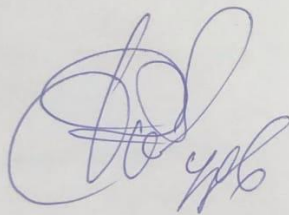


Министерство образования Российской Федерации
Пензенский государственный университет
Кафедра «Вычислительная техника»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе по курсу «Распределенные вычисления»
на тему «Разработка параллельной программы для кластера облачного типа»



30.04.2021 г.

Выполнил:
студент группы 17ВВ2
Аняев А. Р.

Принял:
к. т. н., доцент Федюнин Р. Н.

Пенза, 2021

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет Вычислительной техники

Кафедра «Вычислительная техника»

«УТВЕРЖДАЮ»

Зав. кафедрой ВТ

М. А. Митрохин

«__» __ 20__ г.

ЗАДАНИЕ

на курсовое проектирование по курсу

Распределенные вычисления

Студенту Аняеву Андрею Романовичу Группа 17ВВ2
Тема проекта Разработка параллельной программы для кластера облачного типа. Вариант 4.

Исходные данные (технические требования) на проектирование

- 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 диапазон 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:

$$\int \operatorname{tg}(x) dx \quad \text{Шаг} = 0,000002$$

- 7) Для интегрирования должен использоваться метод 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++ / Java, среда разработки Microsoft Visual Studio.

Объем работы по курсу

1. Расчетная часть
Кластер должен решать задачу вычисления определенного интеграла:

$$\int tg(x) dx \quad \text{Шаг} = 0,000002$$

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: « 03 » марта 2021

Дата защиты проекта: « 30 » апреля 2021

Руководитель

Задание получил « 03 » марта

2021 г.

Студент

Иванов И.П. Иванова И.П.

Содержание

Введение	5
1 Постановка задачи	6
2 Выбор решения	8
3 Описание разработки программы	9
4 Разработка диаграмм UML	10
4.1 Разработка диаграммы вариантов использования	10
4.2 Разработка диаграммы развертывания	11
4.3 Разработка диаграммы последовательности	12
4.4 Разработка диаграммы деятельности	12
4.5 Разработка диаграммы классов	15
5 Отладка и тестирование	17
6 Описание работы программы	18
7 Руководство пользователя	19
Заключение	22
Список используемых источников	23
Приложение А Клиент	24
Файл «Client.java»	24
Приложение Б Сервер	34
Файл «Server.java»	34

Введение

В настоящее время использование параллельного программирования становится наиболее необходимым, поскольку оно позволяет максимально эффективно использовать возможности многоядерных процессоров и многопроцессорных систем. По ряду причин, включая повышение потребления энергии и ограничения пропускной способности памяти, увеличивать тактовую частоту современных процессоров стало невозможно. Вместо этого производители процессоров стали увеличивать их производительность за счет размещения в одном чипе нескольких вычислительных ядер, не меняя или даже снижая тактовую частоту. Поэтому для увеличения скорости работы приложений теперь следует по-новому подходить к организации кода, а именно - оптимизировать программы под многоядерные системы.

Целью исследования, проводимого в рамках данной курсовой работы, является разработка параллельной программы для кластера облачного типа с пользовательским интерфейсом.

1 Постановка задачи

В ходе выполнения курсовой работы необходимо разработать программы клиента и сервера, 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 интеграла:

$$\int tg(x)dx \quad \text{Шаг} = 0,000002$$

- 7) Для интегрирования должен использоваться методом левых прямоугольников.

- 8) Приложения (клиент и сервер) должны обладать интуитивно понятным графическим интерфейсом.
- 9) Для обмена данными должны использоваться протокол TCP/IP.
- 10) Язык программирования Java, среда разработки NetBeans.

2 Выбор решения

Для реализации графического интерфейса было решено воспользоваться редактором форм NetBeans. С помощью данной среды, были созданы графические интерфейсы для ввода и вывода информации, необходимой для решения поставленной задачи.

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.

3 Описание разработки программы

Для написания программы будет использован язык программирования Java, в качестве среды программирования был выбран программный продукт NetBeans 8.2. Графический интерфейс как серверного, так и клиентского приложения представлен в виде оконного приложения `Forms`.

Для того, чтобы приложение не зависало в циклах сетевого взаимодействия, было принято решение перенести его в отдельные нити.

Таким образом, пользователь может остановить работу программ во время их выполнения, что предотвращает их блокировку на время вычисления.

В клиентском приложении создаётся 2 нити: первая отвечает за приложение, вторая — для расчета и обмена с сервером.

Серверное приложение состоит из 3 нитей. Первая нить — само приложение, вторая нить отвечает за подключение клиента в любое время работы сервера. Третья нить для взаимодействия с клиентом. Если клиентов будет больше чем 1, то для взаимодействия с каждым будет создаваться нить.

Для разработки сетевого взаимодействия между приложениями было необходимо проанализировать принципы работы с протоколом TCP/IP. Протокол TCP/IP позволяет гарантировано доставить данные на указанный адрес за счет устанавливаемого соединения, в отличие от протокола UDP. Таким образом для взаимодействия между приложениями достаточно создания двух сокетов: на стороне клиента и на стороне сервера.

Также были созданы методы: для решения задачи интегрирования методом левых прямоугольников; для деления данной задачи между клиентами, подключенными к серверу.

4 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

В обоих приложениях, как в серверном, так и в клиентском, пользователь является единственным участником системы.

Клиент может в любой момент осуществить попытку подключиться к серверу, а также в любой момент отключиться от сервера. Диаграмма вариантов использования клиентского приложения показана на рисунке 1.

