МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий и робототехники

Кафедра программного обеспечения информационных систем и технологий

**КУРСОВОЙ ПРОЕКТ**

по дисциплине: «Компьютерные системы и сети**»**

на тему: Сетевая утилита для тестирования связи с удалённым хостом

Выполнил**:** ст. гр. 10701221 Гайдуков С.Ю.

Приняла**:** ст. преподаватель Белова С.В.

Минск 2023

Белорусский национальный технический университет

Кафедра программного обеспечения информационных систем и технологий

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к курсовому проекту**

по дисциплине: **«**Компьютерные системы и сети**»**

на тему: Сетевая утилита для тестирования связи с удалённым хостом

Выполнил**:** ст. гр. 10701221 Гайдуков С.Ю.

Приняла**:** ст. преподаватель Белова С.В.

Минск 2023

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 5](#_Toc134138076)

[1 ОПИСАНИЕ И АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ. ОБЗОР АНАЛОГОВ 5](#_Toc134138077)

[2 ПОСТАНОВКА ЗАДАЧИ. ОПИСАНИЕ ТРЕБОВАНИЙ К ПО. 9](#_Toc134138078)

[3 ПРОЕКТИРОВАНИЕ ПО 13](#_Toc134138079)

[3.1 Диаграмма вариантов использования 10](#_Toc134138080)

[3.2 Архитектура ПО. Описание клиентской и серверной части. 10](#_Toc134138081)

[3.3 Описание протокола взаимодействия клиента и сервера. Структура передаваемых сообщений. 10](#_Toc134138082)

[3.4 Проектирование интерфейса пользователя 10](#_Toc134138083)

[3.5 Модель данных 10](#_Toc134138084)

[3.6 Выбор средств разработки 10](#_Toc134138085)

[4 РЕАЛИЗАЦИЯ ПО 22](#_Toc134138086)

[4.1 Реализация серверной части 10](#_Toc134138087)

[4.2 Реализация клиентской части 10](#_Toc134138088)

[5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 9](#_Toc134138089)

[ЗАКЛЮЧЕНИЕ 12](#_Toc134138090)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 13](#_Toc134138091)

[ПРИЛОЖЕНИЕ А ГРАФИЧЕСКАЯ ЧАСТЬ 14](#_Toc134138092)

[ПРИЛОЖЕНИЕ Б ЛИСТИНГ КЛИЕНТСКОЙ И СЕРВЕРНОЙ ЧАСТИ 20](#_Toc134138093)

**ВВЕДЕНИЕ**

Для разработки графических интерфейсов на платформе .Net используются различные технологии, такие как Windows Forms, WPF и приложения для магазина Windows Store. Однако самой простой и удобной для использования является Windows Forms.

В Windows Forms форма представляет собой визуальную область, где отображается информация для пользователя. Обычно Windows Forms приложения создаются путем размещения элементов управления на форме и написания кода для реагирования на действия пользователя, такие как клики мыши или нажатия клавиш. Элементы управления - это отдельные компоненты пользовательского интерфейса, предназначенные для отображения или ввода данных.

Когда пользователь выполняет какое-либо действие на форме или на одном из ее элементов управления, возникает событие.

Для обработки этих событий приложение использует код. Приложение реагирует на события, выполняя соответствующие действия.

В Visual Studio, при использовании функции перетаскивания, доступен конструктор Windows Forms, который значительно упрощает процесс создания приложений. Достаточно выбрать нужный элемент управления и разместить его на форме в нужном месте. Чтобы облегчить выравнивание элементов управления, конструктор предлагает инструменты, такие как сетки и линии привязки.

Таким образом, Windows Forms предоставляет простой и интуитивно понятный способ создания графических интерфейсов на платформе .Net. Он позволяет разработчикам быстро размещать элементы управления на форме и реагировать на действия пользователей с помощью написанного кода.

# 1 ОПИСАНИЕ И АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ. ОБЗОР АНАЛОГОВ

# Протоколы TCP и ICMP

Протокол TCP (Transmission Control Protocol) является одним из основных протоколов Интернета. Он обеспечивает надежную доставку данных между узлами в компьютерных сетях. Вот несколько основных особенностей, достоинств и недостатков протокола TCP:

Особенности TCP:

1. Надежность: TCP гарантирует, что данные будут доставлены в правильном порядке и без ошибок. Если какие-либо пакеты потеряны или повреждены в процессе передачи, TCP повторно отправляет их, чтобы гарантировать правильность доставки.

2. Управление потоком: TCP контролирует скорость передачи данных, чтобы избежать перегрузки сети. Он использует механизм подтверждений и оконного управления, чтобы гарантировать, что получатель готов принимать данные.

3. Установка соединения: перед началом передачи данных TCP устанавливает соединение между отправителем и получателем, чтобы обеспечить надежность и управление потоком. Это трехэтапный процесс, включающий установку соединения, передачу данных и разрыв соединения.

Достоинства TCP:

1. Надежность: TCP гарантирует доставку данных без потерь или ошибок.

Управление потоком: TCP эффективно управляет потоком данных, предотвращая перегрузку сети.

2. Отслеживание состояния: TCP поддерживает отслеживание состояния соединения и контроль ошибок, что делает его надежным протоколом для передачи данных.

Недостатки TCP:

1. Значительная нагрузка: TCP требует дополнительных ресурсов для обработки своих механизмов надежности и управления потоком, что может вызывать задержки и использование большего объема пропускной способности сети.

2. Потеря эффективности при высоких задержках: в случае больших задержек в сети TCP может столкнуться с проблемой «медленного старта», когда скорость передачи данных медленно увеличивается.

TCP применяется во множестве сетевых протоколов и приложений, включая веб-браузеры, электронную почту, файловые передачи, потоковое видео и другие приложения, где надежность и управление потоком данных критически важны.

Протокол ICMP (Internet Control Message Protocol) является одним из основных протоколов в семействе протоколов Интернета (TCP/IP). Он используется для передачи сообщений об ошибках, управления сетью и диагностики проблем связи между сетевыми узлами.

Некоторые особенности протокола ICMP:

1. Сообщения об ошибках: ICMP используется для отправки сообщений об ошибках, например, когда сетевой узел недоступен или происходит фрагментация пакетов.

2. Управление сетью: ICMP также позволяет управлять сетью. Например, он может использоваться для передачи сообщений о необходимости уменьшить скорость передачи данных, когда сеть перегружена.

3. Диагностика сетевых проблем: ICMP предоставляет инструменты для диагностики проблем связи между сетевыми узлами. Он может быть использован для проверки доступности узлов (ping), измерения задержки (traceroute) и определения маршрутов в сети.

4. Низкая нагрузка на сеть: ICMP-сообщения являются относительно небольшими по размеру и требуют минимальных ресурсов для передачи, что делает протокол эффективным для отправки небольших управляющих и диагностических сообщений.

Достоинства протокола ICMP:

1. Простота и эффективность: ICMP предоставляет простые механизмы обмена сообщениями об ошибках и управления сетью.

2. Широкая поддержка: Протокол ICMP широко поддерживается в различных сетевых устройствах и операционных системах, что обеспечивает его универсальность.

Недостатки протокола ICMP:

1. Потенциальная уязвимость: ICMP может быть использован злоумышленниками для атак на сеть, таких как атаки типа ICMP flood, когда сетевые ресурсы забираются некорректными или избыточными ICMP-запросами.

2. Блокировка на межсетевых экранах (firewall): Некоторые межсетевые экраны (firewalls) могут блокировать или ограничивать ICMP-сообщения, что может затруднить диагностику сетевых проблем.

3. Ограниченная информация: ICMP-сообщения не предоставляют полной информации о состоянии сети или о возникающих проблемах. Они служат только для передачи базовых данных об ошибках или для простой диагностики.

Протокол ICMP применяется в различных сценариях, включая следующие:

1. Проверка доступности сетевых узлов (ping): ICMP может быть использован для проверки доступности удаленных узлов в сети.

2. Диагностика маршрутизации: Использование утилиты traceroute, которая использует ICMP, позволяет определить путь, по которому пакеты проходят от отправителя к получателю и выявить возможные проблемы маршрутизации.

3. Управление сетью: ICMP может быть использован для уведомления сетевых узлов о проблемах, таких как перегрузка или ограничение скорости передачи данных.

Определение наилучшего момента для использования протокола ICMP зависит от конкретных требований и задач. Он обычно применяется при диагностике сетевых проблем, проверке доступности узлов или определении маршрутов в сети. Также может быть полезным для мониторинга сетевых узлов и обнаружения аномалий в сети.

# Сокеты

Сокеты - это программный интерфейс для обмена данными между процессами, выполняющимися на одном компьютере или разных компьютерах в компьютерной сети. Они обеспечивают универсальный способ общения между приложениями, независимо от операционной системы и языка программирования.

