

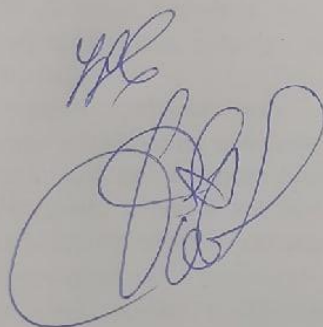
Министерство образования и науки РФ
Федеральное государственное бюджетное
образовательное учреждение высшего образования
Пензенский государственный университет Кафедра
«Вычислительная техника»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

по курсу «Распределённые вычисления»

на тему «Разработка параллельной программы для кластера
облачного типа»



30.04.2021

Выполнил:

студент группы 17ВВ2

Башкирцев Д.В.

Принял:

к.т.н., доцент

Федюнин Р.Н.

Пенза 2021

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет Вычислительной техники

Кафедра "Вычислительная техника"

"УТВЕРЖДАЮ"

Зав. кафедрой ВТ

Профессор М.А. Митрохин

« » 2021 г.

ЗАДАНИЕ

на курсовое проектирование по курсу

Распределенные вычисления

Студенту Башкирцеву Денису Васильевичу

Группа 17ВВ2

Тема проекта «Разработка параллельной программы для кластера облачного типа»

Исходные данные (технические требования) на проектирование

Требования к разработке:

- 1) Клиент-серверное приложение для управления вычислительным процессом должно состоять из двух частей: клиента и сервера, которые должны образовывать единую распределенную вычислительную систему облачного типа.
- 2) Серверное приложение должно обеспечивать одновременное подключение нескольких клиентов.
- 3) Клиентское приложение должно обеспечивать подключение к серверу (порт и IP-адреса сервера вводятся в окне клиента) и иметь возможность осуществить отключение от сервера в любой момент.
- 4) В окне серверного приложения должен отображаться прогресс решения вычислительной задачи и примерное время, оставшееся до ее решения с учетом текущей скорости вычислительного процесса, а также указываться список подключившихся клиентов и шаг интегрирования.
- 5) Для каждого клиента должна быть предусмотрена возможность указания предельно допустимого уровня использования вычислительных ресурсов (процессорного времени) в процентах. Соответствующее значение вводится на клиентской стороне.
- 6) Кластер должен решать задачу вычисления определенного интеграла

$$\int_{0.5}^1 \frac{e^x}{x} dx$$

7) Для интегрирования должен использоваться методом левых прямоугольников.

8) Приложения (клиент и сервер) должны обладать интуитивно понятным графическим интерфейсом.

9) Для обмена данными использовать протокол TCP/IP и библиотека WinSock.

10) Язык программирования C++, среда разработки Microsoft Visual Studio

Объем работы по курсу

1. Расчетная часть

Кластер должен решать задачу вычисления определенного интеграла

$$\int_{0.5}^1 \frac{e^x}{x} dx$$

2. Графическая часть

Диаграммы языка UML:

- 1) диаграмма классов.
- 2) диаграмма деятельности.
- 3) диаграмма вариантов использования.
- 4) диаграмма развёртывания.
- 5) диаграмма последовательности.

3. Экспериментальная часть

Тестирование распределенной вычислительной системы

Срок выполнения проекта по разделам

В соответствии с графиком

Дата выдачи задания "3" марта

Дата защиты проекта "30" апреля

Руководитель

Задание получил "03" марта 2021г.

Студент Билимурцев Д.Н.

Оглавление

Введение.....	
Выбор решения.....	
Описание разработ	
Разработка диаграм	
4.1. Разработка диа	
4.2. Разработка диа	
4.3. Разработка диа	
4.4. Разработка диа	
4.5. Разработка диа	
Отладка и тестиров	
Описание работы с	
Руководство польз	
Заключение.....	
Список используем	
Приложение А. Фа	
Приложение В- Фа	

Оглавление

Введение.....	5
Выбор решения	9
Описание разработки программы.....	10
Разработка диаграмм UML	13
4.1. Разработка диаграммы вариантов использования.....	13
4.2. Разработка диаграммы развертывания	16
4.3. Разработка диаграммы последовательности	17
4.4. Разработка диаграммы деятельности	18
4.5. Разработка диаграммы классов.....	20
Отладка и тестирование.....	22
Описание работы с программой	23
Руководство пользователя	25
Заключение	29
Список используемых источников	30
Приложение А. Файл «Client.cpp».....	31
Приложение В– Файл «Server.cpp»	37

Введение

Вычислительные системы облачного типа представляют собой особый вид распределенных вычислительных систем, которые характеризуются высокой отказоустойчивостью, динамическим масштабированием и эффективным использованием вычислительных ресурсов. При этом вычислительные узлы такой системы обладают (в общем случае) противоположными характеристиками в частности неоднородностью и низкой надежностью.

Прежде, чем рассмотреть каким же образом происходит совмещение столь антагонистичных на первый взгляд свойств необходимо определить каждое из этих свойств.

Высокая отказоустойчивость предполагает непрерывное обеспечение вычислительного процесса, до тех пор, пока хотя бы один вычислительный узел будет функционировать, при этом должны быть сохранены промежуточные результаты вычислений, которые уже были произведены на отказавших узлах. Динамическое масштабирование предполагает возможность подключения и отключения вычислительных узлов в процессе работы и обеспечение перераспределения вычислительной нагрузки между узлами.

Эффективное использование вычислительных ресурсов предполагает распределение задач между узлами в соответствии с их аппаратными характеристиками для обеспечения максимально допустимого уровня их использования.

Неоднородность узлов предполагает различные вычислительные возможности каждого из узлов вычислительной системы.

Низкая надежность предполагает возможность отказа (либо отключения) любого из узлов в любой момент времени.

Для того чтобы совместить указанные свойства вычислительной системы предлагается использовать подход порционной выдачи заданий. Данный подход заключается в следующем. Начальная задача разделяется на множество подзадач, которые могут быть выполнены независимо. Каждая подзадача должна выполняться любым узлом вычислительной системы за сравнительно небольшое время (размер задачи определяется эмпирически). Формируется очередь подзадач. После решения одним из узлов очередной подзадачи ее результат фиксируется на сервере, а затем из очереди извлекается новая подзадача и передается узлу для решения. Если в процессе решения задачи узел отказал (отключился от сервера), то подзадача, которая была ему передана, вновь добавляется в очередь. Таким образом, с одной стороны происходит фиксация промежуточных результатов вычислений, а с другой предотвращается потеря подзадач. При этом каждый из узлов использует все доступные ему вычислительные ресурсы при решении очередной подзадачи.

При такой организации единственным уязвимым для отказов узлом является сервер. В действительности часто используется структура, когда в качестве управляющего сервера выступает целая группа (пул) аппаратно

независимых серверов, которые синхронизируют промежуточные результаты вычислений и при отказе одного из них клиентские узлы перераспределяются между оставшимися серверами.

Для простоты в данной лабораторной работе будем считать, что необходимо реализовать вычислительную систему облачного типа, в которой обеспечивать программную отказоустойчивость серверной части не требуется.

1. Постановка задачи

Необходимо разработать программы клиент и сервер, для работы с кластером облачного типа. Кластер должен состоять из запущенных программ-клиентов. Сервер должен обеспечивать подключение нескольких клиентов одновременно.

Разрабатываемым программам предъявлены следующие требования:

1) Клиент-серверное приложение для управления вычислительным процессом должно состоять из двух частей: клиента и сервера, которые должны образовывать единую распределенную вычислительную систему облачного типа.

2) Серверное приложение должно обеспечивать одновременное подключение нескольких клиентов.

3) Клиентское приложение должно обеспечивать подключение к серверу (порт и IP сервера вводятся в окне клиента) и иметь возможность осуществить отключение от сервера в любой момент.

4) В окне серверного приложения должен отображаться прогресс решения вычислительной задачи и примерное время, оставшееся до ее решения с учетом текущей скорости вычислительного процесса, а также указываться список подключившихся клиентов и шаг интегрирования.

5) Для каждого клиента должна быть предусмотрена возможность указания предельно допустимого уровня использования вычислительных ресурсов (процессорного времени) в процентах. Соответствующее значение вводится на серверной стороне.

6) Кластер должен решать задачу вычисления определенного интеграла

$$\int_{0,5}^1 \frac{e^{x^2}}{x} dx$$

7) Для интегрирования должен использоваться методом средних прямоугольников.

8) Приложения (клиент и сервер) должны обладать интуитивно понятным графическим интерфейсом.

9) Для обмена данными использовать протокол TCP/IP и библиотека

10) Язык программирования C++, среда разработки Microsoft Visual Studio.

В связи с требованиями, предъявленными выше, были приняты следующие положения по разработке:

1) Организация правильного взаимодействия компьютеров в локальной вычислительной сети создается на клиент-серверной основе. Взаимодействие сервера с каждым клиентом осуществляется в отдельном потоке.

2) Устройство ввода информации – клавиатура.

3) Операционная система – MS Windows.

4) Среда программирования – MS Visual Studio 2019.

5) Язык программирования – C++.

Выбор решения

Работа с сетью производится с использованием протокола TCP/IP, поэтому были описаны методы передачи и приёма информации, позволяющие общаться программам внутри сети.

Для реализации графического интерфейса было решено воспользоваться редактором форм Microsoft Visual Studio 2017. С его помощью, были заданы положение и размеры элементов графического интерфейса для ввода и вывода информации, необходимой для решения поставленной задачи. Для реализации логической части ПО были создан набор классов обеспечивающих корректную работу программ. Работа с сетью производится с использованием протокола TCP/IP, в связи с чем были описаны методы передачи и приёма информации, позволяющие программам внутри сети обмениваться необходимыми данными.

Описание разработки программы

Для написания данной программы будет использован язык программирования C++. Этот язык является распространённым языком программирования.

Язык C++ включает в себя возможности языка C, однако в нем была добавлена возможность применения объектно-ориентированного подхода в проектировании и программировании.

В языке C предусмотрено много операций, непосредственно исполняемых большинством микропроцессоров; в то же время он даёт программисту возможность выражать свои алгоритмы и данные наиболее подходящими средствами. Язык C обеспечивает возможности структурирования данных. Он позволяет разрабатывать большие, но структурированные программы, предоставляя возможность отдельной разработки подпрограмм.

C – мобильный язык, который можно было бы использовать для разработки системного программного обеспечения. Программы, написанные на языке высокого уровня, мобильны, поскольку язык должен быть одним и тем же независимо от того, на каком компьютере и в какой операционной системе он используется.

Дополнительное свойство языка C, называемое условной компиляцией, позволяет программисту изолировать машинно-зависимые операторы и контролировать их компиляцию в другой среде. Это дополнительно повышает мобильность программного обеспечения, написанного на языке C.

В качестве среды программирования был выбран программный продукт Microsoft Visual Studio 2019.

Интерфейс как серверного, так и клиентского приложения представлен в виде оконного приложения Windows Forms.

Применялся нисходящий метод проектирования: были определены методы взаимодействия пользователя с методами классов сервера и клиента посредством

графического интерфейса, затем были реализованы сами классы.

Графический интерфейс приложений сервера и клиента реализованы с помощью CLR. Common Language Runtime (англ. CLR — общезыковая исполняющая среда) — исполняющая среда для байт-кода CIL (MSIL), в которой компилируются программы, написанные на .NET-совместимых языках программирования (C#, Managed C++, Visual Basic .NET, F# и прочие). CLR является одним из основных компонентов пакета Microsoft .NET Framework.

Для написания сетевых приложений требовалось освободить главный поток, чтобы интерфейс не зависал. Таким образом, каждая программа выполняет общение по сети в отдельных потоках. Клиентское приложение при подключении к серверу создаёт новый поток для получения и решения задачи. Серверное приложение при своём запуске порождает два потока, один из которых обрабатывает входящие соединения с клиентами, а другой производит выдачу заданий уже доступным клиентам.

Разработка программы началась с изучения и анализа работы WinSock. Протокол TCP/IP по установлению соединения позволяет гарантировано доставить информацию на указанный компьютер. Изучив принцип работы WinSock протоколов TCP/IP было принято решение создать именно 2 сокета: серверный — для приема сообщений, клиентский — для отправки сообщений.