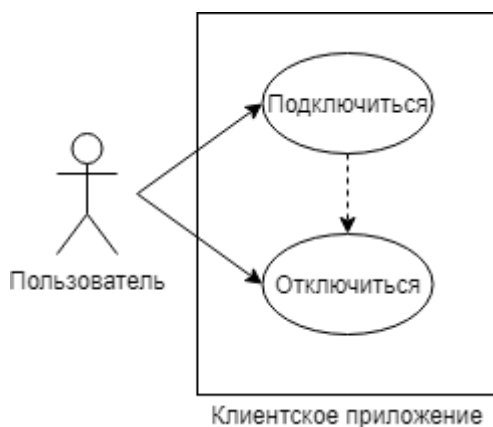


Рисунок 1 — UML-диаграмма вариантов использования клиентского приложения

Диаграмма вариантов использования серверного приложения показана на рисунке 2.

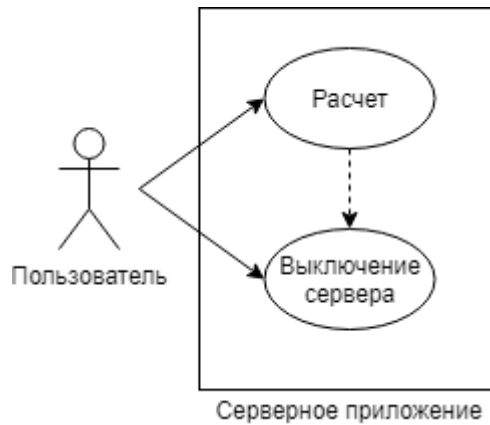


Рисунок 2 — UML-диаграмма вариантов использования серверного приложения

4.2 Разработка диаграммы развертывания

Была разработана диаграмма развертывания, показанная на рисунке 3.

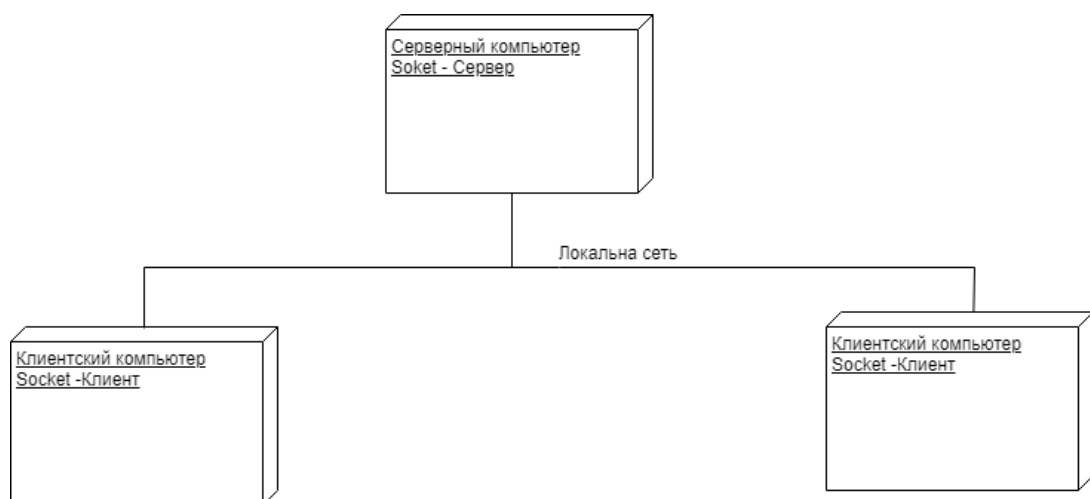


Рисунок 3 – UML-диаграмма развёртывания

Компьютеры с клиентскими сокетами подключаются к компьютеру с серверным сокетом по локальный вычислительной сети. Также возможна работа, когда Компьютер 1 может иметь как серверный, так и клиентский сокет. Таким образом вся система вычислений будет развернута на одном компьютере, который будет выступать в роли и сервера и клиента. Для этого при запуске клиента нужно указывать IP-адрес 127.0.0.1.

4.3 Разработка диаграммы последовательности

На рисунке 4 приведён пример работы серверного и клиентских приложений в виде UML-диаграммы последовательности.

