастую описывают лишь разные стороны одного типа связи. Названия «протокол» и «стек протоколов» также указывают на программное обеспечение, которым реализуется протокол.

Сетевые протоколы предписывают правила работы компьютерам, которые подключены к сети. Они строятся по многоуровневому принципу. Transmission Control Protocol/Internet Protocol (TCP/IP) - это промышленный стандарт стека протоколов, разработанный для глобальных сетей.

TCP/IP - это набор протоколов, который задает стандарты связи между компьютерами и содержит подробные соглашения о маршрутизации и межсетевом взаимодействии. TCP/IP широко применяется в Internet, поэтому с его помощью могут общаться пользователи из исследовательских институтов, школ, университетов, правительственных учреждений и промышленных предприятий.

TCP/IP обеспечивает связь подключенных к сети компьютеров, обычно называемых хостами. Любую сеть можно подключить к другой сети и организовать связь с ее хостами. Несмотря на то, что существуют различные сетевые технологии, многие из которых основаны на коммутации пакетов и потоковом режиме передачи, набор протокол TCP/IP обладает одним важным преимуществом: он обеспечивает аппаратную независимость.

Так как в протоколах Internet определяется только блок передачи и способ его отправки, TCP/IP не зависит от особенностей сетевого аппаратного обеспечения, позволяя организовать обмен информацией между сетями с различной технологией передачи данных. Система IP-адресов позволяет установить соединение между любыми двумя машинами сети. Кроме того, в TCP/IP также определены стандарты для многих служб связи, предназначенных для конечных пользователей.

TCP/IP обеспечивает средства, позволяющие вашему компьютеру выступать в роли хоста Internet, который может подключиться к сети и установить соединение с любым другим хостом Internet.

## 3.2 Алгоритмы работы модулей

Курсовая работа состоит из следующих проектов:

* Client;
* Server.

На рисунке 3 показано взаимодействие проектов.



Рисунок 3 – Взаимодействие проектов

## 3.2.1 Проект Server

Проект реализует серверное приложение, которое будет обслуживать клиентов. Клиенты подключаются к серверу и отправляют запросы на сервер, а сервер в свою очередь отправляет результат обработки запроса. На рисунке 4 показана архитектура серверного приложения и взаимодействия основных классов.



Рисунок 4 – Архитектура серверного приложения

Основные классы проекта:

TcpServer – класс, осуществляющий обмен данными с клиентом.

ClientObject – класс, который осуществляет обработку запроса от клиента в отдельном потоке

Сonfigurations – класс, который служит для получение данных для отправки клиенту .

## 3.2.2 Проект Client

Проект реализует клиентское приложение, которое будет подключаться к серверу и иметь возможность обмена сообщениями и сервером. После того как клиент подключается к серверу, он может отправлять данные. На рисунке 5 показана архитектура клиентского приложения и взаимодействия основных классов.



Рисунок 5 – Архитектура клиентского приложения

Основные классы проекта:

TCPClient – главное окно программы, организовывает работу клиента и сервера.

# 4 Программная реализация

## 4.1 Физическая структура и разработанные классы сервера

Физическая структура приложений состоит из двух разных проектов, не объединённые в один общий модуль.

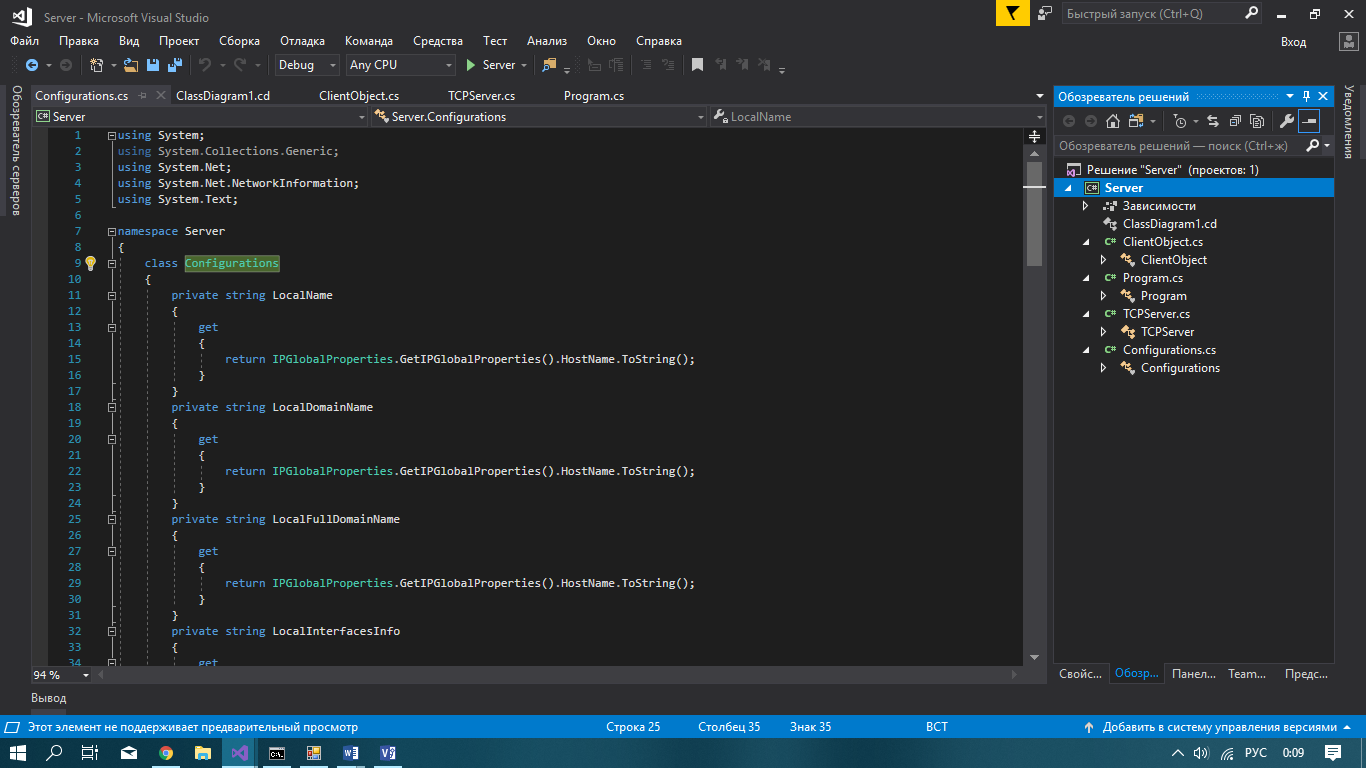


Рисунок 6 – Физическая модель Сервера

Серверное приложение состоит из четырех классов (рисунок 6).