Затем в проект была внедрена функция реагирования на поступивший запрос и выбор действий над ним, появилась возможность подключаться к серверу.

После была создана возможность решения задачи интегрирования методом средних прямоугольником и деление данной задачи на несколько одновременно работающих клиентов.

Разработка диаграмм UML

С целью упрощения и визуализации процесса проектирования программы были разработаны следующие диаграммы UML:

- 1) диаграмма вариантов использования;
- 2) диаграмма развёртывания;
- 3) диаграмма последовательности;
- 4) диаграмма деятельности;
- 5) диаграмма классов.

4.1. Разработка диаграммы вариантов использования

Как в клиентском, так и в серверном приложении, единственным участником системы является пользователь.

Диаграмма вариантов использования клиентского приложения показана на рисунке 1.

Диаграмма вариантов использования серверного приложения показана на рисунке 2.

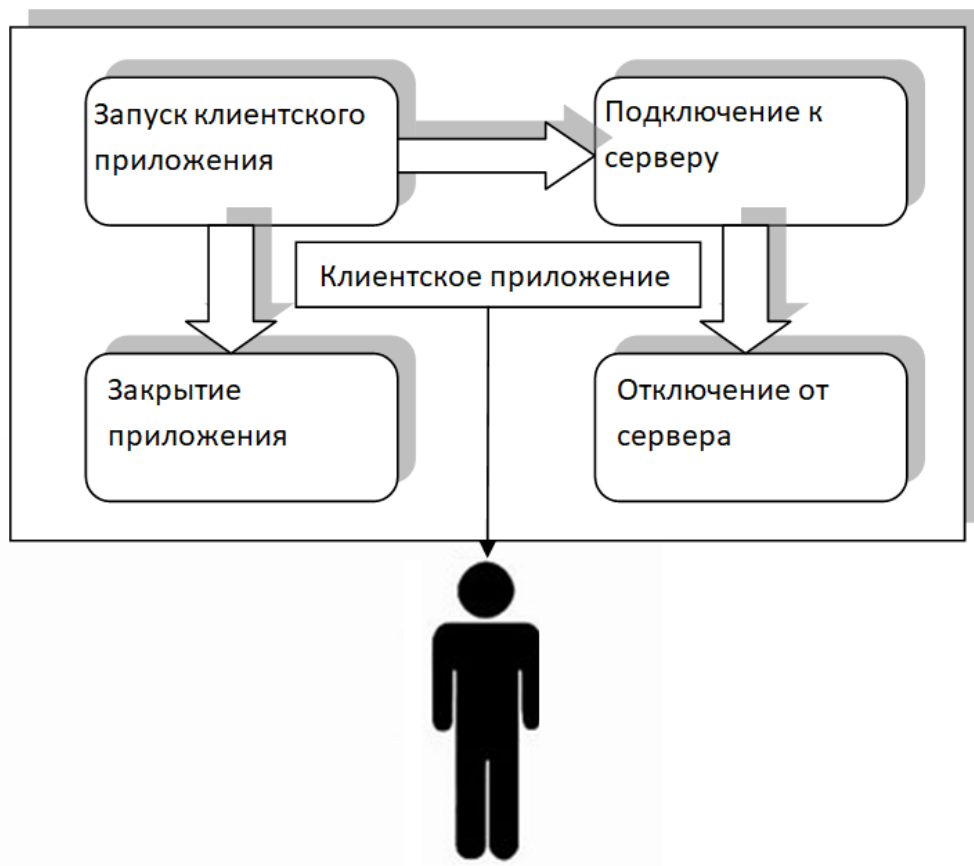


Рисунок 1 – UML-диаграмма вариантов использования клиентского приложения

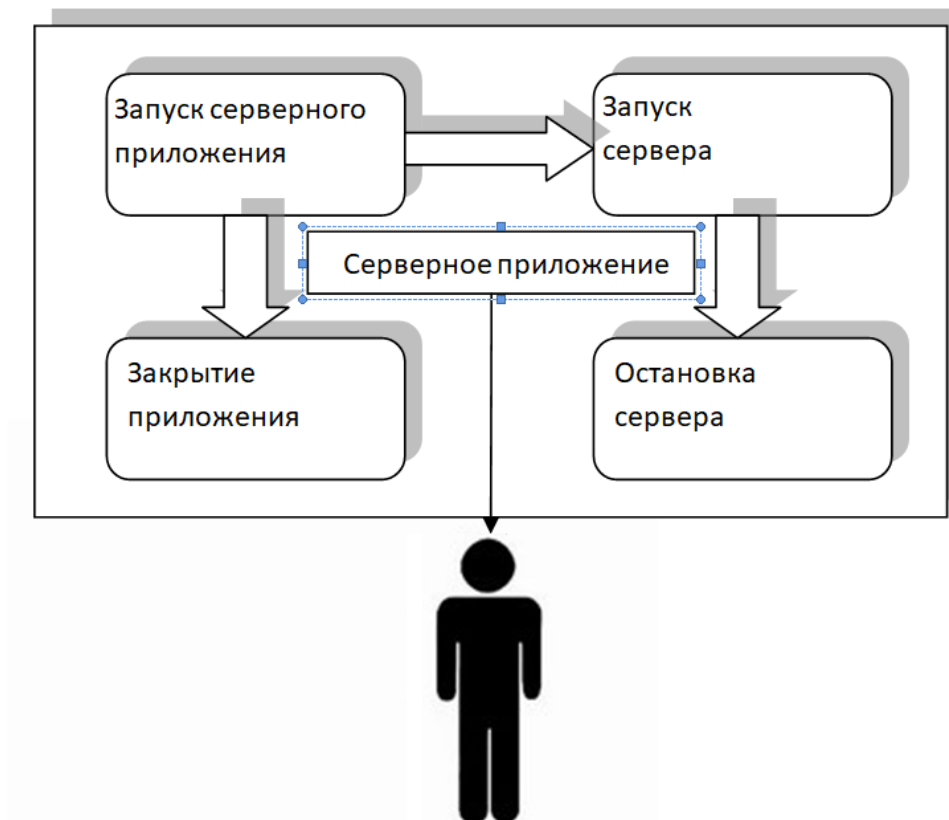


Рисунок 2 – UML-диаграмма вариантов использования серверного приложения

На рисунках 1 и 2 показаны UML-диаграммы использования двух приложений. Клиент может в любой момент осуществить попытку подключиться к серверу и в любой момент разорвать с ним соединение. Сервер так же может быть запущен и остановлен в любой момент.

4.2. Разработка диаграммы развертывания

В связи с требованиями к программному обеспечению, была разработана диаграмма развертывания, показанная на рисунке 3.

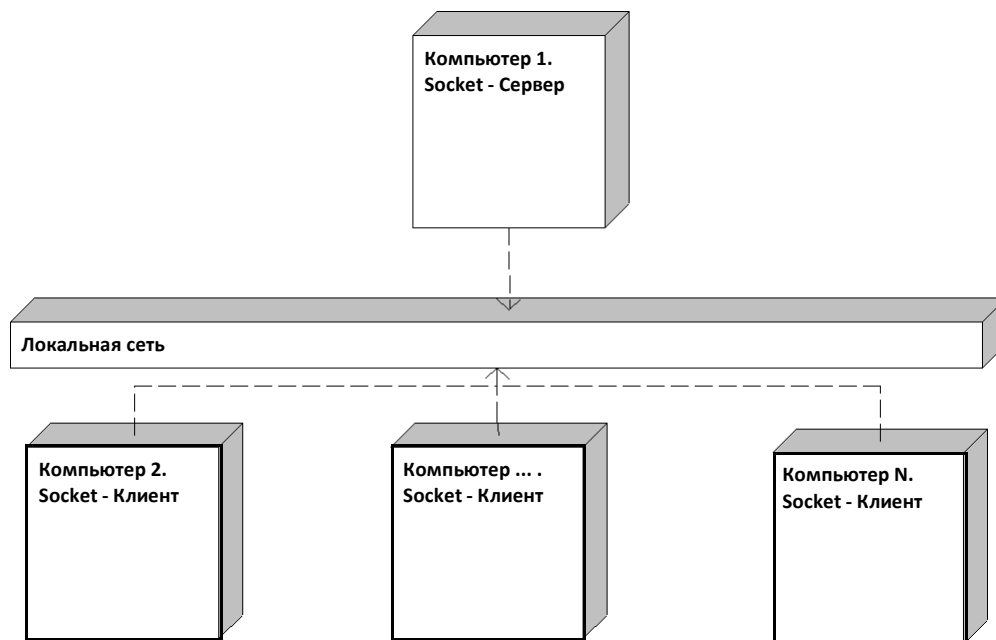


Рисунок 3 – UML-диаграмма развёртывания

Стоит отметить, что компьютеры с клиентскими сокетами подключаются к компьютеру с серверным сокетом по локальной вычислительной сети. Также

возможна работа таким образом, что Компьютер 1 диаграммы может иметь как серверный, так и клиентский сокет. Таким образом развертывание возможно на одном компьютере, а членом локальной вычислительной сети будет только один компьютер, посылающий сообщения себе по адресу localhost.

4.3. Разработка диаграммы последовательности

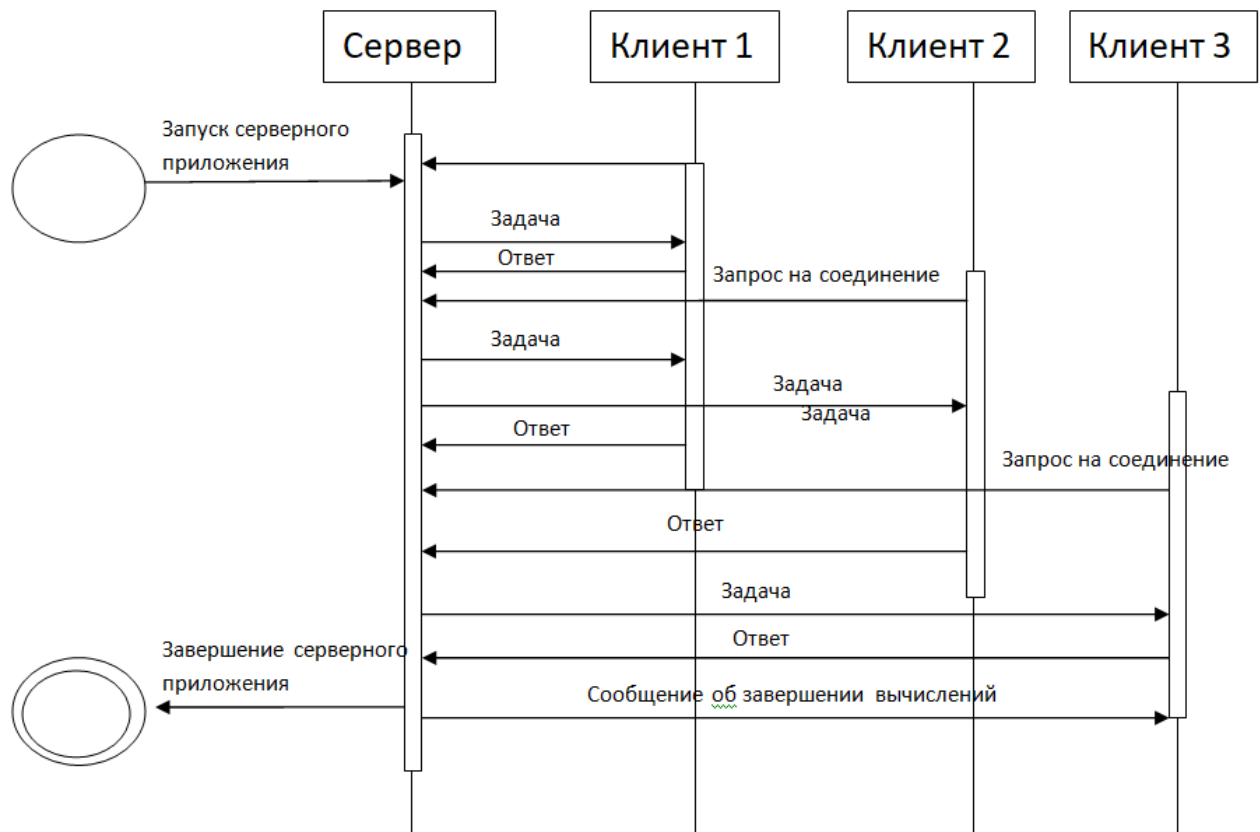


Рисунок 4 – UML-диаграмма последовательности

На рисунке 4 приведён пример работы серверного и клиентских приложений в виде UML-диаграммы последовательности. В примере показано что сервер ожидает возвращение результатов вычислений от всех клиентов, которым были отправлены задания. Это ожидание может быть нарушено при сообщении об ошибке на клиенте, при этом клиент удаляется из списка, а его задачи возвращаются в очередь задач.