Существуют два основных типа сокетов:

1. Сокеты на основе протокола UDP (User Datagram Protocol) - это протокол без установления соединения, ориентированный на передачу датаграмм. Датаграммы представляют собой пакеты данных, которые могут быть отправлены по сети без необходимости установления постоянного соединения между отправителем и получателем. Сокеты UDP обеспечивают не надежную, но быструю передачу данных, что делает их полезными для приложений, где небольшие задержки более приемлемы, чем гарантированная доставка данных.

2. Сокеты на основе протокола TCP (Transmission Control Protocol) - это протокол с установлением соединения, ориентированный на надежную передачу потока данных. Передача данных через сокеты TCP осуществляется путем установления соединения между отправителем и получателем, а затем последовательной передачи данных в виде потока байтов. Сокеты TCP гарантируют, что данные доставлены в том же порядке, в котором они были отправлены, и обеспечивают механизм контроля надежности для обнаружения потерянных или поврежденных пакетов данных.

Оба типа сокетов имеют свои преимущества и применяются в различных сценариях. Сокеты UDP обычно используются в приложениях, где небольшие задержки являются ключевыми, например, в потоковом видео, играх или передаче данных в режиме реального времени. Сокеты TCP широко применяются в приложениях, где важна надежная доставка данных и гарантия порядка их получения, например, в веб-браузерах, электронной почте или передаче файлов.

Однако помимо двух основных существует и Raw-сокет. Raw-сокеты предоставляют доступ к полному заголовку сетевого уровня, что позволяет приложению полностью контролировать создание, отправку и получение пакетов на более низком уровне сетевой стек. Обычно взаимодействие приложений с сетевыми данными происходит на более высоком уровне, где TCP или UDP обрабатывают передачу данных. Но в случаях, когда приложению требуется более низкоуровневый доступ к сетевым пакетам, оно может использовать raw-сокеты. С помощью raw-сокетов приложения могут создавать и отправлять собственные сетевые пакеты, а также принимать и анализировать входящие пакеты. Это дает разработчикам большую гибкость и контроль над обработкой сетевых данных, но также требует от них большего понимания протоколов сетевого уровня и безопасности.

Семейство адресов является важным параметром сокета, указывающим используемый в данный момент сетевой протокол и ограничивающим применение других параметров сокета.

При использовании протоколов TCP/IP адрес сокета представляет собой комбинацию IP-адреса и номера порта прикладной службы. IP-адрес является уникальным в Интернете, а номер порта уникален на отдельном компьютере. Таким образом, адрес сокета обеспечивает уникальность для взаимодействия удаленных процессов и основан на адресе сокета.

Приложение, использующее сокеты, состоит из распределенной программы, выполняемой на обоих концах связи. Инициирующую передачу программу называют клиентом, а сервер представляет собой модуль, пассивно ожидающий входящих запросов на установление соединений от удаленных клиентов. Обычно серверное приложение загружается при запуске системы и прослушивает свой порт, ожидая входящих соединений. Клиентские приложения пытаются установить соединение с сервером, после чего начинается обмен данными. По завершении сеанса связи клиент, как правило, разрывает соединение.

# Описание приложения для тестирования связи с удалённым хостом

В качестве сетевой утилиты была разработана утилита для тестирования связи с удалённым хостом. Приложение работает по принципу клиент-серверного взаимодействия. Сначала необходимо запустить сервера, которые ожидают подключения клиента. Утилита предназначена для просмотра информации о доступных серверах, их данных и взаимодействия с ними аналогами утилит Ping, Tracert и Nslookup.

Все действия клиент производит через кнопки, размещенные на форме. Клиент представлен в виде приложения Windows Forms. Сервера представлены в виде консольных приложений.

Клиентская часть состоит из двух форм. На первой форме клиент получает список доступных серверов, имеет возможность найти новый сервер по имени или IP-адресу, выбирает сервер из списка для дальнейшего взаимодействия, то есть выполнение Ping, Tracert и Nslookup. Вторая форма является вспомогательной: она предназначена для добавления нового сервера в список серверов путем введения в специальное поле имени сервера или его IP-адреса.

Серверная часть представлена консольным приложением. При запуске сервера он имеет заранее назначенный IP (например 127.0.0.3) и находится в ожидании подключения клиента. При подключении клиента сервер добавляется в список доступных серверов на клиентской форме. Также в качестве сервера может выступать существующий удалённый хост, например, [www.google.com](http://www.google.com). В таком случае подобный сервер необходимо добавлять вручную через кнопку “Find other” и ввести имя хоста или его IP-адрес. После нажатия кнопки “Add” в случае успеха нахождения сервера он добавляется в список серверов на основной форме.

# 2 ПОСТАНОВКА ЗАДАЧИ. ОПИСАНИЕ ТРЕБОВАНИЙ К ПО

В рамках данного курсового проекта необходимо реализовать сетевую утилиту для тестирования связи с удалённым хостом.

Разрабатываемое приложение должно иметь архитектуру клиент-сервер, графический интерфейс для пользователя, а также многопользовательский режим.

Клиентская и серверная часть должна иметь графический интерфейс.

Требования к реализации работы:

- операционная система – MS Windows 10;

- среда выполнения – Visual Studio;

- язык программирования – C#;

- Сетевое приложение;

- графический интерфейс;

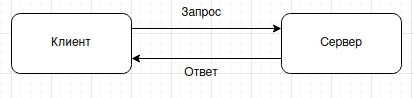
- многопользовательский режим.

Взаимодействие клиента и сервера будет осуществляться с помощью технологии протокола TCP и ICMP.

# 3 ПРОЕКТИРОВАНИЕ ПО

3.1 Диаграмма вариантов использования

Сетевое приложение представляет собой распределенную программу, т. е. программу, которая состоит из нескольких взаимодействующих частей, каждая из которых выполняется на отдельном компьютере сети, либо на одном компьютере, если для связи используется петля обратной связи (см. рис. 3.1.1).



«Рисунок 3.1.1- Взаимодействие частей распределённого приложения»

Распределенная программа использует технологию клиент-сервер.

# Архитектура ПО. Описание клиентской и серверной части.

Архитектура программного обеспечения - это структура программы или вычислительной системы, которая включает компоненты программы, их видимые свойства и взаимосвязи между ними.

Программное обеспечение с тщательно спроектированной архитектурой способно эффективно выполнять поставленные задачи, обеспечивая высокую производительность, безопасность, надежность и другие факторы

Сетевая утилита для тестирования связи с удалённым хостом, разработанная в рамках текущего курсового проекта, представляет собой многопользовательское приложение на основе клиент-серверной технологии, использующее многопоточную модель взаимодействия.

Для клиентов с несколькими серверами такой подход работает хорошо и легко реализуется. Однако многопоточный клиент не масштабируется хорошо из-за большого числа создаваемых и уничтожаемых потоков. При примерно 1000 соединениях начинают возникать исключения, указывающие на нехватку памяти, поскольку каждый поток имеет собственный стек размером 1 МБ по умолчанию.

Чтобы избежать создания отдельного потока для каждого соединения, можно заранее создать пул потоков. При подключении сервера берется поток из пула и в нем происходит взаимодействие с сервером. Задачи выполняются параллельно, и если потоки не обмениваются данными, то не возникает накладных расходов на синхронизацию, что обеспечивает достаточно быструю работу. По завершении работы поток не удаляется, а остается в пуле, ожидая следующей задачи. Это позволяет избежать накладных расходов на создание и удаление потоков. Этот подход подходит для недолговечных соединений. Однако в случае соединений с серверами, которые остаются открытыми в течение длительного времени (например, сервер-чат), ограничение числа соединений числом потоков в пуле становится неприемлемым.

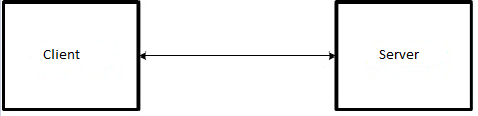
Для решения этой проблемы можно использовать асинхронный сервер. Асинхронность позволяет обрабатывать несколько соединений без создания отдельных потоков для каждого из них. Вместо этого, клиент сможет обрабатывать запросы асинхронно, переключаясь между разными задачами в процессе ожидания ввода-вывода.

# Описание протокола взаимодействия клиента и сервера. Структура передаваемых сообщений

Курсовая работа состоит из следующих модулей:

- Server;

- Client.

На рисунке 3.3.1 показано взаимодействие главных модулей.

«Рисунок 3.3.1- Взаимодействие главных модулей»

Сетевая утилита для тестирования связи с удалённым хостом реализует клиентское приложение, которое будет обслуживать сервера, которые подключаются к этому приложению по многопоточной модели, то есть для нового клиента новый поток.

При подключении клиента к серверу начинается обмен данными между ними на основе технологии сокетов.

Взаимодействие клиента и сервера осуществляется при помощи посылки эхо-пакетов.

