

РЕФЕРАТ

Бакалаврская работа содержит 69 листов, 23 рисунка, 5 источников, 3 приложения.

СЕРВЕР, КЛИЕНТ, БАЗА ДАННЫХ, ЛОКАЛЬНАЯ СЕТЬ, ОБРАБОТКА, ЗАГРУЗКА КОМПЬЮТЕРА, КЛИЕНТ-СЕРВЕР, VISUAL STUDIO, C#, SQL SERVER.

Цель работы – разработать клиент-серверное приложение, позволяющее осуществлять сбор статистических данных по загрузке персональных компьютеров в локальной сети.

Объектом разработки является программный интерфейс для приложения-сервера и приложения-клиента, которые должны обмениваться между собой некоторой информацией (запросами и ответами).

В результате проведенной работы создан проект, удовлетворяющий поставленной задаче. А именно, создано приложение-клиент, которое собирает информацию с текущего компьютера по загрузке процессора, оперативной памяти, сети, диска, а затем отправляет эти данные на сервер. Приложение-сервер принимает данные от подключенных к нему клиентов и записывает их в базу данных, а также имеет возможность просматривать записанные данные в виде графиков, структурируя их по пользователям и по времени.

					ПГУ 1.09.03.01.07.001 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата				
Разраб.		Жирков К.Д.			Система сбора статистических данных по загрузке персональных компьютеров в локальной сети. Пояснительная записка	Лит.	Лист	Листов
Провер.		Дорошенко И.Н.					4	69
Н. Контр		Кучин А.В.				ФВТ, каф. ВТ гр. 13ВВ1		
.								
Утверд.		Пащенко Д.В.						

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	7
1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ.....	9
1.1 Постановка задачи	9
1.2 Обзор существующих программных решений.....	9
1.3 Актуальность использования	11
2 ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЯ.....	13
2.1 Этап проектирования	13
2.1.1 Проектирование структуру системы.....	13
2.1.2 Проектирование интерфейса клиентской части	14
2.1.3 Проектирование интерфейса серверной части	14
2.1.4 Проектирование структуры базы данных.....	16
2.2 Этап разработки программы.....	17
2.2.1 Выбор средств разработки программы.....	17
2.2.2 Описание разработки клиентского приложения.....	19
2.2.3 Описание разработки серверного приложения.....	21
3 ОТЛАДКА И ТЕСТИРОВАНИЕ	24
3.1 Руководство пользователя	24
3.2 Отладка и тестирование	35
ЗАКЛЮЧЕНИЕ	40
СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ	41
ПРИЛОЖЕНИЕ А ЛИСТИНГ	42
Приложение А.1 Файл «server.cs»	42
Приложение А.2 Файл «client.cs»	53
ПРИЛОЖЕНИЕ В UML ДИАГРАММЫ	58
Приложение В.1 Диаграмма вариантов использования	58
Приложение В.2 Диаграмма последовательностей.....	60
Приложение В.3 Диаграмма развертывания	61

Приложение В.4 Диаграмма классов	61
Приложение В.5 Диаграмма состояния	62
ПРИЛОЖЕНИЕ С ПРЕЗЕНТАЦИЯ.....	63

					ПГУ 1.09.03.01.07.001 ПЗ	Лист
						6
Изм.	Лист	№ докум.	Подпись	Дата		

ВВЕДЕНИЕ

Сегодня персональные компьютеры всё больше и больше внедряются в жизнь человека. Компьютеры в наше время используются практически во всех областях жизнедеятельности: начиная от личных потребностей, заканчивая применением в научных областях и медицине.

Не так давно об Интернете никто не слышал, а локальная сеть считалась верхом компьютерных технологий. Хотя передовые методики разработки ПО в последние годы быстро развивались, общепринятые практически стояли на месте [1]. С развитием Интернета всё более остро вставал вопрос необходимости организации распределенных вычислений, так как нужно было соединить разработки различных компаний, которые появились за годы компьютерной революции.

В наше время большую популярность снискали клиент-серверные приложения, так как большинство операций на вычислительных машинах производится с помощью интернета. Сегодня сложно найти компьютер, не подключенный к глобальной паутине или по крайней мере, не являющийся частью локальной сети.

Область применения клиент-серверных приложений распространяется не только на интернет, множество применений таких приложений приходится на локальную сеть. К таким приложениям можно отнести различные средства администрирования, сканирования сети, программы для работы с локальными серверами, а также локальные чаты. Большую часть таких программ составляют средства для работы внутри закрытой локальной сети, так как это дает наибольшую безопасность для пользователей и данных внутри такой сети. Также зачастую локальная сеть используется для организации высокопроизводительной вычислительной системы, где множество узлов соединены между собой и задачи равномерно распределяются по всем компьютерам сети, для достижения максимальной

					ПГУ 1.09.03.01.07.001 ПЗ	Лист
						7
Изм.	Лист	№ докум.	Подпись	Дата		

производительности. Именно для такой системы будет необходима данная программа.

В данной работе будет разработано клиент-серверное приложение, позволяющее собирать данные с компьютеров в сети, сохранять их в базу данных, а также просматривать полученную информацию в удобном виде.

					ПГУ 1.09.03.01.07.001 ПЗ	Лист
						8
Изм.	Лист	№ докум.	Подпись	Дата		

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Постановка задачи

Требуется разработать клиент-серверное приложение, позволяющее осуществлять сбор статистических данных по загрузке персональных компьютеров в локальной сети.

Основное назначение разработки – проектирование системы с функциями сбора, хранения и представления статистических данных по загрузке компьютеров в локальной сети.

Требуется разработать клиент-серверное приложение, работающее в локальной сети, а именно приложение-клиент, которое собирает данные с компьютеров, приложение-сервер, которое выполняет обработку, хранение и вывод полученных от приложений-клиентов данных. Также необходимо разработать структуру БД для хранения статистических данных.

Прежде всего необходимо разработать механизм сбора статистических данных по загрузке компьютера. Приложение-клиент должно собирать и отправлять информацию на сервер. Приложение-сервер должно получать информацию, обрабатывать и сохранять в базу данных. Необходимо проработать способы вывода информации в удобном для пользователя виде.

Сетевое взаимодействие между клиентом и сервером должно быть построено на основе протокола ТСР.

1.2 Обзор существующих программных решений

Решение поставленной задачи требует изучение способов сбора статистических данных на компьютере, формат их записи, определиться со способом передачи данных между сервером и клиентом, а также с протоколом управления передачи данных. Также необходимо определиться с

					ПГУ 1.09.03.01.07.001 ПЗ	Лист
						9
Изм.	Лист	№ докум.	Подпись	Дата		

выбором решения хранения данных и способом просмотра сохраненной информации на стороне сервера.

Чтобы решить данные задачи было решено изучить текущие программы похожей тематики, для того чтобы посмотреть их технические решения и способы реализации конкретных задач. К сожалению, не удалось найти программы, которая бы отвечала всем нужным требованиям, однако были найдены нужные решения в отдельных программах.

На общую производительность компьютера влияет множество факторов — вычислительная мощность процессора, количество доступной оперативной памяти, скорость дисковой подсистемы, пропускная способность сетевых интерфейсов. Для мониторинга было решено выбрать процентную загрузку систем, а конкретно процессора, оперативной памяти, сети и диска. При решении вопроса, как собрать эти данные, рассматривалось несколько технических решений. Самым оптимальным было использование счетчиков, так называемых Performance Counters, так как встроенных средств Windows Forms было достаточно для их реализации.

При связи клиента и сервера решено было использовать протокол ТСР/ІР, из-за того, что он обеспечивает надежную связь между сетевым оборудованием от различных производителей, а также независим от сетевой технологии. Данный протокол предлагает приложениям сервис надежной и упорядоченной передачи. Все отправленные данные подлежат обязательному подтверждению встречной стороной, причем формируются подтверждения не для каждого конкретного успешно полученного пакета, а для всех данных от начала посылки до некоторого порядкового номера. Протокол ТСР, поддерживающийся практически всеми приложениями Интернета, за прошедшие годы был значительно усовершенствован для обеспечения надежности и производительности в сетях различной емкости и качества.

Для хранения данных наиболее удобным решением было бы использование базы данных, так как это позволило бы записывать, хранить и читать информацию в удобном виде, а также записывать данные большого

					ПГУ 1.09.03.01.07.001 ПЗ	Лист
						10
Изм.	Лист	№ докум.	Подпись	Дата		

объема. Была выбрана встроенная в MS Visual Studio база данных SQL Server. Для работы с ней есть встроенные средства, что упрощает создание новых таблиц и полей прямо внутри среды разработки, что было бы невозможно при использовании сторонней СУБД. Используя структурированный язык SQL можно обращаться к созданной БД, для того чтобы заполнять ее данными, изменять их, а также извлекать информацию для последующего ее использования сервером.

При просмотре похожих программ по данной тематике было установлено, что часто они имеют очень неудобную форму представления информации. Зачастую данные выводились в виде стен текста, что далеко не способствовало читаемости и наглядности информации. Также во многих случаях не было удобной сортировки информации по пользователям и по времени. Для решения этой проблемы было решено для вывода информации использовать графики, с возможностью выбора предмета загрузки. Для удобства необходимо было бы ввести возможность просмотра информации по всем пользователям, а также для каждого пользователя в отдельности. При просмотре графиков необходимо ввести промежуток времени для которого отображались бы данные. Возможность изменять время необходимо было бы ввести вплоть до секунд. Очень полезной функцией было бы вывод информации на графиках в реальном времени, так как это способствовало бы принятию оперативных решений. Для этого предлагается ввести функцию обновления информации из базы данных. Ее можно принудительно привязать к определенному промежутку времени, либо дать пользователю возможность распоряжаться ей самостоятельно.

1.3 Актуальность использования

При использовании локальных сетей и множества компьютеров для обработки больших массивов данных или выполнении трудоемких вычислений существует необходимость мониторинга состояния

					ПГУ 1.09.03.01.07.001 ПЗ	Лист
						11
Изм.	Лист	№ докум.	Подпись	Дата		

компьютеров, для того чтобы администратор сети мог знать какие из элементов локальной сети находятся в наибольшей загрузке, а какие простаивают, для того чтобы была возможность максимально эффективно распределить задачи между компьютерами. Программа должна вычислять не только общую производительность системы, но и конкретно загрузку всех узлов, влияющих на общую производительность компьютера. Такими факторами являются загрузка процессора, загрузка оперативной памяти, состояние дисковой подсистемы, а также загруженность сети и отдельных интерфейсов.

Для удобства пользования такими системами необходимо выводить перечисленные факторы как в реальном времени, так и записывать эти данные, для того, чтобы администратор мог в любое время просмотреть информацию по интересующем его узлам, а также выбрать временной диапазон просмотра данных. Это позволит ему отслеживать ошибки сети незамедлительно, и, что немаловажно, даст возможность проанализировать уже прошедшую ситуацию и сделать выводы на основе полученных данных. Это дает администратору свободу действий и освобождает от постоянной необходимости находиться за рабочим местом.

При выполнении трудоемких вычислений такая система позволит находить так называемое «бутылочное горлышко», т. е. узкое место в локальной сети или выделенных компьютерах, что позволит администратору проанализировать ситуацию и в дальнейшем произвести замену устаревшего оборудования или перераспределить нагрузку сети равномерно: слишком загруженным системам дать меньше задач, а более свободным отдать наиболее трудоемкие задания.

При необходимости данную систему можно использовать как общую на всю доступную локальную сеть, так и распределить ее по сегментам, выделив на каждый сегмент по серверу, для обработки и хранения информации.

2 ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЯ

2.1 Этап проектирования

2.1.1 Проектирование структуры системы

При разработке крупных программных систем принято использовать модульный подход программирования. Его идеей является упрощение системы. Для этого исходную программу разбивают на части, которые называются программными модулями. Каждый модуль программируется, компилируется и отлаживается отдельно от других модулей программы и физически разделен с ними. Также, в некоторых системах программный модуль может использоваться в составе разных программ, тем самым уменьшается количество кода, которое необходимо для разработки систем.

По заданию данная программа является клиент-серверным приложением, отсюда сразу можно выделить два модуля: клиент и сервер. Также для хранения информации предполагается использовать базу данных, а значит для удобства можно выделить еще один модуль – базу данных.

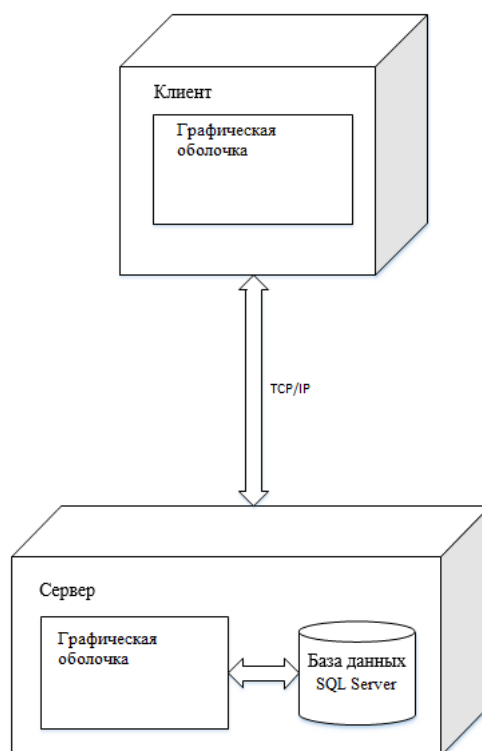


Рисунок 1 - Структура взаимодействия модулей приложения

На рисунке 1 можно увидеть структуру взаимодействия модулей приложения. Клиент взаимодействует с сервером посредством протокола ТСР/ІР. Сервер обрабатывает информацию, а затем сохраняет в базу данных. Параллельно с сохранением сервер имеет возможность загружать данные из базы данных.

2.1.2 Проектирование интерфейса клиентской части

В основную задачу клиентской части программы должен входить сбор информации по загрузке компьютера. Он будет осуществляться автоматически, поскольку клиенту не нужно выбирать какую информацию он будет отсылать на сервер, она будет передаваться в полном объеме. Для подключения к серверу клиенту необходимо знать ІР адрес и номер порта сервера. Для этого необходимо ввести возможность ввода этой информации на стороне клиента.

При подключении и отключении, а также при других вариантах работы будет меняться состояние системы. Для этого необходимо придумать инструмент, который бы оповещал клиента о статусе программы. Лучше всего для таких целей подойдет текстовое поле, в которое будет выводиться текстовая информация. Эта информация будет оповещать клиента о том, что происходит в данный момент.

Для подключения к серверу предлагается ввести кнопку. То же самое можно сделать и для отключения системы. Так пользователь сможет сам контролировать момент начала и конца передачи данных серверу.

2.1.3 Проектирование интерфейса серверной части

В данной системе сервер выполняет основную роль, он принимает информацию от клиента, сохраняет ее в базу данных, параллельно с этим может загружать данные из базы данных и выводить в виде графиков. Для

удобства использования имеет смысл разбить интерфейс серверной части на два больших раздела: прием данных и вывод данных. При этом разделы должны быть самостоятельными и работать независимо друг от друга. Реализовать такое разделение лучше всего с помощью вкладок.

На первой вкладке предлагается сделать вывод информации, ее основой будет являть график, который займет основное пространство на всей области приложения. Над графиком необходимо будет разместить различные элементы управления. Для переключения типа отображаемого графика будет удобно сделать кнопки, каждая будет отвечать за свой тип данных: процессор, оперативная память, диск и сеть. По нажатию на любую из этих кнопок график будет заново отрисовываться и выводить информацию, соответствующую нажатой кнопке. Так как к серверу может подключаться множество разных компьютеров, то было бы неудобно отображать данные всех пользователей одновременно. Для этого можно использовать выпадающий список, в котором будет список всех пользователей. При выборе какого – либо пункта на графике будет отображаться информация только этого пользователя. Также предлагается оставить вкладку «все», для вывода общей статистики обо всех пользователях.

Так как в приложении будет функция вывода данных за определенный период, предлагается для этого ввести два поля выбор даты и времени: один для выбора с какой даты начать отображение, второй для окончания временного периода.