4.4. Разработка диаграммы деятельности

Разработаны диаграммы деятельности для клиентского и серверного приложения, показанные на рисунке 5 и 6, соответственно.

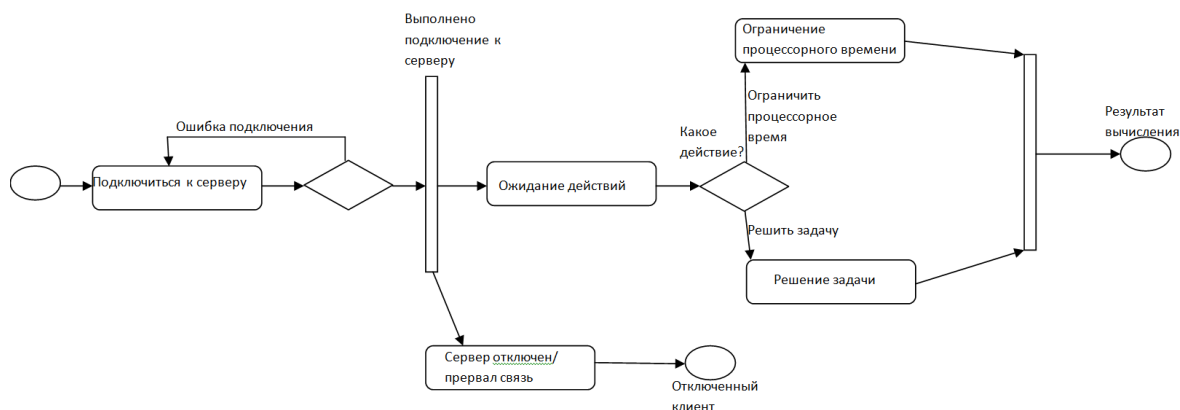


Рисунок 5 – UML-диаграмма деятельности клиентского приложения

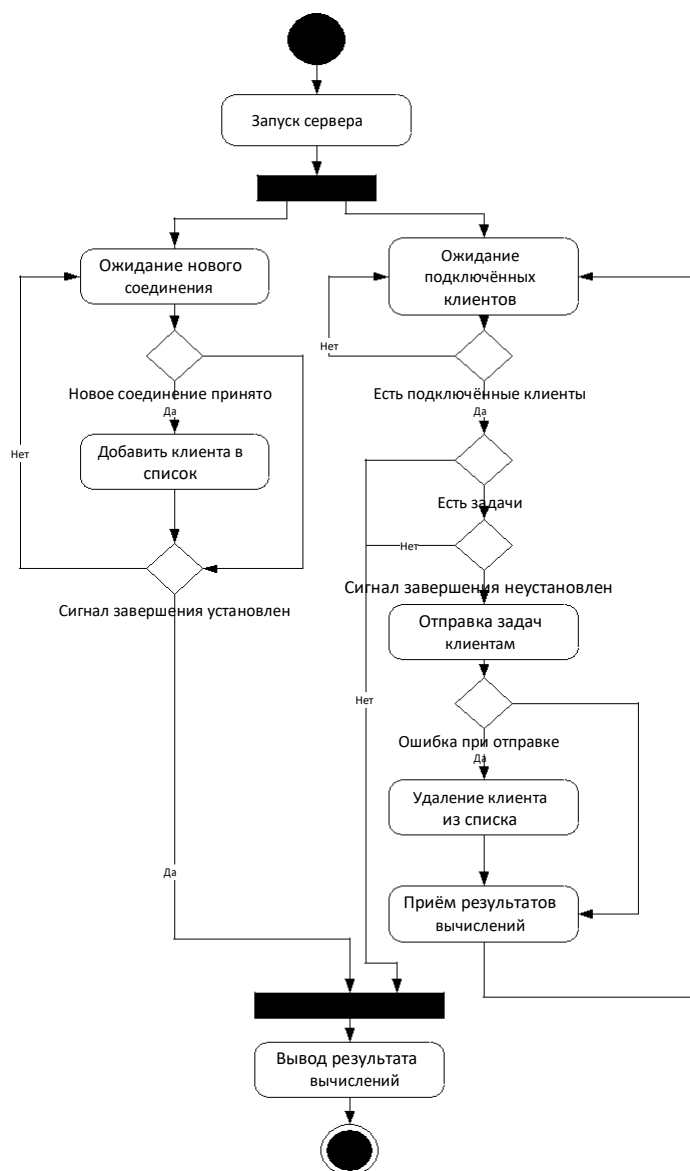


Рисунок 6 – UML-диаграмма деятельности

UML-диаграмма деятельности, показанная на рисунке 6, отображает работу нитей на основе методов класса IntegrServer: listenThread и workThread. Слева отображена работа метода listenThread – выполняется ожидание входящего запроса на соединение, соединение устанавливается и данные нового клиента заносятся в список клиентов, по которому затем производится отправка заданий и приём результатов.

Метод workThread отправляет клиентам задания и принимает от них результаты вычислений. При возникновении ошибки на одном из двух этапов происходит удаление конкретного клиента из списка и возвращение задач, которые были отправлены, но не были завершены, обратно в список задач. На рисунке показан такой случай только при ошибке во время отправления задач клиенту. Отправка сообщений и приём результатов происходит пока не были выполнены эти действия для всех клиентов в списке.

При установлении сигнала завершения нити завершаются и затем объект класса удаляется.

4.5. Разработка диаграммы классов

Была разработана диаграмма классов для клиентского и серверного приложений, показанная на рисунке 7.

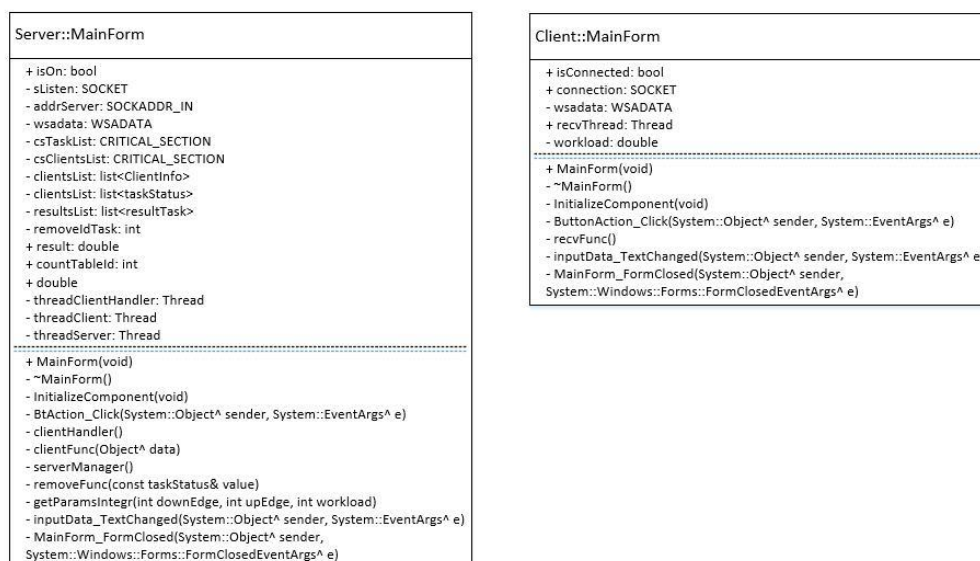


Рисунок 7 – UML-диаграмма классов

На рисунке 7 предоставлена UML-диаграмма классов. На ней изображены класс `Server::MainForm`, который реализует функции сервера, и класс `Client::MainForm`, выполняющий непосредственные вычисления. В классах имеются несколько методов, работающих в отдельных потоках.

Отладка и тестирование

В качестве среды разработки выступает программный продукт MS Visual Studio 2017. Программа предоставляет все средства необходимые при разработке и отладке разрабатываемых модулей и программ. Для отладки использовались такие инструменты как точка остановки, выполнение кода по шагам, анализ содержимого локальных и глобальных (для определенного класса) переменных.

Тестирование проводилось в рабочем порядке, в процессе разработки, после завершения написания программы. В ходе тестирования было выявлено и исправлено множество проблем, связанных с взаимодействием сервера и клиента, ограничения процессорного времени и реагированием сервера на подключение и отключение клиентов.

Описание работы с программой

Visual Studio 2019 предоставляет возможности для создания программ с графическим пользовательским интерфейсом.

Описание элементов графического интерфейса производится в файлах, которые представляют собой генерируемую автоматически часть класса. Они генерируются Visual Studio 2019 на основе созданного в графическом режиме интерфейса и описывают положение, размеры и другие свойства элементов графического интерфейса.

При запуске сервера необходимо произвести его инициализацию: задать прослушиваемый порт, указать шаг интегрирования и нагрузку на вычислительные возможности клиентов. После нажатия кнопки «Включить сервер» будет запущен отдельный поток `ThreadClientHandler` в котором выполняется обработка всех входящих подключений по порту прослушивания. Для каждого подключившегося клиента создаётся отдельный поток `threadClient`, где осуществляется приём результатов вычисления и обновление элементов интерфейса прогресса (процент выполнения и оставшееся время до выполнения задачи).

На клиентском приложении предлагается ввести адрес и порт сервера. После успешного подключения к серверу создастся новый поток `recvThread`, где происходит приём очередной подзадачи и её выполнение с учётом поставленного ограничения на использования центрального процессора.

Методы `inputData_TextChanged` выполняют проверку введенных данных в форму на их корректность. Таким образом, если в поля вводить буквы или символы кнопка действия будет заблокирована до тех пор, пока данные не будут удовлетворять условиям.