* 1. **Проектирование интерфейса пользователя**

Интерфейс пользователя в Visual Studio создавался с использованием интегрированной среды разработки. Эта среда разработки поддерживает множество языков программирования и обеспечивает широкий набор инструментов, таких как анализ кода, графический отладчик, возможность запуска юнит-тестов и поддержка веб-разработки.

С помощью Visual Studio можно выполнять стандартные задачи, характерные для интегрированных сред разработки, такие как просмотр, редактирование, запуск и отладка программ на языке C#. Редактор кода в Visual Studio также предоставляет подсветку синтаксиса.

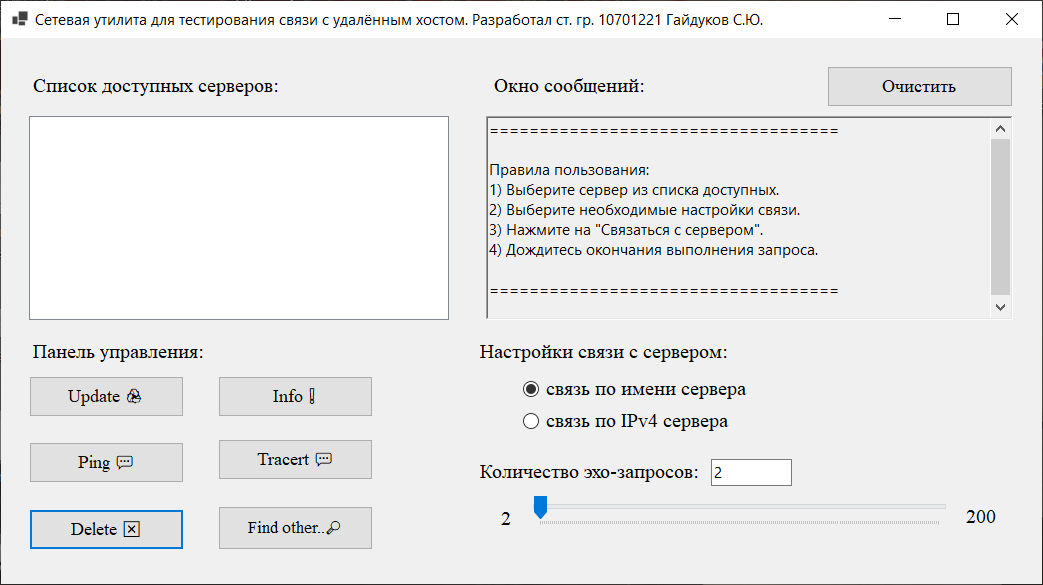
При открытии приложения перед нами появляется загрузочный экран (см. 3.4.1).



«Рисунок 3.4.1 – Загрузочный экран»

Пока пользователю показывается загрузочный экран, в приложении осуществляется поиск всех доступных хостов в локальной сети и получение информации о них.

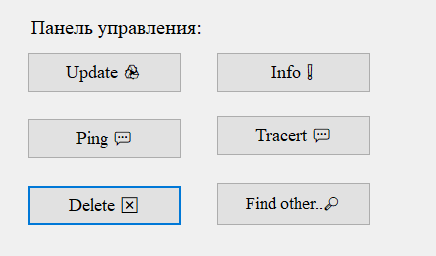
После загрузки информации перед нами появляется основное меню приложения (см. 3.4.2).



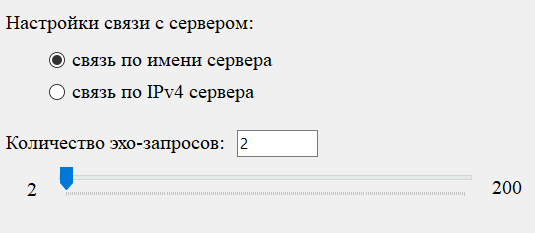
«Рисунок 3.4.2 – Основное меню»

Выбрав доступный сервер по отношению к нему доступны следующие функции на панели управления (см. 3.4.3): Ping, Tracert, Info(Nslookup), а также имеется возможность обновить список серверов путем нажатия “Update” и удалить выбранный сервер из списка путем нажатия “Delete”.

Для утилиты Ping реализована панель настроек связи сервером (см. 3.4.4): можно выбрать Ping по имени сервера либо по IP-адресу, а также выбрать количество посылаемых эхо-пакетов.

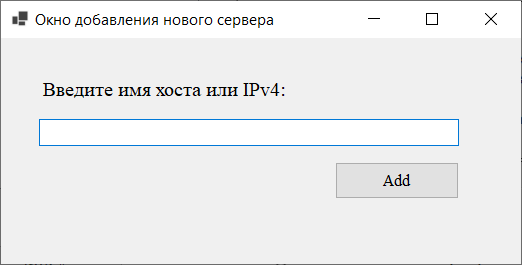


«Рисунок 3.4.3 – Панель управления»



«Рисунок 3.4.4 – Панель настроек связи с сервером»

На отдельной форме (см. 3.4.5) по нажатию кнопки “Find other…” реализовано добавление нового сервера. При нажатии кнопки “Add” если сервер будет найден он добавится в список серверов основной формы.



«Рисунок 3.4.5 – форма добавления сервера»

* 1. **Модель данных**



«Рисунок 3.5.1 - Модель данных»

# Выбор средств разработки

Интерфейс пользователя в Visual Studio создавался с использованием интегрированной среды разработки. Эта среда разработки поддерживает множество языков программирования и обеспечивает широкий набор инструментов, таких как анализ кода, графический отладчик, возможность запуска юнит-тестов и поддержка веб-разработки.

С помощью Visual Studio можно выполнять стандартные задачи, характерные для интегрированных сред разработки, такие как просмотр, редактирование, запуск и отладка программ на языке C#. Редактор кода в Visual Studio также предоставляет подсветку синтаксиса.

# 4 РЕАЛИЗАЦИЯ ПО

**4.1 Реализация серверной части**

Реализация серверной части основана на абстрактном класс Server, от которого наследуются все реализованные сервера. Полями родительного класса являются:

// Длинна очереди

protected const int \_lengthQueue = 10;

// Размер сообщения

protected const int SIZE = 1024;

// Порт

public static int PORT { get; protected set; }

// IP-адресс

public static string IPAdress { get; protected set; }

Реализация работы сервера находится в статическом методе Start().

Листинг метода Start():

protected static void Start()

{

Console.WriteLine($"Запуск сервера IP: {IPAdress}, порт: {PORT}");

// Создание конечной точки по IP и порту

IPEndPoint \_iPEndPoint = new IPEndPoint(IPAddress.Parse(IPAdress), PORT);

// Создание сокета(v4, потоковый, TCP)

Socket \_socket1 = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

// Связь с конечной локальной точкой для ожидания вход. запросов

\_socket1.Bind(\_iPEndPoint);

// Включение прослушивания

\_socket1.Listen(\_lengthQueue);

// Вывод инфо о сервере

Console.WriteLine("Дата и время: " + DateTime.Now + "\r\n");

Console.WriteLine($"Прослушивающий сокет:\r\n Дескриптор: {\_socket1.Handle}\r\n IPv4: {\_iPEndPoint.Address}\r\n Порт: {\_iPEndPoint.Port}\r\n");

// Ожидание подключения

Console.WriteLine("Сервер в режиме ожидания\r\n");

// Инициализация клиентского сокета в случае подключения клиента к серверу

Socket \_socket2 = \_socket1.Accept();

// Инициализация переменной для сообщения от сервера

String \_dataRec = "";

while (true)

{

Console.WriteLine("Дата и время: " + DateTime.Now + "\r\n");

Console.WriteLine($"Получение запроса от клиента:\n Дескриптор: {\_socket2.Handle}\n IPv4: {((IPEndPoint)\_socket2.RemoteEndPoint).Address}\n Порт: {((IPEndPoint)\_socket2.RemoteEndPoint).Port}\r\n");

byte[] \_byteRec = new byte[SIZE];

// Приём сообщения от клиента, запись сообщения и его длинны

int \_lenBytesReciver = \_socket2.Receive(\_byteRec);

// Декодировка

\_dataRec += Encoding.ASCII.GetString(\_byteRec, 0, \_lenBytesReciver);

// Есть ли в сообщении еще символы

if (\_dataRec.IndexOf('.') > -1)

{

break;

}

}

Console.WriteLine($"Получено сообщение от клиента: {\_dataRec}\r\n");

// Инициализация сообщения для клиента

string dataSend = $"\n {Dns.GetHostName()}";

// Кодировка сообщения

byte[] byteSend = Encoding.ASCII.GetBytes(dataSend);

// Отправка сообщения клиенту

int lenBytesSend = \_socket2.Send(byteSend);

Console.WriteLine($"Отправка клиенту {lenBytesSend} bytes");

// Инициирование закрытия сокета клиента

\_socket2.Shutdown(SocketShutdown.Both);

// Закрытие сокета клиента

\_socket2.Close();