Класс Program создает сервер для прослушивания запросов от клиентов на заданном порте.

|  |
| --- |
| class Program  {  const int port = 8888;  const string adrress = "127.0.0.1";  static void Main(string[] args)  {  TCPServer server = new TCPServer();  server.InfoServer();  server.Create(adrress, port);  server.StartServer();  }  } |

В данном случаи используется константные переменные для порта и адреса сервера. Как видно, что в данном классе создается новый класс TCPServer, который отвечает за прослушивание заданного порта по данную адресу запросов и клиентов.

|  |
| --- |
| public void StartServer()  {  try  {  listener = new TcpListener(IPAddress.Parse(IPaddress), port);  listener.Start();  Console.WriteLine("Ожидание подключений...");  while (true)  {  TcpClient client = listener.AcceptTcpClient();  ClientObject clientObject = new ClientObject(client);  // создаем новый поток для обслуживания нового клиента  Thread clientThread = new Thread(new ThreadStart(clientObject.Process));  clientThread.Start();  }  }  catch (Exception ex)  {  Console.WriteLine(ex.Message);  }  finally  {  if (listener != null)  listener.Stop();  }  } |

В данном классе главное внимание уделяется методу StartServer, которые и запускает данную работу сервера. Когда клиент запрашивает запрос у сервера, то сервер создает отдельный поток для обработки данного запроса, это необходимо для того, чтобы можно было подключаться нескольким клиентам.

|  |
| --- |
| public void Process()  {  NetworkStream stream = null;  try  {  stream = client.GetStream();  byte[] data = new byte[64]; // буфер для получаемых данных  // получаем сообщение  StringBuilder builder = new StringBuilder();  int bytes = 0;  do  {  bytes = stream.Read(data, 0, data.Length);  builder.Append(Encoding.Unicode.GetString(data, 0, bytes));  }  while (stream.DataAvailable);  string message = builder.ToString();  Console.WriteLine("Запрос от клиента {0}", message);  // отправляем обратно сообщение в верхнем регистре  message = сonfigurations.IPConfig(message);  data = Encoding.Unicode.GetBytes(message);  stream.Write(data, 0, data.Length);  }  catch (Exception ex)  {  Console.WriteLine(ex.Message);  }  finally  {  if (stream != null)  stream.Close();  if (client != null)  client.Close();  }  } |

Данный метод работает в отдельном потоке. В данном методе сервер обрабатывает запрос клиента и отправляет результат обратно клиенту.

## 4.2 Физическая структура и разработанные классы клиента

Физическая структура приложений состоит из двух разных проектов, не объединённые в один общий модуль.

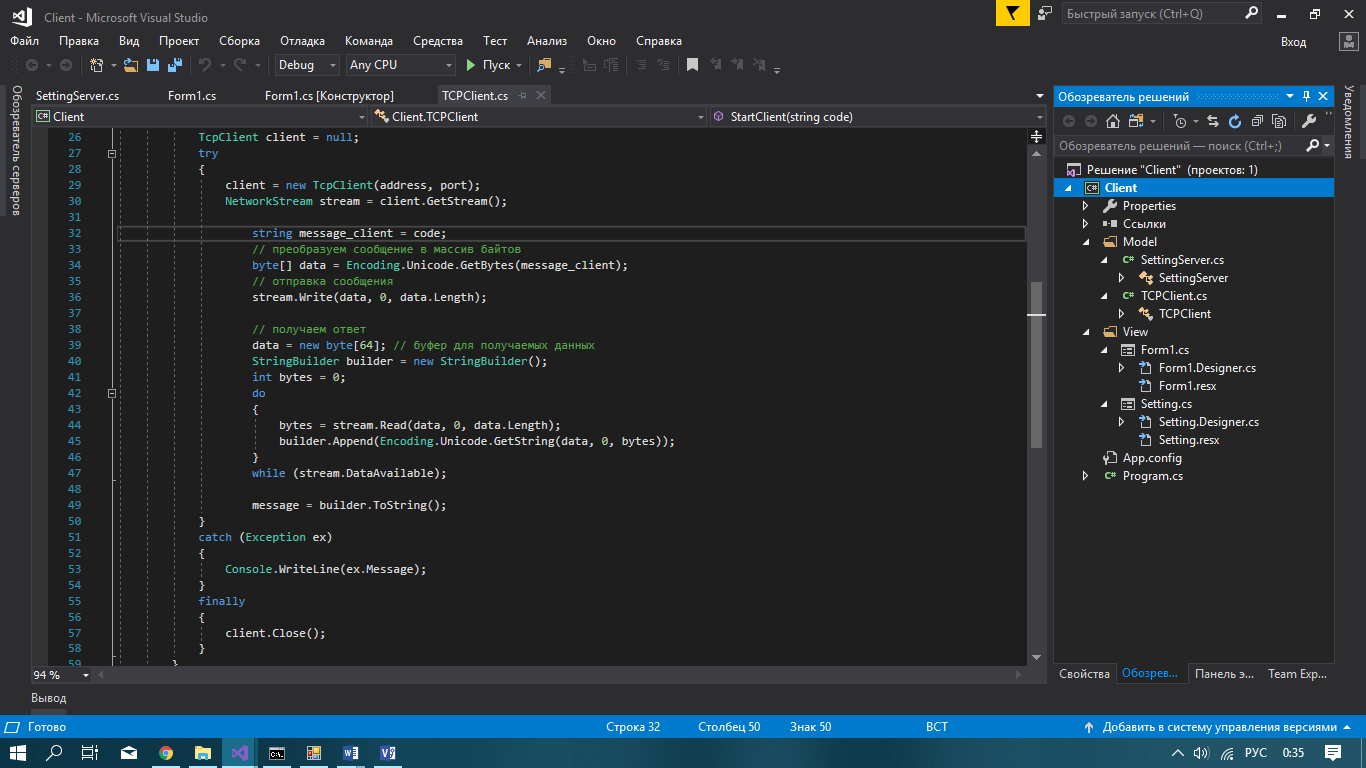


Рисунок 7 – Физическая структура клиента

Данное приложение состоит из двух форм и двух независимых классов, не учитывая классы, создаваемые от форм.

Первая форма Form1 является основной. В данной форме выводится вся информация, которые была запрошена у сервера, а также и отправлена на сервер. На данной форме можно вызвать вторую форму, которая отвечает за настройки данного приложения.

|  |
| --- |
| class TCPClient  {  int port;  string address;    string message = "";  public void CreateClient(string ip\_address, int ip\_port)  {  port = ip\_port;  address = ip\_address;  }    public void StartClient(string code)  {  TcpClient client = null;  try  {  client = new TcpClient(address, port);  NetworkStream stream = client.GetStream();  string message\_client = code;  // преобразуем сообщение в массив байтов  byte[] data = Encoding.Unicode.GetBytes(message\_client);  // отправка сообщения  stream.Write(data, 0, data.Length);  // получаем ответ  data = new byte[64]; // буфер для получаемых данных  StringBuilder builder = new StringBuilder();  int bytes = 0;  do  {  bytes = stream.Read(data, 0, data.Length);  builder.Append(Encoding.Unicode.GetString(data, 0, bytes));  }  while (stream.DataAvailable);  message = builder.ToString();  }  catch (Exception ex)  {  Console.WriteLine(ex.Message);  }  finally  {  client.Close();  }  }  public string GetMessage()  {  return message;  }  } |

В данном методе создается запрос на получение информации от сервера. Для этого формируется заявка на запрос и отправляется данный запрос серверу. Потом клиент переходит в режим ожидания, пока не получит ответ от сервера.