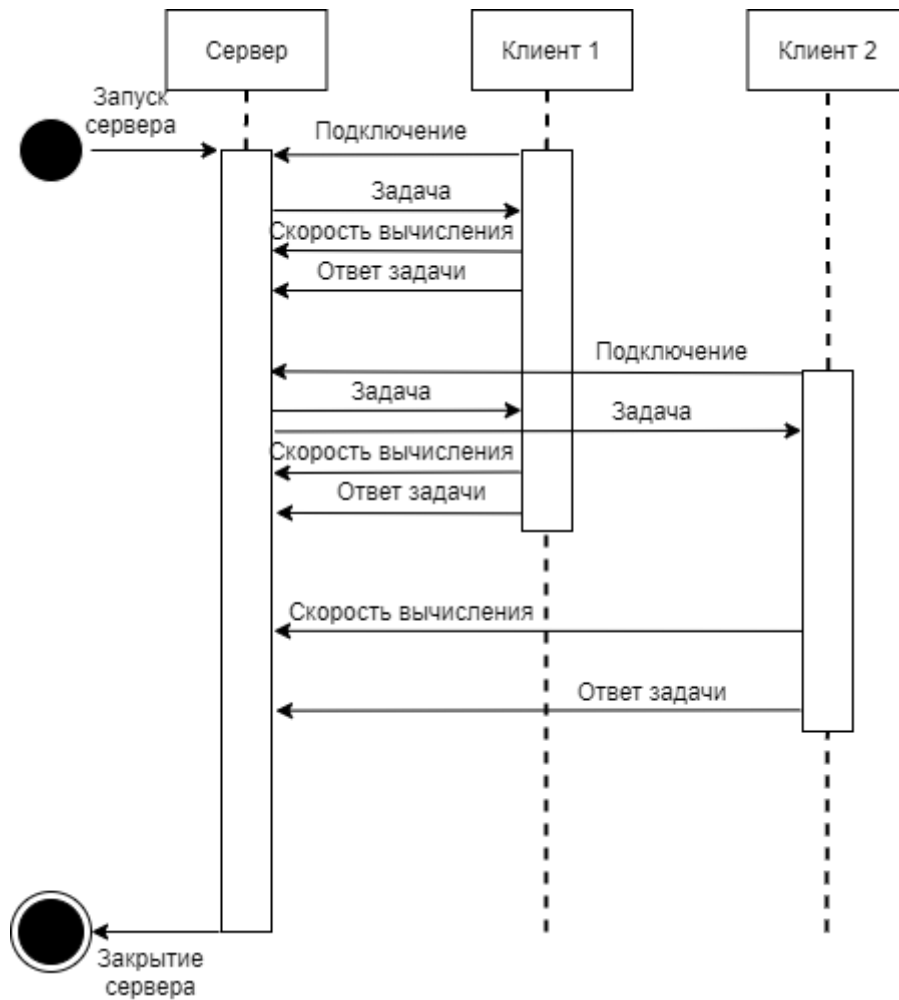


Рисунок 4 – UML-диаграмма последовательности

Из диаграммы видно, что сервер отправляет задание каждому клиенту, после чего получает от них скорость вычислений, а только затем решение.

4.4 Разработка диаграммы деятельности

UML-диаграмма деятельности показана ниже, на рисунке 5.

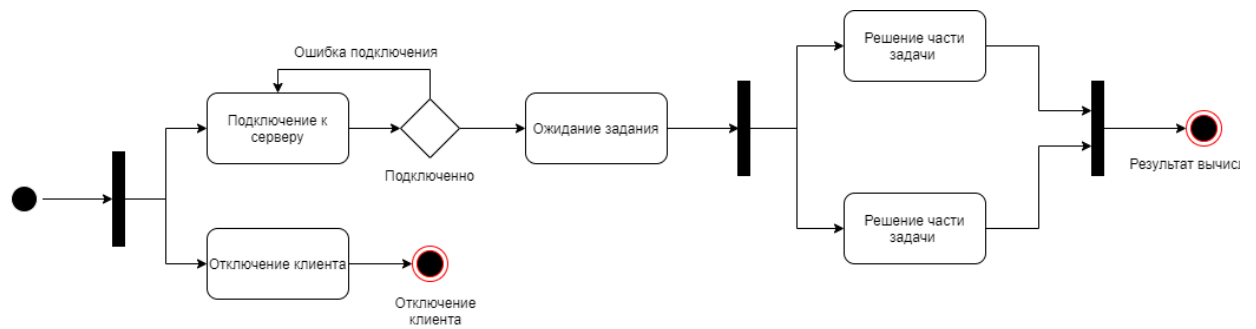


Рисунок 5 – UML-диаграмма деятельности клиентского приложения

Когда клиент захочет подключиться к серверу, создается новая нить. Она будет работать параллельно основной, не давая зависнуть программе и производить общение с сервером. Если клиент получает задание от сервера, то создаются новые потоки, количество которых зависит от числа ядер процессора. Эти потоки выполняют задание параллельно для большей производительности.

UML-диаграмма деятельности, показанная на рисунке 6, отображает работу нитей серверного приложения.