Console.WriteLine("Общение с клиентом остановлено");

}

Реализация всех локальных серверов представляет собой следующий код:

Листинг кода Server\_4:

internal class Server\_4 : Servers.Server

{

public static void Main(string[] args)

{

IPAdress = "127.0.0.4";

PORT = 8888;

Start();

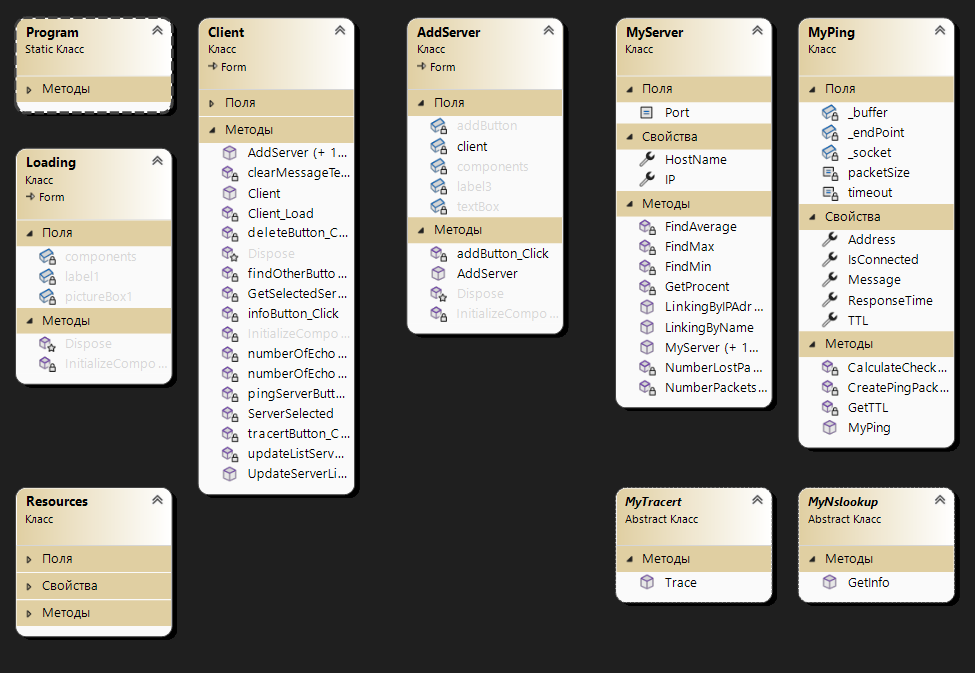
}

}

Аналогичным образом реализованы и остальные локальные сервера.

4.2 Реализация клиентской части

Реализация клиентской части представляет собой куда более объёмный код. Ниже приведена диаграмма классов клиентской части(см. 4.2.1)



«Рисунок 4.2.1 - Диаграмма классов клиентской части»

Класс Client реализует в себе удобную среду для взаимодействия с серверами, отображение серверов, вывод текстовой информации – результата работы различных утилит, вывод MessageBox’ов – сообщений программы об ошибках, создание объектов классов – утилит и получение данных от них. Листинг класса Client представлен в Приложении Б, стр. 36-41.

Класс MyServer является способом реализации хранения информации о серверах и взаимодействия с ними. Он имеет публичные поля, задаваемые при объявлении объекта класса: HostName – имя хоста, IP – IP-адрес, а также публичные методы PingByName(), PingByAdress(), Tracert() и Nslookup() – данные методы реализуют взаимодействие между объектами класса Server и классами, реализующими утилиты. Листинг класса MyServer представлен в Приложении Б, стр. 41-44.

В качестве утилит реализованы следующие классы: MyPing, MyTracert, MyNslookup. Основными методами данных класов являются MyPing(), Trace() и GetInfo() соответственно. Их листинг представлен в Приложении Б, стр. 44-48.

# 5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

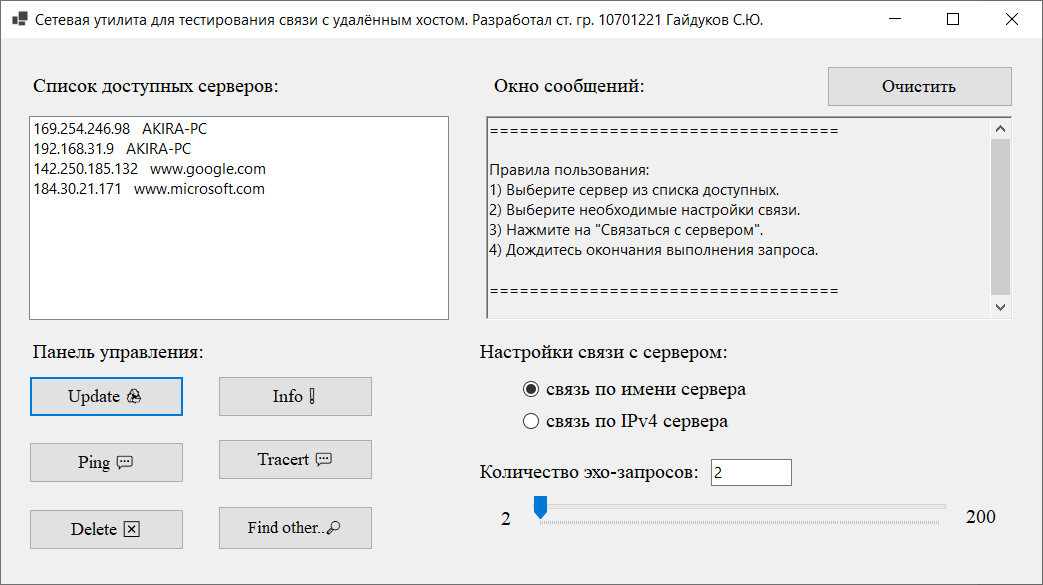
При открытии приложения перед нами появляется загрузочный экран (см. 5.1).



«Рисунок 5.1 – Загрузочный экран»

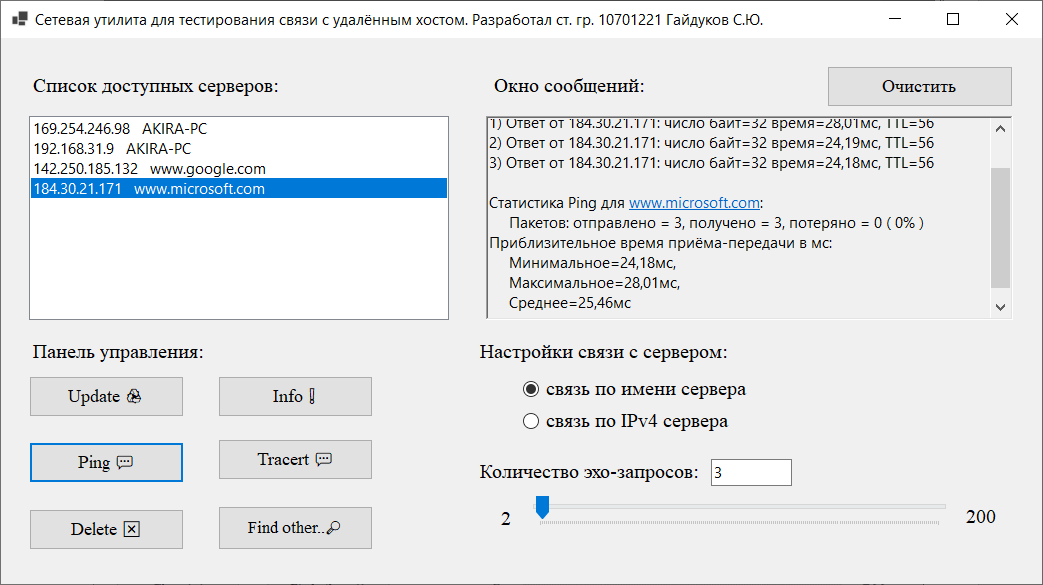
Пока пользователю показывается загрузочный экран, в приложении осуществляется поиск всех доступных хостов в локальной сети и получение информации о них.

После загрузки информации перед нами появляется основное меню приложения (см. 5.2).