Для того, чтобы пользователь имел контроль над базой данных, было решено ввести кнопку, которая позволяла бы обновить данные из базы данных, тем самым информация, которая отображена на графике меняла бы свое значение. Было решено не вводить функцию автоматического обновления базы данных, через определенный промежуток времени, так как это могло замедлять работу системы при работе с большими массивами данных.

Вторая вкладка будет полностью посвящена приему информации от клиентов. Для того чтобы клиенты могли узнать IP адрес сервера предлагается его выводить в интерфейсе сервера. Также для гибкости приложения на стороне сервера нужно сделать возможность выбора порта для подключения. Чтобы выводить информацию о состоянии подключений и состоянии сервера предлагается сделать текстовое поле, в котором была бы возможность вывода всей необходимой информации. Для управления состоянием сервера предлагается сделать две кнопки: для запуска режима прослушивания и для его остановки.

При подключении других компьютеров к серверу можно отображать их в удобном виде. Для этого лучше всего подойдет ListBox. Список подключенных пользователей будет отображаться там.

2.1.4 Проектирование структуры базы данных

Структура базы данных напрямую будет зависеть от данных, которые там будут храниться. Предлагается использовать одну таблицу, так как между данными нет сложных связей и их набор не так велик. Первым полем, как и в любой таблице предлагается использовать ID. Он будет полезен при сортировке и поиске записей. Затем необходимо использовать поле имени, в нем будет записываться имя компьютера, его формат лучше сделать текстовым. Для того чтобы сортировать данные по времени необходимо ввести поле даты. Подходящим форматом данных будет datetime. Далее идут данные по загрузке системы, так как эти данные могут иметь значение с плавающей точкой, то для их хранения лучше использовать float. Сюда можно включить загрузку процессора, загрузку оперативной памяти, процент используемых дисков данных и процент занятости сети. Собрав все вышесказанное получим следующие данные:

- ID INT,
- Name TEXT ,

- Date DATETIME,
- Proc_usage FLOAT,
- Mem_usage FLOAT,
- Net_usage FLOAT,
- Disk_usage FLOAT.

Также для поля ID следует включить автоинкрементацию значения, для того чтобы самому не записывать значение этого поля, а также чтобы не было пустых значений.

2.2 Этап разработки программы

2.2.1 Выбор средств разработки программы

Перед началом выполнения процесса разработки программы стояла задача выбора языка программирования, на котором бы писался основной код приложения. Также было важно выбрать среду разработки программы, так чтобы она наиболее хорошо подходила к основным требованиям. Помимо этого, важно было определиться, где будет создаваться база данных и какие средства нужно будет использовать для ее обслуживания.

В качестве языка программирования был выбран C#. Он относится к семье языков с C-подобным синтаксисом, из них его синтаксис наиболее близок к C++ и Java. Язык имеет статическую типизацию, поддерживает полиморфизм, перегрузку операторов (в том числе операторов явного и неявного приведения типа), делегаты, атрибуты, события, свойства, обобщённые типы и методы, итераторы, анонимные функции с поддержкой замыканий, LINQ, исключения, комментарии в формате XML [2]. Выбор языка C# был сделан не зря, так как он обладает рядом преимуществ:

- язык программирования C# претендует на подлинную объектную ориентированность (всякая языковая сущность претендует на то, чтобы быть объектом);

- компонентно-ориентированный подход к программированию, способствующий меньшей машинно-архитектурной зависимости результирующего программного кода, гибкости, переносимости и легкости повторного использования (фрагментов) программ;
- ориентация на безопасность кода (в сравнении с С и С++);
- унифицированная система типизации;
- расширенная поддержка событийно-ориентированного программирования.

Для разработки на языке С# было удобнее всего использовать MS Visual Studio, из-за того, что эта студия имеет нативную поддержку языка. Кроме того, MS Visual Studio включает в себя редактор исходного кода с поддержкой технологии IntelliSense и возможностью простейшего рефакторинга кода. Встроенный отладчик может работать как отладчик уровня исходного кода, так и отладчик машинного уровня. Остальные встраиваемые инструменты включают в себя редактор форм для упрощения создания графического интерфейса приложения, дизайнер классов и дизайнер схемы базы данных. MS Visual Studio позволяет создавать и подключать сторонние дополнения (плагины) для расширения функциональности практически на каждом уровне, включая добавление поддержки систем контроля версий исходного кода (как, например, Subversion и Visual SourceSafe), добавление новых наборов инструментов (например, для редактирования и визуального проектирования кода на предметно-ориентированных языках программирования) или инструментов для прочих аспектов процесса разработки программного обеспечения. Важным фактором в выборе также является то, что при использовании Visual Studio есть возможность использовать Windows Forms. Это интерфейс программирования приложений, отвечающий за графический интерфейс пользователя и являющийся частью Microsoft .NET Framework. Данный

интерфейс упрощает доступ к элементам интерфейса Microsoft Windows за счет создания обёртки для существующего Win32 API в управляемом коде [4]. Причём управляемый код — классы, реализующие API для Windows Forms, не зависят от языка разработки. Это дает гибкость приложениям и возможность использовать его в различных сферах.

В качестве систему управления базой данных было решено использовать MS SQL Server. Данная система имеет встроенную поддержку MS Visual Studio, что позволяет без посторонних средств управлять базами данных. SQL Server - это хорошо масштабируемый, полностью реляционный, быстродействующий многопользовательский сервер баз данных масштаба предприятия, способный обрабатывать большие объемы данных для клиент-серверных приложений [5]. Основными преимуществами являются:

- многопользовательская поддержка;
- многоплатформенность;
- поддержка 64-разрядной архитектуры;
- масштабируемость (многопроцессорная обработка и поддержка терабайтных БД;
- параллельные архивирование и восстановление БД;
- репликация данных;
- распределенные запросы;
- распределенные транзакции;
- динамические блокировки.

2.2.2 Описание разработки клиентского приложения

При проектировании приложения используется Windows Form, поэтому для работы с ней создается класс Form1, который наследуется от класса Form, это необходимо для того, чтобы можно было работать с элементами, находящимися в графическом интерфейсе. Внутри этого класса

и будут происходить все необходимые задачи, нужные для работы клиента. В процессе работы клиент будет выполнять два основных процесса, это: передача информации на сервер и прием данных от сервера. Чтобы разделить эти функции создаем два метода: SendMessage и ReceiveMessage.

Прежде чем передавать или принимать информацию необходимо подключиться к серверу. Это происходит в методе MainProgram. Сначала мы производим подключение к серверу с помощью функции Connect. Аргументами этой функции являются IP адрес и порт. В интерфейсе данной программы реализована функции самостоятельного задания необходимого адреса и порта, поэтому мы считываем необходимые данные из интерфейса и передаем их в функцию Connect. После этого мы запускаем методы ReceiveMessage и SendMessage в отдельных потоках. Они работают, параллельно считывая и отправляю информацию.

Метод SendMessage собирает данные о загрузке системы, а затем отправляет их на сервер. Для получения информации о текущих загрузках используются Perfomance counters. Это удобное средство, которое позволяет узнать практически любую информацию о системе, также существует возможность создавать собственные счетчики производительности для специализированных задач. Собранная информация переводится в двоичный код, а затем передается в поток. Метод разработан так, чтобы в бесконечном цикле каждую секунду он срабатывал снова, то есть заново собирал данные и отправлял на сервер.

ReceiveMessage работает до тех пор, пока подключение не будет прервано. Внутри него считывается двоичный поток и записывается в буфер, размером 64 байта, затем информация из буфера переводится в формат String. Основной функцией данного метода является то, что он принимает сигналы от сервера, тем самым может остановить отправку данных от клиента, если сервер прекратит свою работу.

Также стоит упомянуть про метод Disconnect. Он отключает поток передачи данных, отключает клиента, а также останавливает потоки

программы клиента. Данный метод используется при завершении передачи данных и отключения от сервера. Его может вызвать ReceiveMessage после того как получит информацию от сервера, о том, что он остановил прием данных. Также пользователь клиентского приложения может вызвать метод Disconnect принудительно нажав соответствующую кнопку в интерфейсе.

При нажатии пользователем кнопки «подключиться» обработчик событий отрабатывает нажатие и запускает метод MainProgram в новом потоке. При закрытии приложения вызывается Disconnect.

2.2.3 Описание разработки серверного приложения

Также, как и в клиентской части сервер использует интерфейс Windows Forms, из-за этого основной класс, который содержит все методы и вложенные классы наследуется от класса Form. Все методы можно разделить на две части, в первую включить те, которые используются для организации сервера, т. е. операции подключения новых клиентов, отключения клиентов, передачи и приема данных, отображения подключенных пользователей. Во вторую категорию необходимо включить методы, которые отвечают за вывод данных, отбора информации, обновления базы данных и т. д.

Рассмотрим первую группу. Туда входят два вложенных класса ClientObject и ServerObject. Первый отвечает за операции, которые происходят с клиентами на сервере, а второй включает все задачи, выполняемые сервером в процессе работы.

Класс ClientObject имеет несколько методов. GetMessage получает данные, которые передаются на сервер и возвращает их в виде строкового значения. Метод Process выполняет несколько функций, во-первых, он получает от клиента информацию об удачном подключении, а также имя подключенного компьютера, во-вторых, он запускает бесконечный цикл, который принимает данные по загрузке компьютера от клиента и сразу записывает их в базу данных. Если клиент отключается, то данный метод

прерывает свой цикл и выдает соответствующее сообщение, а также совершает необходимые действия по отключению клиента на сервере. Последний метод Close используется для закрытия потока передачи и клиента.

Класс ServerObject предназначен для управления списком клиентов и обслуживания подключений. Переменная clients является списком и служит для хранения объектов подключенных клиентов. Метод AddConnection добавляет нового клиента в список. Метод RemoveConnection удаляет клиента из списка по его ID. Метод Listen создает листенера, который прослушивает подключения по указанному порту и, когда клиент подключается, то для него создаётся объект класса ClientObject и затем в новом потоке запускается метод Process. Метод класса BroadcastMessage позволяет отправлять сообщения всем подключенным клиентам. Disconnect останавливает листенер, и отключает всех клиентов, тем самым останавливая работу сервера.

Стоит упомянуть про структуру DBInformation, которая дублирует поля базы данных. Она необходима для промежуточного хранения данных и используется при выводе информации на экран. Структура содержит следующие поля:

- String Name;
- DateTime Date;
- Double Proc_usage;
- Double Mem_usage;
- Double Net_usage;
- Double Disk_usage;

Каждое поле соответствует такому же в базе данных.

Теперь рассмотрим группу, содержащую методы, отвечающие за вывод информации. GetNameFromBD на вход получает структуру, затем отбирает уникальные значения в поле структуры Name и записывает их в список. Возвращаемым значением является данный список. ReadDB, метод, который

считывает всю находящуюся информацию из базы данных и записывает ее в структуру. Для этого он использует SqlConnection и с помощью стандартных sql команд делает запрос в базу данных, на выходе получая необходимые значения. Метод DrawChart отвечает за отрисовку графика Chart1 в пользовательском интерфейсе, учитывает выбранные флаги, что позволяет рисовать графики по различным данным, в зависимости от выбранных параметров. Метод writeAndDraw в новом потоке запускает ReadDB, а затем обновляет все данные на интерфейсе в соответствии с информацией, полученной из базы данных. Сюда включается выпадающий список с подключенными компьютерами, обновление текущего времени в блоке dateTimePicker, а также заново рисуется график. В интерфейсе существует 4 кнопки, каждая из которых отвечает за выбор данных для рисовки графика: процессор, оперативная память, сеть, диск. При нажатии на любую из этих кнопок срабатывает обработчик событий и запускает один из методов. В каждом таком методе запускается функция DrawChart, при этом сначала изменяется значение переменной trigger. Она отвечает за то, какой тип данных будет выводиться на графике. При изменении даты в блоках dateTimePicker или при выборе другого пункта в ComboBox, запускается метод DrawChart, тем самым обновляя график. Если нажать на кнопку «обновить БД», то вызовется метод writeAndDraw и данные внутри структуры обновятся, тем самым обновится и график.

3 ОТЛАДКА И ТЕСТИРОВАНИЕ

3.1 Руководство пользователя

Данная система предназначена для сбора статистических данных по загрузке компьютеров, находящихся в локальной сети. Она включает в себя приложение-клиент и приложение-сервер. Приложение клиент собирает информацию с текущего компьютера по загрузке процессора, оперативной памяти, сети, диска, а затем отправляет эти данные на сервер. Приложение-сервер принимает данные от подключенных к нему клиентов и записывает их в базу данных, а также имеет возможность просматривать записанные данные в виде графиков, структурируя их по пользователям и по времени.

Прежде чем приступить к работе необходимо установить на используемые компьютеры приложение-клиент, а также установить приложение-сервер на компьютер, который будет собирать данные. Далее следует убедиться, что все пользовательские компьютеры подключены по локальной сети к главному компьютеру.

Запускаем серверную программу на основном компьютере. Изначально будет открыта вкладка «Вывод данных», здесь мы увидим элементы управления вывода данных: кнопки, выпадающий список, поля выбора времени. Также на этой вкладке отображается графика вывода данных. Для того чтобы использовать серверные функции необходимо перейти на другую вкладку, чтобы это сделать выберем «Получение данных» (см. рисунок 2). После этого откроется новая вкладка (см. рисунок 3).

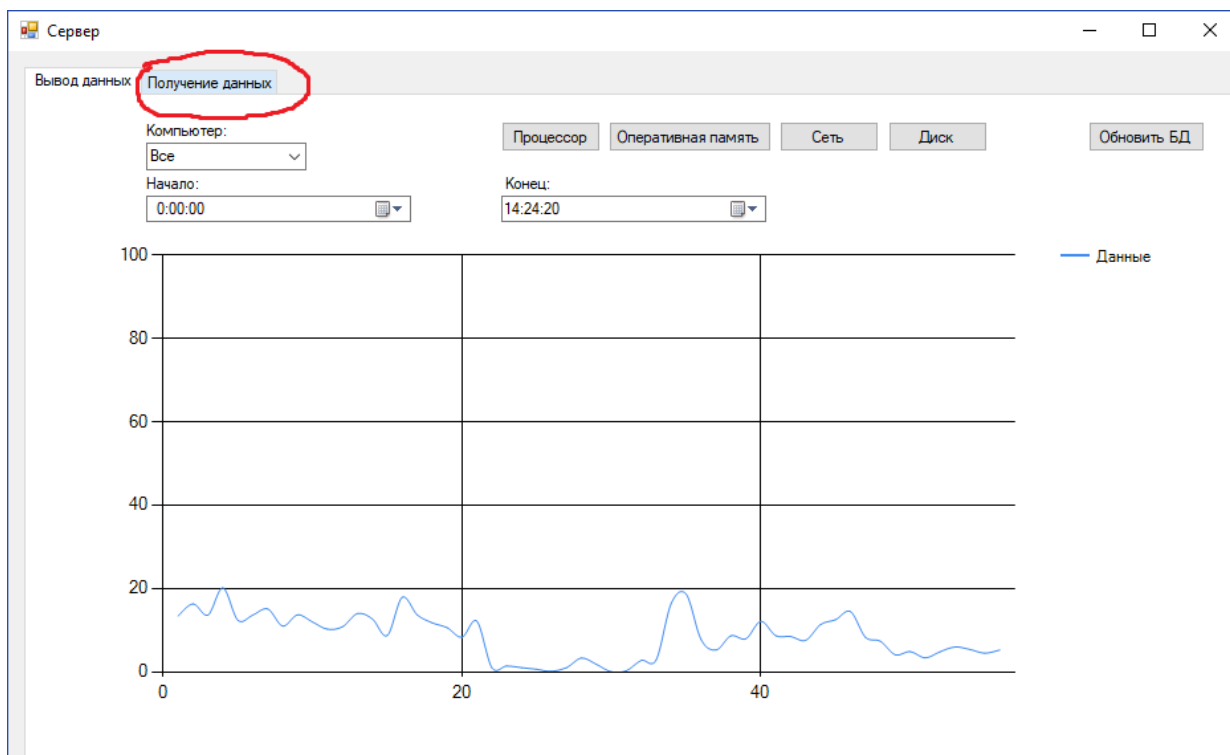


Рисунок 2 – Выбор вкладки «Получение данных»

Рисунок 3 – Окно получения данных

Здесь мы видим текущий IP-адрес сервера, также есть поле ввода порта, его можно поменять. После того как все предварительные действия завершены, можно запускать сервер, для этого нажимаем кнопку «Начать». Надпись говорит о том, что сервер успешно запущен (см. рисунок 4).