После запуска клиента вы увидите окно, показанное на рисунке ниже

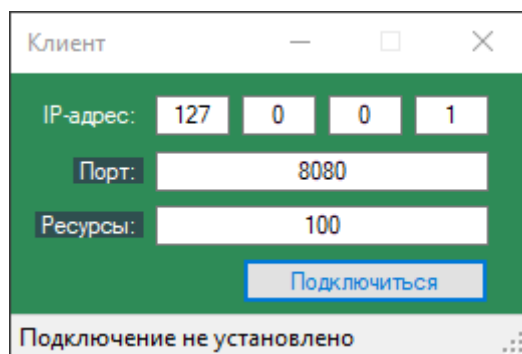


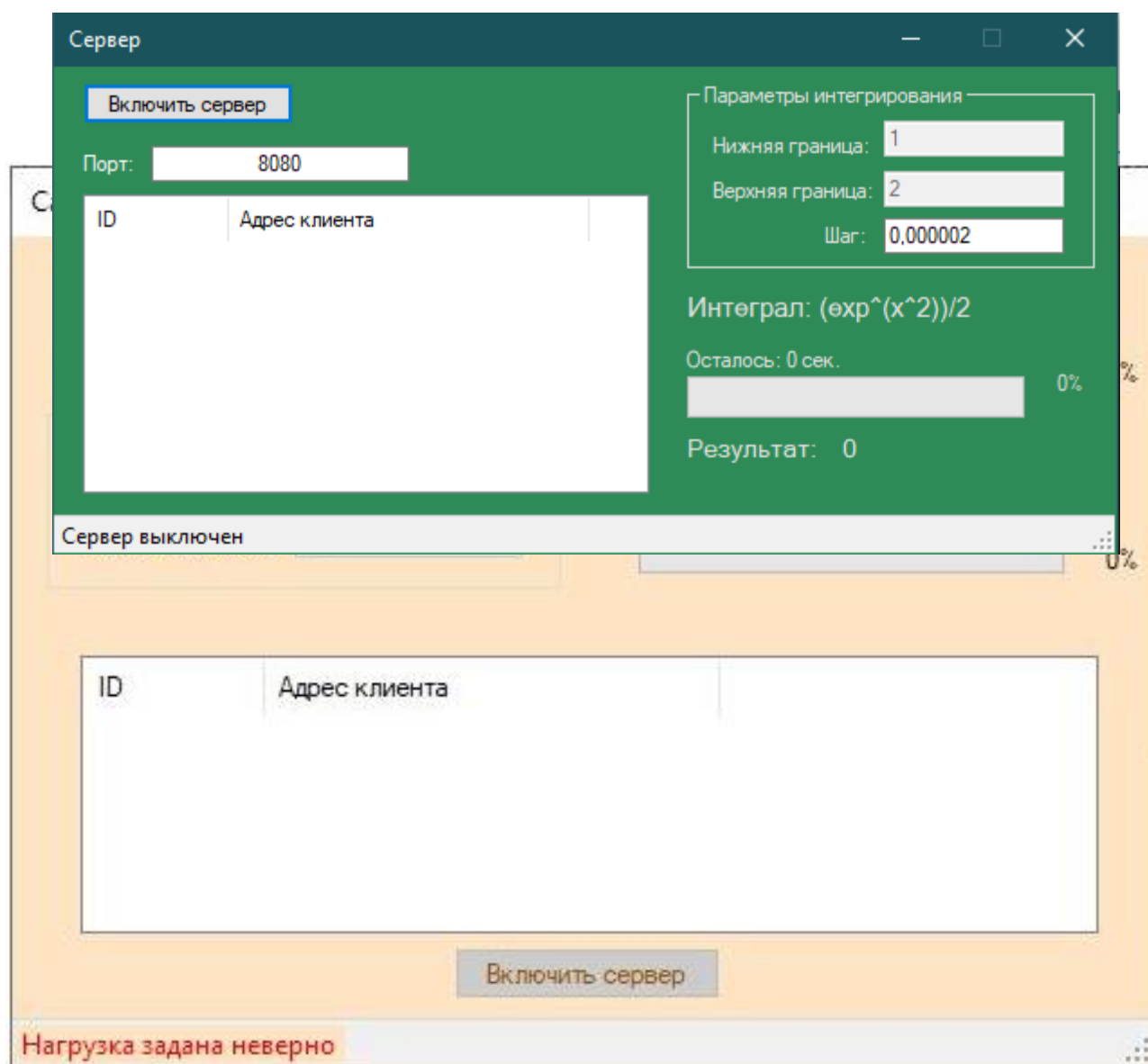
Рисунок 8 – Окно клиентского приложения после запуска

После запуска сервера вы увидите окно, показанное на рисунке ниже.

Руководство пользователя

Данный проект образует единую распределенную вычислительную системы облачного типа и состоит из двух программ: сервера и клиента.

Для начала работы требуется запустить программу сервера. При старте программы в поля формы уже будут занесены значения по умолчанию, как показано на рисунке 9.



После того, как все данные введены, необходимо нажать кнопку «Включить сервер». Все поля ввода будут заблокированы, появится кнопка «Выключить сервер» и в нижнем меню статусе появится соответствующее сообщение, как показано на рисунке 10.

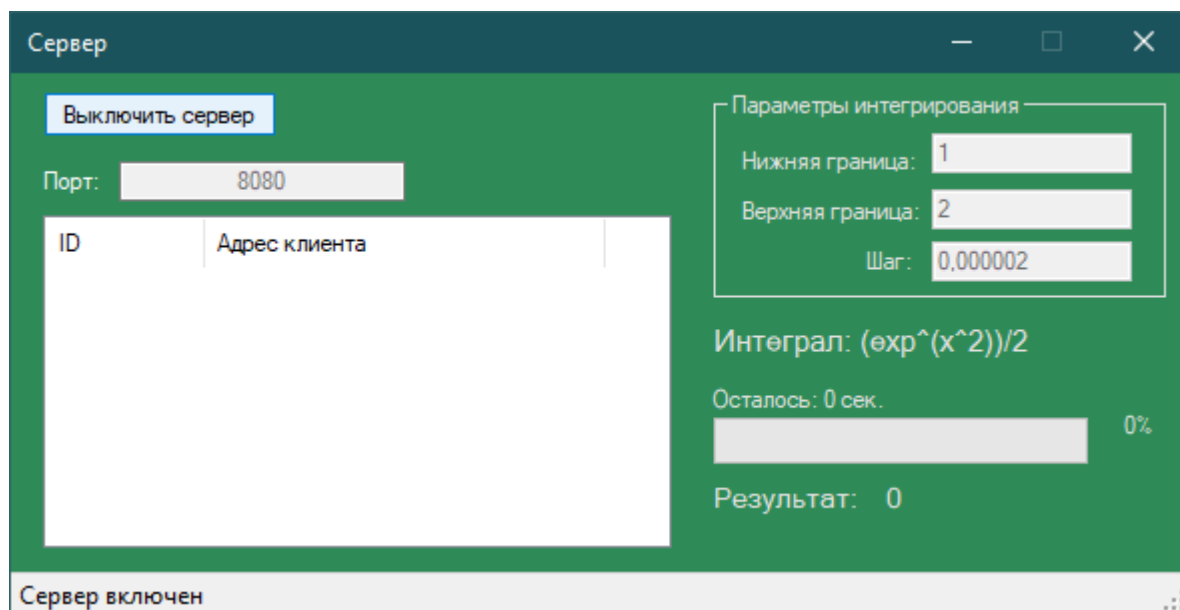


Рисунок 10 – Сервер запущен

На данном этапе сервер ожидает подключения клиентов, чтобы начать выполнение вычислений. После того, как клиент подключится к серверу, начнутся вычисления, о чём будет свидетельствовать элемент прогресса и счётчик оставшегося времени, как показано на рисунке 11.

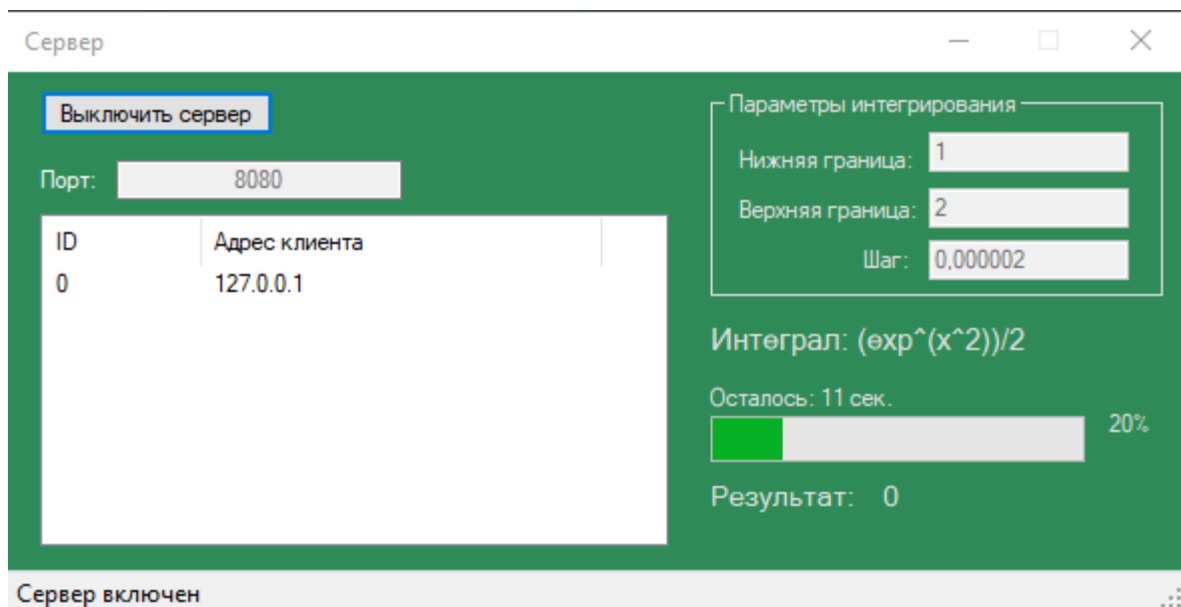


Рисунок 11 – Сервер в процессе вычисления

После того, как клиенты закончат вычисления, результат будет отображён в соответствующем поле сервера, как показано на рисунке 12. Для повторного запуска вычислений необходимо выключить сервер и запустить по новой.

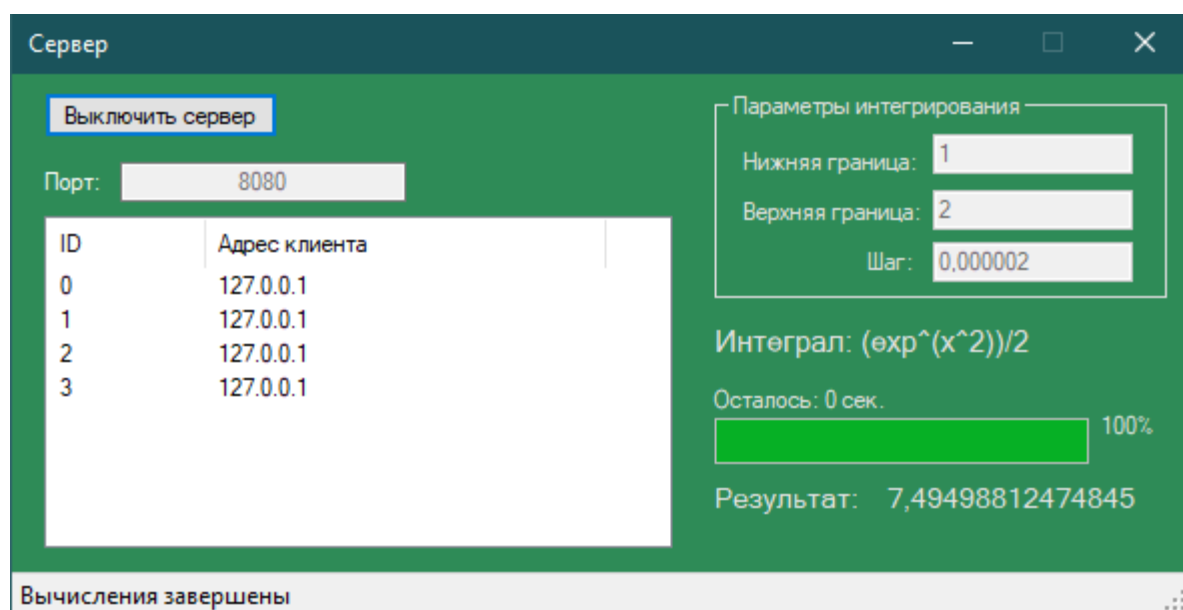


Рисунок 12 – вычисления были завершены с 4 клиентами

Программа клиент представляет из себя форму, где пользователю необходимо ввести IP-адрес и порт сервера, как показано на рисунке 13. Поля ввода должны соответствовать следующим правилам:

- В полях ввода используются только цифры;
- Каждая часть IP-адреса задаётся числом в диапазоне 0-255.

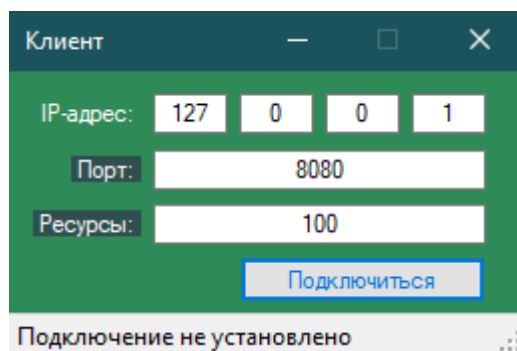


Рисунок 13 – Программа «Клиент»

После нажатия кнопки «Подключиться» будет произведена попытка подключиться к серверу по указанному IP-адресу и порту. В случае успеха, поля ввода будут заблокированы, появится кнопка «Отключиться» и в нижнем меню статусе появится соответствующее сообщение, как показано на рисунке 14.