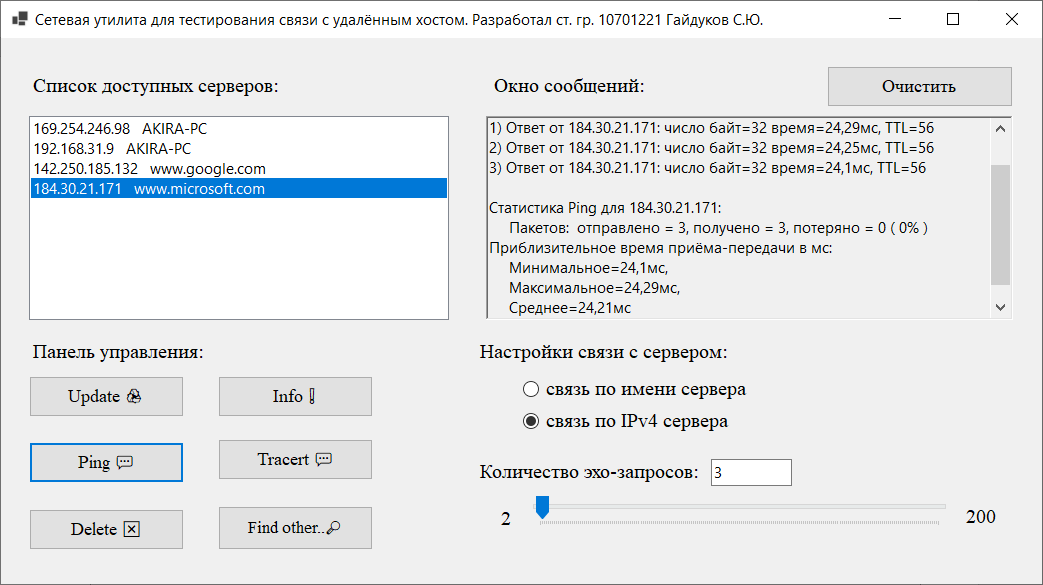


«Рисунок 5.2 – Основное меню»

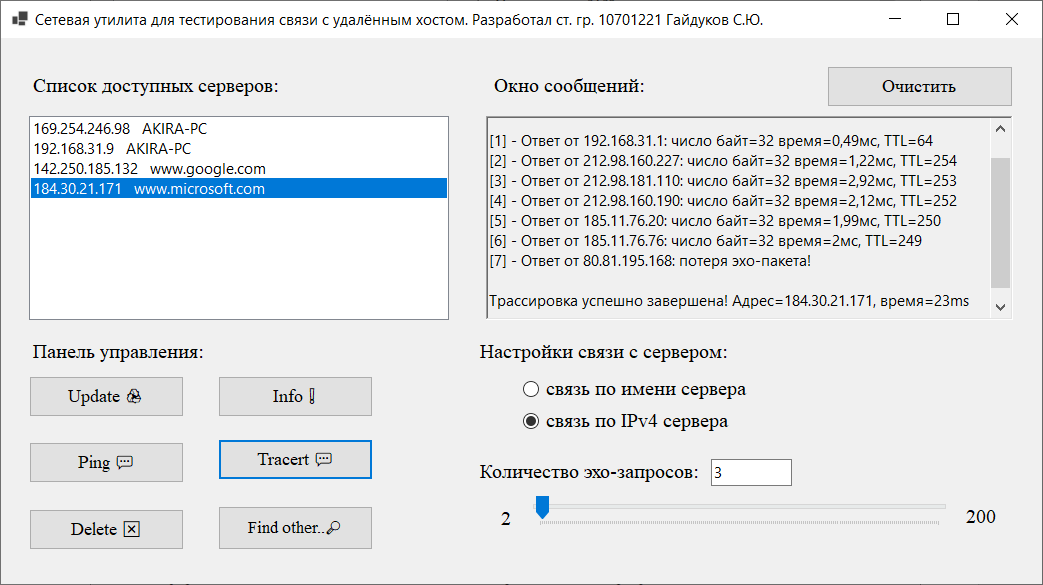
Выбрав доступный сервер по отношению к нему доступны следующие функции: Ping по имени сервера(см. 5.3), Ping по IPv4 адресу сервера(см. 5.4), Tracert (см. 5.5), Info(Nslookup) (см. 5.6).



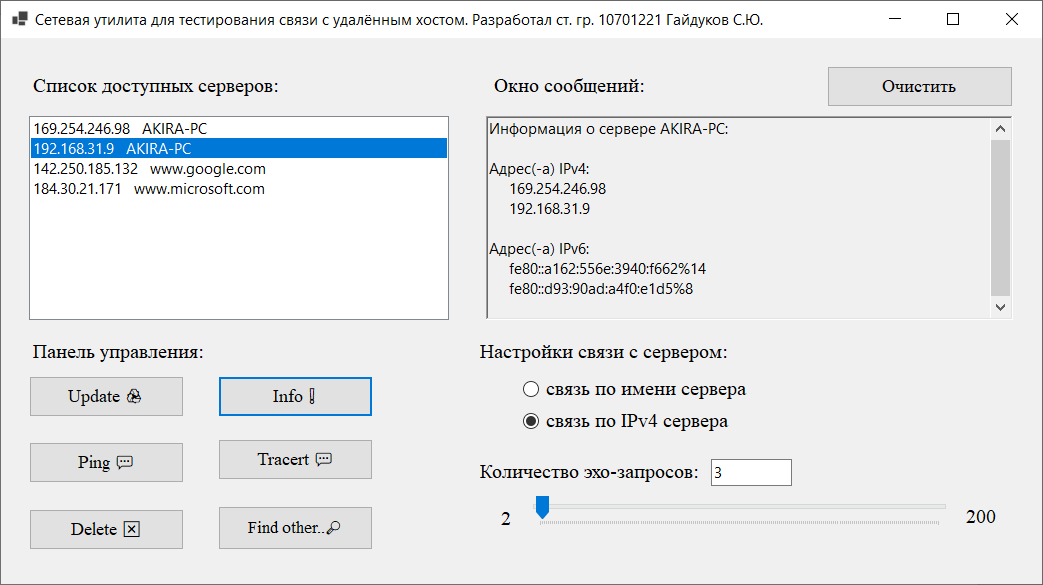
«Рисунок 5.3 – Ping по имени сервера»



«Рисунок 5.4 – Ping по IPv4 сервера»



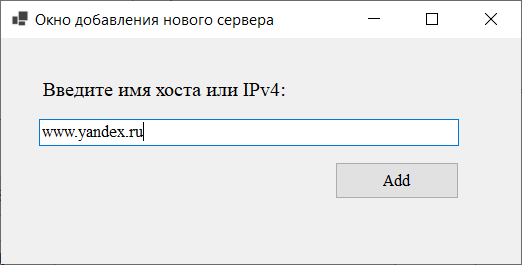
«Рисунок 5.5 – применение утилиты Tracert»



«Рисунок 5.6 – применение утилиты Nslookup»

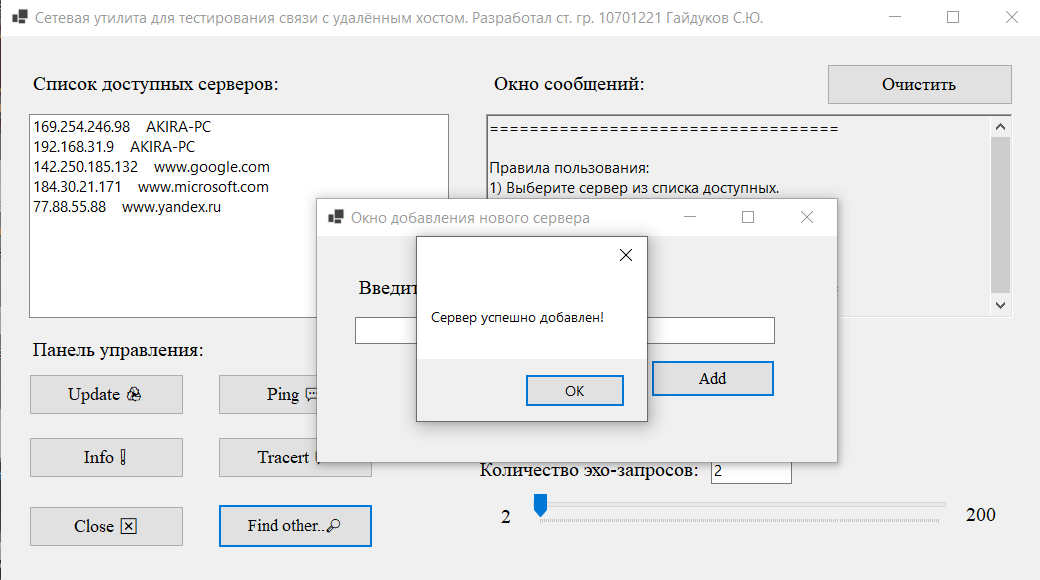
Также имеется возможность обновить список доступных серверов с помощью кнопки “Update”. Пользователю вновь будет показан загрузочный экран, а приложение вновь проведёт поиск доступных сетевых интерфейсов в локальной сети. При этом вручную добавленные сервера будут “забыты” – кнопку можно применять в качестве очистки списка серверов.

Помимо прочего в приложении реализовано добавление хостов, не находящихся в локальной сети, к которому подключено устройство, а в сети Интернет. По нажатию на кнопку “Find other..” перед пользователем появится форма для добавления нового сервера(см. рисунок 5.7).