## 4.3 Библиотечные классы и методы

Для реализации курсовой работы используется среда Microsoft Visual Studio 2017. Она имеет пользовательский интерфейс, предоставляет удобные средства для написания программы. Необходимо отметить, что данный программный продукт представляет собой довольно мощный инструмент для разработки программ подобного типа.

В реализации данного проекта были использованы стандартные классы и методы, которые представлены в таблице 1, предоставляемого программной платформой .NET Framework версии 4.5. Последний класс в пространстве имен System.Net.Sockets – это сам класс Socket. Он обеспечивает базовую функциональность приложения сокета.

Таблица 1 – Стандартные классы и методы, используемые в проекте.

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Класс** | **Метод** | **Описание** |
|  | System.Net.Sockets | Bind | Связывает объект [Socket](https://msdn.microsoft.com/ru-ru/library/system.net.sockets.socket(v=vs.110).aspx) с локальной конечной точкой. |
|  | System.Net.Sockets | Listen | Устанавливает объект  [Socket](https://msdn.microsoft.com/ru-ru/library/system.net.sockets.socket(v=vs.110).aspx)  в состояние прослушивания. |
|  | System.Net.Sockets | Accept | Создает новый объект [Socket](https://msdn.microsoft.com/ru-ru/library/system.net.sockets.socket(v=vs.110).aspx) для заново созданного подключения. |
|  | System.Net.Sockets | Shutdown | Блокирует передачу и получение данных для объекта [Socket](https://msdn.microsoft.com/ru-ru/library/system.net.sockets.socket(v=vs.110).aspx). |
|  | System.Net.Sockets. NetworkStream | Close | Закрывает соединение и освобождает все связанные ресурсы. |

Это основные методы, которые необходимы для работы и установки соединения между клиентом и сервером.

# 5 Руководство пользователя

Для запуска приложения необходимо иметь на компьютере ОС Windows версии 7 и выше и .Net Framework версии 4.5 и выше, а также скопировать файлы Client.exe и Server.exe в удобный каталог. Для запуска сервера нужно запустить Server.exe. Для запуска клиента запустить приложение Client.exe.

## Руководство пользователя сервера

После запуска приложения появится главная форма представленная на рисунке 8.

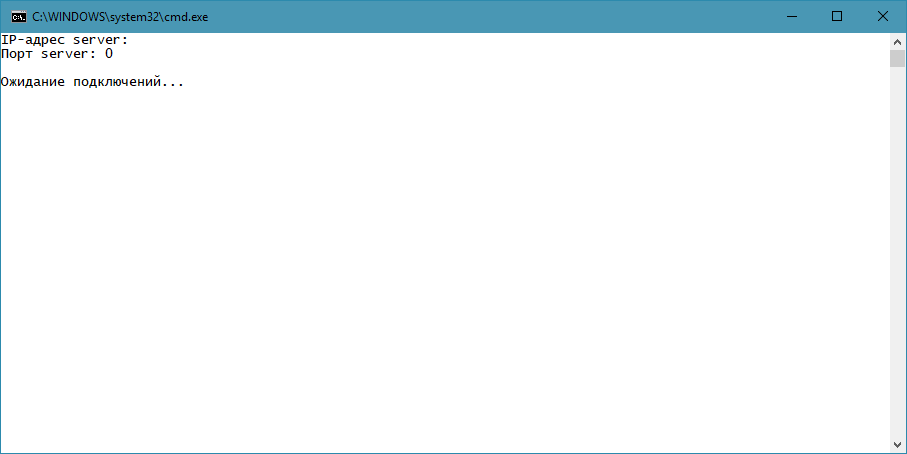


Рисунок 8 – Главное меню сервера

Как только сервер получит данные, то покажется данный запрос от клиента (рисунок 9).

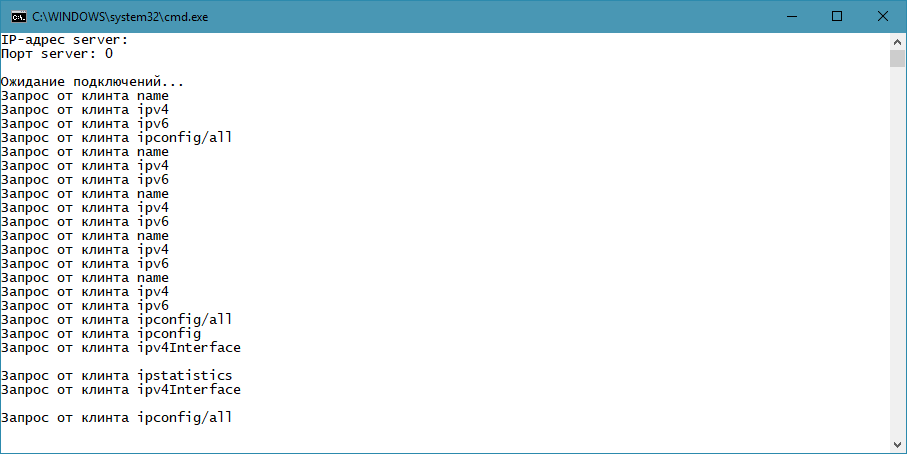


Рисунок 9 – Получение запросов от клиентов

## Руководство пользователя клиента

Для запуска клиентской части нужно запусти файл exe. После запуска появится следующее окно (рисунок 10).

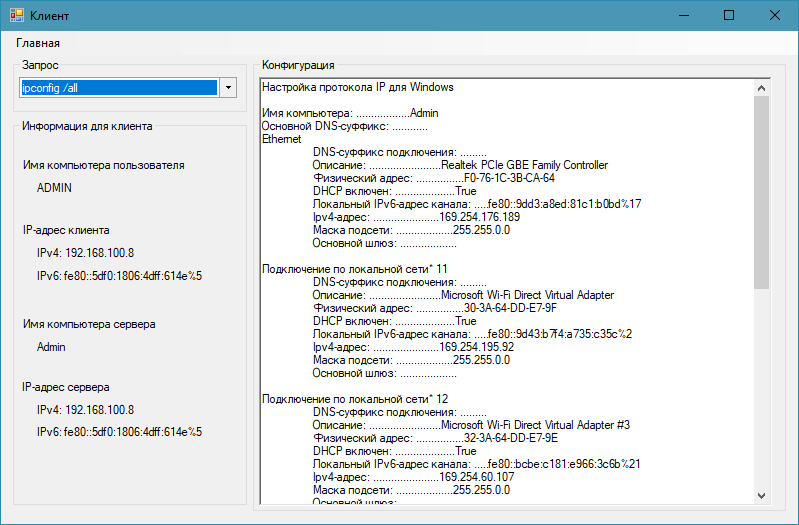


Рисунок 10 – Главное меню клиента

Во вкладке «Главная» можно выбрать настройки, очистку текстового поля, а также выход из приложения (рисунок 11).