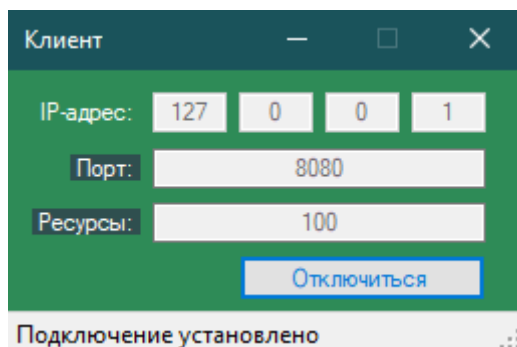


Рисунок 14 – Клиенту удалось подключиться к серверу

Заключение

При выполнении данной курсовой работы были получены навыки разработки клиент-серверных программ, позволяющих работать в рамках локальной вычислительной сети. Были освоены принципы работы с сокетами и протоколом TCP/IP и с другими функциями и объектами Win32 API. Получены навыки программирования на языке C++. Изучены возможности средств программирования и отладки среды программирования MS Visual Studio 2019.

В рамках выполнения курсовой работы была разработана программа "Параллельная программа для кластера облачного типа". Программа носит практический характер и предназначена для одновременного распределенного решения задачи несколькими компьютерами, подсоединенными к серверу. Сервер только получает данные и разделяет задачу на несколько клиентов. Клиенты производят полное решение поставленной им задачи.

В дальнейшем программу можно улучшить, усовершенствовав интерфейс пользователя, улучшив алгоритм работы как клиентского, так и серверного приложения.

Список используемых источников

1. В. Г. Олифер, Н. А. Олифер. «Компьютерные сети. Принципы, технологии, протоколы» 2016.
2. Р. Лафоре «Объектно-ориентированное программирование в С++», 4-е издание, Издательство Питер, 2004.
3. Стивен Прата. Язык программирования С++ (С++11). Лекции и упражнения, 6-е издание — М.: Вильямс, 2012. — 1248 с.
4. Б. Страуструп. Язык программирования С++. Специальное издание. Пер. с англ. — М.: Издательство Бином, 2011 г. — 1136 с.
5. Андрей Александреску. Современное проектирование на С++. Обобщенное программирование и прикладные шаблоны проектирования. Перевод с английского — Издательский дом «Вильямс», 2002 г. 336 с.

Приложение А. Файл «Client.cpp»

```
#pragma once
#pragma comment(lib, "ws2_32.lib")
#include <winsock2.h>
#include <time.h>
#include <msclr/marshal.h>

#include "Header.h"

namespace Client {

    using namespace System;
    using namespace System::ComponentModel;
    using namespace System::Collections;
    using namespace System::Windows::Forms;
    using namespace System::Data;
    using namespace System::Drawing;
    using namespace System::Threading;
    using namespace msclr::interop;

    using namespace std;

    /// <summary>
    /// Сводка для MainForm
    /// </summary>
    public ref class MainForm : public System::Windows::Forms::Form
    {
    public:
        bool isConnected;
        SOCKET connection;
    private: System::Windows::Forms::TextBox^ tbWorkload;
    private: System::Windows::Forms::Label^ label1;
    public:
        Thread ^ recvThread;

        MainForm(void)
        {
            InitializeComponent();

            isConnected = false;
            MainForm::CheckForIllegalCrossThreadCalls = false;
        }

    protected:
        /// <summary>
        /// Освободить все используемые ресурсы.
        /// </summary>
        ~MainForm()
        {
            if (components)
            {
                delete components;
            }
        }
    private: System::Windows::Forms::Label^ labelIpAddress;
    private: System::Windows::Forms::TextBox^ tbIpAddress0;
    private: System::Windows::Forms::TextBox^ tbIpAddress3;

    private: System::Windows::Forms::TextBox^ tbIpAddress2;

    private: System::Windows::Forms::TextBox^ tbIpAddress1;

    private: System::Windows::Forms::StatusStrip^ statusStrip;
    private: System::Windows::Forms::ToolStripStatusLabel^ statusLabel;

    private: System::Windows::Forms::Label^ labelPort;
    private: System::Windows::Forms::TextBox^ tbPort;

    private: System::Windows::Forms::Button^ btAction;
```

```

private:
    /// <summary>
    /// Обязательная переменная конструктора.
    /// </summary>
    System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code
    /// <summary>
    /// Требуемый метод для поддержки конструктора — не изменяйте
    /// содержимое этого метода с помощью редактора кода.
    /// </summary>
    void InitializeComponent(void)
    {
        this->labelIpAddress = (gcnew System::Windows::Forms::Label());
        this->tbIpAddress0 = (gcnew System::Windows::Forms::TextBox());
        this->tbIpAddress3 = (gcnew System::Windows::Forms::TextBox());
        this->tbIpAddress2 = (gcnew System::Windows::Forms::TextBox());
        this->tbIpAddress1 = (gcnew System::Windows::Forms::TextBox());
        this->statusStrip = (gcnew
System::Windows::Forms::StatusStrip());
        this->statusLabel = (gcnew
System::Windows::Forms::ToolStripStatusLabel());
        this->labelPort = (gcnew System::Windows::Forms::Label());
        this->tbPort = (gcnew System::Windows::Forms::TextBox());
        this->btAction = (gcnew System::Windows::Forms::Button());
        this->tbWorkload = (gcnew System::Windows::Forms::TextBox());
        this->label1 = (gcnew System::Windows::Forms::Label());
        this->statusStrip->SuspendLayout();
        this->SuspendLayout();
        //
        // labelIpAddress
        //
        this->labelIpAddress->AutoSize = true;
        this->labelIpAddress->ForeColor =
System::Drawing::SystemColors::ControlLightLight;
        this->labelIpAddress->Location = System::Drawing::Point(13,
14);

        this->labelIpAddress->Name = L"labelIpAddress";
        this->labelIpAddress->Size = System::Drawing::Size(53, 13);
        this->labelIpAddress->TabIndex = 0;
        this->labelIpAddress->Text = L"IP-адрес:";
        //
        // tbIpAddress0
        //
        this->tbIpAddress0->Location = System::Drawing::Point(72, 11);
        this->tbIpAddress0->MaxLength = 3;
        this->tbIpAddress0->Name = L"tbIpAddress0";
        this->tbIpAddress0->Size = System::Drawing::Size(37, 20);
        this->tbIpAddress0->TabIndex = 1;
        this->tbIpAddress0->Text = L"127";
        this->tbIpAddress0->TextAlign =
System::Windows::Forms::HorizontalAlignment::Center;
        this->tbIpAddress0->TextChanged += gcnew
System::EventHandler(this, &MainForm::inputData_TextChanged);
        //
        // tbIpAddress3
        //
        this->tbIpAddress3->Location = System::Drawing::Point(201, 11);
        this->tbIpAddress3->MaxLength = 3;
        this->tbIpAddress3->Name = L"tbIpAddress3";
        this->tbIpAddress3->Size = System::Drawing::Size(37, 20);
        this->tbIpAddress3->TabIndex = 4;
        this->tbIpAddress3->Text = L"1";
        this->tbIpAddress3->TextAlign =
System::Windows::Forms::HorizontalAlignment::Center;
        this->tbIpAddress3->TextChanged += gcnew
System::EventHandler(this, &MainForm::inputData_TextChanged);
        //
        // tbIpAddress2

```

```

//
this->tbIpAddress2->Location = System::Drawing::Point(158, 11);
this->tbIpAddress2->MaxLength = 3;
this->tbIpAddress2->Name = L"tbIpAddress2";
this->tbIpAddress2->Size = System::Drawing::Size(37, 20);
this->tbIpAddress2->TabIndex = 3;
this->tbIpAddress2->Text = L"0";
this->tbIpAddress2->TextAlign =
System::Windows::Forms::HorizontalAlignment::Center;
this->tbIpAddress2->TextChanged += gcnew
System::EventHandler(this, &MainForm::inputData_TextChanged);
//
// tbIpAddress1
//
this->tbIpAddress1->Location = System::Drawing::Point(115, 11);
this->tbIpAddress1->MaxLength = 3;
this->tbIpAddress1->Name = L"tbIpAddress1";
this->tbIpAddress1->Size = System::Drawing::Size(37, 20);
this->tbIpAddress1->TabIndex = 2;
this->tbIpAddress1->Text = L"0";
this->tbIpAddress1->TextAlign =
System::Windows::Forms::HorizontalAlignment::Center;
this->tbIpAddress1->TextChanged += gcnew
System::EventHandler(this, &MainForm::inputData_TextChanged);
//
// statusStrip
//
this->statusStrip->Items->AddRange(gcnew cli::array<
System::Windows::Forms::ToolStripItem^ >(1) { this->statusLabel });
this->statusStrip->Location = System::Drawing::Point(0, 120);
this->statusStrip->Name = L"statusStrip";
this->statusStrip->Size = System::Drawing::Size(258, 22);
this->statusStrip->TabIndex = 50;
this->statusStrip->Text = L"statusStrip";
//
// statusLabel
//
this->statusLabel->BackColor =
System::Drawing::Color::Transparent;
this->statusLabel->ForeColor =
System::Drawing::SystemColors::ControlText;
this->statusLabel->Name = L"statusLabel";
this->statusLabel->Size = System::Drawing::Size(174, 17);
this->statusLabel->Text = L"Подключение не установлено";
//
// labelPort
//
this->labelPort->AutoSize = true;
this->labelPort->BackColor =
System::Drawing::Color::DarkSlateGray;
this->labelPort->ForeColor =
System::Drawing::SystemColors::ControlLightLight;
this->labelPort->Location = System::Drawing::Point(31, 42);
this->labelPort->Name = L"labelPort";
this->labelPort->Size = System::Drawing::Size(35, 13);
this->labelPort->TabIndex = 51;
this->labelPort->Text = L"Порт:";
this->labelPort->TextAlign =
System::Drawing::ContentAlignment::TopRight;
//
// tbPort
//
this->tbPort->Location = System::Drawing::Point(72, 39);
this->tbPort->MaxLength = 5;
this->tbPort->Name = L"tbPort";
this->tbPort->Size = System::Drawing::Size(166, 20);
this->tbPort->TabIndex = 5;
this->tbPort->Text = L"8080";
this->tbPort->TextAlign =
System::Windows::Forms::HorizontalAlignment::Center;
this->tbPort->TextChanged += gcnew System::EventHandler(this,
&MainForm::inputData_TextChanged);

```

```

        //
        // btAction
        //
        this->btAction->ForeColor =
System::Drawing::SystemColors::Highlight;
        this->btAction->Location = System::Drawing::Point(115, 92);
        this->btAction->Name = L"btAction";
        this->btAction->Size = System::Drawing::Size(123, 23);
        this->btAction->TabIndex = 0;
        this->btAction->Text = L"Подключиться";
        this->btAction->UseVisualStyleBackColor = true;
        this->btAction->Click += gcnew System::EventHandler(this,
&MainForm::ButtonAction_Click);
        //
        // tbWorkload
        //
        this->tbWorkload->Location = System::Drawing::Point(72, 66);
        this->tbWorkload->Name = L"tbWorkload";
        this->tbWorkload->Size = System::Drawing::Size(166, 20);
        this->tbWorkload->TabIndex = 52;
        this->tbWorkload->Text = L"100";
        this->tbWorkload->TextAlign =
System::Windows::Forms::HorizontalAlignment::Center;
        //
        // label1
        //
        this->label1->AutoSize = true;
        this->label1->BackColor =
System::Drawing::Color::DarkSlateGray;
        this->label1->ForeColor =
System::Drawing::SystemColors::ControlLightLight;
        this->label1->Location = System::Drawing::Point(12, 69);
        this->label1->Name = L"label1";
        this->label1->Size = System::Drawing::Size(54, 13);
        this->label1->TabIndex = 53;
        this->label1->Text = L"Ресурсы:";
        this->label1->TextAlign =
System::Drawing::ContentAlignment::TopRight;
        //
        // MainForm
        //
        this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);
        this->AutoScaleMode =
System::Windows::Forms::AutoScaleMode::Font;
        this->BackColor = System::Drawing::Color::SeaGreen;
        this->ClientSize = System::Drawing::Size(258, 142);
        this->Controls->Add(this->label1);
        this->Controls->Add(this->tbWorkload);
        this->Controls->Add(this->btAction);
        this->Controls->Add(this->tbPort);
        this->Controls->Add(this->labelPort);
        this->Controls->Add(this->statusStrip);
        this->Controls->Add(this->tbIpAddress3);
        this->Controls->Add(this->tbIpAddress2);
        this->Controls->Add(this->tbIpAddress1);
        this->Controls->Add(this->tbIpAddress0);
        this->Controls->Add(this->labelIpAddress);
        this->ForeColor =
System::Drawing::SystemColors::ControlDarkDark;
        this->FormBorderStyle =
System::Windows::Forms::FormBorderStyle::FixedSingle;
        this->MaximizeBox = false;
        this->Name = L"MainForm";
        this->ShowIcon = false;
        this->StartPosition =
System::Windows::Forms::FormStartPosition::CenterScreen;
        this->Text = L"Клиент";
        this->FormClosed += gcnew
System::Windows::Forms::FormClosedEventHandler(this,
&MainForm::MainForm_FormClosed);
        this->Load += gcnew System::EventHandler(this,
&MainForm::MainForm_Load);