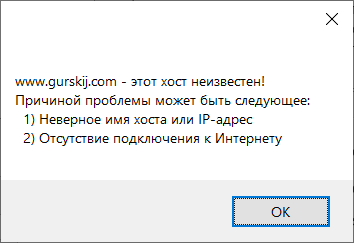
«Рисунок 5.7 – форма для добавления нового сервера»

В случае успеха (если будет найден сервер по заданному доменному имени или IP-адресу) сервер будет добавлен в список доступных серверов, а пользователю будет выведено сообщение о том, что сервер успешно добавлен (см. рисунок 5.8)



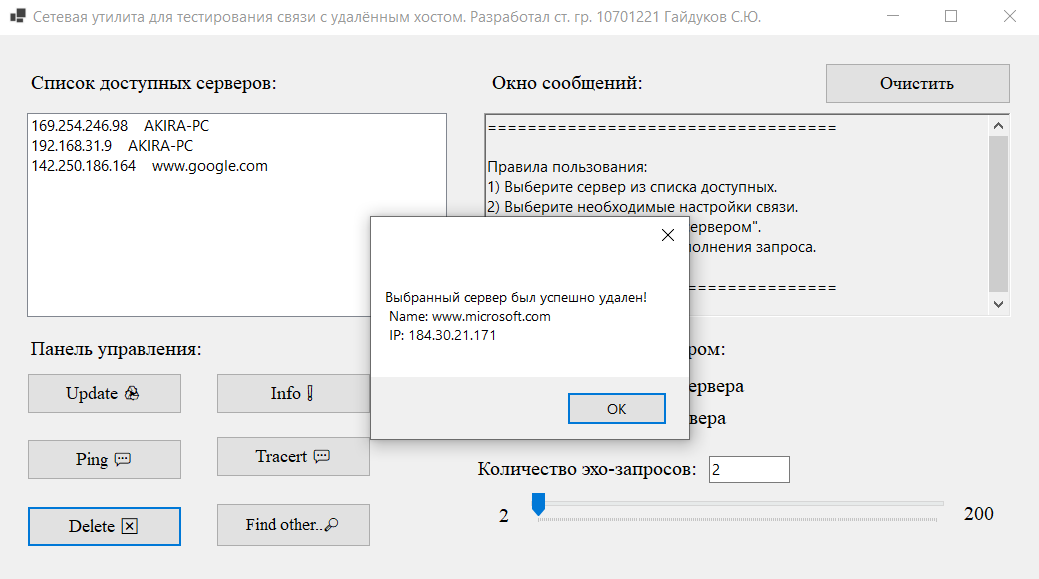
«Рисунок 5.8 – успешное добавление сервера»

В том случае, если по заданному имени или IP сервера не удастся обнаружить хост, будет выведено предупреждающее окно (см. рисунок 5.9) с советами о том, какая причина могла стать результатом неуспешного поиска.



«Рисунок 5.9 – Предупреждающее окно»

Так же реализована функция удаления сервера из списка. Для этого необходимо выбрать сервер и нажать кнопку “Delete” (см. рисунок 5.10).



«Рисунок 5.10 – Удаление сервера»

**ЗАКЛЮЧЕНИЕ**

В ходе выполнения данного курсового проекта были закреплены знания: принцип работы клиент-серверного приложения, средства C# для разработки клиент-серверного приложения на базе сокетов, также был изучен протокол передачи данных ICMP.

Результатом выполнения курсового проекта стало сетевая утилита для тестирования связи с удалённым хостом, разработанное на языке C#. Это приложение предоставляет возможность работать нескольким клиентам в режиме реального времени.

Клиентская часть обладает широкими возможностями, такими как получение, отправка и обработка данных. Она основана на технологии ICMP, которая используется для передачи сообщений об ошибках, управления сетью и диагностики проблем связи между сетевыми узлами.

Серверная часть приложения позволяет быть доступным для обнаружения и дальнейших действий клиентской частью. Приложение имеет интуитивно понятный, простой и эффективный интерфейс который делает работу с серверами понятной и приятной.

В целом, разработка этого клиент-серверного приложения на базе сокетов с использованием языка программирования C# и протокола ICMP привнесла глубокое понимание принципов работы сетевых утилит и расширила набор навыков в области разработки приложений.

**СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ**

1. Компьютерные сети. Принципы, технологии, протоколы: Учебник для вузов. 2-е изд. / В.Г.Олифер, Н.А.Олифер. — СПб: «Питер», 2003. — 864с.
2. Бишоп, Дж. C# в кратком изложении / Дж. Бишоп, Н. Хорспул. - М.: Бином. Лаборатория знаний, 2013. - 472 c.
3. Зиборов, В.В. Visual C# 2012 на примерах / В.В. Зиборов. - М.: БХВ-Петербург, 2013. - 480 c.
4. Петцольд Ч. Программирование с использованием Microsoft Windows Forms / Ч. Петцольд – М.: Русская редакция, 2012 – 426 с.
5. Албахари Д. C# 5.0. Справочник. Полное описание языка / Д. Албахари, Б.Албахари – М.: Вильямс, 2013. – 1008 с.
6. "C# 9 and .NET 5 – Modern Cross-Platform Development" by Mark J. Price, Packt Publishing, 2020 - 816 с.
7. "C# in Depth" by Jon Skeet: Manning Publications, 2019 – 526 с..
8. "CLR via C#" by Jeffrey Richter: Microsoft Press, 2012 – 896 с.
9. "C# Programming Yellow Book" by Rob Miles: Rob Miles, 2014 – 256 с..
10. "Pro C# 9 with .NET 5" by Andrew Troelsen and Philip Japikse: Apress, 2021 – 1536 c.

**ПРИЛОЖЕНИЕ А ГРАФИЧЕСКАЯ ЧАСТЬ**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | | | |
|  |  |  |  |  | КП—1070122108–2023–01 | | | | | |
|  |  |  |  |  |
| Изм. | Лист | № документа | Подпись | Дата |
| Разраб. | | Гайдуков |  |  | *Схема взаимодействия клиентской и серверной части* | Лит | | | Лист | Листов |
| Руковод. | | Белова |  |  |  | У |  | *1* |  |
| Консульт. | | Белова |  |  | 1-40 01 01 БНТУ  г. Минск | | | | |
| Н.контр. | | Белова |  |  |
| Зав.каф. | | Полозков |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | |
|  | Фамилия | Подпись | Дата | *Модель данных* | Лист | Листов |
| Студент | Гайдуков |  |  |
| Руководитель | Белова |  |  | *2* |  |
|  | | | | | | |
|  | Фамилия | Подпись | Дата | *Диаграмма классов клиентской части* | Лист | Листов |
| Студент | Гайдуков |  |  |
| Руководитель | Белова |  |  | *3* |  |

**ПРИЛОЖЕНИЕ Б ЛИСТИНГ КЛИЕНТСКОЙ И СЕРВЕРНОЙ ЧАСТИ**

**Файл Client.cs**

using System;

using System.Net;

using System.Net.Sockets;

using System.Net.NetworkInformation;

using Client.MyClasses;

namespace Client

{

public partial class Client : Form

{

public Client()

{

InitializeComponent();

}

private const int \_port = 8888;

private const int \_size = 1024;

List<MyServer> \_servers = new List<MyServer>();

//

private int \_numberOfEchoRequests = 2;

private readonly int \_numberOfEchoRequestsMaxValue = 200;

private readonly int \_numberOfEchoRequestsMinValue = 2;

//

private Socket \_socket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

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

{

\_servers.Clear();

serversListBox.Items.Clear();

Loading loading = new Loading();

loading.Show();

this.Hide();

loading.Update();

IPEndPoint[] allIPEndPoints = IPGlobalProperties.GetIPGlobalProperties().GetActiveTcpListeners();

List<IPEndPoint> iPEndPoints = new List<IPEndPoint>();

foreach (IPEndPoint iPEndPoint in allIPEndPoints)

{

if (iPEndPoint.Address.ToString() != IPAddress.None.ToString() && iPEndPoint.Address.ToString() != IPAddress.Any.ToString() && iPEndPoint.AddressFamily == AddressFamily.InterNetwork)

{

iPEndPoints.Add(iPEndPoint);

}

}

for (int i = 0; i < iPEndPoints.Count; i++)

{

\_servers.Add(new MyServer(iPEndPoints[i].Address));

}

try

{

IPEndPoint iPEnd = new IPEndPoint(Dns.GetHostByName("www.google.com").AddressList[0], \_port);

\_servers.Add(new MyServer("www.google.com"));

iPEnd = new IPEndPoint(Dns.GetHostByName("www.microsoft.com").AddressList[0], \_port);

\_servers.Add(new MyServer("www.microsoft.com"));

}

catch

{

MessageBox.Show("Отсутствует соединение с Интернетом! Невозможно добавить хост www.google.com");

}

this.Show();

loading.Close();

for (int i = 0; i < \_servers.Count; i++)

{

serversListBox.Items.Add($"{\_servers[i].IP} {\_servers[i].HostName}");