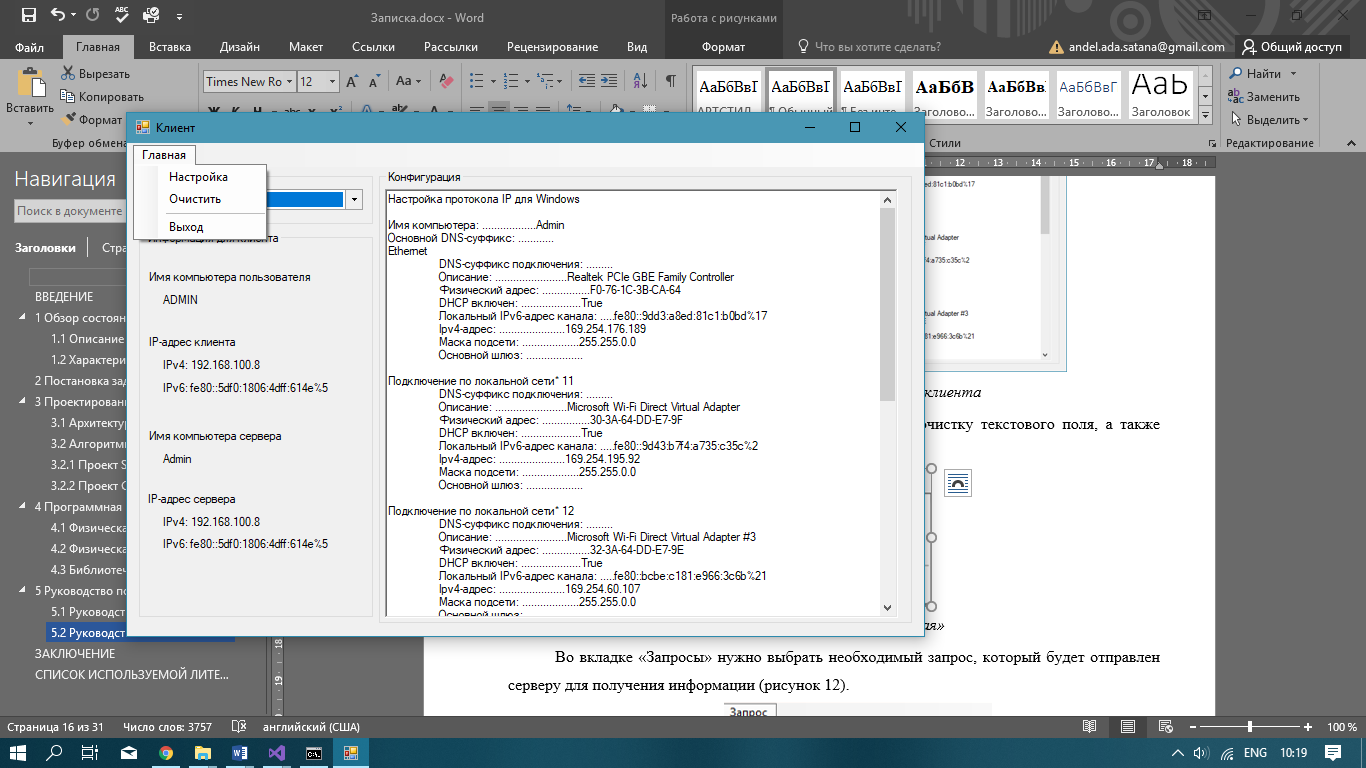


Рисунок 11 – Вкладки «Главная»

Во вкладке «Запросы» нужно выбрать необходимый запрос, который будет отправлен серверу для получения информации (рисунок 12).

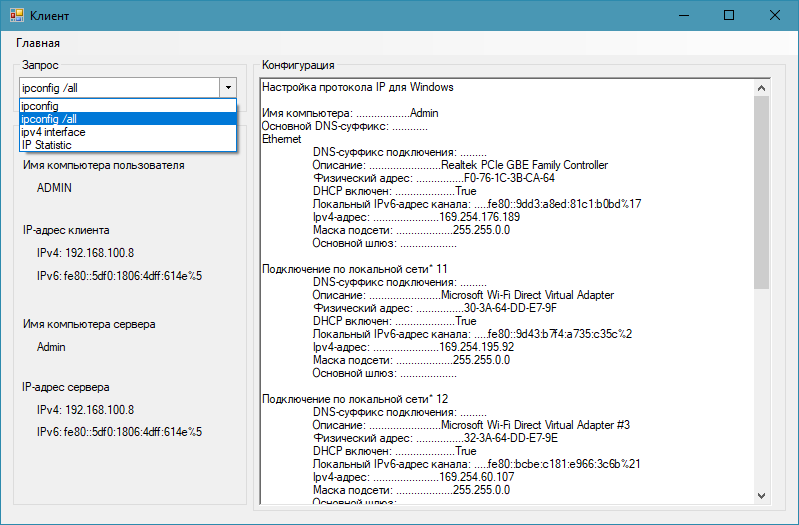


Рисунок 12 – Вкладка «Запросы»

Если нажать «Настройка» во вкладке «Главная», то появится новое окно с выбором определенных действий (рисунок 13). Первая графа служит для подключения к серверу по заданному IP-адресу и прослушивающему порту. Во второй графе можно выбрать цвет текста и фона, для этого нужно нажать на соответствующие кнопки рядом, а также размер текста.

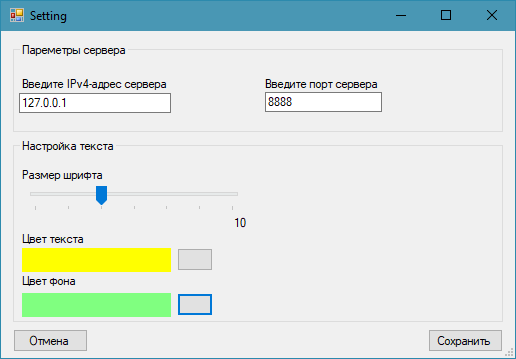


Рисунок 13 – Окно настроек

После выбора можно сохранить параметры или отменить. В обоих случаях окно закроется.