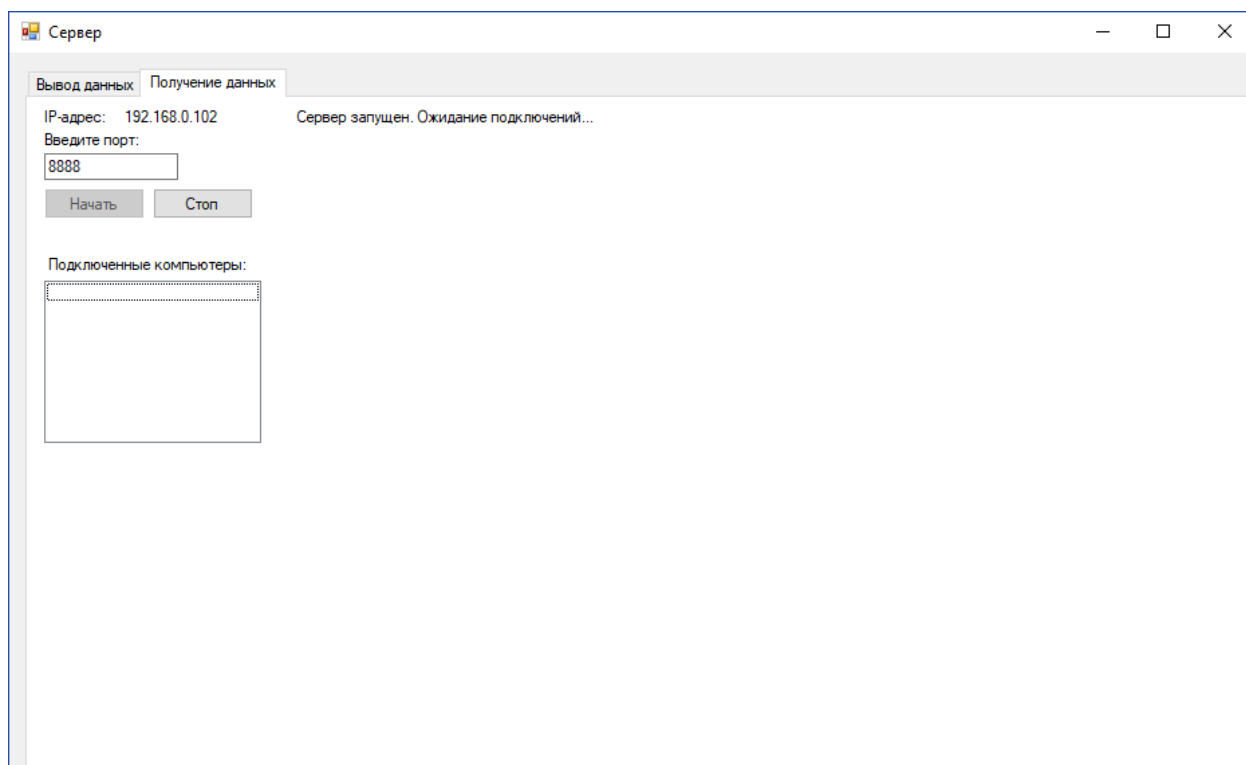


Рисунок 4 – Запуск сервера

Теперь перейдем к клиент-приложению. Его интерфейс довольно прост, так как для большинства действий, которые происходят в данном приложении не нужно участие клиента. Ему будет необходимо только вводить данные для подключения, а также производить соединение с сервером или прерывает его в случае необходимости. На рисунке 5 показано, что увидят клиенты после запуска программы.

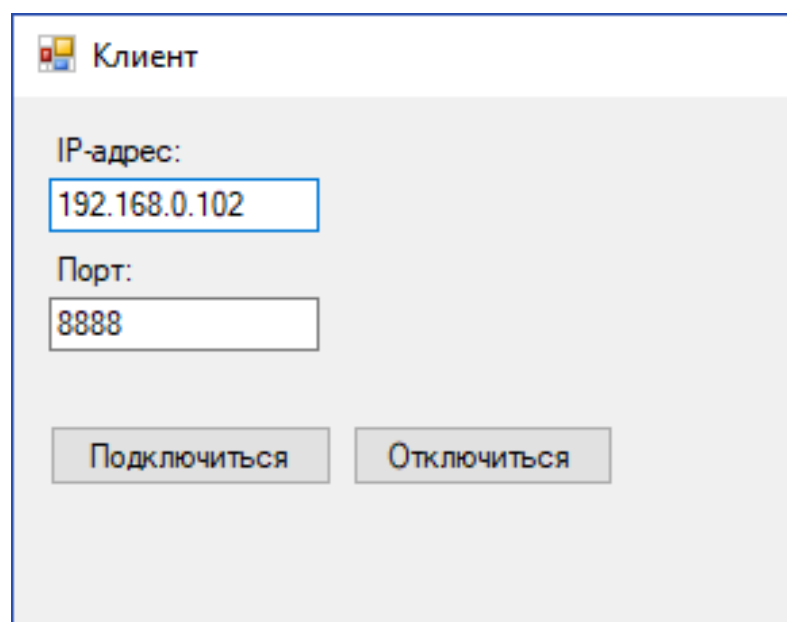


Рисунок 5 – Приложение-клиент

Для подключения, клиентам необходимо ввести IP-адрес сервера, а также порт в соответствующие поля. После этого нажимаем кнопку «Подключиться». При удачном подключении программа выведет соответствующее сообщение (см. рисунок 6).

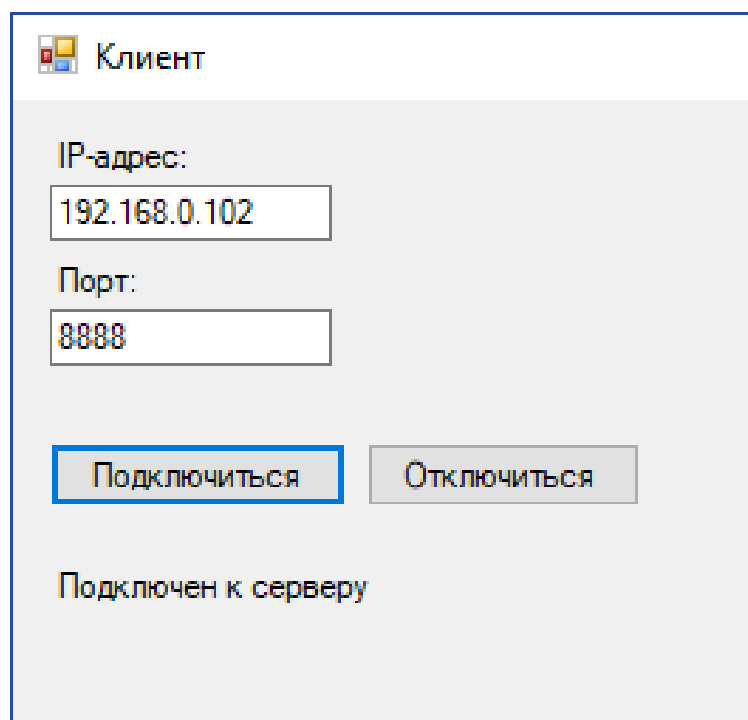


Рисунок 6 – Подключение клиента к серверу

Перейдем к программе-серверу, при подключении клиентов на нем выводятся сообщения, а также заполняется окно подключенных клиентов (см. рисунок 7).

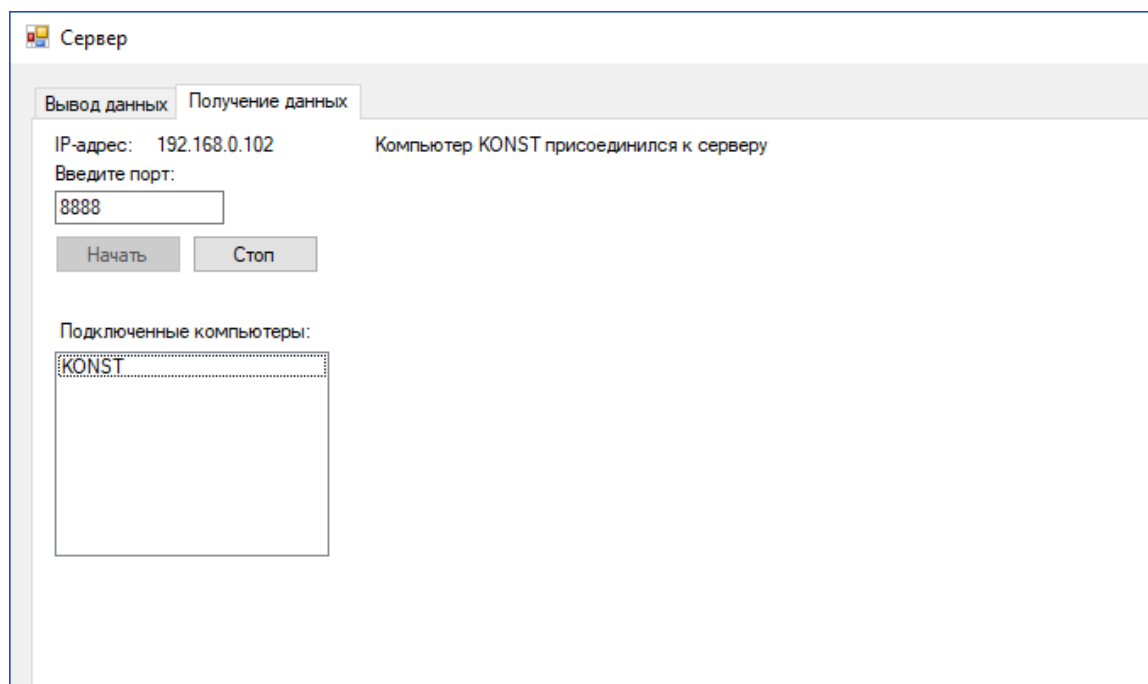


Рисунок 7 – Вывод сообщения о подключении клиента

После окончания работы клиент может нажать кнопку «Отключиться» (см. рисунок 8).

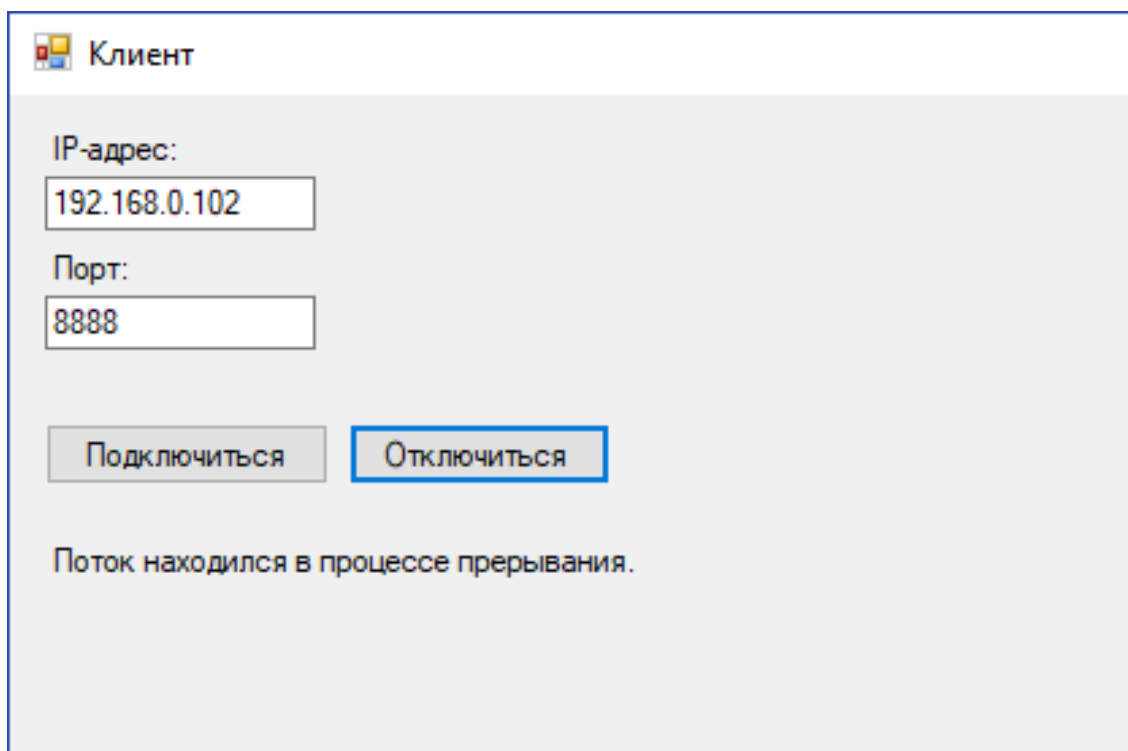


Рисунок 8 – Отключение клиента от сервера

Сообщение об этом событии появится на сервере (см. рисунок 9).:

Сервер

Вывод данных Получение данных

IP-адрес: 192.168.0.102 Компьютер KONST: отключился от сервера

Введите порт:
8888

Начать Стоп

Подключенные компьютеры:

Рисунок 9 – Вывод сообщения об отключении клиента

Теперь перейдем к рассмотрению режима просмотра данных. Для того, чтобы начать просмотр выберем соответствующую вкладку (см. рисунок 10).

Сервер

Вывод данных Получение данных

IP-адрес: 192.168.0.102 Компьютер KONST: отключился от сервера

Введите порт:
8888

Начать Стоп

Подключенные компьютеры:

Рисунок 10 – Выбор вкладки «Вывод данных»

Перед нами появится график и элементы управления (см. рисунок 11).

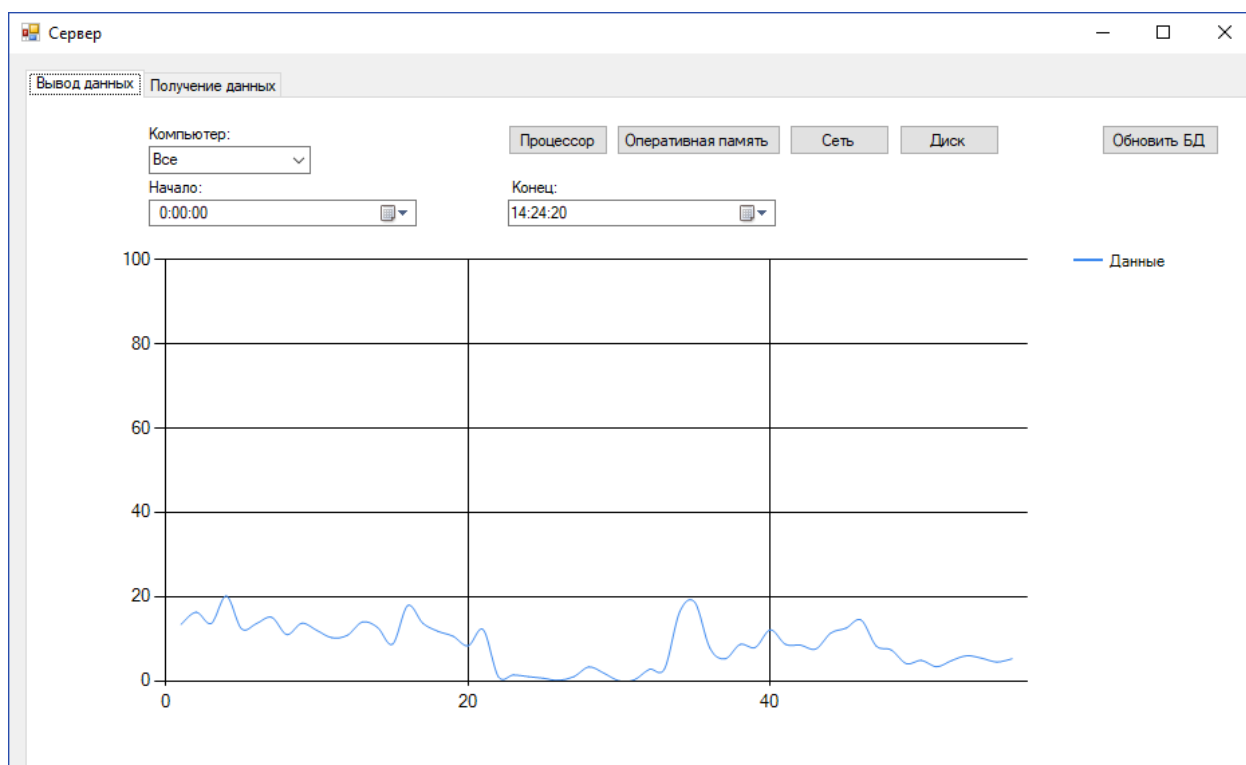


Рисунок 11 – Окно вывода данных

Нажимая на кнопки: «Процессор», «Оперативная память», «Сеть» и «Диск», мы будем менять данные, которые отображаются на графике. На рисунке 12 показано отображение графика оперативной памяти, а на рисунке 13 выводится график загрузки диска.