```

```

        this->statusStrip->ResumeLayout(false);
        this->statusStrip->PerformLayout();
        this->ResumeLayout(false);
        this->PerformLayout();
    }
#pragma endregion

    // Обработчик кнопки "Подключиться/Отключиться"
    private: System::Void ButtonAction_Click(System::Object^ sender,
        System::EventArgs^ e) {
        if (isConnected) {
            isConnected = false;
            closesocket(connection);
            WSACleanup();
            tbIpAddress0->Enabled = true;
            tbIpAddress1->Enabled = true;
            tbIpAddress2->Enabled = true;
            tbIpAddress3->Enabled = true;
            tbPort->Enabled = true;
            tbWorkload->Enabled = true;
            statusLabel->Text = "Подключение не установлено";
            btAction->Text = "Подключиться";
        }
        else {
            if (WSAStartup(MAKEWORD(2, 0), &wsadata) != 0) //
инициализировать библиотеку WinSock
            {
                statusLabel->Text = "Ошибка инициализации, повторите
попытку";
                return;
            }

            if ((connection = socket(AF_INET, SOCK_STREAM, NULL)) ==
INVALID_SOCKET)
            {
                statusLabel->Text = "Ошибка инициализации, повторите
попытку";
                WSACleanup();
                return;
            }

            String^ ip = tbIpAddress0->Text;
            ip += "." + tbIpAddress1->Text + "." + tbIpAddress2->Text + "."
+ tbIpAddress3->Text;
            marshal_context^ ctx = gcnew marshal_context();

            SOCKADDR_IN addr;
            addr.sin_addr.s_addr = inet_addr(ctx->marshal_as<const
char*>(ip));
            addr.sin_port = htons(Convert::ToUInt16(tbPort->Text));
            addr.sin_family = AF_INET;

            if (connect(connection, (SOCKADDR*)& addr, sizeof(addr))) {
                statusLabel->Text = "Сервер не отвечает";
                closesocket(connection);
                WSACleanup();
                return;
            }

            recvThread = gcnew Thread(gcnew ThreadStart(this,
&MainForm::recvFunc));
            recvThread->Start();

            isConnected = true;
            tbIpAddress0->Enabled = false;
            tbIpAddress1->Enabled = false;
            tbIpAddress2->Enabled = false;
            tbIpAddress3->Enabled = false;
            tbPort->Enabled = false;
            workload = (double)Convert::ToInt32(tbWorkload->Text) / 100;
            tbWorkload->Enabled = false;

```

```

        statusLabel->Text = "Подключение установлено";
        btAction->Text = "Отключиться";
    }

    double f(double currentX)
    {
        return (exp(currentX*currentX)/currentX);
    }

    System::Void recvFunc() {
        char msg[1024];
        while (1)
        {
            // Если сервер не отвечает - ошибка
            if (recv(connection, &msg[0], sizeof(msg), 0) <
1)
            {
                break;
            }

            task* recvTask = (task*)& msg[0];

            clock_t tBegin, tEnd;

            //Время работы и задержки
            double tFull = 0, tSleep = 0, tDeltaSleep = 0;

            double currentX = recvTask->downEdge;
            double result = 0;

            double step = (recvTask->upEdge - recvTask-
>downEdge) / recvTask->stepcount;

            tBegin = clock();

            for (int i = 0; i < recvTask->stepcount; i++)
            {
                // Десять раз измеряем время работы, исходя
из результатов формируем соответствующее время задержки
                if (i % (10000) == 0)
                {
                    tFull = (double(clock() - tBegin)) /
CLOCKS_PER_SEC;
                    tDeltaSleep = ((1 - workload) *
tFull) - tSleep;

                    if (tDeltaSleep >= 0)
                    {
                        clock_t tSleepBegin = clock();
                        Sleep(tDeltaSleep * 1000);
                        tSleep += (double(clock() -
tSleepBegin)) / CLOCKS_PER_SEC;
                    }

                    }
                // Вычисления
                //double result1 = (currentX* currentX) +
1;
                result += exp(pow(currentX,2))/2;
                currentX += step;
            }

            // Корректировка результата
            result = step * (result + (exp(pow(recvTask-
>downEdge,2))/2));

            resultTask resultData;
            resultData.result = result;
            resultData.time = tFull + tSleep;
            resultData.idTask = recvTask->idTask;
            // Отправка результата серверу

```

```

        send(connection, (char*)& resultData,
sizeof(resultData), 0);
    }

    // Отключились от сервера
    isConnected = false;
    closesocket(connection);
    WSACleanup();
    tbIpAddress0->Enabled = true;
    tbIpAddress1->Enabled = true;
    tbIpAddress2->Enabled = true;
    tbIpAddress3->Enabled = true;
    tbPort->Enabled = true;
    statusLabel->Text = "Подключение не установлено";
    btAction->Text = "Подключиться";
}

// Проверяет корректность введённых данных
private: System::Void inputData_TextChanged(System::Object^ sender,
System::EventArgs^ e) {
    try
    {
        Convert::ToUInt16(tbPort->Text);
        if ((Convert::ToUInt16(tbIpAddress0->Text) > 255) ||
            (Convert::ToUInt16(tbIpAddress1->Text) > 255) ||
            (Convert::ToUInt16(tbIpAddress2->Text) > 255) ||
            (Convert::ToUInt16(tbIpAddress3->Text) > 255))
        {
            statusLabel->ForeColor = Color::Red;
            statusLabel->Text = "IP-адрес некорректен";
            btAction->Enabled = false;
        }
        else {
            btAction->Enabled = true;
            statusLabel->ForeColor = Color::Black;
            statusLabel->Text = "Подключение не установлено";
        }
    }
    catch (...)
    {
        statusLabel->ForeColor = Color::Red;
        statusLabel->Text = "Введённые данные не корректны";
        btAction->Enabled = false;
    }
}

private: System::Void MainForm_FormClosed(System::Object^ sender,
System::Windows::Forms::FormClosedEventArgs^ e) {
    closesocket(connection);
    WSACleanup();
}

private: System::Void MainForm_Load(System::Object^ sender,
System::EventArgs^ e) {
}

};
}

```

Приложение В– Файл «Server.cpp»

```

#pragma once
#pragma comment(lib, "ws2_32.lib")
#include <WinSock2.h>
#include <list>
#include <ws2tcpip.h>
#include <msclr/marshal.h>

```

```

#include "Header.h"

namespace Server {

using namespace System;
using namespace System::ComponentModel;
using namespace System::Collections;
using namespace System::Windows::Forms;
using namespace System::Data;
using namespace System::Drawing;
using namespace System::Threading;
using namespace msclr::interop;

using namespace std;

/// <summary>
/// Сводка для MainForm
/// </summary>
public ref class MainForm : public System::Windows::Forms::Form
{
public:
    MainForm(void)
    {
        InitializeComponent();
        isOn = false;

        MainForm::CheckForIllegalCrossThreadCalls = false;
    }

protected:
    /// <summary>
    /// Освободить все используемые ресурсы.
    /// </summary>
    ~MainForm()
    {
        if (components)
        {
            delete components;
        }
    }

private: System::Windows::Forms::Button^ btAction;
private: System::Windows::Forms::Label^ labelPort;
private: System::Windows::Forms::TextBox^ tbPort;

private: System::Windows::Forms::StatusStrip^ statusStrip;
private: System::Windows::Forms::ToolStripStatusLabel^ statusLabel;

private: System::Windows::Forms::ProgressBar^ progressBar;
private: System::Windows::Forms::Label^ labelProgress;
private: System::Windows::Forms::Label^ lbTimeLeft;
private: System::Windows::Forms::Label^ lbProgress;

private: System::Windows::Forms::Label^ labelIntegr;
private: System::Windows::Forms::ListView^ lvClients;

```

```

private: System::Windows::Forms::Label^ labelResult;

private: System::Windows::Forms::Label^ lbResult;


private: System::Windows::Forms::ColumnHeader^ columnId;
private: System::Windows::Forms::ColumnHeader^ columnName;
private: System::Windows::Forms::Label^ label1;
private: System::Windows::Forms::TextBox^ step;
private: System::Windows::Forms::Label^ labelUpEdge;
private: System::Windows::Forms::TextBox^ tbDownEdge;
private: System::Windows::Forms::TextBox^ tbUpEdge;
private: System::Windows::Forms::Label^ labelDownEdge;
private: System::Windows::Forms::GroupBox^ gbParamsIntegr;


private:
    /// <summary>
    /// Обязательная переменная конструктора.
    /// </summary>
    System::ComponentModel::Container^ components;

#pragma region Windows Form Designer generated code
    /// <summary>
    /// Требуемый метод для поддержки конструктора — не изменяйте
    /// содержимое этого метода с помощью редактора кода.
    /// </summary>
    void InitializeComponent(void)
    {
        this->btAction = (gcnew System::Windows::Forms::Button());
        this->labelPort = (gcnew System::Windows::Forms::Label());
        this->tbPort = (gcnew System::Windows::Forms::TextBox());
        this->statusStrip = (gcnew
System::Windows::Forms::StatusStrip());
        this->statusLabel = (gcnew
System::Windows::Forms::ToolStripStatusLabel());
        this->progressBar = (gcnew
System::Windows::Forms::ProgressBar());
        this->labelProgress = (gcnew System::Windows::Forms::Label());
        this->lbTimeLeft = (gcnew System::Windows::Forms::Label());
        this->lbProgress = (gcnew System::Windows::Forms::Label());
        this->labelIntegr = (gcnew System::Windows::Forms::Label());
        this->lvClients = (gcnew System::Windows::Forms::ListView());
        this->columnId = (gcnew
System::Windows::Forms::ColumnHeader());
        this->columnName = (gcnew
System::Windows::Forms::ColumnHeader());
        this->labelResult = (gcnew System::Windows::Forms::Label());
        this->lbResult = (gcnew System::Windows::Forms::Label());
        this->label1 = (gcnew System::Windows::Forms::Label());
        this->step = (gcnew System::Windows::Forms::TextBox());
        this->labelUpEdge = (gcnew System::Windows::Forms::Label());
        this->tbDownEdge = (gcnew System::Windows::Forms::TextBox());
        this->tbUpEdge = (gcnew System::Windows::Forms::TextBox());
        this->labelDownEdge = (gcnew System::Windows::Forms::Label());
        this->gbParamsIntegr = (gcnew
System::Windows::Forms::GroupBox());
        this->statusStrip->SuspendLayout();
        this->gbParamsIntegr->SuspendLayout();
        this->SuspendLayout();
    }
    //