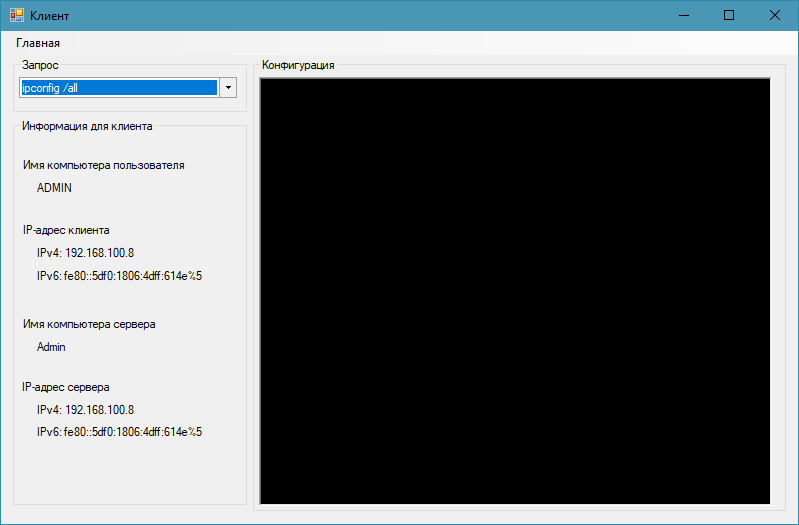


Рисунок 14 – После применение настроек

Если были изменены параметры и нажата кнопка сохранить, то параметры применятся в форме (рисунок 14).

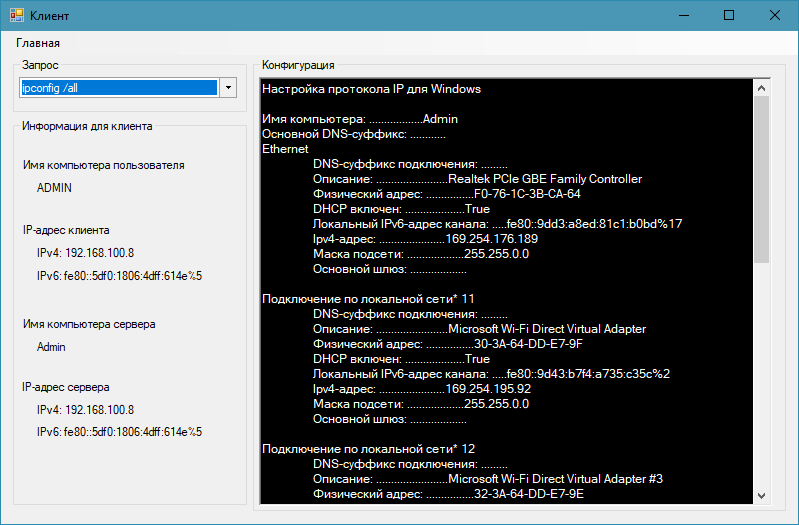


Рисунок 15 – Полученный ответ на запрос

На рисунке 15 показано результат отправленного запроса на сервер и возврат ответа на сервер с его последующем отображением в поле ответа.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения данной курсовой работы были изучены: принцип работы клиент-серверного приложения, средства .NET Framework для разработки клиент-серверного приложения на базе сокетов.

Разработано приложение, демонстрирующее клиент-серверное приложение в ОС Windows, являющееся примером ПО, позволяющего клиентам получать информацию об утилите ipconfig c удаленного узла используя средства .NET, и предназначенное для обмена информацией между клиентскими приложениями и сервером.

Серверное приложение обладает такими возможностями, как: подключение к нему клиентов, прием и обработка запросов клиентов.

Клиентское приложение обладает такими возможностями, как: подключение к серверу, отправка запросов серверу, прием ответа.

Данный программный продукт отвечает поставленным требованиям, имеет простой и дружественный интерфейс, ориентирован на широкий круг пользователей.

# СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Компьютерные сети. Принципы, технологии, протоколы: Учебник для вузов. 2-е изд./В.Г.Олифер, Н.А.Олифер. — СПб: «Питер», 2003. — 864с

2. Белова, С.В. Компьютерные сети: лабораторный практикум / С.В. Белова. – Минск: БНТУ, 2012. – 1-8 с.

3. Олег Герман, Юлия Герман. Программирование на Java и на C# для студента. Санкт-Петербург: «БХВ-Петербург», 2005 – 1-507.

4. Интересные публикации [Электронный ресурс] / UML — диаграмма вариантов использования (use case diagram). Василев А.К. – Электрон. дан. – 2008. – Режим доступа: <http://www.habrahabr.ru/post/47940/>.

# 

**Диаграмма классов сервера**

Изм.

Лист

№ докум.

Подпись

Дата

Лист

1

ДП-1070111409-2019-02

Разраб.

Берестнева О.Я.

Провер.

Белова

*Консульт.*

*Консултант*

Белова

Диаграмма классов сервера

Лит.