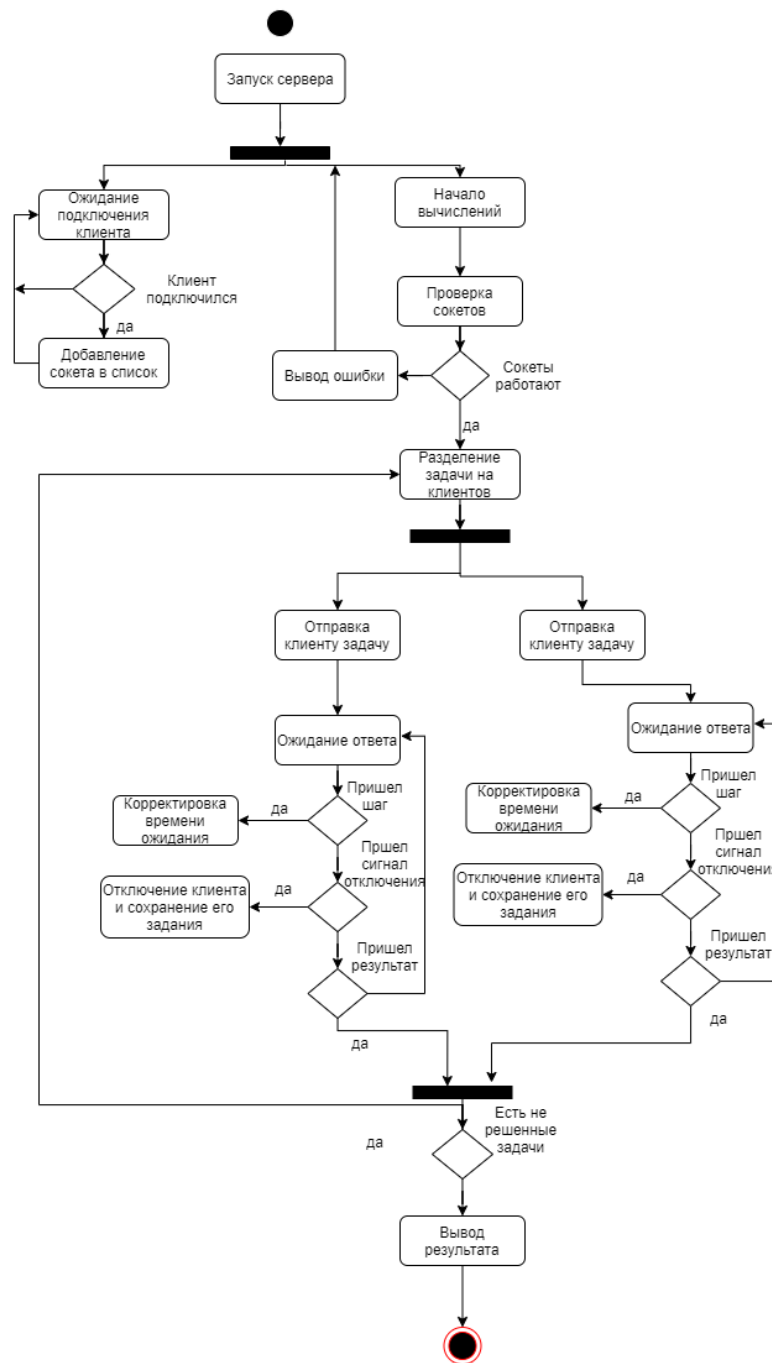


Рисунок 6 – UML-диаграмма деятельности

Во время запуска сервера сразу создаётся дополнительная нить для обработки подключения клиентов. Основная нить приложения обрабатывает введенные данные и формирует задания для клиентов, после для каждого клиента формируется новая нить для параллельно взаимодействия. По окончании вычислений клиентские нити закрываются и основной поток выводит результат.

4.5 Разработка диаграммы классов

Были разработаны диаграммы классов для клиентского и серверного приложений, они показаны на рисунке 7 и рисунке 8.