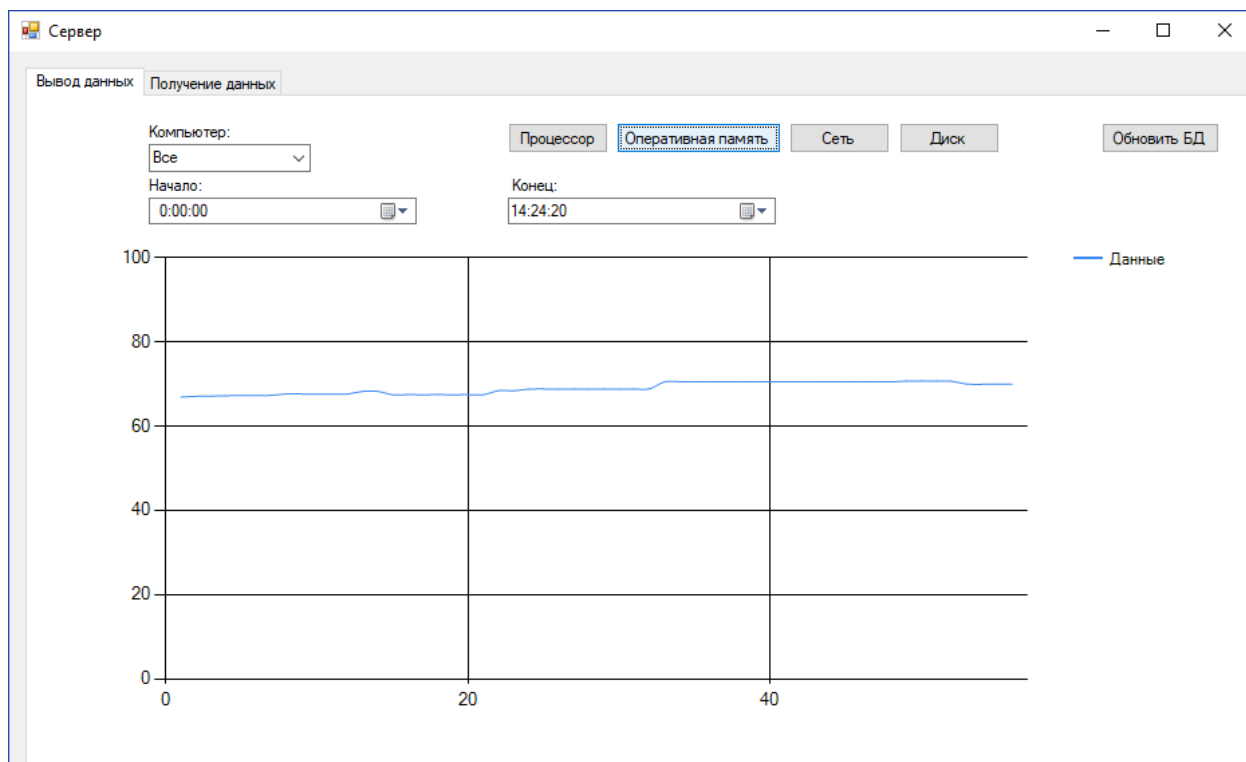


Рисунок 12 – График загрузки оперативной памяти

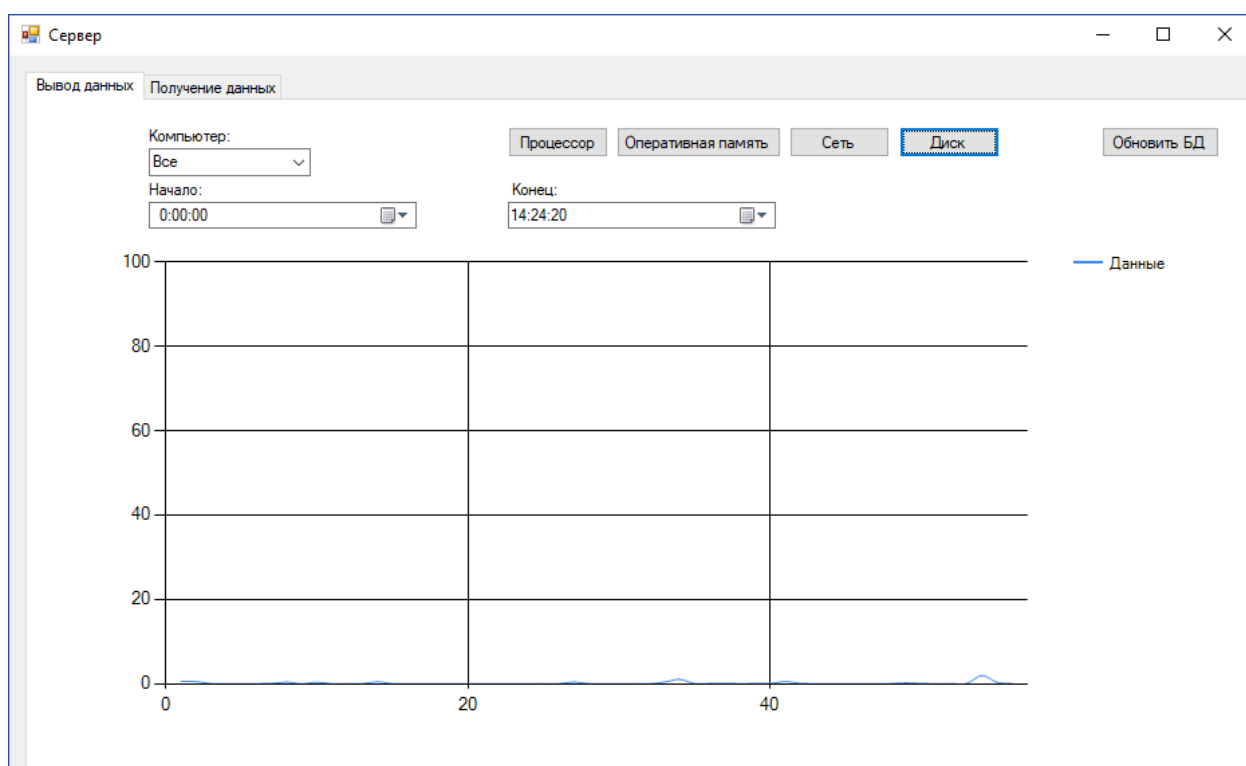


Рисунок 13 – График загрузки Диска

В программе есть возможность выбирать значения какого компьютера выводить на график, для этого в выпадающем списке нужно выбрать интересующий компьютер (см. рисунок 14).

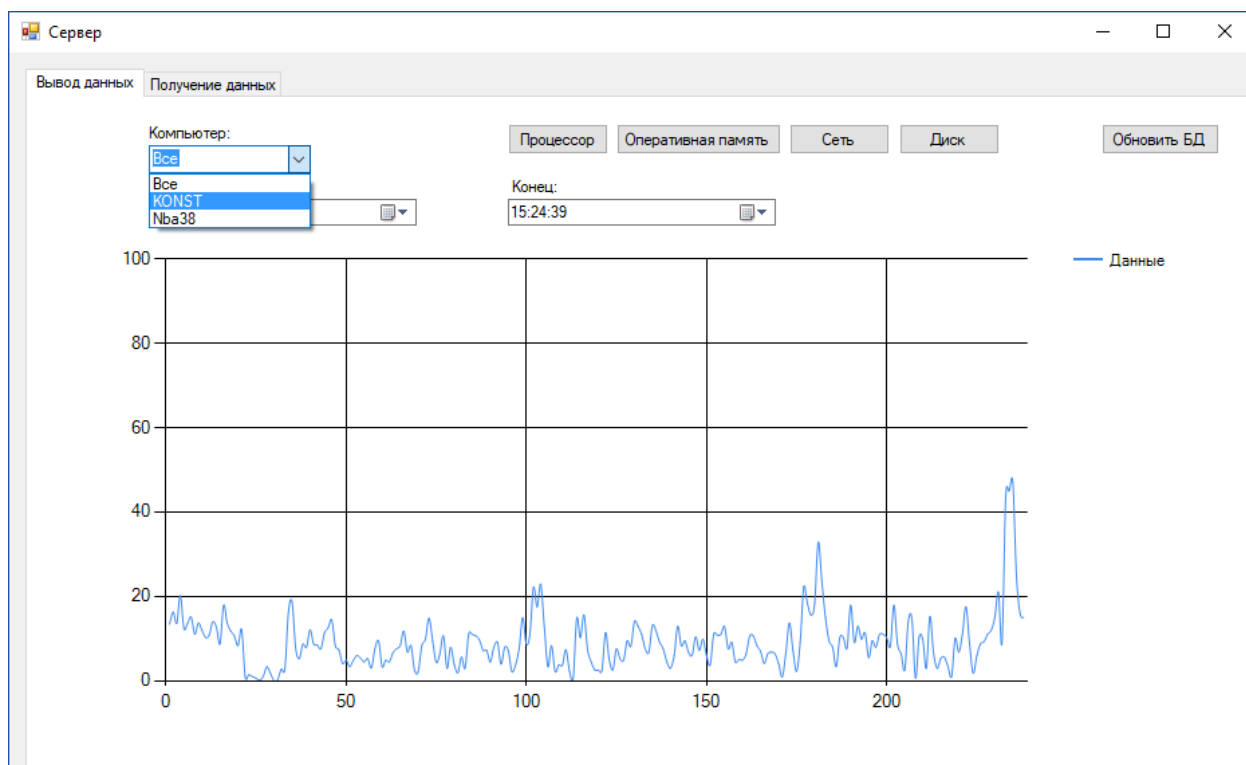


Рисунок 14 – Выбор компьютера

При выборе разных пользователей данные меняются (см. рисунок 15).

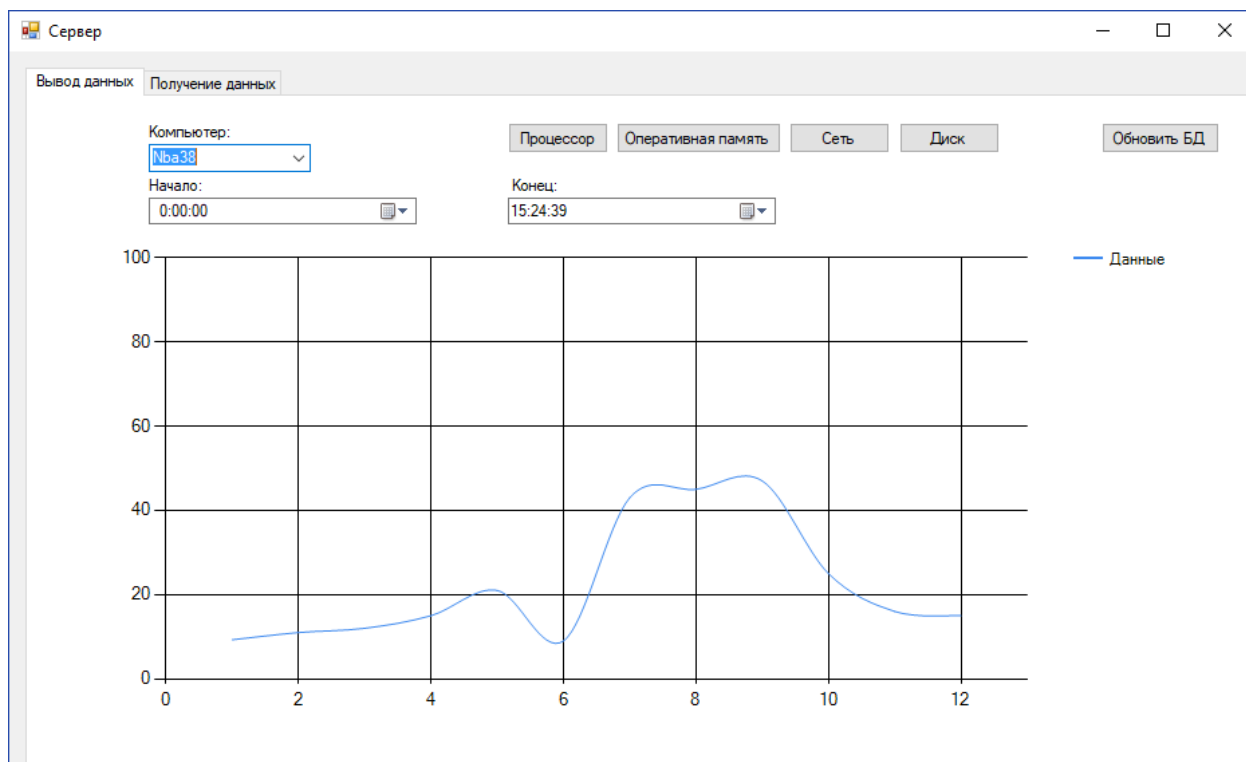


Рисунок 15 – Данные для «Nba38»

Для отображаемых данных можно менять временной промежуток, выбирать дату и время начала и конца отображения данных (см. рисунок 16).

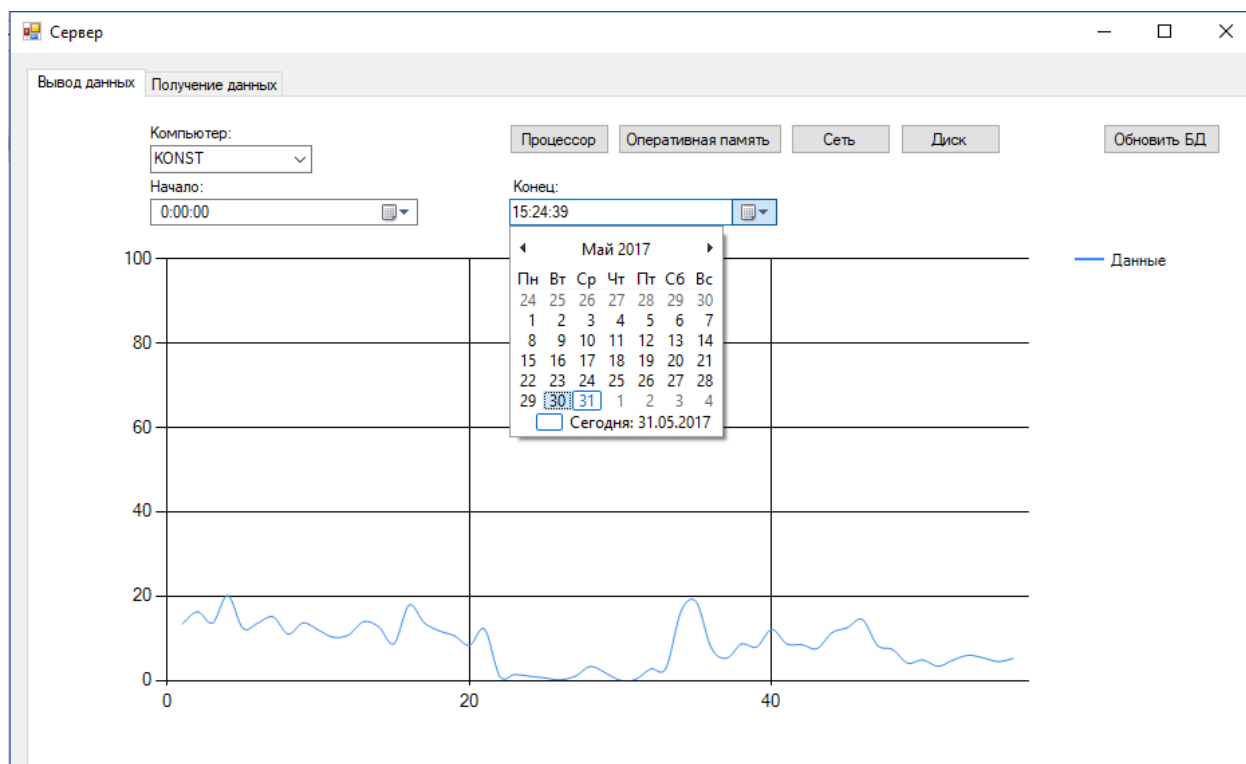


Рисунок 16 – Выбор даты

С помощью кнопки «Обновить БД» пользователь может обновлять данные, т. е. брать всю новую информацию, которая могла появиться в базе данных (см. рисунок 17).