```

```

        // btAction
        //
        this->btAction->Location = System::Drawing::Point(16, 9);
        this->btAction->Name = L"btAction";
        this->btAction->Size = System::Drawing::Size(117, 23);
        this->btAction->TabIndex = 0;
        this->btAction->Text = L"Включить сервер";
        this->btAction->UseVisualStyleBackColor = true;
        this->btAction->Click += gcnew System::EventHandler(this,
&MainForm::BtAction_Click);
        //
        // labelPort
        //
        this->labelPort->AutoSize = true;
        this->labelPort->ForeColor =
System::Drawing::SystemColors::ControlLightLight;
        this->labelPort->Location = System::Drawing::Point(13, 47);
        this->labelPort->Name = L"labelPort";
        this->labelPort->Size = System::Drawing::Size(35, 13);
        this->labelPort->TabIndex = 1;
        this->labelPort->Text = L"Порт:";
        //
        // tbPort
        //
        this->tbPort->Location = System::Drawing::Point(54, 44);
        this->tbPort->Name = L"tbPort";
        this->tbPort->ScrollBars =
System::Windows::Forms::ScrollBars::Both;
        this->tbPort->Size = System::Drawing::Size(143, 20);
        this->tbPort->TabIndex = 2;
        this->tbPort->Text = L"8080";
        this->tbPort->TextAlign =
System::Windows::Forms::HorizontalAlignment::Center;
        this->tbPort->TextChanged += gcnew System::EventHandler(this,
&MainForm::inputData_TextChanged);
        //
        // statusStrip
        //
        this->statusStrip->Items->AddRange(gcnew cli::array<
System::Windows::Forms::ToolStripItem^ >(1) { this->statusLabel });
        this->statusStrip->Location = System::Drawing::Point(0, 249);
        this->statusStrip->Name = L"statusStrip";
        this->statusStrip->Size = System::Drawing::Size(589, 22);
        this->statusStrip->TabIndex = 7;
        this->statusStrip->Text = L"statusStrip1";
        //
        // statusLabel
        //
        this->statusLabel->BackColor =
System::Drawing::Color::Transparent;
        this->statusLabel->Name = L"statusLabel";
        this->statusLabel->Size = System::Drawing::Size(108, 17);
        this->statusLabel->Text = L"Сервер выключен";
        //
        // progressBar
        //
        this->progressBar->Location = System::Drawing::Point(351,
172);
        this->progressBar->Name = L"progressBar";
        this->progressBar->Size = System::Drawing::Size(187, 23);
        this->progressBar->TabIndex = 9;
        //
        // labelProgress
        //

```

```

        this->labelProgress->AutoSize = true;
        this->labelProgress->ForeColor =
System::Drawing::SystemColors::ControlLight;
        this->labelProgress->Location = System::Drawing::Point(348,
156);
        this->labelProgress->Name = L"labelProgress";
        this->labelProgress->Size = System::Drawing::Size(59, 13);
        this->labelProgress->TabIndex = 10;
        this->labelProgress->Text = L"Осталось:";
        //
        // lbTimeLeft
        //
        this->lbTimeLeft->AutoSize = true;
        this->lbTimeLeft->ForeColor =
System::Drawing::SystemColors::ControlLight;
        this->lbTimeLeft->Location = System::Drawing::Point(403, 156);
        this->lbTimeLeft->Name = L"lbTimeLeft";
        this->lbTimeLeft->Size = System::Drawing::Size(37, 13);
        this->lbTimeLeft->TabIndex = 11;
        this->lbTimeLeft->Text = L"0 сек.";
        //
        // lbProgress
        //
        this->lbProgress->AutoSize = true;
        this->lbProgress->ForeColor =
System::Drawing::SystemColors::ControlLight;
        this->lbProgress->Location = System::Drawing::Point(541, 169);
        this->lbProgress->MaximumSize = System::Drawing::Size(33, 13);
        this->lbProgress->MinimumSize = System::Drawing::Size(33, 13);
        this->lbProgress->Name = L"lbProgress";
        this->lbProgress->Size = System::Drawing::Size(33, 13);
        this->lbProgress->TabIndex = 12;
        this->lbProgress->Text = L"0%";
        this->lbProgress->TextAlign =
System::Drawing::ContentAlignment::MiddleRight;
        //
        // labelIntegr
        //
        this->labelIntegr->AutoSize = true;
        this->labelIntegr->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 11,
System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,
        static_cast<System::Byte>(204)));
        this->labelIntegr->ForeColor =
System::Drawing::SystemColors::ControlLight;
        this->labelIntegr->Location = System::Drawing::Point(348,
124);
        this->labelIntegr->Name = L"labelIntegr";
        this->labelIntegr->Size = System::Drawing::Size(165, 18);
        this->labelIntegr->TabIndex = 13;
        this->labelIntegr->Text = L"Интервал:  $(\exp^x - 1)/2$ ";
        this->labelIntegr->Click += gcnew System::EventHandler(this,
&MainForm::labelIntegr_Click);
        //
        // lvClients
        //
        this->lvClients->Columns->AddRange(gcnew cli::array<
System::Windows::Forms::ColumnHeader^ >(2) { this->columnId, this->
columnName });
        this->lvClients->HeaderStyle =
System::Windows::Forms::ColumnHeaderStyle::Nonclickable;
        this->lvClients->HideSelection = false;
        this->lvClients->LabelWrap = false;
        this->lvClients->Location = System::Drawing::Point(16, 71);

```

```

        this->lvClients->MultiSelect = false;
        this->lvClients->Name = L"lvClients";
        this->lvClients->Size = System::Drawing::Size(314, 166);
        this->lvClients->TabIndex = 14;
        this->lvClients->TabStop = false;
        this->lvClients->UseCompatibleStateImageBehavior = false;
        this->lvClients->View = System::Windows::Forms::View::Details;
        this->lvClients->SelectedIndexChanged += gcnew
System::EventHandler(this, &MainForm::lvClients_SelectedIndexChanged);
        //
        // columnId
        //
        this->columnId->Text = L"ID";
        this->columnId->Width = 79;
        //
        // columnName
        //
        this->columnName->Text = L"Адрес клиента";
        this->columnName->Width = 200;
        //
        // labelResult
        //
        this->labelResult->AutoSize = true;
        this->labelResult->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 10,
System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,
        static_cast<System::Byte>(204)));
        this->labelResult->ForeColor =
System::Drawing::SystemColors::ControlLight;
        this->labelResult->Location = System::Drawing::Point(348,
203);

        this->labelResult->Name = L"labelResult";
        this->labelResult->Size = System::Drawing::Size(80, 17);
        this->labelResult->TabIndex = 15;
        this->labelResult->Text = L"Результат:";
        //
        // lbResult
        //
        this->lbResult->AutoSize = true;
        this->lbResult->Font = (gcnew
System::Drawing::Font(L"Microsoft Sans Serif", 10,
System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,
        static_cast<System::Byte>(204)));
        this->lbResult->ForeColor =
System::Drawing::SystemColors::ControlLight;
        this->lbResult->Location = System::Drawing::Point(434, 203);
        this->lbResult->Name = L"lbResult";
        this->lbResult->Size = System::Drawing::Size(16, 17);
        this->lbResult->TabIndex = 16;
        this->lbResult->Text = L"0";
        //
        // label1
        //
        this->label1->AutoSize = true;
        this->label1->ForeColor =
System::Drawing::SystemColors::ControlLight;
        this->label1->Location = System::Drawing::Point(73, 78);
        this->label1->Name = L"label1";
        this->label1->Size = System::Drawing::Size(30, 13);
        this->label1->TabIndex = 20;
        this->label1->Text = L"War:";
        this->label1->Click += gcnew System::EventHandler(this,
&MainForm::label1_Click);
        //

```

```

        // step
        //
        this->step->Location = System::Drawing::Point(109, 75);
        this->step->MaxLength = 3;
        this->step->Name = L"step";
        this->step->Size = System::Drawing::Size(100, 20);
        this->step->TabIndex = 22;
        this->step->Text = L"0,000002";
        this->step->TextChanged += gcnew System::EventHandler(this,
&MainForm::textBox2_TextChanged);
        //
        // labelUpEdge
        //
        this->labelUpEdge->AutoSize = true;
        this->labelUpEdge->ForeColor =
System::Drawing::SystemColors::ControlLight;
        this->labelUpEdge->Location = System::Drawing::Point(11, 53);
        this->labelUpEdge->Name = L"labelUpEdge";
        this->labelUpEdge->Size = System::Drawing::Size(96, 13);
        this->labelUpEdge->TabIndex = 6;
        this->labelUpEdge->Text = L"Верхняя граница:";
        //
        // tbDownEdge
        //
        this->tbDownEdge->Enabled = false;
        this->tbDownEdge->Location = System::Drawing::Point(109, 21);
        this->tbDownEdge->Name = L"tbDownEdge";
        this->tbDownEdge->Size = System::Drawing::Size(100, 20);
        this->tbDownEdge->TabIndex = 3;
        this->tbDownEdge->Text = L"1";
        this->tbDownEdge->TextChanged += gcnew
System::EventHandler(this, &MainForm::inputData_TextChanged);
        //
        // tbUpEdge
        //
        this->tbUpEdge->Enabled = false;
        this->tbUpEdge->Location = System::Drawing::Point(109, 49);
        this->tbUpEdge->Name = L"tbUpEdge";
        this->tbUpEdge->Size = System::Drawing::Size(100, 20);
        this->tbUpEdge->TabIndex = 4;
        this->tbUpEdge->Text = L"2";
        this->tbUpEdge->TextChanged += gcnew
System::EventHandler(this, &MainForm::inputData_TextChanged);
        //
        // labelDownEgde
        //
        this->labelDownEgde->AutoSize = true;
        this->labelDownEgde->ForeColor =
System::Drawing::SystemColors::ControlLight;
        this->labelDownEgde->Location = System::Drawing::Point(11,
28);
        this->labelDownEgde->Name = L"labelDownEgde";
        this->labelDownEgde->Size = System::Drawing::Size(94, 13);
        this->labelDownEgde->TabIndex = 5;
        this->labelDownEgde->Text = L"Нижняя граница:";
        //
        // gbParamsIntegr
        //
        this->gbParamsIntegr->Controls->Add(this->step);
        this->gbParamsIntegr->Controls->Add(this->label1);
        this->gbParamsIntegr->Controls->Add(this->labelDownEgde);
        this->gbParamsIntegr->Controls->Add(this->tbUpEdge);
        this->gbParamsIntegr->Controls->Add(this->tbDownEdge);
        this->gbParamsIntegr->Controls->Add(this->labelUpEdge);

```

```

        this->gbParamsIntegr->ForeColor =
System::Drawing::SystemColors::ControlLight;
        this->gbParamsIntegr->Location = System::Drawing::Point(351,
9);
        this->gbParamsIntegr->Name = L"gbParamsIntegr";
        this->gbParamsIntegr->Size = System::Drawing::Size(226, 104);
        this->gbParamsIntegr->TabIndex = 8;
        this->gbParamsIntegr->TabStop = false;
        this->gbParamsIntegr->Text = L"Параметры интегрирования";
        //
        // MainForm
        //
        this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);
        this->AutoScaleMode =
System::Windows::Forms::AutoScaleMode::Font;
        this->BackColor = System::Drawing::Color::SeaGreen;
        this->ClientSize = System::Drawing::Size(589, 271);
        this->Controls->Add(this->lbResult);
        this->Controls->Add(this->labelResult);
        this->Controls->Add(this->lvClients);
        this->Controls->Add(this->labelIntegr);
        this->Controls->Add(this->lbProgress);
        this->Controls->Add(this->lbTimeLeft);
        this->Controls->Add(this->labelProgress);
        this->Controls->Add(this->progressBar);
        this->Controls->Add(this->gbParamsIntegr);
        this->Controls->Add(this->statusStrip);
        this->Controls->Add(this->tbPort);
        this->Controls->Add(this->labelPort);
        this->Controls->Add(this->btAction);
        this->FormBorderStyle =
System::Windows::Forms::FormBorderStyle::FixedSingle;
        this->MaximizeBox = false;
        this->Name = L"MainForm";
        this->ShowIcon = false;
        this->StartPosition =
System::Windows::Forms::FormStartPosition::CenterScreen;
        this->Text = L"Сервер";
        this->FormClosed += gcnew
System::Windows::Forms::FormClosedEventHandler(this,
&MainForm::MainForm_FormClosed);
        this->Load += gcnew System::EventHandler(this,
&MainForm::MainForm_Load);
        this->statusStrip->ResumeLayout(false);
        this->statusStrip->PerformLayout();
        this->gbParamsIntegr->ResumeLayout(false);
        this->gbParamsIntegr->PerformLayout();
        this->ResumeLayout(false);
        this->PerformLayout();

    }
#pragma endregion

public:
    bool isOn;
    double result;
    int countTableId;

    value class Client {
    public:
        int id;
        SOCKET sClient;
        bool isFree;
    };
};