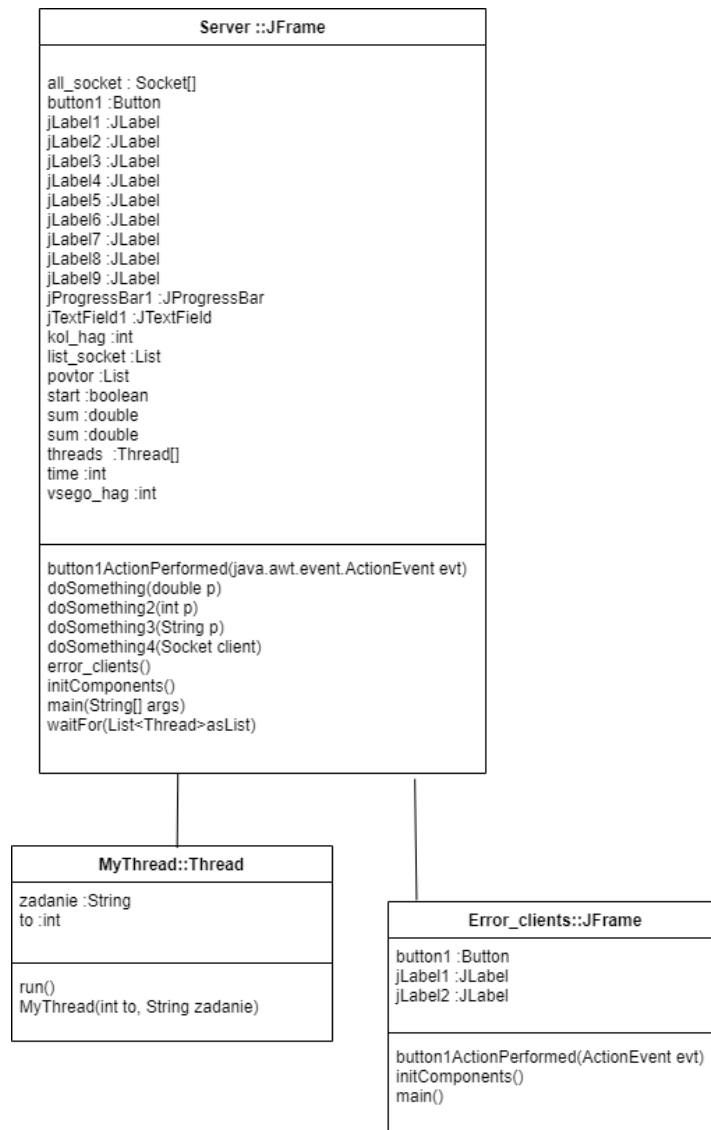


Рисунок 7 – UML-диаграмма классов серверного приложения

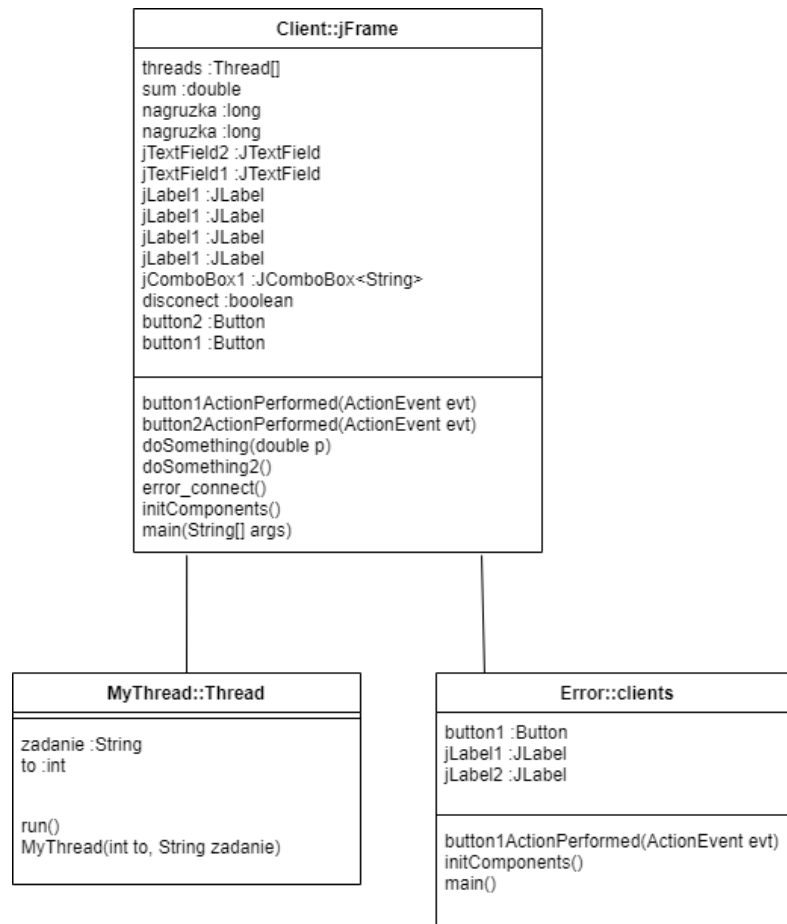


Рисунок 8 – UML-диаграмма классов клиентского приложения

5 Отладка и тестирование

В качестве среды разработки выступает программный продукт NetBeans. 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 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.

При запуске сервера пользователь должен ввести шаг интегрирования и дождаться подключения клиентов. Когда все будет готово, необходимо нажать на кнопку рассчитать. В этот момент основной поток проверит всех подключенных клиентов, путем отправки им небольшого сообщения. При неудачной отправке клиент будет исключен из расчетов. Если остался хотя бы один клиент, сервер разделяет весь промежуток интегрирования на равные части на всех клиентов.

Для взаимодействия с каждым клиентом создается новая нить. Она отвечает за принятие и отправку сообщений. Данная нить может принять от клиента 3 вида сообщений: количество шагов, которые успел рассчитать клиент (данное сообщение отправляется каждые 100 мс). На основе этих данных рассчитываются примерно время окончания расчетов и их прогресс; сообщение об отключении — клиент удаляется из списка сервера и его задание помечается как не сделанное; решение задачи.

После окончания взаимодействия всех нитей с клиентами происходит проверка на нерешенные задачи. Если такие присутствуют, то они распределяются на оставшихся клиентов. Если нет, то выводится результат расчетов.

После запуска клиента пользователь вводит данные подключения и максимальную загрузку процессора. Для подключения и взаимодействия с сервером создается новая нить. Если подключение было успешным — клиент ожидает задачи. Когда задачу прислали, клиент узнает сколько ядер в процессоре и в зависимости от этого формирует максимально количество параллельных нитей для расчетов. Основная нить каждые 100 мс фиксирует количество рассчитанных шагов и отправляет их на сервер. По окончании расчетов на сервер отправляется результат.

Проект образует единую распределенную вычислительную систему облачного типа и состоит из двух частей: серверного приложения и клиентского приложения.

Для начала требуется запустить серверное приложение. При старте программы в поля формы уже будут занесены значения по умолчанию.

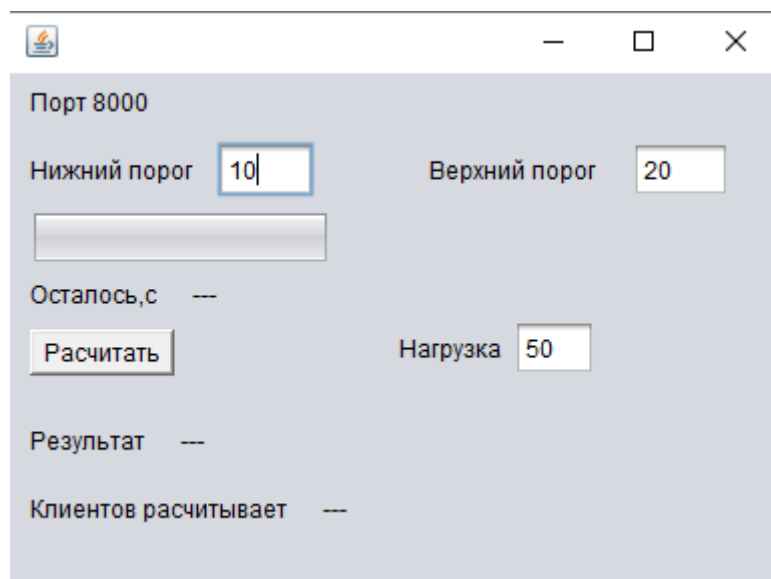


Рисунок 9 – Запуск сервера

В данном окне пользователю необходимо ввести нижний и верхний пороги интегрирования.

Если есть подключенные клиенты, и данные введены правильно, можно производить вычисления. Во время вычислений будет показано примерное время, оставшееся до окончания, количество клиентов, которые участвуют в расчетах и шкала прогресса.

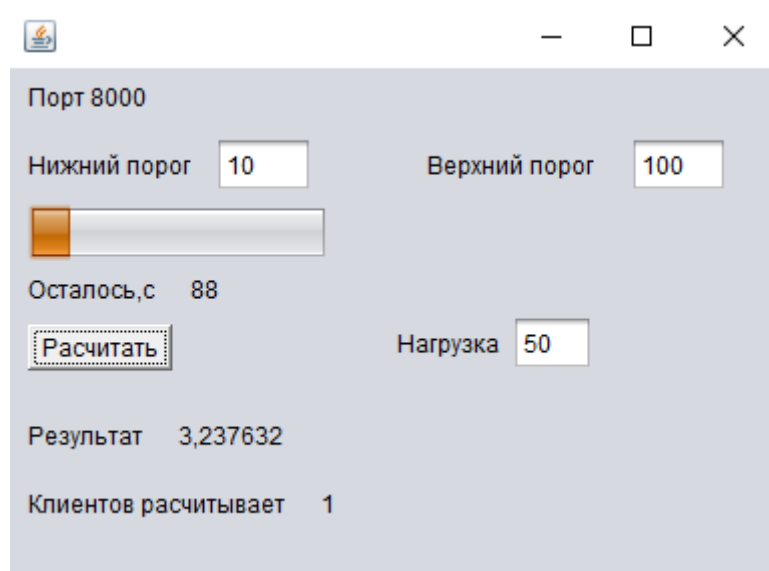


Рисунок 10 — Выполнение расчётов

После того как вычисления будут выполнены, результат будет выведен в специальное поле.