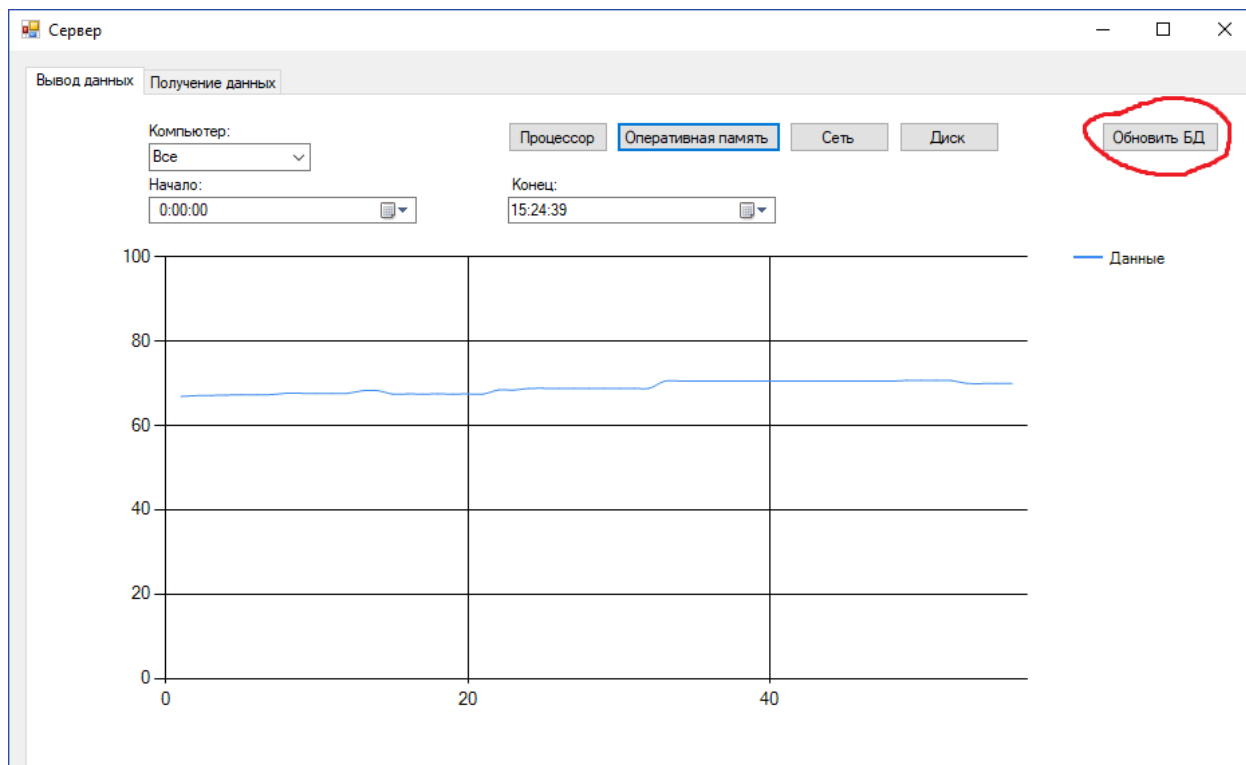


Рисунок 17 – Обновление базы данных

3.2 Отладка и тестирование

Для отладки приложения использовался встроенный в MS Visual Studio отладчик. Тестирование проводилось на нескольких компьютерах, подключенных между собой по локальной сети.

Сначала проверялось наличие подключения, для этого на разных компьютерах запускались приложение-клиент и приложение-сервер. Соединение было удачное, никаких ошибок не найдено (см. рисунок 18).

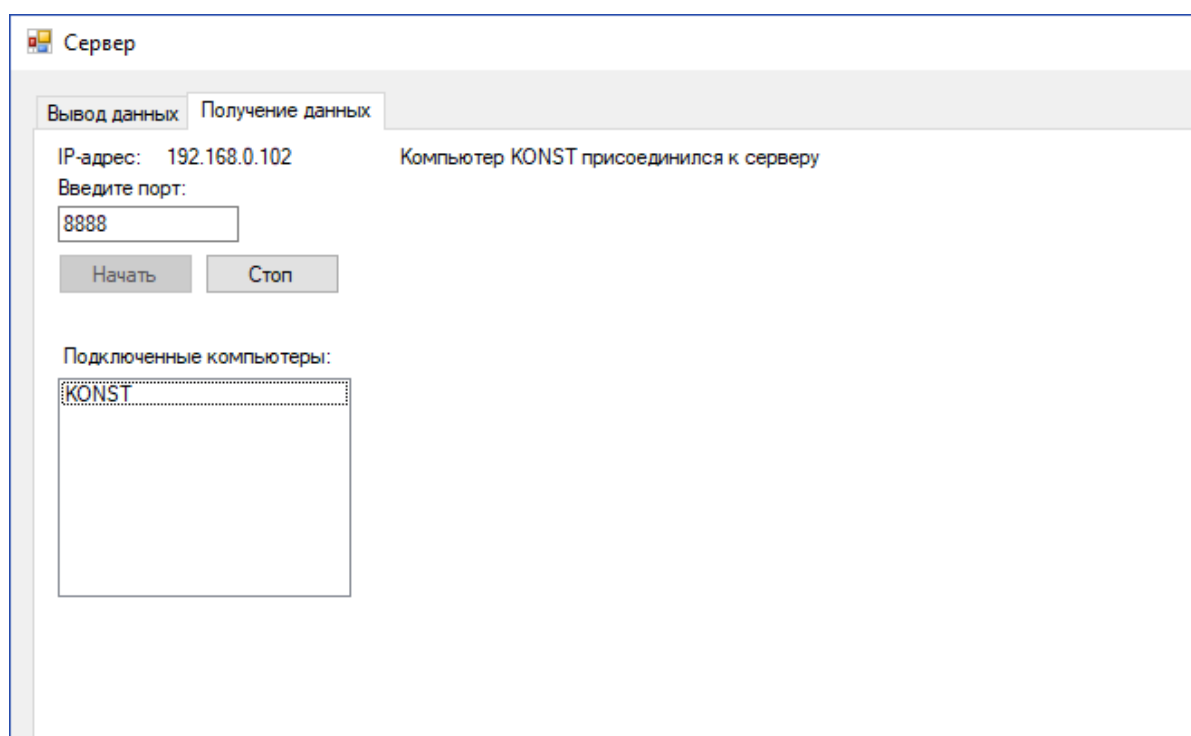


Рисунок 18 – Проверка соединения

Затем тестировался факт передачи данных, а также результат записи их в базу данных. Информация передавалась, а также удачно записывалась в базу данных. Это мы можем увидеть на рисунке 19.

	Id	Name	Date	Proc_usage	Mem_usage	Net_usage	Disk_usage
	912	KONST	31.05.2017 16:13:12	4,098093032836...	62,79148101806...	2,111395929205...	0,041432078927...
	913	KONST	31.05.2017 16:13:13	2,157604932785...	62,76359176635...	0	0,019340720027...
	914	KONST	31.05.2017 16:13:14	5,633338928222...	62,794189453125	0	0,019336109980...
	915	KONST	31.05.2017 16:13:15	3,332007884979...	62,6083984375	2,111404029392...	0,035126999020...
	916	KONST	31.05.2017 16:13:16	3,330621004104...	62,64068984985...	0	0,018924359232...
	917	KONST	31.05.2017 16:13:17	1,657516002655...	62,63261032104...	0	0,019252240657...
	918	KONST	31.05.2017 16:13:18	3,315848112106...	62,64469909667...	2,111700041496...	0,019349109381...
	919	KONST	31.05.2017 16:13:19	6,783936977386...	62,67399978637...	0	0,385642886161...
	920	KONST	31.05.2017 16:13:20	4,481715202331...	62,67890930175...	5,892317034295...	0,043759498745...
	921	KONST	31.05.2017 16:13:21	6,762832164764...	62,62355041503...	2,111949015670...	0,019075330346...
	922	KONST	31.05.2017 16:13:22	1,028687953948...	62,63174819946...	0	0,041333369910...
	923	KONST	31.05.2017 16:13:23	1,267089009284...	62,63312149047...	0	0,018951360136...
	924	KONST	31.05.2017 16:13:24	0,645795285701...	61,79066085815...	2,111489010303...	0,325554013252...
	925	KONST	31.05.2017 16:13:25	4,099081993103...	61,79296875	0	0,024157820269...
	926	KONST	31.05.2017 16:13:26	4,482182979583...	61,79719161987...	0	0,778917074203...
	927	KONST	31.05.2017 16:13:27	2,309420108795...	61,25494003295...	2,108595964500...	0,018996870145...
	928	KONST	31.05.2017 16:13:28	1,267678022384...	61,26887130737...	4,923268988932...	0,019070120528...
	929	KONST	31.05.2017 16:13:29	1,276203989982...	61,26815032958...	9,747714102559...	0,018920330330...
	930	KONST	31.05.2017 16:13:30	0	61,26298904418...	3,694714905577...	0,045399870723...
	931	KONST	31.05.2017 16:13:31	1,412214994430...	61,25819015502...	0	0,019012330099...
	932	KONST	31.05.2017 16:13:32	4,077500820159...	61,20792007446...	1,001869023298...	0,041440971195...
	933	KONST	31.05.2017 16:13:33	7,550129890441...	61,24953079223...	2,111231935941...	0,019051579758...
	934	KONST	31.05.2017 16:13:34	2,179627895355...	61,28419876098...	0	0,020239660516...
	935	KONST	31.05.2017 16:13:35	3,563899993896...	61,28229141235...	5,854786877534...	0,339548498392...
	936	KONST	31.05.2017 16:13:36	6,784092903137...	61,27861022949...	2,111386976366...	0,019129900261...
	937	KONST	31.05.2017 16:13:37	2,563298940658...	61,28499984741...	0	0,019149759784...
	938	KONST	31.05.2017 16:13:38	4,098497867584...	61,27947998046...	0	0,262457698583...
	939	KONST	31.05.2017 16:13:39	3,715029001235...	61,30635833740...	3,299556112779...	0,038908030837...
	940	KONST	31.05.2017 16:13:40	2,947608947753...	61,29000854492...	0	0,042510818690...

Рисунок 19 – База данных

Для надежности системы необходимо было протестировать аварийные случаи завершения программы. Были выбраны разные сценарии: отключение клиента, отключение сервера, закрытие программы-сервера, закрытие программы-клиента, а также снятия задач программы сервера и клиента с помощью диспетчера задач. Во всех этих случаях прерывание соединения отрабатывалось и в программах выводились соответствующие сообщения (см. рисунок 20 и рисунок 21).

Сервер

Вывод данных Получение данных

IP-адрес: 192.168.0.102 Компьютер KONST: отключился от сервера

Введите порт:

8888

Начать Стоп

Подключенные компьютеры:

Рисунок 20 – Отключение компьютера от сервера

Клиент

IP-адрес:

192.168.0.102

Порт:

8888

Подключиться Отключиться

Подключение прервано!

Рисунок 21 – Соединение прервано

Также был протестирован случай, когда клиент подключался к несуществующему IP-адресу или порту. В этом случае выводилось сообщение о неудаче (см. рисунок 22 и рисунок 23).