}

}

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

{

\_numberOfEchoRequests = numberOfEchoRequestsTrackBar.Value;

numberOfEchoRequestsTextBox.Text = \_numberOfEchoRequests.ToString();

}

private void numberOfEchoRequestsTextBox\_TextChanged(object sender, EventArgs e)

{

try

{

int \_value = Convert.ToInt32(numberOfEchoRequestsTextBox.Text);

if (\_value < \_numberOfEchoRequestsMinValue)

{

numberOfEchoRequestsTextBox.Text = \_numberOfEchoRequestsMinValue.ToString();

\_value = \_numberOfEchoRequestsMinValue;

}

else if(\_value > \_numberOfEchoRequestsMaxValue)

{

numberOfEchoRequestsTextBox.Text = \_numberOfEchoRequestsMaxValue.ToString();

\_value = \_numberOfEchoRequestsMaxValue;

}

\_numberOfEchoRequests = \_value;

numberOfEchoRequestsTrackBar.Value = \_value;

}

catch

{

MessageBox.Show("Ошибка ввода!");

int \_value = \_numberOfEchoRequestsMinValue;

numberOfEchoRequestsTrackBar.Value = \_value;

numberOfEchoRequestsTextBox.Text = \_value.ToString();

}

}

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

{

if (ServerSelected())

{

if (contactByIPRadioButton.Checked)

{

\_servers[serversListBox.SelectedIndex].LinkingByIPAdress(messageTextBox, \_numberOfEchoRequests);

}

else if (contactByNameRadioButton.Checked)

{

\_servers[serversListBox.SelectedIndex].LinkingByName(messageTextBox, \_numberOfEchoRequests);

}

MessageBox.Show("Выполнение утилиты Ping завершено!");

}

else

{

MessageBox.Show("Выберите сервер из списка для взаимодействия!");

}

}

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

{

}

private void Client\_Load(object sender, EventArgs e)

{

updateListServersButton\_Click(sender, e);

}

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

{

messageTextBox.Text = "";

}

private bool ServerSelected()

{

return serversListBox.SelectedIndex > -1;

}

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

{

if(ServerSelected())

{

messageTextBox.Text = MyNslookup.GetInfo(\_servers[serversListBox.SelectedIndex]);

}

else

{

MessageBox.Show("Выберите сервер из списка для взаимодействия!");

}

}

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

{

AddServer addServerForm = new AddServer(this);

addServerForm.Show();

}

public void AddServer(IPAddress iP)

{

\_servers.Add(new MyServer(iP));

}

public void AddServer(string hostname)

{

\_servers.Add(new MyServer(hostname));

}

public void UpdateServerListBox()

{

serversListBox.Items.Clear();

foreach(var server in \_servers)

{

serversListBox.Items.Add($"{server.IP} {server.HostName}");

}

}

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

{

if (ServerSelected())

{

MyTracert.Trace(\_servers[serversListBox.SelectedIndex], messageTextBox);

MessageBox.Show("Выполнение утилиты Tracert завершено!");

}

else

{

MessageBox.Show("Выберите сервер из списка для взаимодействия!");

}

}

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

{

if (ServerSelected())

{

string serverName = \_servers[serversListBox.SelectedIndex].HostName;

IPAddress serverIP = \_servers[serversListBox.SelectedIndex].IP;

\_servers[serversListBox.SelectedIndex] = null;

List<MyServer> \_updateServers = new List<MyServer>();

for (int i = 0; i < \_servers.Count; i++)

{

if (\_servers[i] != null)

{

\_updateServers.Add(\_servers[i]);

}

}

\_servers = \_updateServers;

UpdateServerListBox();

MessageBox.Show($"Выбранный сервер был успешно удален!\r\n Name: {serverName}\r\n IP: {serverIP}");

}

else

{

MessageBox.Show("Выберите сервер из списка для взаимодействия!");

}

}

}

}

**Файл MyServer.cs**

using System.Net.Sockets;

using System.Net;

namespace Client.MyClasses

{

public class MyServer

{

public const int Port = 8888;

public IPAddress IP { get; private set; }

public string HostName { get; private set; }

public MyServer(string hostName)

{

HostName = hostName;

IPAddress[] iPs = Dns.GetHostAddresses(HostName);

bool findIPv4 = false;

for (int i = 0; i < iPs.Length; i++)

{

if (iPs[i].AddressFamily == AddressFamily.InterNetwork)

{

IP = iPs[i];

findIPv4 = true;

break;

}

}

if (findIPv4 == false)

{

IP = iPs[0];

}

}

public MyServer(IPAddress address)

{

IP = address;

HostName = Dns.GetHostByAddress(IP).HostName;

}

public void LinkingByIPAdress(RichTextBox textBox, int numberOfEchoRequests)

{

textBox.Text = "Выполнение проверки соединения с сервером по IPv4\r\n\r\n";

try

{

List<float> time = new List<float>();

List<bool> connectedEchoPackage = new List<bool>();

for (int i = 0; i < numberOfEchoRequests; i++)

{

Thread.Sleep(50);

MyPing myPing = new MyPing(IP);

connectedEchoPackage.Add(myPing.IsConnected);

string message = $"{i + 1}) " + myPing.Message;

textBox.AppendText(message);

if (myPing.IsConnected)

{

time.Add(myPing.ResponseTime);

}

textBox.Update();

}

textBox.AppendText($"\r\nСтатистика Ping для {IP}:");

textBox.AppendText($"\r\n Пакетов: отправлено = {numberOfEchoRequests}, получено = {NumberPacketsReceived(connectedEchoPackage)}, потеряно = {NumberLostPackets(connectedEchoPackage, numberOfEchoRequests)} ( {GetProcent(NumberLostPackets(connectedEchoPackage, numberOfEchoRequests), numberOfEchoRequests)}% )");

if (time.Count > 0)

{

textBox.AppendText($"\r\nПриблизительное время приёма-передачи в мс:\r\n Минимальное={FindMin(time)}мс,\r\n Максимальное={FindMax(time)}мс,\r\n Среднее={FindAverage(time)}мс\r\n");

}

}

catch (Exception ex)

{

textBox.Text += $"При проверке связи не удалось подключиться к выбранному узлу\r\n";

textBox.Update();

}

}

public void LinkingByName(RichTextBox textBox, int numberOfEchoRequests)

{

textBox.Text = "Выполнение проверки соединения с сервером по имени хоста\r\n\r\n";

try

{

IPAddress[] addresses = Dns.GetHostByName(HostName).AddressList;

IPAddress addressv4 = null;

foreach (var adress in addresses)

{

if (adress.AddressFamily == AddressFamily.InterNetwork)

{

addressv4 = adress;

}

}

if (addressv4 != null)

{

List<float> time = new List<float>();

List<bool> connectedEchoPackage = new List<bool>();

for (int i = 0; i < numberOfEchoRequests; i++)

{

Thread.Sleep(50);

MyPing myPing = new MyPing(IP);

textBox.Text += $"{i + 1}) " + myPing.Message;

textBox.Update();

connectedEchoPackage.Add(myPing.IsConnected);

if (myPing.IsConnected)

{

time.Add(myPing.ResponseTime);

}

}

textBox.Text += $"\r\nСтатистика Ping для {HostName}:";

textBox.Text += $"\r\n Пакетов: отправлено = {numberOfEchoRequests}, получено = {NumberPacketsReceived(connectedEchoPackage)}, потеряно = {NumberLostPackets(connectedEchoPackage, numberOfEchoRequests)} ( {GetProcent(NumberLostPackets(connectedEchoPackage, numberOfEchoRequests), numberOfEchoRequests)}% )";

if (time.Count > 0)

{

textBox.Text += $"\r\nПриблизительное время приёма-передачи в мс:\r\n Минимальное={FindMin(time)}мс,\r\n Максимальное={FindMax(time)}мс,\r\n Среднее={FindAverage(time)}мс\r\n";

}

}

else

{

textBox.Text += "Не найден IPv4 для подключения\r\n";

textBox.Update();

}

}

catch (Exception ex)

{

textBox.Text += "При проверке связи не удалось подключиться к выбранному узлу\r\n";

textBox.Update();

}

}

private float FindMin(List<float> list)

{

float min = list[0];

foreach (var item in list)

{

if (item < min)

{

min = item;

}

}

return min;

}

private float FindMax(List<float> list)

{

float max = list[0];

foreach (var item in list)

{

if (item > max)

{

max = item;

}

}

return max;

}

private float FindAverage(List<float> list)

{

float sum = 0;

foreach (var item in list)

{

sum += item;

}

return (float)Math.Round(sum / list.Count, 2);

}

private int NumberPacketsReceived(List<bool> list)

{

int count = 0;

foreach (bool connect in list)

{

if (connect == true)

{

count++;

}

}

return count;

}

private int NumberLostPackets(List<bool> list, int numberOfEchoRequests)

{

return numberOfEchoRequests - NumberPacketsReceived(list);