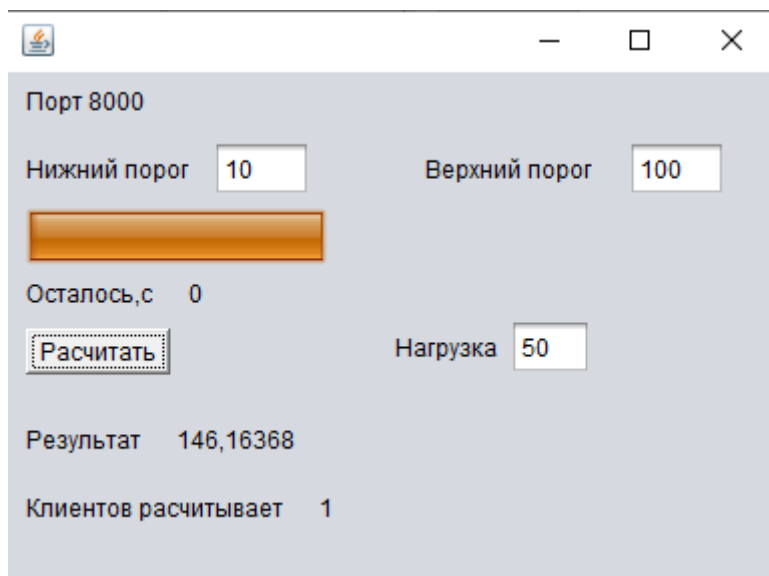


Рисунок 11 — Результат вычислений

Для подключения клиента требуется ввести IP-адрес и порт сервера, а также максимальную нагрузку на сервер.

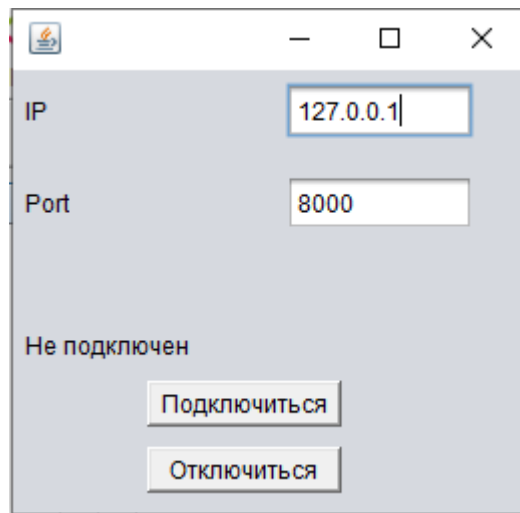


Рисунок 12 — Подключение клиента

После нажатия кнопки «Подключиться» будет произведена попытка подключения к серверу по указанному IP-адресу и порту. В случае успеха надпись «Не подключен» сменится на «Подключен».

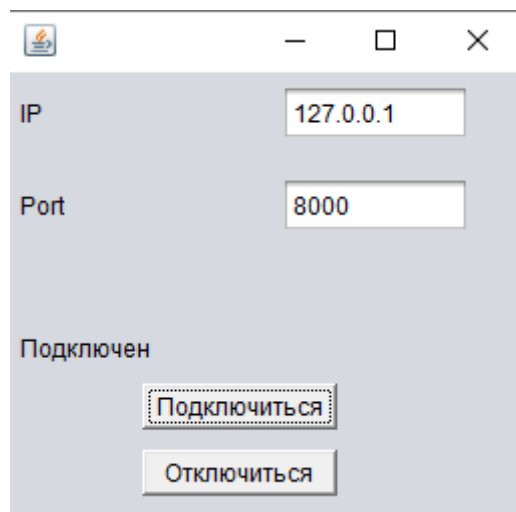


Рисунок 13 — Подключение к серверу

Заключение

В ходе выполнения данной курсовой работы было разработано клиент-серверное приложение, позволяющее производить вычисления в рамках локальной вычислительной сети. Программа носит практический характер и предназначена для одновременного распределенного решения единой задачи несколькими компьютерами. Также были освоены принципы сетевого взаимодействия, принципы работы с сокетами и протоколом TCP/IP.

В дальнейшем программу можно улучшить, дав пользователю самостоятельно выбрать подынтегральную функцию, дополнительно оптимизировав алгоритм работы как серверного, так и клиентского приложения.

Список используемых источников

1. "Java. Методы программирования" И.Н Блинов, В.С. Романчик – Минск: УниверсалПресс, 2007 – 704 с.
2. "Java. Руководство для начинающих", Герберт Шилдт – ООО «Диалектика», 2019 – 816 с.
3. Форум программистов и сисадминов Киберфорум [Электронный ресурс]. Режим доступа: <http://www.cyberforum.ru/>, свободный (Дата обращения 01.07.2021).