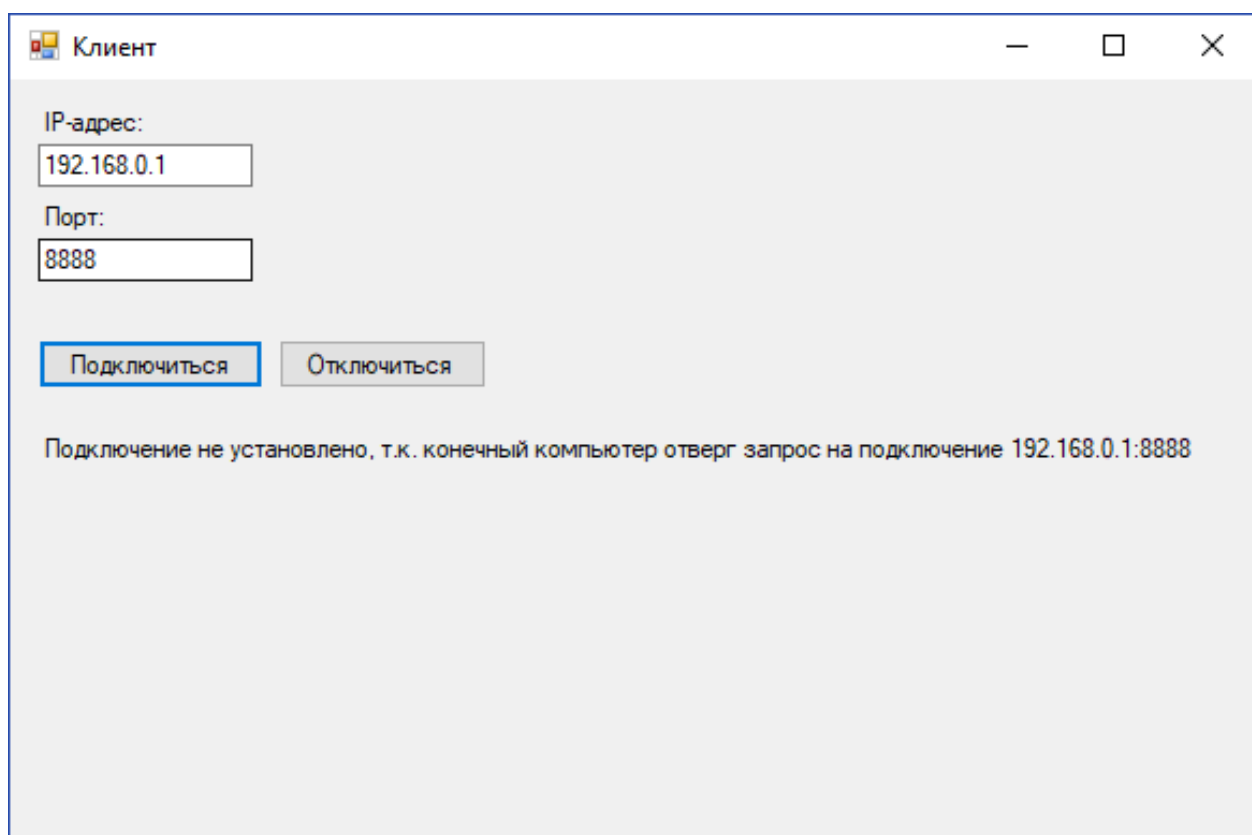


Рисунок 22 – Подключение к несуществующему IP-адресу

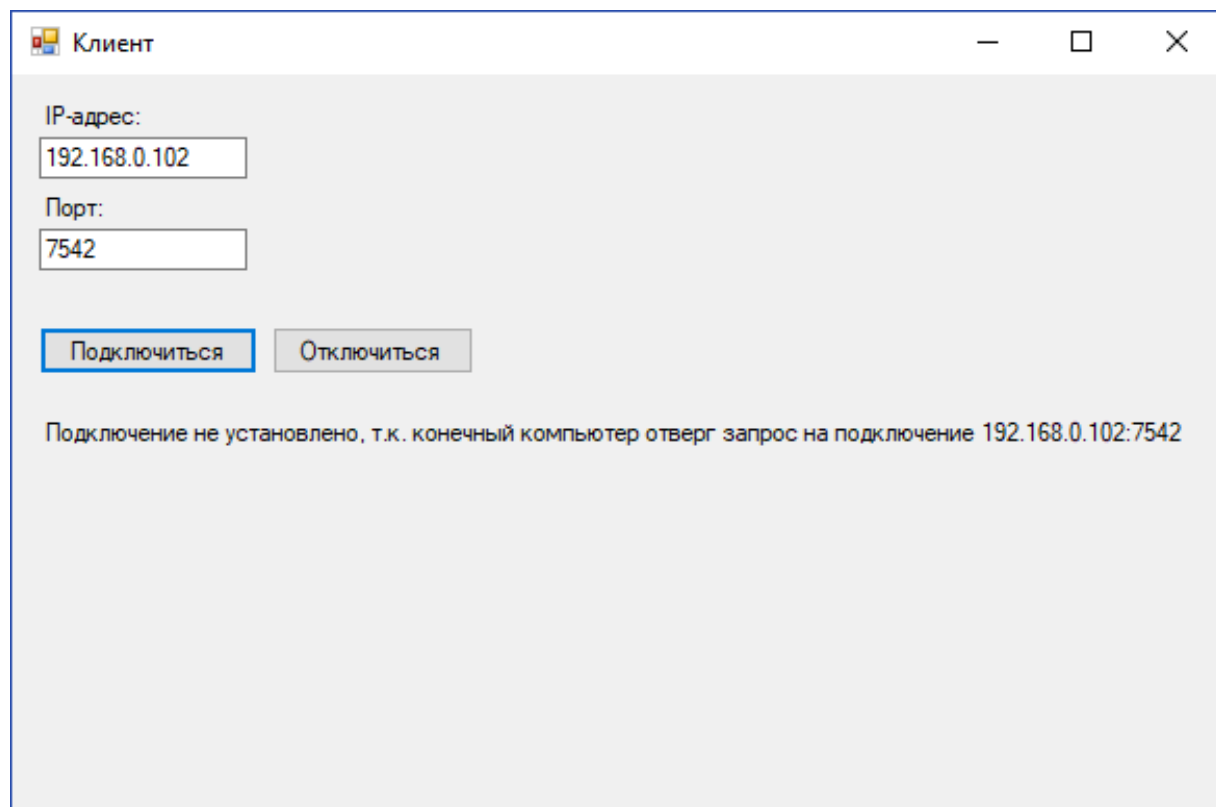


Рисунок 23 – Подключение к неправильному порту

ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы было разработано клиент-серверное приложение. Во время разработки был создан алгоритм решения задачи, составлена и отлажена программа, оформлена документация на программу. Проведенные испытания показали, что программа работает правильно.

Проектирование приложения показало, что, развивая приложение в будущем, можно привнести в него дополнительные функции. К ним можно отнести: добавление новых ресурсов, о которых будет собираться информация, введение возможности подключать различные базы данных, функция удаленного подключения к базе данных, хранение информации в облаке, передача клиенту прав выбирать, какую информацию отсылать на сервер.

За время работы выпускной квалификационной работы был значительно повышен уровень практических навыков по созданию клиент-серверных приложений, освоен процесс разработки баз данных для приложения. Были рассмотрены средства разработки: MS Visual Studio, MS SQL Server; изучены: языки C#, Transact-SQL; был разобран интерфейс программирования приложений Windows Forms.

					ПГУ 1.09.03.01.07.001 ПЗ	Лист
						40
Изм.	Лист	№ докум.	Подпись	Дата		

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Макконнелл С. Совершенный код. Мастер-класс / Пер. с англ. — М. : Издательство «Русская редакция», 2010. — 896 стр.
2. Wikipedia [Электронный ресурс]. — Электрон.дан. — URL: https://ru.wikipedia.org/wiki/C_Sharp
3. Wikipedia [Электронный ресурс]. — Электрон.дан. — URL: https://ru.wikipedia.org/wiki/Microsoft_Visual_Studio
4. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C#, Джеффри Рихтер. 4-е издание. 2017 г.
5. Learning SQL. Алан Бьюли. 2007г.

ПРИЛОЖЕНИЕ А ЛИСТИНГ

Приложение А.1 Файл «server.cs»

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Collections;
using System.Threading;
using System.Net.Sockets;
using System.IO;
using System.Net;

namespace infoOutput
{
    public struct DBInformation //структура для записи данных из БД
    {
        public String Name;
        public DateTime Date;
        public Double Proc_usage;
        public Double Mem_usage;
        public Double Net_usage;
        public Double Disk_usage;
    };

    public partial class Form1 : Form
    {
        public String sqlConnectionValue =
            @"Data Source = (LocalDB)\MSSQLLocalDB;
AttachDbFilename=C:\Users\Konstantin\Desktop\Диплом\infoOutput\infoOutput\
Database1.mdf;Integrated Security = True";
        static ServerObject server; // сервер
        static Thread listenThread; // поток для прослушивания
```

					ПГУ 1.09.03.01.07.001 ПЗ	Лист
						42
Изм.	Лист	№ докум.	Подпись	Дата		

```

List<DBInformation> DBInfo; // лист структуры
String all = "Все";
short trigger = 0; // переменная для определения текущего графика
public Form1()
{

    InitializeComponent();
    dateTimePicker1.Format = DateTimePickerFormat.Time;
    dateTimePicker2.Format = DateTimePickerFormat.Time;

    String host = System.Net.Dns.GetHostName();
    System.Net.IPAddress ip =
System.Net.Dns.GetHostByName(host).AddressList[0]; // получения ip
компьютера
    label6.Text = ip.ToString();

    writeAndDraw(); // отрисовка charta
}

private HashSet<String> GetNameFromBD(List<DBInformation>
DBInfo) // возвращает имена всех компьютеров
{
    HashSet<String> names = new HashSet<string>();
    names.Add(all);
    foreach (DBInformation element in DBInfo)
    {
        names.Add(element.Name);
    }
    return names;
}

private async Task<List<DBInformation>> ReadDB() // читает БД
{
    SqlConnection sqlConnection = new
SqlConnection(sqlConnectionValue);
    SqlCommand sqlCommand = new SqlCommand("SELECT Name,
Date, Proc_usage, Mem_usage, Net_usage, Disk_usage FROM [Table];",
sqlConnection);
    sqlCommand.Connection.Open();
    SqlDataReader reader = sqlCommand.ExecuteReader();
    List<DBInformation> DBInfo = new List<DBInformation>();
    DBInformation buf = new DBInformation();

```

```

try
{
    while (reader.Read())
    {
        Console.WriteLine(String.Format("{0}, {1}, {2}, {3}, {4},
{5}", reader[0], reader[1], reader[2], reader[3], reader[4], reader[5]));

        buf.Name = (String)reader[0];
        buf.Date = (DateTime)reader[1];
        buf.Proc_usage = (Double)reader[2];
        buf.Mem_usage = (Double)reader[3];
        buf.Net_usage = (Double)reader[4];
        buf.Disk_usage = (Double)reader[5];
        DBInfo.Add(buf);

    }
}
finally
{
    reader.Close();
    sqlCommand.Connection.Close();
}
return DBInfo;
}

private void DrawChart(List<DBInformation> DBInfo, short trigger) //
отрисовка графика
{

    chart1.Series[0].Points.Clear();
    foreach (DBInformation element in DBInfo)
    {
        double changeValue = 0;
        if (trigger == 0)
        {
            changeValue = element.Proc_usage;
        }
        else if (trigger == 1)
        {
            changeValue = element.Mem_usage;
        }
        else if (trigger == 2)
        {
            changeValue = element.Net_usage;
        }
    }
}

```

```

    }
    else if (trigger == 3)
    {
        changeValue = element.Disk_usage;
    }

    if (element.Date > dateTimePicker1.Value && element.Date <
dateTimePicker2.Value)
    {
        if (element.Name == (String)comboBox1.SelectedValue)
        {
            chart1.Series[0].Points.AddXY(0, changeValue);

        }
        else if ((String)comboBox1.SelectedValue == all)
        {
            chart1.Series[0].Points.AddXY(0, changeValue);
        }
    }
}

private void button1_Click_1(object sender, EventArgs e)
{
    trigger = 0;
    DrawChart(DBInfo, trigger);
}

private void button2_Click_1(object sender, EventArgs e)
{
    trigger = 1;
    DrawChart(DBInfo, trigger);
}

private void button3_Click_1(object sender, EventArgs e)
{
    trigger = 2;
    DrawChart(DBInfo, trigger);
}

private void button4_Click_1(object sender, EventArgs e)
{
    trigger = 3;
    DrawChart(DBInfo, trigger);
}

```

```

    }

    private void comboBox1_SelectedIndexChanged_1(object sender,
EventArgs e)
    {
        DrawChart(DBInfo, trigger);
    }

    private void dateTimePicker1_ValueChanged_1(object sender,
EventArgs e)
    {
        DrawChart(DBInfo, trigger);
    }

    private void dateTimePicker2_ValueChanged_1(object sender,
EventArgs e)
    {
        DrawChart(DBInfo, trigger);
    }

    private void Form1_Load(object sender, EventArgs e)
    {
    }

    private void Form1_FormClosed(object sender, FormClosedEventArgs
e)
    {
        if(server !=null) server.Disconnect();
    }

    private void button5_Click(object sender, EventArgs e)
    {
        button6.Enabled = false;
        try
        {
            server = new ServerObject(this);
            listenThread = new Thread(new ThreadStart(server.Listen));
            listenThread.Start(); //старт потока
        }
        catch (Exception ex)
        {
            server.Disconnect();
        }
    }

```

```

        Console.WriteLine(ex.Message);
    }
}

private void button6_Click(object sender, EventArgs e)
{
    server.Disconnect();
    label7.Text = "Сервер остановлен.";
    button5.Enabled = true;
    button6.Enabled = false;
}

private void button7_Click(object sender, EventArgs e)
{
    writeAndDraw();
}

private async void writeAndDraw()
{
    DBInfo = await ReadDB();
    DrawChart(DBInfo, trigger);
    List<String> listName =
GetNameFromBD(DBInfo).ToList<String>();
    comboBox1.DataSource = listName; // вывод имен компьютеров
    comboBox1.DisplayMember = "Proc_usage";
    dateTimePicker2.Value = DateTime.Now;
}

public class ClientObject
{
    SqlConnection sqlConnection;
    SqlCommand sqlCommand;
    Form1 form = new Form1();
    protected internal string Id { get; private set; }
    protected internal NetworkStream Stream { get; private set; }
    string userName = "";
    TcpClient client;
    ServerObject server; // объект сервера
    Form1 instance;

    public ClientObject(TcpClient tcpClient, ServerObject serverObject,
    Form1 _instance)

```

```

    {
        instance = _instance;
        Id = Guid.NewGuid().ToString();
        client = tcpClient;
        server = serverObject;
        serverObject.AddConnection(this);
    }

    public void Process()
    {
        sqlConnection = new SqlConnection(form.sqlConnectionValue);
        try
        {
            float buffer;
            String strBufer;
            Stream = client.GetStream();
            // получаем имя пользователя
            string message = GetMessage();
            StringReader rbf = new StringReader(message);
            userName = rbf.ReadLine();
            if(userName != null)
            {
                message = "Компьютер " + userName + " присоединился к
серверу";

                //server.BroadcastMessage(message, this.Id);
                Console.WriteLine(message);
                instance.label7.BeginInvoke((MethodInvoker)() =>
instance.label7.Text = message));
                instance.listBox1.BeginInvoke((MethodInvoker)() =>
instance.listBox1.Items.Add(userName));
            }

            // в бесконечном цикле получаем сообщения от клиента
            while (true)
            {
                try
                {
                    message = GetMessage();
                    Console.WriteLine(String.Format("{0}:\n{1}",
DateTime.Now, message));
                    message = String.Format(message);
                    StringReader reader = new StringReader(message);
                    sqlCommand = new SqlCommand("INSERT INTO
[Table] (Name, Date, Proc_usage, Mem_usage, Net_usage,

```

```

Disk_usage)VALUES(@Name,    @Date,    @Proc_usage,    @Mem_usage,
@Net_usage, @Disk_usage)", sqlConnection);
        sqlCommand.Connection.Open();
        strBufer = reader.ReadLine();
        sqlCommand.Parameters.AddWithValue("Name",
strBufer);
        sqlCommand.Parameters.AddWithValue("Date",
DateTime.Now);
        buffer = Single.Parse(reader.ReadLine());
        sqlCommand.Parameters.AddWithValue("Proc_usage",
buffer);
        buffer = Single.Parse(reader.ReadLine());
        sqlCommand.Parameters.AddWithValue("Mem_usage",
buffer);
        buffer = Single.Parse(reader.ReadLine());
        sqlCommand.Parameters.AddWithValue("Net_usage",
buffer);
        buffer = Single.Parse(reader.ReadLine());
        sqlCommand.Parameters.AddWithValue("Disk_usage",
buffer);

        sqlCommand.ExecuteNonQuery();
        sqlCommand.Connection.Close();

    }
    catch
    {
        message = String.Format("Компьютер {0}: отключился
от сервера", userName);
        Console.WriteLine(message);
        instance.label7.BeginInvoke((MethodInvoker)(() =>
instance.label7.Text = message));
        instance.listBox1.BeginInvoke((MethodInvoker)(() =>
instance.listBox1.Items.Remove(userName)));
        break;
    }

}
}
catch (Exception e)
{
    Console.WriteLine(e.Message);
}
finally

```

```

    {
        // в случае выхода из цикла закрываем ресурсы
        server.RemoveConnection(this.Id);
        sqlConnection.Close();
        Close();
    }
}

// чтение входящего сообщения и преобразование в строку
private string GetMessage()
{
    byte[] 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);

    return builder.ToString();
}

// закрытие подключения
protected internal void Close()
{
    if (Stream != null)
        Stream.Close();
    if (client != null)
        client.Close();
}
}

public class ServerObject
{
    static TcpListener tcpListener; // сервер для прослушивания
    List<ClientObject> clients = new List<ClientObject>(); // все
подключения
    Form1 instance;

    public ServerObject(Form1 _instance)
    {

```

```

        instance = _instance;
    }

    protected internal void AddConnection(ClientObject clientObject)
    {
        clients.Add(clientObject);
    }

    protected internal void RemoveConnection(string id)
    {
        // получаем по id закрытое подключение
        ClientObject client = clients.FirstOrDefault(c => c.Id == id);
        // и удаляем его из списка подключений
        if (client != null)
            clients.Remove(client);
    }

    // прослушивание входящих подключений
    protected internal void Listen()
    {
        try
        {
            int port = Int32.Parse(instance.textBox1.Text);
            tcpListener = new TcpListener(IPAddress.Any, port);
            tcpListener.Start();
            Console.WriteLine("Сервер запущен. Ожидание
подключений...");
            instance.label7.BeginInvoke((MethodInvoker)(() =>
instance.label7.Text = "Сервер запущен. Ожидание подключений..."));
            instance.button5.BeginInvoke((MethodInvoker)(() =>
instance.button5.Enabled = false));
            instance.button6.BeginInvoke((MethodInvoker)(() =>
instance.button6.Enabled = true));

            while (true)
            {
                TcpClient tcpClient = tcpListener.AcceptTcpClient();

                ClientObject clientObject = new ClientObject(tcpClient, this,
instance);

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

```

```

        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
            Disconnect();
        }
    }

    // трансляция сообщения подключенным клиентам
    protected internal void BroadcastMessage(string message, string id)
    {
        byte[] data = Encoding.Unicode.GetBytes(message);
        for (int i = 0; i < clients.Count; i++)
        {
            if (clients[i].Id != id) // если id клиента не равно id
отправляющего
            {
                clients[i].Stream.Write(data, 0, data.Length); //передача
данных
            }
        }
    }