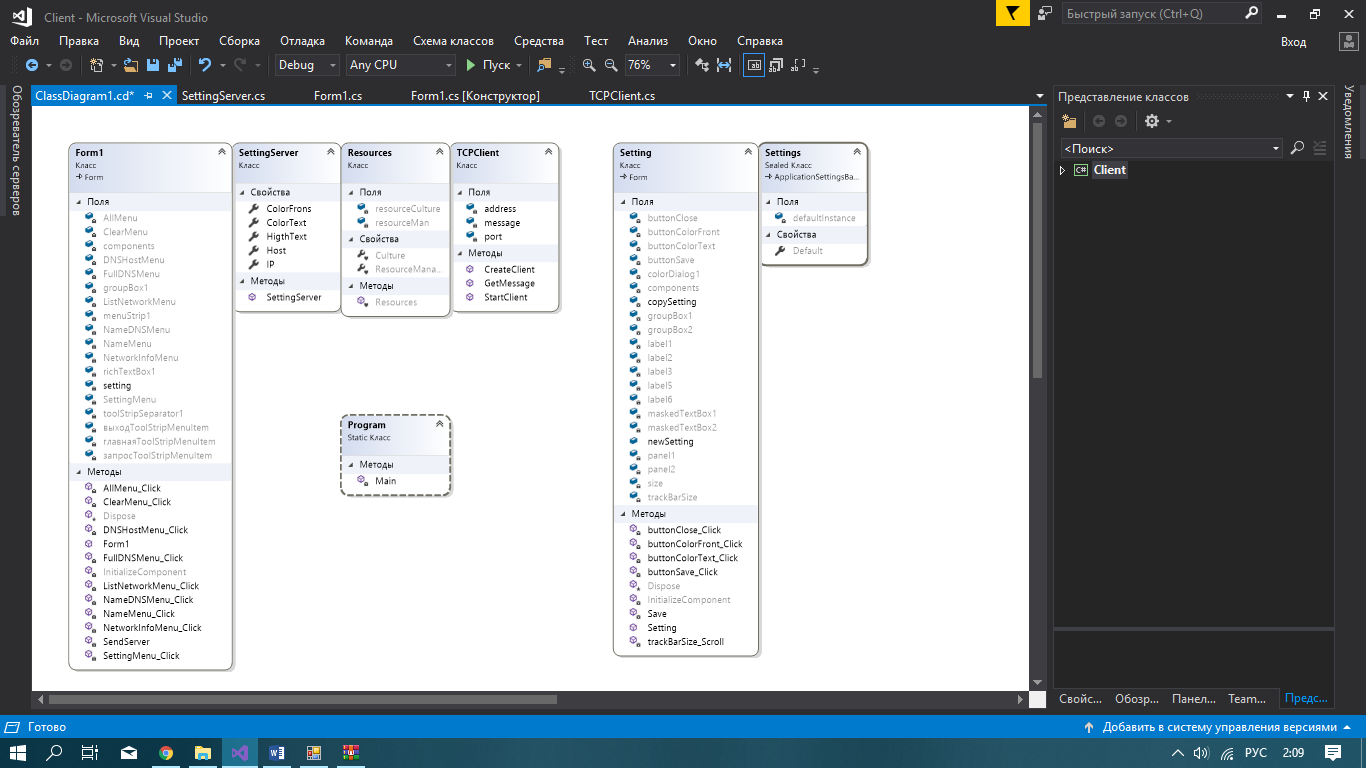
Листов

1

1-40 01 01

БНТУ, г. Минск

У



**Диаграмма Классов клиента**

Изм.

Лист

№ докум.

Подпись

Дата

Лист

1

ДП-1070111409-2019-02

Разраб.

Берестнева О.Я.

Провер.

Белова

*Консульт.*

*Консултант*

Белова

Диаграмма Классов клиента

Лит.

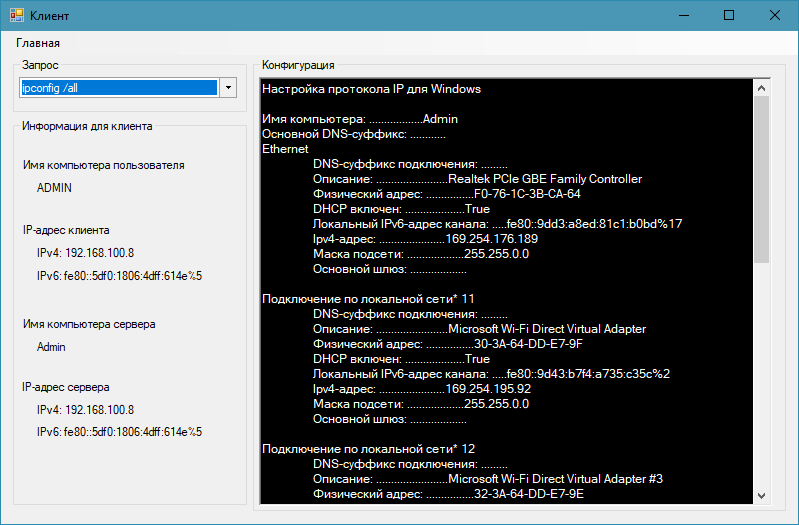
Листов

1

1-40 01 01

БНТУ, г. Минск

У



**Начальное окно приложения клиента**

Изм.

Лист

№ докум.

Подпись

Дата

Лист

1

ДП-1070111409-2019-02

Разраб.

Берестнева О.Я.

Провер.

Белова

*Консульт.*

*Консултант*

Белова

Начальное окно приложения клиента

Лит.

Листов

1

1-40 01 01

БНТУ, г. Минск

У

# Приложение А

Файл Program.cs

class Program

{

const int port = 8888;

const string adrress = "127.0.0.1";

static void Main(string[] args)

{

TCPServer server = new TCPServer();

server.InfoServer();

server.Create(adrress, port);

server.StartServer();

}

}

Файл TCPServer.cs

public class TCPServer

{

static string IPaddress { get; set; }

static int port { get; set; }

static TcpListener listener;

public void Create(string adress, int ip\_port)

{

IPaddress = adress;

port = ip\_port;

}

public void InfoServer()

{

Console.WriteLine("IP-адрес server: {0}", IPaddress);

Console.WriteLine("Порт server: {0}", port);

Console.WriteLine("");

}

public void StartServer()

{

try

{

listener = new TcpListener(IPAddress.Parse(IPaddress), port);

listener.Start();

Console.WriteLine("Ожидание подключений...");

while (true)

{

TcpClient client = listener.AcceptTcpClient();

ClientObject clientObject = new ClientObject(client);

// создаем новый поток для обслуживания нового клиента

Thread clientThread = new Thread(new ThreadStart(clientObject.Process));

clientThread.Start();

}

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

finally

{

if (listener != null)

listener.Stop();

}

}

}

Файл ClientObject.cs

class ClientObject

{

public TcpClient client;

private Сonfigurations сonfigurations = new Сonfigurations();

public ClientObject(TcpClient tcpClient)

{

client = tcpClient;

}

public void Process()

{

NetworkStream stream = null;

try

{

stream = client.GetStream();

byte[] data = new byte[64]; // буфер для получаемых данных

// получаем сообщение

StringBuilder builder = new StringBuilder();

int bytes = 0;

do

{

bytes = stream.Read(data, 0, data.Length);

builder.Append(Encoding.Unicode.GetString(data, 0, bytes));

}

while (stream.DataAvailable);

string message = builder.ToString();

Console.WriteLine("Запрос от клинта {0}",message);

// отправляем обратно сообщение в верхнем регистре

message = сonfigurations.IPConfig(message);

data = Encoding.Unicode.GetBytes(message);

stream.Write(data, 0, data.Length);

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

finally

{

if (stream != null)

stream.Close();

if (client != null)

client.Close();

}

}

}

Файл Сonfigurations.cs

namespace Server

{

class Сonfigurations

{

private string LocalName

{

get

{

return IPGlobalProperties.GetIPGlobalProperties().HostName.ToString();

}

}

private string IPv4

{

get

{

return Dns.GetHostAddresses(Environment.MachineName)[0].ToString();

}

}

private string IPv6

{

get

{

return Dns.GetHostAddresses(Environment.MachineName)[1].ToString();

}

}

private string IPconfig

{

get

{

string text = "";

text = "Настройка протокола IP для Windows \n\n";

text += GetNetworkInterfacesInfo(NetworkInterface.GetAllNetworkInterfaces());

return text;

}

}

private string ToAll

{

get

{

string text = "";

text = "Настройка протокола IP для Windows \n\n";

text += "Имя компьютера: .................." + LocalName + "\n";

text += "Основной DNS-суффикс: ............" + DNSSuffix + "\n";

text += GetNetworkInterfacesInfoAll(NetworkInterface.GetAllNetworkInterfaces());

return text;

}

}

private string DNSSuffix

{

get

{

NetworkInterface[] adapters = NetworkInterface.GetAllNetworkInterfaces();

IPInterfaceProperties properties = adapters[0].GetIPProperties();

return properties.DnsSuffix;