}

private float GetProcent(int a, int b)

{

return (float)Math.Round((float)a / b \* 100, 2);

}

}

}

**Файл MyPing.cs**

using System.Net;

using System.Net.Sockets;

namespace Client.MyClasses

{

public class MyPing

{

public IPAddress Address { get; private set; }

public bool IsConnected { get; private set; }

public float ResponseTime { get; private set; }

public byte TTL { get; private set; }

public string Message { get; private set; }

private byte[] \_buffer;

private EndPoint \_endPoint;

private Socket \_socket;

const int timeout = 1000;

const int packetSize = 32;

public MyPing(IPAddress address)

{

Address = address;

\_buffer = new byte[packetSize];

new Random().NextBytes(\_buffer);

\_socket = new Socket(AddressFamily.InterNetwork, SocketType.Raw, ProtocolType.Icmp);

\_socket.SetSocketOption(SocketOptionLevel.Socket, SocketOptionName.ReceiveTimeout, timeout);

\_endPoint = new IPEndPoint(address, 0);

DateTime sendTime = DateTime.Now;

\_socket.SendTo(CreatePingPacket(\_buffer), \_endPoint);

byte[] responseBuffer = new byte[\_buffer.Length + 28];

try

{

\_socket.ReceiveFrom(responseBuffer, ref \_endPoint);

DateTime receiveTime = DateTime.Now;

ResponseTime = (float)Math.Round(((float)(receiveTime.Ticks - sendTime.Ticks)) / 10000, 2);

TTL = GetTTL(responseBuffer);

IsConnected = true;

Message = $"Ответ от {Address}: число байт={\_buffer.Length} время={ResponseTime}мс, TTL={TTL}\r\n";

}

catch (SocketException ex)

{

IsConnected = false;

Message = $"Ответ от {Address}: потеря эхо-пакета!\r\n";

}

}

private static byte GetTTL(byte[] ipHeader)

{

// IP-заголовок имеет фиксированный размер 20 байт (для IPv4)

// Восьмой байт в заголовке содержит TTL

const int ttlOffset = 8;

return ipHeader[ttlOffset];

}

private static byte[] CreatePingPacket(byte[] buffer)

{

const int icmpHeaderSize = 8;

byte[] packet = new byte[icmpHeaderSize + buffer.Length];

packet[0] = 8; // Эхо-запрос ICMP

packet[1] = 0; // Всегда ноль

Array.Copy(BitConverter.GetBytes(0), 0, packet, 2, 2); // Всегда ноль

Array.Copy(BitConverter.GetBytes(0), 0, packet, 4, 2); // Идентификатор

Array.Copy(BitConverter.GetBytes(0), 0, packet, 6, 2); // Номер последовательности

Array.Copy(buffer, 0, packet, icmpHeaderSize, buffer.Length);

int checksum = CalculateChecksum(packet);

Array.Copy(BitConverter.GetBytes(checksum), 0, packet, 2, 2);

return packet;

}

private static int CalculateChecksum(byte[] buffer)

{

int sum = 0;

for (int i = 0; i < buffer.Length; i += 2)

{

sum += (int)BitConverter.ToUInt16(buffer, i);

}

sum = (sum >> 16) + (sum & 0xffff);

sum += sum >> 16;

return ~sum;

}

}

}

**Файл MyTracert.cs**

using System.Net;

using System.Text;

namespace Client.MyClasses

{

using System.Net;

using System.Net.NetworkInformation;

public abstract class MyTracert

{

public static void Trace(MyServer server, RichTextBox textBox)

{

textBox.Clear();

IPAddress address = server.IP;

textBox.AppendText($"Трассировка маршрута к {server.HostName} [{address}]\r\n\r\n");

int ttl = 1;

while (true)

{

PingReply reply = new Ping().Send(address, 5000, new byte[] { 0 }, new PingOptions(ttl, true));

if (reply.Status == IPStatus.Success)

{

textBox.AppendText($"\r\nТрассировка успешно завершена! Адрес={reply.Address}, время={reply.RoundtripTime}ms\r\n");

break;

}

else

{

MyPing myPing = new MyPing(reply.Address);

textBox.AppendText($"[{ttl}] - {myPing.Message}");

}

if (ttl++ > 30)

{

textBox.AppendText("Трассировка прервана.\r\n");

break;

}

}

}

}

}

**Файл MyNslookup.cs**

using System.Net;

using System.Net.Sockets;

namespace Client.MyClasses

{

internal abstract class MyNslookup

{

public static string GetInfo(MyServer server)

{

string message = $"Информация о сервере {server.HostName}:\r\n\r\n";

try

{

IPHostEntry host = Dns.GetHostByName(server.HostName);

IPAddress[] ips = host.AddressList;

List<IPAddress> ipV4 = new List<IPAddress>();

List<IPAddress> ipV6 = new List<IPAddress>();

foreach (IPAddress ip in ips)

{

if (ip.AddressFamily == AddressFamily.InterNetwork)

{

ipV4.Add(ip);

}

else if (ip.AddressFamily == AddressFamily.InterNetworkV6)

{

ipV6.Add(ip);

}

}

if (ipV4.Count > 0)

{

message += "Адрес(-а) IPv4:\r\n";

foreach (var ip in ipV4)

{

message += " " + ip.ToString() + "\r\n";

}

}

if (ipV6.Count > 0)

{

message += "\r\nАдрес(-а) IPv6:\r\n";

foreach (var ip in ipV6)

{

message += " " + ip.ToString() + "\r\n";

}

}

string[] aliesNames = host.Aliases;

if (aliesNames.Length > 0)

{

message += "\r\nAlias-имена домена:\r\n";

foreach (string aliesName in aliesNames)

{

message += " " + aliesName + "\r\n";

}

}

return message;

}

catch (Exception ex)

{

message += "Произошла ошибка получения информации о сервере!";

}

return message;

}

}

}}

**Файл Server.cs**

using System.Net;

using System.Net.Sockets;

using System.Text;

namespace Servers

{

internal abstract class Server

{

// Длинна очереди

protected const int \_lengthQueue = 10;

// Размер сообщения

protected const int SIZE = 1024;

public static int PORT { get; protected set; }

public static string IPAdress { get; protected set; }

protected static void Start()

{

Console.WriteLine($"Запуск сервера IP: {IPAdress}, порт: {PORT}");

// Создание конечной точки по IP и порту

IPEndPoint \_iPEndPoint = new IPEndPoint(IPAddress.Parse(IPAdress), PORT);

// Создание сокета(v4, потоковый, TCP)

Socket \_socket1 = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

// Связь с конечной локальной точкой для ожидания вход. запросов

\_socket1.Bind(\_iPEndPoint);

// Включение прослушивания

\_socket1.Listen(\_lengthQueue);

// Вывод инфо о сервере

Console.WriteLine("Дата и время: " + DateTime.Now + "\r\n");

Console.WriteLine($"Прослушивающий сокет:\r\n Дескриптор: {\_socket1.Handle}\r\n IPv4: {\_iPEndPoint.Address}\r\n Порт: {\_iPEndPoint.Port}\r\n");

// Ожидание подключения

Console.WriteLine("Сервер в режиме ожидания\r\n");

// Инициализация клиентского сокета в случае подключения клиента к серверу

Socket \_socket2 = \_socket1.Accept();

// Инициализация переменной для сообщения от сервера

String \_dataRec = "";

while (true)

{

Console.WriteLine("Дата и время: " + DateTime.Now + "\r\n");

Console.WriteLine($"Получение запроса от клиента:\n Дескриптор: {\_socket2.Handle}\n IPv4: {((IPEndPoint)\_socket2.RemoteEndPoint).Address}\n Порт: {((IPEndPoint)\_socket2.RemoteEndPoint).Port}\r\n");

byte[] \_byteRec = new byte[SIZE];

// Приём сообщения от клиента, запись сообщения и его длинны

int \_lenBytesReciver = \_socket2.Receive(\_byteRec);

// Декодировка

\_dataRec += Encoding.ASCII.GetString(\_byteRec, 0, \_lenBytesReciver);

// Есть ли в сообщении еще символы

if (\_dataRec.IndexOf('.') > -1)

{

break;

}

}

Console.WriteLine($"Получено сообщение от клиента: {\_dataRec}\r\n");

// Инициализация сообщения для клиента

string dataSend = $"\n {Dns.GetHostName()}";

// Кодировка сообщения

byte[] byteSend = Encoding.ASCII.GetBytes(dataSend);

// Отправка сообщения клиенту

int lenBytesSend = \_socket2.Send(byteSend);

Console.WriteLine($"Отправка клиенту {lenBytesSend} bytes");

// Инициирование закрытия сокета клиента

\_socket2.Shutdown(SocketShutdown.Both);

// Закрытие сокета клиента

\_socket2.Close();

Console.WriteLine("Общение с клиентом остановлено");

}