    // отключение всех клиентов
    protected internal void Disconnect()
    {
        tcpListener.Stop(); //остановка сервера

        for (int i = 0; i < clients.Count; i++)
        {
            clients[i].Close(); //отключение клиента
        }
        //Environment.Exit(0); //завершение процесса
    }
}
}

```

Приложение А.2

Файл «client.cs»

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Drawing;
using System.Linq;
using System.Net.NetworkInformation;
using System.Net.Sockets;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace clientGUI
{
    public partial class Form1 : Form
    {
        static string userName;
        private string host = "127.0.0.1";
        private int port = 8888;
        static TcpClient client;
        static NetworkStream stream;
        static Thread receiveThread, mainProg;

        public Form1()
        {
            InitializeComponent();
            userName = Environment.MachineName;
        }

        // отправка сообщений
        static void SendMessage()
        {
            NetworkInterface[] interfaces =
NetworkInterface.GetAllNetworkInterfaces();
            string NameInterface = interfaces[0].Description;
            PerformanceCounter cpuCount =
PerformanceCounter("Процессор", "% загрузки процессора", "_Total");
```

					ПГУ 1.09.03.01.07.001 ПЗ	Лист
						53
Изм.	Лист	№ докум.	Подпись	Дата		

```

        PerformanceCounter memCount = new
PerformanceCounter("Память", "% использования выделенной памяти");
        PerformanceCounter netAllCount = new
PerformanceCounter("Сетевой интерфейс", "Текущая пропускная
способность", NameInterface);
        PerformanceCounter netValueCount = new
PerformanceCounter("Сетевой интерфейс", "Всего байт/с", NameInterface);
        PerformanceCounter diskCount = new
PerformanceCounter("Логический диск", "% активности диска", "_Total");
        while (true)
        {
            try {
                int value = 1;
                float cpu = 0;
                float mem = 0;
                float netAll = 0;
                float netValue = 0;
                float net = 0;
                float disk = 0;
                for (int i = 0; i < value; i++)
                {
                    cpu += cpuCount.NextValue();
                    mem += memCount.NextValue();
                    netValue += netValueCount.NextValue();
                    disk += diskCount.NextValue();
                    Thread.Sleep(1);
                }
                netAll = netAllCount.NextValue();

                cpu = cpu / value;
                mem = mem / value;
                net = netValue / netAll;
                disk = disk / value;
                string messageToServer = userName + "\n" + cpu.ToString() +
"\n" + mem.ToString() + "\n" + net.ToString() + "\n" + disk.ToString();
                byte[] data = Encoding.Unicode.GetBytes(messageToServer);
                stream.Write(data, 0, data.Length);
                Thread.Sleep(1000);
            } catch (Exception e)
            {
                Disconnect();
            }
        }
    }
}

```

```

// получение сообщений
void ReceiveMessage()
{
    while (true)
    {
        try
        {
            byte[] 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);

            string message = builder.ToString();
            Console.WriteLine(message); // вывод сообщения
        }
        catch
        {
            Console.WriteLine("Подключение прервано!"); // соединение
            было прервано
            this.label3.BeginInvoke((MethodInvoker)(() => this.label3.Text
            = "Подключение прервано!"));
            Disconnect();
        }
    }
}

static void Disconnect()
{
    if (stream != null)
        stream.Close(); // отключение потока
    if (client != null)
        client.Close(); // отключение клиента
    // Environment.Exit(0); // завершение процесса
    if (receiveThread != null) receiveThread.Abort();
    if (mainProg != null) mainProg.Abort();
}

private void Form1_Load(object sender, EventArgs e)
{

```

```

    }

    private void MainProgram()
    {
        client = new TcpClient();
        try
        {
            host = textBox1.Text;
            port = Int32.Parse(textBox2.Text);
            client.Connect(host, port); //подключение клиента
            stream = client.GetStream(); // получаем поток

            string message = userName;
            //byte[] data = Encoding.Unicode.GetBytes(message);
            // stream.Write(data, 0, data.Length);

            // запускаем новый поток для получения данных
            receiveThread = new Thread(new ThreadStart(ReceiveMessage));
            receiveThread.Start();
            Console.WriteLine("Подключен к серверу", userName);
            this.label3.BeginInvoke((MethodInvoker)() => this.label3.Text =
"Подключен к серверу"));
            SendMessage();
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
            this.label3.BeginInvoke((MethodInvoker)() => this.label3.Text =
ex.Message));
        }
        finally
        {
            Disconnect();
        }
    }

    private void button1_Click(object sender, EventArgs e)
    {
        mainProg = new Thread(new ThreadStart(MainProgram));
        mainProg.Start();
    }

```

```

private void button2_Click(object sender, EventArgs e)
{
    Disconnect();
}

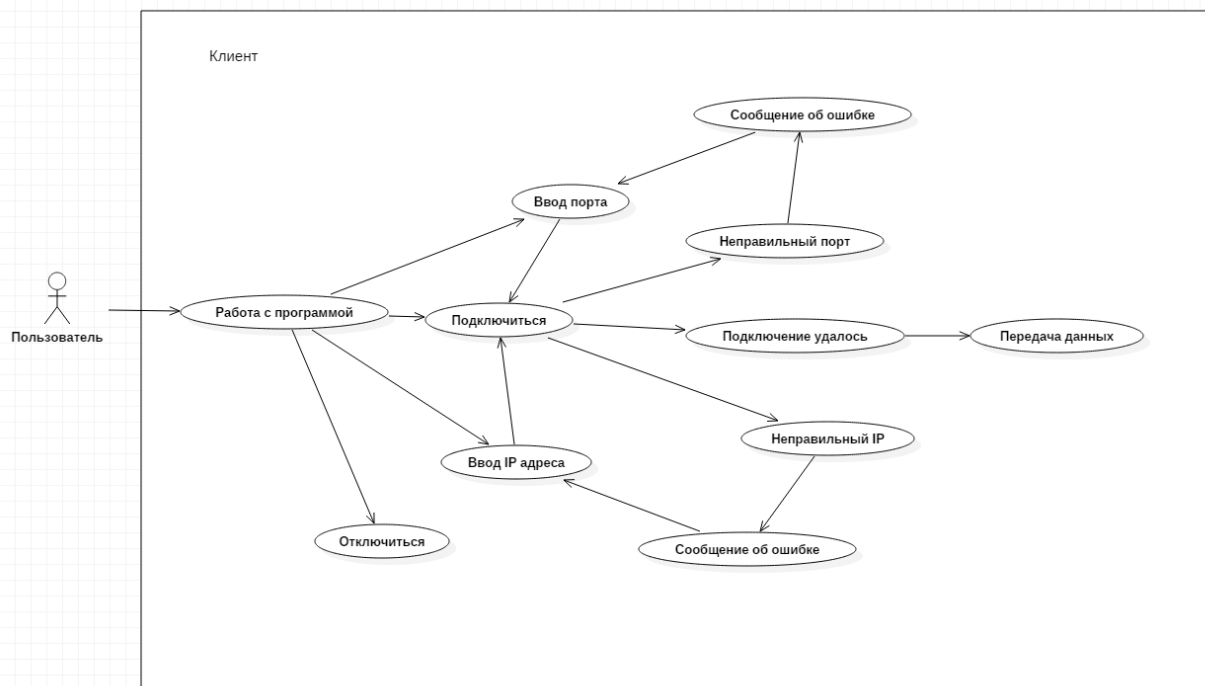
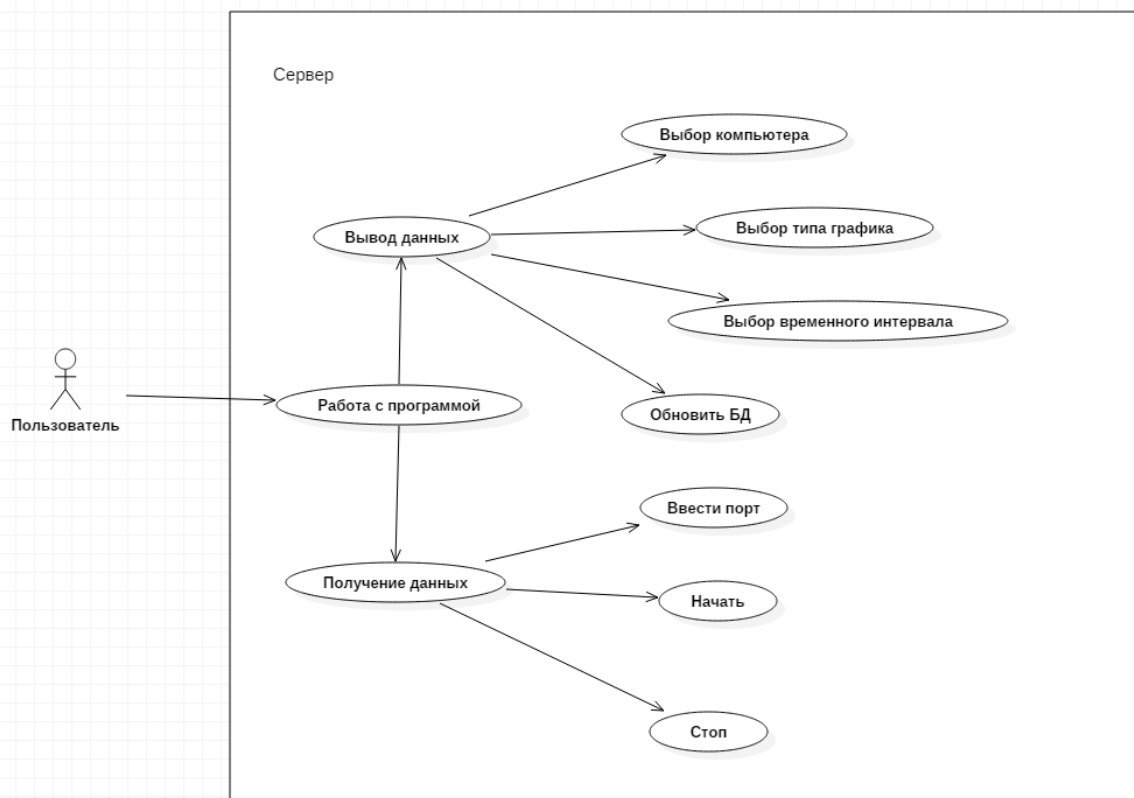
private void Form1_FormClosed(object sender, FormClosedEventArgs
e)
{
    Disconnect();
    Environment.Exit(0);
}
}

```

ПРИЛОЖЕНИЕ В UML ДИАГРАММЫ

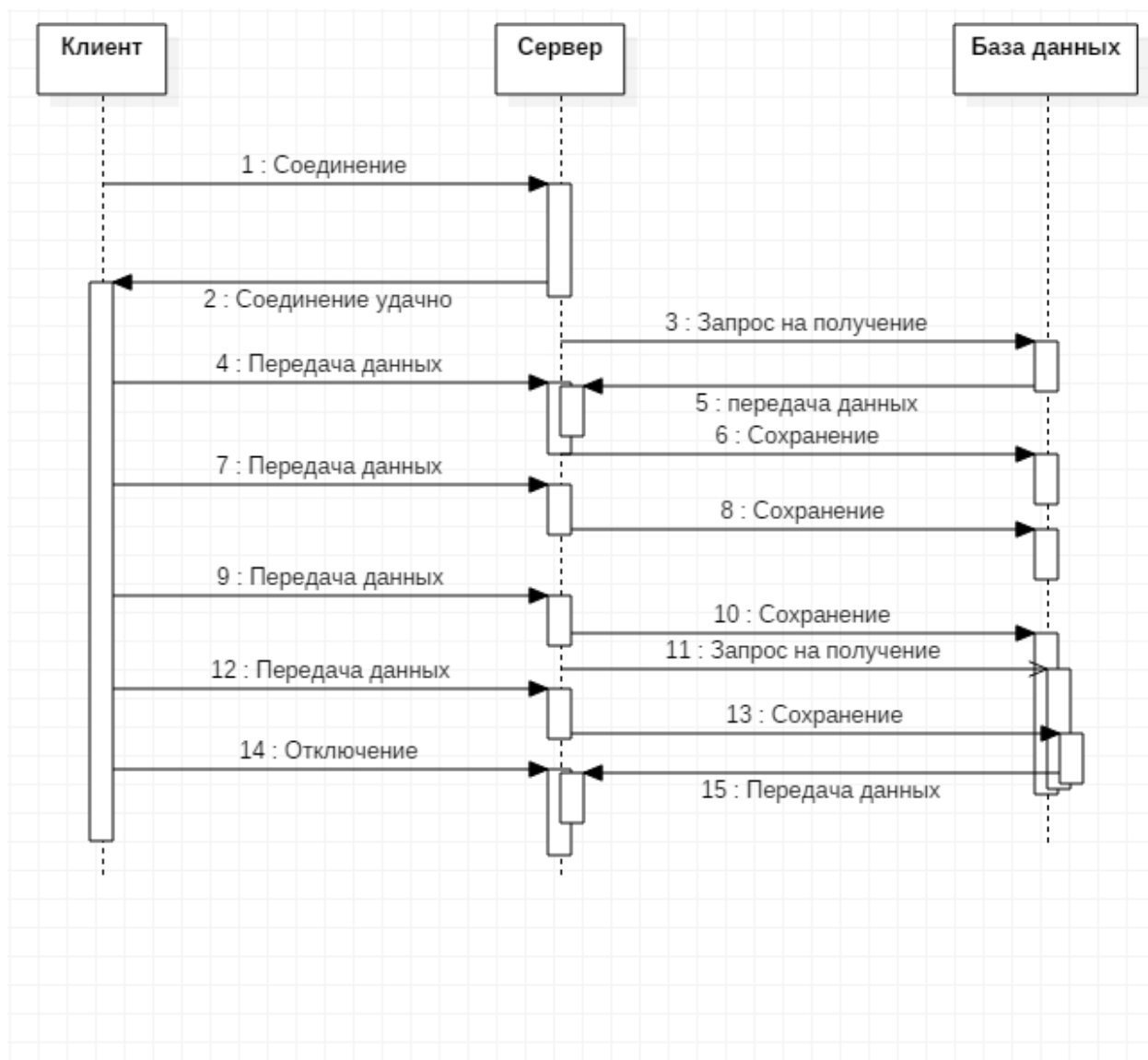
Приложение В.1
Диаграмма вариантов использования