}

}

private static string GetNetworkInterfacesInfoAll(NetworkInterface[] interfaces)

{

StringBuilder info = new StringBuilder();

foreach (NetworkInterface networkInterface in interfaces)

{

info.Append(networkInterface.Name);

info.Append("\n\t DNS-суффикс подключения: ........." + networkInterface.GetIPProperties().DnsSuffix);

info.Append("\n\t Описание: ........................" + networkInterface.Description);

info.Append("\n\t Физический адрес: ................" + BitConverter.ToString(networkInterface.GetPhysicalAddress().GetAddressBytes()));

info.Append("\n\t DHCP включен: ...................." + networkInterface.GetIPProperties().GetIPv4Properties().IsDhcpEnabled);

info.Append("\n\t Локальный IPv6-адрес канала: ....." + networkInterface.GetIPProperties().UnicastAddresses[0].Address.ToString());

info.Append("\n\t Ipv4-адрес: ......................" + networkInterface.GetIPProperties().UnicastAddresses[1].Address.ToString());

info.Append("\n\t Маска подсети: ..................." + networkInterface.GetIPProperties().UnicastAddresses[1].IPv4Mask.ToString());

info.Append("\n\t Основной шлюз: ...................");

info.Append(GatewayAddress(networkInterface));

info.Append("\n\n");

}

return info.ToString();

}

private static string GetNetworkInterfacesInfo(NetworkInterface[] interfaces)

{

StringBuilder info = new StringBuilder();

foreach (NetworkInterface networkInterface in interfaces)

{

info.Append(networkInterface.Name);

info.Append("\n\t DNS-суффикс подключения: ........." + networkInterface.GetIPProperties().DnsSuffix);

info.Append("\n\t Локальный IPv6-адрес канала: ....." + networkInterface.GetIPProperties().UnicastAddresses[0].Address.ToString());

info.Append("\n\t Ipv4-адрес: ......................" + networkInterface.GetIPProperties().UnicastAddresses[1].Address.ToString());

info.Append("\n\t Маска подсети: ..................." + networkInterface.GetIPProperties().UnicastAddresses[1].IPv4Mask.ToString());

info.Append("\n\t Основной шлюз: ...................");

info.Append(GatewayAddress(networkInterface));

info.Append("\n\n");

}

return info.ToString();

}

private static string GatewayAddress(NetworkInterface networkInterface)

{

StringBuilder info = new StringBuilder();

IPInterfaceProperties adapterProperties = networkInterface.GetIPProperties();

GatewayIPAddressInformationCollection addresses = adapterProperties.GatewayAddresses;

if (addresses.Count > 0)

{

foreach (GatewayIPAddressInformation address in addresses)

{

info.Append(address.Address.ToString() + "\n");

}

}

return info.ToString();

}

public static string DisplayIPv4NetworkInterfaces

{

get

{

StringBuilder info = new StringBuilder();

NetworkInterface[] nics = NetworkInterface.GetAllNetworkInterfaces();

IPGlobalProperties properties = IPGlobalProperties.GetIPGlobalProperties();

info.Append("IPv4 interface information for" + properties.HostName + "." + properties.DomainName + "\n");

foreach (NetworkInterface adapter in nics)

{

// Only display informatin for interfaces that support IPv4.

if (adapter.Supports(NetworkInterfaceComponent.IPv4) == false)

{

continue;

}

info.Append(adapter.Description + "\n");

// Underline the description.

info.Append(String.Empty.PadLeft(adapter.Description.Length, '=') + "\n");

IPInterfaceProperties adapterProperties = adapter.GetIPProperties();

// Try to get the IPv4 interface properties.

IPv4InterfaceProperties p = adapterProperties.GetIPv4Properties();

if (p == null)

{

info.Append("No IPv4 information is available for this interface." + "\n");

continue;

}

// Display the IPv4 specific data.

info.Append(" Index ............................. : " + p.Index + "\n");

info.Append(" MTU ............................... : " + p.Mtu + "\n");

info.Append(" APIPA active....................... : " + p.IsAutomaticPrivateAddressingActive + "\n");

info.Append(" APIPA enabled...................... : " + p.IsAutomaticPrivateAddressingEnabled + "\n");

info.Append(" Forwarding enabled................. : " + p.IsForwardingEnabled + "\n");

info.Append(" Uses WINS ......................... : " + p.UsesWins + "\n");

}

return info.ToString();

}

}

public static string ShowIPStatistics

{

get

{

NetworkInterfaceComponent version = new NetworkInterfaceComponent();

StringBuilder info = new StringBuilder();

IPGlobalProperties properties = IPGlobalProperties.GetIPGlobalProperties();

IPGlobalStatistics ipstat = null;

switch (version)

{

case NetworkInterfaceComponent.IPv4:

ipstat = properties.GetIPv4GlobalStatistics();

info.Append(Environment.NewLine + " IPv4 Statistics " + "\n");

break;

case NetworkInterfaceComponent.IPv6:

ipstat = properties.GetIPv4GlobalStatistics();

info.Append(Environment.NewLine + " IPv6 Statistics " + "\n");

break;

default:

throw new ArgumentException("version");

// break;

}

info.Append(" Forwarding enabled ...................... : " + ipstat.ForwardingEnabled + "\n");

info.Append(" Interfaces .............................. : " + ipstat.NumberOfInterfaces + "\n");

info.Append(" IP addresses ............................ : " + ipstat.NumberOfIPAddresses + "\n");

info.Append(" Routes .................................. : " + ipstat.NumberOfRoutes + "\n");

info.Append(" Default TTL ............................. : " + ipstat.DefaultTtl + "\n");

info.Append("\n");

info.Append(" Inbound Packet Data:" + "\n");

info.Append(" Received ............................ : " + ipstat.ReceivedPackets + "\n");

info.Append(" Forwarded ........................... : " + ipstat.ReceivedPacketsForwarded + "\n");

info.Append(" Delivered ........................... : " + ipstat.ReceivedPacketsDelivered + "\n");

info.Append(" Discarded ........................... : " + ipstat.ReceivedPacketsDiscarded + "\n");

info.Append(" Header Errors ....................... : " + ipstat.ReceivedPacketsWithHeadersErrors + "\n");

info.Append(" Address Errors ...................... : " + ipstat.ReceivedPacketsWithAddressErrors + "\n");

info.Append(" Unknown Protocol Errors ............. : " + ipstat.ReceivedPacketsWithUnknownProtocol + "\n");

info.Append("\n");

info.Append(" Outbound Packet Data:" + "\n");

info.Append(" Requested ........................... : " + ipstat.OutputPacketRequests + "\n");

info.Append(" Discarded ........................... : " + ipstat.OutputPacketsDiscarded + "\n");

info.Append(" No Routing Discards ................. : " + ipstat.OutputPacketsWithNoRoute + "\n");

info.Append(" Routing Entry Discards .............. : " + ipstat.OutputPacketRoutingDiscards + "\n");

info.Append("\n");

info.Append(" Reassembly Data:" + "\n");

info.Append(" Reassembly Timeout .................. : " + ipstat.PacketReassemblyTimeout + "\n");

info.Append(" Reassemblies Required ............... : " + ipstat.PacketReassembliesRequired + "\n");

info.Append(" Packets Reassembled ................. : " + ipstat.PacketsReassembled + "\n");

info.Append(" Packets Fragmented .................. : " + ipstat.PacketsFragmented + "\n");

info.Append(" Fragment Failures ................... : " + ipstat.PacketFragmentFailures + "\n");

info.Append("\n");

return info.ToString();