Приложение А Клиент

Файл «Client.java»

```
package client_kursach_fedunin;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import static java.lang.Math.log;
import java.net.Socket;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.Timer;

/**
 *
 * @author User
 */
public class Client extends javax.swing.JFrame {

    /**
     * Creates new form Client
     */
    public Client() {
        initComponents();
    }

    public static double sum = new Double(0);
    public static boolean disconnect = false;
    public static int stepCount = new Integer(0);
    public static Thread[] threads = new Thread[100];
    public static long processorLoad = new Long(0);

    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
}
```

```

@SuppressWarnings("unchecked")

// <editor-fold defaultstate="collapsed" desc="Generated Code">

private void initComponents() {

    button1 = new java.awt.Button();
    jLabel1 = new javax.swing.JLabel();
    jLabel2 = new javax.swing.JLabel();
    jTextField1 = new javax.swing.JTextField();
    jTextField2 = new javax.swing.JTextField();
    button2 = new java.awt.Button();
    jLabel4 = new javax.swing.JLabel();

    setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

    button1.setLabel("Подключиться");
    button1.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            button1ActionPerformed(evt);
        }
    });

    jLabel1.setText("IP");

    jLabel2.setText("Port");

    jTextField1.setText("127.0.0.1");

    jTextField2.setText("8000");

    button2.setLabel("Отключиться");
    button2.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            button2ActionPerformed(evt);
        }
    });

    jLabel4.setText("Не подключен");

```



```

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
    .addComponent(jLabel1)
    .addComponent(jTextField1, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
    .addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
    .addComponent(jLabel2)
    .addComponent(jTextField2, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
    .addGap(49, 49, 49)
    .addComponent(jLabel4)
    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
    .addComponent(button1, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
    .addComponent(button2, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
    .addContainerGap())
);

pack();
} // </editor-fold>

public static void error_connect() {

    client_kursach_fedunin.Error.main();

}

private void button1ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    new Thread(() -> {
        String ip = jTextField1.getText();
        String portS = jTextField2.getText();
        int port = Integer.parseInt(portS);
        try (Socket socket = new Socket(ip, port);
            BufferedWriter writer = new BufferedWriter(new
OutputStreamWriter(socket.getOutputStream())));

```

```

        BufferedReader reader = new BufferedReader(new
InputStreamReader(socket.getInputStream()));) {

    System.err.println("conected");
    jLabel4.setText("Подключен");
    while (true) {
        stepCount = 0;
        double otvet = 0;

        String response = reader.readLine();
        System.err.println(response);
        String[] fullString = response.split("/");
        double a = Double.parseDouble(fullString[0]);
        if (a == 1) {
            continue;
        }
        double step = Double.parseDouble(fullString[1]);
        double step1 = Double.parseDouble(fullString[2]);
        double step2 = Double.parseDouble(fullString[3]);
        double processorLoad2 = Double.parseDouble(fullString[4]);
        processorLoad = Math.round(processorLoad2);

        //-------------------------------------
        int clientsCount =
Runtime.getRuntime().availableProcessors();//Получаем количество потоков
        double clientsCountDouble = clientsCount;
        double interval = step2 - step1;
        int q = (int) (interval / clientsCountDouble);
        int k = (int) (interval % clientsCountDouble);
        String zadanie = null;
        int threadNumber = 0;//номер нити
        int streamNumber = 0;//номер потока
        sum = 0;
        for (double i = 0; i < interval; i = i + q) {
            if (i + q + k != interval) {
                zadanie = a + "/" + step + "/" + (i + step1) + "/" + (i +
step1 + q);

                System.err.println(threadNumber + " " + zadanie);
                threads[streamNumber] = new MyThread(threadNumber,
zadanie);

                threads[streamNumber].start();
                streamNumber++;
            }
        }
    }
}

```

```

        threadNumber++;
    } else {
        zadanie = a + "/" + step + "/" + (i + step1) + "/" + (i +
step1 + q + k);

        System.err.println(threadNumber + " " + zadanie);
        threads[streamNumber] = new MyThread(threadNumber,
zadanie);

        threads[streamNumber].start();
        streamNumber++;
        break;
    }

}

//-----
while (true) {
    boolean exit = true;
    for (int i = 0; i <= streamNumber; i++) {

        if (threads[i] != null) {
            exit = false;
        }
    }

    try {
        Thread.sleep(100);
    } catch (InterruptedException ex) {
        Logger.getLogger(Client.class.getName()).log(Level.SEVERE,
null, ex);
    }

    if (interval != stepCount) {

        String command = "hag/" + stepCount;
        if (disconnect) {
            command = "disconnect/";
        }

        writer.write(command);
        writer.newLine();
        writer.flush();
        doSomething2();
    }
}

```

```

        if (exit) {
            break;
        }
    }

    String result = "otvet/" + String.valueOf(sum);
    System.err.println("-----" + sum);
    //Thread.sleep(10000);
    writer.write(result);
    writer.newLine();
    writer.flush();

    }
} catch (IOException ex) {
    System.err.println("error");
    error_connect();
    // Logger.getLogger(Client.class.getName()).log(Level.SEVERE, null,
ex);
}

}).start();

}

private void button2ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    disconnect = true;
    try {
        Thread.sleep(500);
    } catch (InterruptedException ex) {
        Logger.getLogger(Client.class.getName()).log(Level.SEVERE, null, ex);
    }
    System.exit(0);
}

class MyThread extends Thread {

    private int to;

```

```

private String task;

public MyThread(int to, String task) {
    this.to = to;
    this.task = task;
}

@Override
public void run() {
    double otvet = 0;
    String[] fullString = task.split("/");
    double a = Double.parseDouble(fullString[0]);
    double step = Double.parseDouble(fullString[1]);
    double step1 = Double.parseDouble(fullString[2]);
    double step2 = Double.parseDouble(fullString[3]);
    for (double i = step1; i < step2; i++) {
        long start = System.currentTimeMillis();
        double proverka = a + (step * i);
        otvet = (Math.tan(proverka));
        doSomething(otvet);
        long finish = System.currentTimeMillis();
        long workTime = finish - start;
        long t = (workTime * (100 / processorLoad) - workTime);
        try {
            sleep(t);
        } catch (InterruptedException ex) {
            Logger.getLogger(Client.class.getName()).log(Level.SEVERE, null,
ex);
        }
    }
    System.err.println(otvet);

    threads[to] = null;
}

public synchronized void doSomething(double p) {
    sum = sum + p;
}