```

```

private:
    Thread ^ threadClientHandler;
    Thread^ threadClient;
    Thread^ threadServer;

    //    Нажали кнопку действия
    System::Void BtAction_Click(System::Object^ sender,
System::EventArgs^ e) {
        if (isOn) {
            isOn = false;
            closesocket(sListen);
            WSACleanup();

            step->Enabled = true;
            tbPort->Enabled = true;
            tbDownEdge->Enabled = true;
            tbUpEdge->Enabled = true;
            //tbWorkload->Enabled = true;
            statusLabel->Text = "Сервер выключен";
            btAction->Text = "Включить сервер";

            lbTimeLeft->Text = "0 сек.";
            lvClients->Items->Clear();
        }
        else {
            InitializeCriticalSection(&csTaskList);
            InitializeCriticalSection(&csClientsList);

            if (WSAStartup(MAKEWORD(2, 0), &wsadata) != 0) //
инициализировать библиотеку WinSock
            {
                statusLabel->Text = "Ошибка инициализации,
повторите попытку";
                return;
            }

            if ((sListen = socket(AF_INET, SOCK_STREAM, NULL)) ==
INVALID_SOCKET)
            {
                statusLabel->Text = "Ошибка инициализации,
повторите попытку";
                WSACleanup();
                return;
            }

            addrServer.sin_family = AF_INET;
            addrServer.sin_port = htons(Convert::ToUInt16(tbPort-
>Text));
            addrServer.sin_addr.s_addr = htonl(INADDR_ANY);

            if (bind(sListen, (sockaddr*)& addrServer,
sizeof(addrServer)) == SOCKET_ERROR)
            {
                statusLabel->Text = "Ошибка инициализации,
повторите попытку";
                closesocket(sListen);
                WSACleanup();
                return;
            }

            if (listen(sListen, 0x100))
            {

```

```

        statusLabel->Text = "Ошибка инициализации,
повторите попытку";
        closesocket(sListen);
        WSACleanup();
        return;
    }

    countTableId = 0;
    // открываем порт
    threadClientHandler = gcnew Thread(gcnew
ThreadStart(this, &MainForm::clientHandler));
    threadClientHandler->Start();
    // настраиваем интерфейс
    isOn = true;
    step->Enabled = false;
    tbPort->Enabled = false;
    tbDownEdge->Enabled = false;
    tbUpEdge->Enabled = false;
    //tbWorkload->Enabled = false;
    statusLabel->Text = "Сервер включен";
    btAction->Text = "Выключить сервер";

    lbResult->Text = "0";
    progressBar->Value = 0;
    lbProgress->Text = "0%";

    threadServer = gcnew Thread(gcnew ThreadStart(this,
&MainForm::serverManager));
    threadServer->Start();
}

}

void clientHandler() {
    SOCKADDR_IN addrClient;
    int sizeofAddrClient = sizeof(addrClient);
    SOCKET sClient;

    // Цикл обработки входящих соединений, пока сервер включен
    while (isOn)
    {
        if ((sClient = accept(sListen, (SOCKADDR*)& addrClient,
&sizeofAddrClient)) != SOCKET_ERROR)
        {
            Client newClient;
            newClient.id = countTableId;
            newClient.sClient = sClient;
            newClient.isFree = true;
            // Добавить в таблицу
            char str[INET_ADDRSTRLEN];
            inet_ntop(AF_INET, &(addrClient.sin_addr), str,
INET_ADDRSTRLEN);
            String^ ipClient = marshal_as<String^>(str);
            ListViewItem^ item = gcnew
ListViewItem(Convert::ToString(countTableId));
            item->SubItems->Add(ipClient);
            lvClients->Items->Add(item);
            countTableId++;
            // Для каждого клиента создается нить
            threadClient = gcnew Thread(gcnew
ParameterizedThreadStart(this, &MainForm::clientFunc));
            threadClient->Start(newClient);
        }
    }
}
}

```

```

void clientFunc(Object^ data)
{
    Client client = Client(data);
    ClientInfo currentClient;
    currentClient.sClient = client.sClient;
    currentClient.isFree = client.isFree;

    clientsList.push_back(currentClient);

    list<ClientInfo>::iterator thisClient = clientsList.end();
    thisClient--;

    char buff[1024];

    // Принимаем результаты от клиента
    while (1)
    {
        if (recv(currentClient.sClient, &buff[0], sizeof(buff),
0) < 1)
        {
            break;
        }

        EnterCriticalSection(&csTaskList);

        resultTask* result = (resultTask*)& buff[0];
        resultsList.push_back(*result);

        removeIdTask = result->idTask;
        tasksList.remove_if(removeFunc);

        // Интерфейс прогресса
        //setTextInProcessStatusTextBox("Выполнено: " + (100 *
(((double)TASKSNUM - (double)tasks_list.size()) /
(double)TASKSNUM)).ToString() + " процентов" + " ( " + (int(result_data-
>execute_time * tasks_list.size()) / clients_list.size()).ToString() + "
секунд осталось)");
        progressBar->Value = (100 * (((double)TASKSCOUNT -
(double)tasksList.size()) / (double)TASKSCOUNT));
        lbProgress->Text = (100 * (((double)TASKSCOUNT -
(double)tasksList.size()) / (double)TASKSCOUNT)).ToString() + "%";
        lbTimeLeft->Text = (int(result->time * tasksList.size())
/ clientsList.size()).ToString() + " сек.";

        LeaveCriticalSection(&csTaskList);

        EnterCriticalSection(&csClientsList);
        thisClient->isFree = true;
        LeaveCriticalSection(&csClientsList);
    }

    // Клиент отключился
    EnterCriticalSection(&csTaskList);

    for (std::list<taskStatus>::iterator iter = tasksList.begin();
iter != tasksList.end(); iter++)
    {
        if (iter->idTask == thisClient->idTask)
        {
            iter->status = statusTask::freeNow;
        }
    }
}

```

```

        LeaveCriticalSection(&csTaskList);

        EnterCriticalSection(&csClientsList);
        clientsList.erase(thisClient);
        LeaveCriticalSection(&csClientsList);

        try
        {
            for (int i = 0; i < lvClients->Items->Count; i++)
            {
                if (Convert::ToInt32(lvClients->Items[i]->Text) ==
client.id)
                    lvClients->Items[i]->Remove();
            }
        }
        catch (...) {}
        closesocket(currentClient.sClient);
    }

    void serverManager()
    {
        task* paramsIntegr =
getParamsIntegr(Convert::ToInt32(tbDownEdge->Text),
Convert::ToInt32(tbUpEdge->Text), 0);

        // Формирование списка задач
        for (int i = 0; i < TASKSCOUNT; i++)
        {
            taskStatus task;
            task.status = statusTask::freeNow;
            task.idTask = i;
            tasksList.push_back(task);
        }

        // Отправляем задачи пока включён сервер и список задач не
пуст
        while (isOn && (tasksList.size() > 0))
        {
            // Подбираем свободного клиента
            list<ClientInfo>::iterator clientFree;
            bool isFoundClient = false;

            while (1)
            {
                EnterCriticalSection(&csClientsList);

                if (isOn && (tasksList.size() > 0))
                {
                    for (list<ClientInfo>::iterator
currentClient = clientsList.begin(); currentClient != clientsList.end();
currentClient++)
                    {
                        if (currentClient->isFree == true)
                        {
                            clientFree = currentClient;
                            isFoundClient = true;
                            break;
                        }
                    }
                }

                if (isFoundClient == true)
                {
                    LeaveCriticalSection(&csClientsList);
                    Sleep(20);
                }
            }
        }
    }

```

```

        break;
    }
}
else
{
    isFoundClient = false;
    break;
}

LeaveCriticalSection(&csClientsList);
Sleep(20);
}

EnterCriticalSection(&csTaskList);
EnterCriticalSection(&csClientsList);

// Подбираем свободному клиенту задачу
if ((tasksList.size() > 0) && isFoundClient)
{
    for (std::list<taskStatus>::iterator currentTask =
tasksList.begin(); currentTask != tasksList.end(); currentTask++)
    {
        if (currentTask->status ==
statusTask::freeNow)
        {
            if (send(clientFree->sClient, (char*)&
paramsIntegr[currentTask->idTask], sizeof(*paramsIntegr), 0) < 1)
            {
                break;
            }
            else
            {
                clientFree->isFree = false;
                clientFree->idTask =
currentTask->idTask;
                currentTask->status =
statusTask::busy;
                break;
            }
        }
    }
}

LeaveCriticalSection(&csClientsList);
LeaveCriticalSection(&csTaskList);

}
// Все задачи решены или что-то произошло
// Задача НЕ БЫЛА РЕШЕНА полностью при отключении сервера
if (tasksList.size() != 0 && !isOn)
{
    statusLabel->Text = "Сервер был отключён";
}
// Задача РЕШЕНА полностью
else
{
    result = 0;
    for (std::list<resultTask>::iterator currentResult =
resultsList.begin(); currentResult != resultsList.end(); currentResult++)
    {
        result += currentResult->result;
    }
    statusLabel->Text = "Вычисления завершены";
    lbResult->Text = result.ToString();
}

```

```

        }

        tasksList.clear();
        resultsList.clear();
        result = 0;
    }

    // Проверка введенных данных во всех полях ввода
private: System::Void inputData_TextChanged(System::Object^ sender,
System::EventArgs^ e) {
    try
    {
        Convert::ToUInt16(tbPort->Text);
        if (Convert::ToUInt16(tbDownEdge->Text) >=
Convert::ToUInt16(tbUpEdge->Text))
        {
            statusLabel->ForeColor = Color::Red;
            statusLabel->Text = "Нижняя граница больше верхней";
            btAction->Enabled = false;
        }
        else {
            btAction->Enabled = true;
            statusLabel->ForeColor = Color::Black;
            statusLabel->Text = "Сервер выключен";
        }
        //if ((Convert::ToUInt16(tbWorkload->Text) > 100) ||
(Convert::ToUInt16(tbWorkload->Text) <= 0))
        //{
            // statusLabel->ForeColor = Color::Red;
            // statusLabel->Text = "Нагрузка задана неверно";
            // btAction->Enabled = false;
        //}
    }
    catch (...)
    {
        statusLabel->ForeColor = Color::Red;
        statusLabel->Text = "Введенные данные не корректны";
        btAction->Enabled = false;
    }
}

private: System::Void MainForm_FormClosed(System::Object^ sender,
System::Windows::Forms::FormClosedEventArgs^ e) {
    isOn = false;
    closesocket(sListen);
    WSACleanup();
}

private: System::Void label1_Click(System::Object^ sender,
System::EventArgs^ e) {
}

private: System::Void textBox1_TextChanged(System::Object^ sender,
System::EventArgs^ e) {
}

private: System::Void textBox2_TextChanged(System::Object^ sender,
System::EventArgs^ e) {
    //if ((Convert::ToUInt16(tbWorkload->Text) > 100) ||
(Convert::ToUInt16(tbWorkload->Text) <= 0, 000002))
    //{
        // statusLabel->ForeColor = Color::Red;
        // statusLabel->Text = "Нагрузка задана неверно";
        // btAction->Enabled = false;
    //}
}

```

```
private: System::Void MainForm_Load(System::Object^ sender,
System::EventArgs^ e) {
}
private: System::Void labelIntegr_Click(System::Object^ sender,
System::EventArgs^ e) {
}
private: System::Void lvClients_SelectedIndexChanged(System::Object^
sender, System::EventArgs^ e) {
}
};

}
```