					ПГУ 1.09.03.01.07.001 ПЗ	Лист
						58
Изм.	Лист	№ докум.	Подпись	Дата		

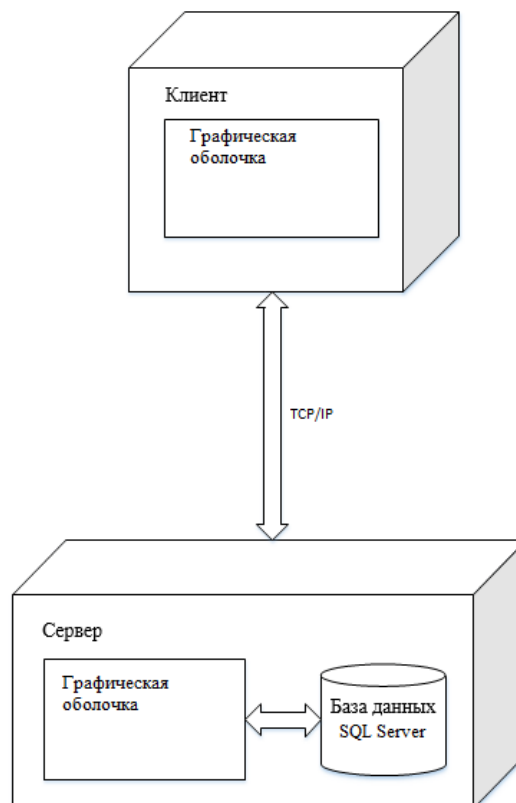


Приложение В.2

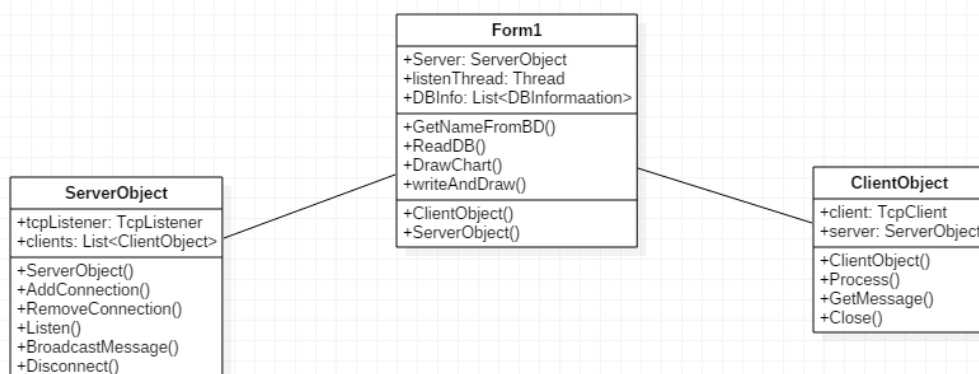
Диаграмма последовательностей



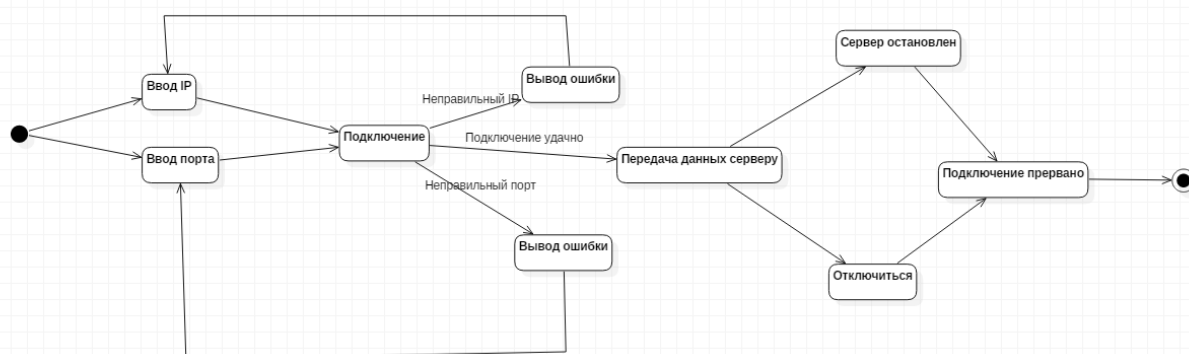
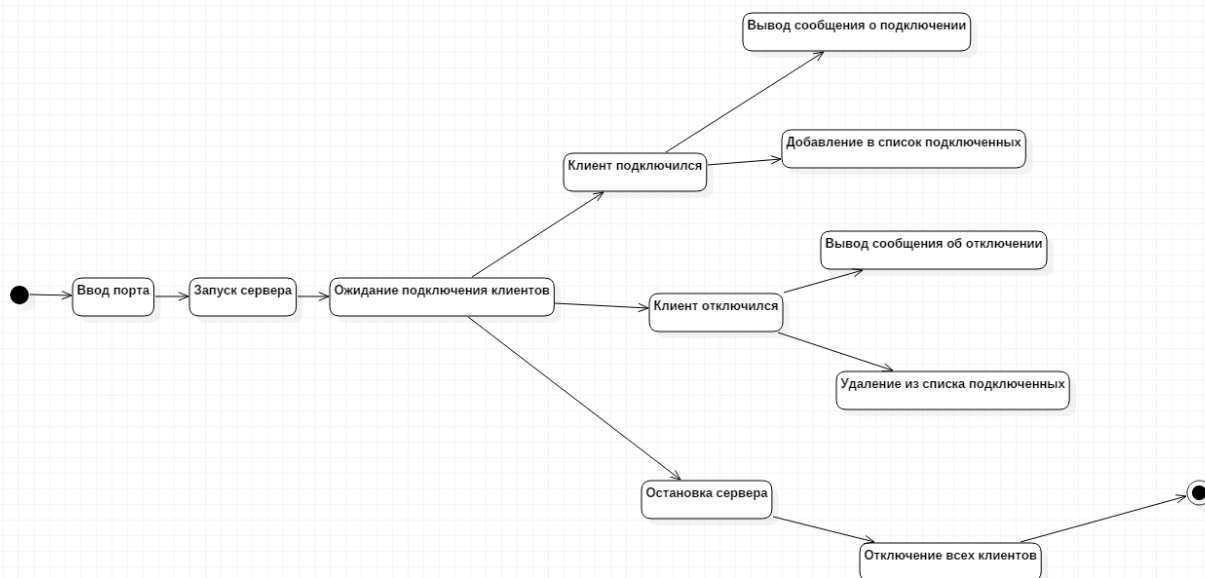
Приложение В.3 Диаграмма развертывания



Приложение В.4 Диаграмма классов



Приложение В.5 Диаграмма состояния



ПРИЛОЖЕНИЕ С ПРЕЗЕНТАЦИЯ

Тема работы «Система сбора
статистических данных по загрузке
персональных компьютеров в локальной
сети»

АВТОР РАБОТЫ: ЖИРКОВ К.Д. 13ВВ1

РУКОВОДИТЕЛЬ РАБОТЫ: ДОЦЕНТ КАФ. ВТ ПГУ ДОРОШЕНКО И.Н.

Постановка задачи

В данной выпускной работе необходимо разработать клиент-серверное приложение, позволяющее осуществлять сбор статистических данных по загрузке персональных компьютеров в локальной сети:

- ▶ приложение-клиент должно собирать данные с компьютеров;
- ▶ приложение-сервер должно получать информацию, обрабатывать и сохранять в базу данных;
- ▶ сетевое взаимодействие между клиентом и сервером должно быть построено на основе протокола TCP.

Диаграмма развертывания

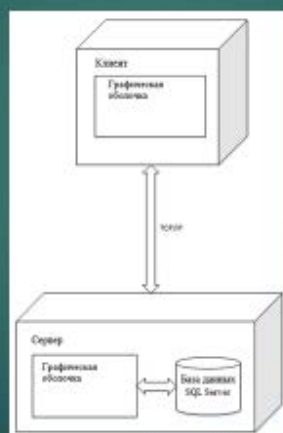


Диаграмма вариантов использования (клиент)

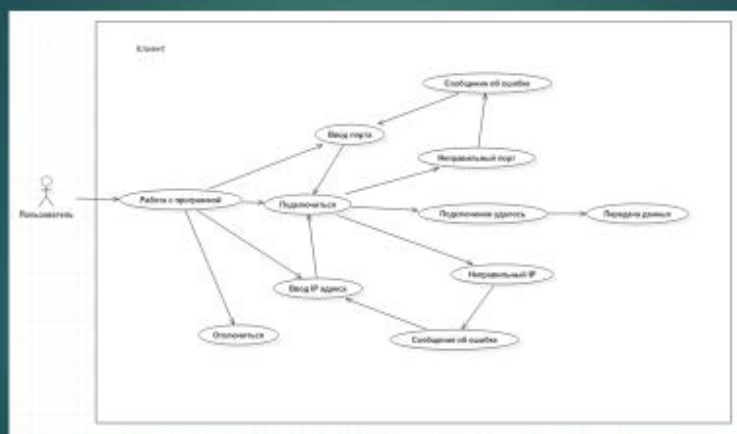


Диаграмма вариантов использования (сервер)

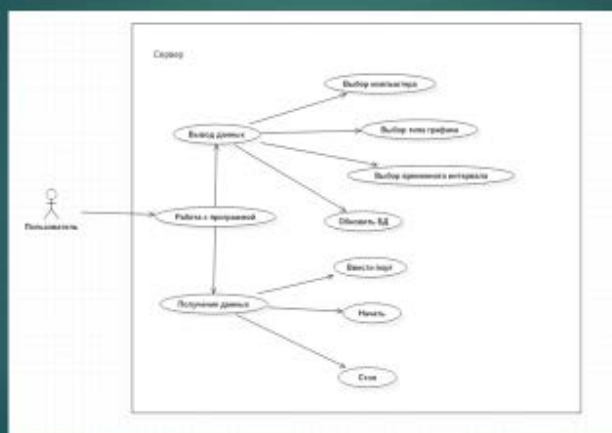


Диаграмма последовательностей

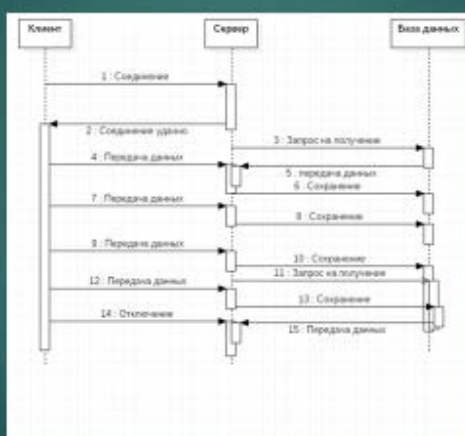


Диаграмма состояний (клиент)

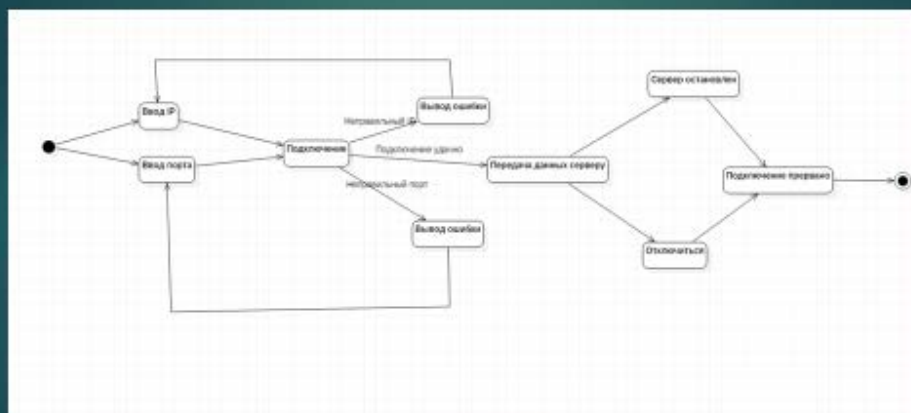
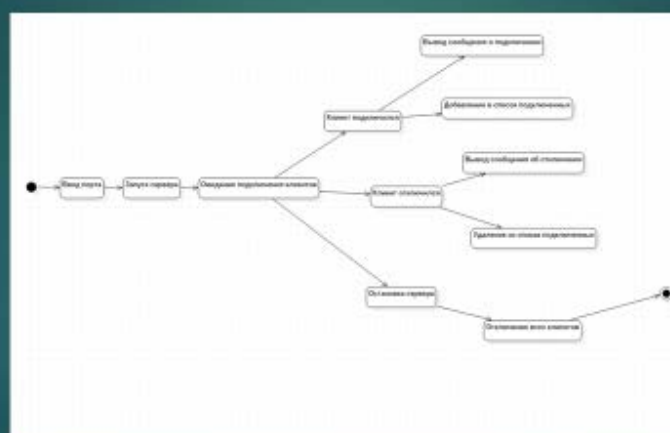
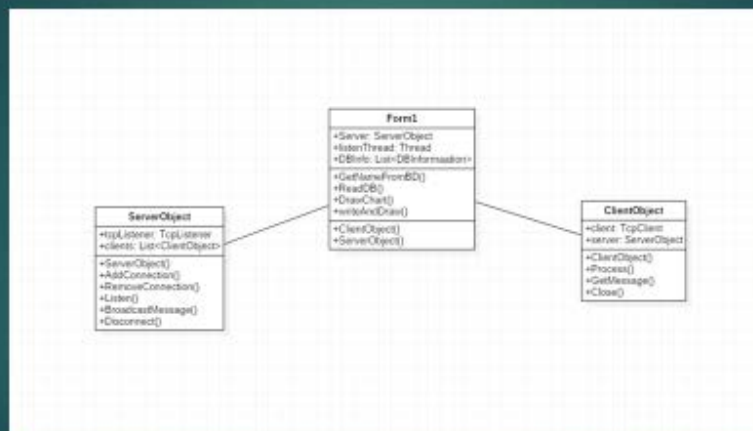
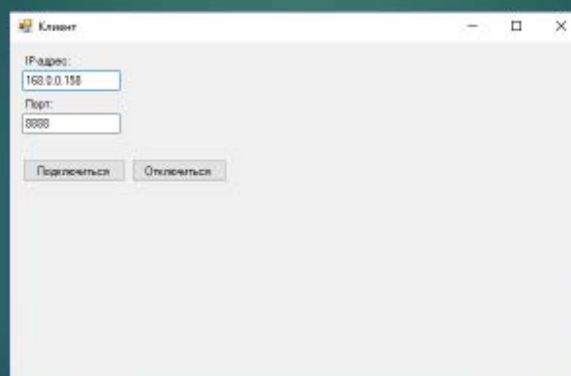


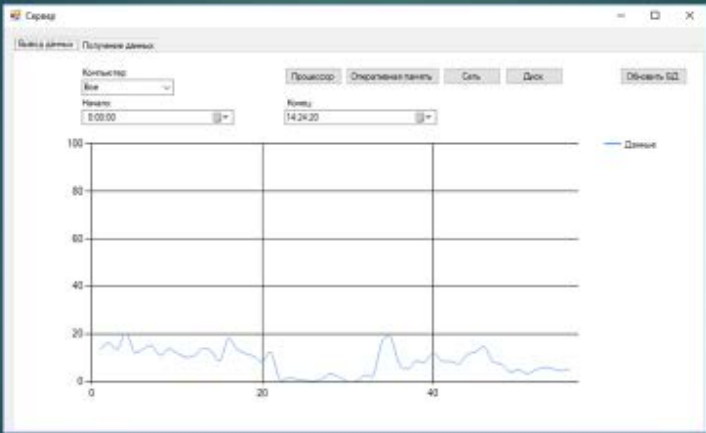
Диаграмма состояний (сервер)







Интерфейс серверного приложения



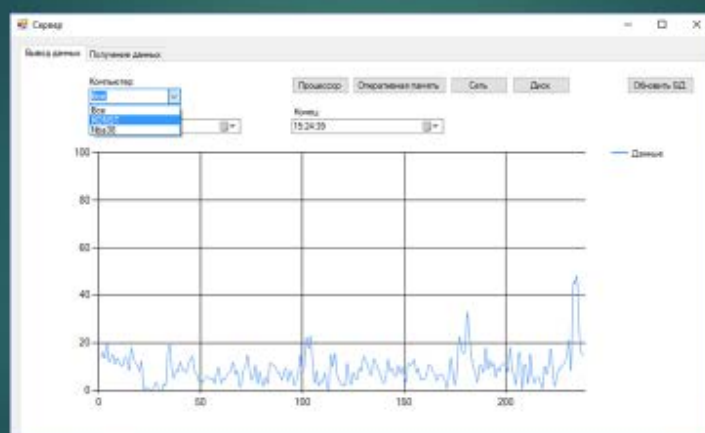
Интерфейс серверного приложения

The screenshot shows the 'Сервер' application window with the 'Получение данных' tab selected. It displays configuration fields: 'IP-адрес' (192.168.0.152), 'Ваш порт' (8888), and a note 'Нажмите кнопку для запуска сервера'. There are 'Начать' and 'Стоп' buttons. Below is a section 'Подключение к компьютеру' with an empty rectangular box.

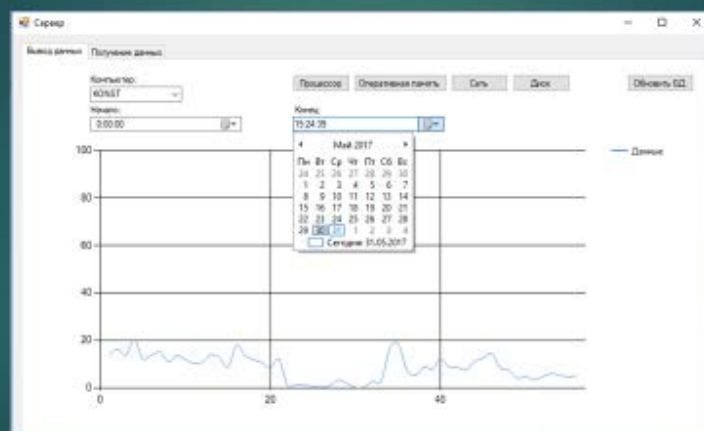
Подключение клиента



Вывод информации об определенном компьютере



Выбор времени вывода информации



Спасибо за
внимание!

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.09.03.01.07.001 ПЗ

Лист

70