}

}

public string IPConfig(string code)

{

switch (code)

{

case "name":

return LocalName;

case "ipv4":

return IPv4;

case "ipv6":

return IPv6;

case "ipconfig":

return IPconfig;

case "ipconfig/all":

return ToAll;

case "ipv4Interface":

return DisplayIPv4NetworkInterfaces;

case "ipstatistics":

return ShowIPStatistics;

default:

return "ERROR: Данная команда отсутствует!!!";

}

}

}

}

# Приложение В

Файл Form1.cs

using Client.Model;

using Client.View;

using System;

using System.Drawing;

using System.Net;

using System.Net.Sockets;

using System.Text;

using System.Windows.Forms;

namespace Client

{

public partial class Form1 : Form

{

Timer timer = new Timer();

SettingServer setting;

public Form1()

{

InitializeComponent();

setting = new SettingServer();

richTextBox1.BackColor = setting.ColorFrons;

richTextBox1.ForeColor = setting.ColorText;

richTextBox1.Font = new Font("Microsoft Sans Serif", setting.HigthText);

timer.Tick += new EventHandler(InfoServer);

timer.Interval += 10000;

timer.Start();

}

private void InfoServer(object sender, EventArgs e)

{

labelNameClient.Text = Environment.MachineName;

labelIPv4.Text = "IPv4: " + Dns.GetHostAddresses(Environment.MachineName)[0].ToString();

labelIPv6.Text = "IPv6: " + Dns.GetHostAddresses(Environment.MachineName)[1].ToString();

labelNameServer.Text = SendServer("name");

labelIPv4Server.Text = "IPv4: " + SendServer("ipv4");

labelIPv6Server.Text = "IPv6: " + SendServer("ipv6");

}

private string SendServer(string code)

{

string configserver = "";

try

{

TCPClient client = new TCPClient();

client.CreateClient(setting.IP, setting.Host);

client.StartClient(code);

configserver = client.GetMessage();

}

catch (SocketException ex)

{

MessageBox.Show(ex.Message);

}

return configserver;

}

private void ClearMenu\_Click(object sender, EventArgs e)

{

richTextBox1.Clear();

}

private void SettingMenu\_Click(object sender, EventArgs e)

{

Setting formSetting = new Setting(setting);

formSetting.ShowDialog();

setting = formSetting.newSetting;

richTextBox1.BackColor = setting.ColorFrons;

richTextBox1.ForeColor = setting.ColorText;

richTextBox1.Font = new Font("Microsoft Sans Serif", setting.HigthText);

}

private void comboBox1\_SelectedIndexChanged(object sender, EventArgs e)

{

if(comboBox1.SelectedIndex == 0)

{

richTextBox1.Text = SendServer("ipconfig");

}

if (comboBox1.SelectedIndex == 1)

{

richTextBox1.Text = SendServer("ipconfig/all");

}

if (comboBox1.SelectedIndex == 2)

{

richTextBox1.Text = SendServer("ipv4Interface");

}

if (comboBox1.SelectedIndex == 3)

{

richTextBox1.Text = SendServer("ipstatistics");

}

}

private void Form1\_FormClosing(object sender, FormClosingEventArgs e)

{

timer.Stop();

timer.Dispose();

}

}

}

Файл Setting.cs

public partial class Setting : Form

{

SettingServer copySetting;

public SettingServer newSetting;

public Setting(SettingServer setting)

{

InitializeComponent();

maskedTextBox1.ValidatingType = typeof(System.Net.IPAddress);

copySetting = newSetting = setting;

maskedTextBox1.Text = newSetting.IP;

maskedTextBox2.Text = newSetting.Host.ToString();

trackBarSize.Value = newSetting.HigthText;

panel1.BackColor = newSetting.ColorText;

panel1.BackColor = newSetting.ColorFrons;

}

private void trackBarSize\_Scroll(object sender, EventArgs e)

{

size.Text = trackBarSize.Value.ToString();

}

private void buttonColorText\_Click(object sender, EventArgs e)

{

if (colorDialog1.ShowDialog() == DialogResult.Cancel)

return;

panel1.BackColor = colorDialog1.Color;

}

private void buttonColorFront\_Click(object sender, EventArgs e)

{

if (colorDialog1.ShowDialog() == DialogResult.Cancel)

return;

panel2.BackColor = colorDialog1.Color;

}

private void Save()

{

newSetting.IP = IPAddress.Parse(maskedTextBox1.Text).ToString();

newSetting.Host = Convert.ToInt32(maskedTextBox2.Text);

newSetting.HigthText = trackBarSize.Value;

newSetting.ColorText = panel1.BackColor;

newSetting.ColorFrons = panel2.BackColor;

}

private void buttonSave\_Click(object sender, EventArgs e)

{

Save();

Close();

}

private void buttonClose\_Click(object sender, EventArgs e)

{

newSetting = copySetting;

Close();

}

}

Файл TCPClient.cs

class TCPClient

{

int port;

string address;

string message = "";

public void CreateClient(string ip\_address, int ip\_port)

{

port = ip\_port;

address = ip\_address;

}

public void StartClient(string code)

{

TcpClient client = null;

try

{

client = new TcpClient(address, port);

NetworkStream stream = client.GetStream();

string message\_client = code;

// преобразуем сообщение в массив байтов

byte[] data = Encoding.Unicode.GetBytes(message\_client);

// отправка сообщения

stream.Write(data, 0, data.Length);

// получаем ответ

data = new byte[64]; // буфер для получаемых данных

StringBuilder builder = new StringBuilder();

int bytes = 0;

do

{

bytes = stream.Read(data, 0, data.Length);

builder.Append(Encoding.Unicode.GetString(data, 0, bytes));

}

while (stream.DataAvailable);

message = builder.ToString();

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

finally

{

client.Close();

}

}

public string GetMessage()

{

return message;

}

}

Файл SettingServer.cs

public class SettingServer

{

public string IP { get; set; }

public int Host { get; set; }

public Color ColorText { get; set; }

public Color ColorFrons { get; set; }

public int HigthText { get; set; }

public SettingServer()

{

IP = "127.0.0.1";

Host = 8888;

ColorFrons = Color.White;

ColorText = Color.Black;

HigthText = 8;

}

}