```

```

        stepCount = stepCount + 1;
    }

    public synchronized void doSomething2() {
        stepCount = 0;
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        /* Set the Nimbus look and feel */
        //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code
(optional) ">
        /* If Nimbus (introduced in Java SE 6) is not available, stay with the default
look and feel.
         * For details see
http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
         */
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(Client.class.getName()).log(java.util.logging.Level
.SEVERE, null, ex);

        } catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(Client.class.getName()).log(java.util.logging.Level
.SEVERE, null, ex);

        } catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(Client.class.getName()).log(java.util.logging.Level
.SEVERE, null, ex);

        } catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(Client.class.getName()).log(java.util.logging.Level
.SEVERE, null, ex);

```

```

    }

    //</editor-fold>

    /* Create and display the form */
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new Client().setVisible(true);

        }

    });
}

// Variables declaration - do not modify
private java.awt.Button button1;
private java.awt.Button button2;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel2;
private javax.swing.JLabel jLabel4;
private javax.swing.JTextField jTextField1;
private javax.swing.JTextField jTextField2;
// End of variables declaration
}

```

Приложение Б Сервер

Файл «Server.java»

```
package serwer_kursach_fedunin;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.ImageIcon;
import javax.swing.JFrame;

/**
 *
 * @author User
 */
public class Server extends javax.swing.JFrame {

    private static void waitFor(List<Thread> asList) {

        throw new UnsupportedOperationException("Not supported yet."); //To change
body of generated methods, choose Tools | Templates.

    }

    /**
     * Creates new form Server
     */
    public Server() {
        initComponents();
    }
}
```

```

public static Socket[] all_socket = new Socket[100];
public static boolean start = false;
public static double sum = 0;
public static int time = 0;
public static int stepCount = 0;
public static int allSteps = 0;
public static Thread[] threads = new Thread[100];
public static List list_socket = new ArrayList();
public static List povtor = new ArrayList();

/**
 * This method is called from within the constructor to initialize the form.
 * WARNING: Do NOT modify this code. The content of this method is always
 * regenerated by the Form Editor.
 */
@SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code">
private void initComponents() {

    jLabel10 = new javax.swing.JLabel();
    jScrollPane1 = new javax.swing.JScrollPane();
    jTextArea1 = new javax.swing.JTextArea();
    progressBar2 = new javax.swing.JProgressBar();
    jTextField1 = new javax.swing.JTextField();
    button1 = new java.awt.Button();
    jLabel1 = new javax.swing.JLabel();
    jLabel2 = new javax.swing.JLabel();
    jLabel3 = new javax.swing.JLabel();
    jLabel4 = new javax.swing.JLabel();
    jLabel6 = new javax.swing.JLabel();
    progressBar1 = new javax.swing.JProgressBar();
    jLabel8 = new javax.swing.JLabel();
    jLabel9 = new javax.swing.JLabel();
    jLabel5 = new javax.swing.JLabel();
    jTextField2 = new javax.swing.JTextField();
    jLabel11 = new javax.swing.JLabel();
    jLabel7 = new javax.swing.JLabel();

```

```

jTextField3 = new javax.swing.JTextField();

jLabel10.setText("jLabel10");

jTextArea1.setColumns(20);
jTextArea1.setRows(5);
jScrollPane1.setViewportView(jTextArea1);

setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

jTextField1.setText("10");

button1.setLabel("Рассчитать");
button1.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        button1ActionPerformed(evt);
    }
});

jLabel1.setText("Нижний порог");

jLabel2.setText("Результат");

jLabel3.setText("---");

jLabel4.setText("Осталось, с");

jLabel6.setText("---");

jLabel8.setText("Клиентов рассчитывает");

jLabel9.setText("---");

jLabel5.setText("Порт 8000");

jTextField2.setText("20");
jTextField2.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {

```

[illegible]

```

        .addGap(18, 18, 18)

        .addComponent(jTextField2,
javax.swing.GroupLayout.PREFERRED_SIZE, 49, javax.swing.GroupLayout.PREFERRED_SIZE)

        .addGap(26, 26, 26))

    .addGroup(layout.createSequentialGroup())

    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

        .addComponent(button1,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)

    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)

        .addComponent(jLabel7)

    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

        .addComponent(jLabel5)

        .addGroup(layout.createSequentialGroup())

            .addComponent(jLabel8)

            .addGap(18, 18, 18)

            .addComponent(jLabel9))

        .addGroup(layout.createSequentialGroup())

            .addComponent(jLabel12)

            .addGap(18, 18, 18)

            .addComponent(jLabel6,
javax.swing.GroupLayout.PREFERRED_SIZE, 160,
javax.swing.GroupLayout.PREFERRED_SIZE))))

    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

        .addComponent(jTextField3,
javax.swing.GroupLayout.PREFERRED_SIZE, 41, javax.swing.GroupLayout.PREFERRED_SIZE)

        .addGap(0, 0, Short.MAX_VALUE)))

);

layout.setVerticalGroup(

    layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

    .addGroup(layout.createSequentialGroup())

        .addContainerGap()

        .addComponent(jLabel5)

        .addGap(12, 12, 12)

    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

        .addComponent(jLabel1)

        .addComponent(jTextField1, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)

```

```

        .addComponent(jTextField2, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)

        .addComponent(jLabel11))

        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

        .addComponent(jProgressBar1, javax.swing.GroupLayout.PREFERRED_SIZE,
28, javax.swing.GroupLayout.PREFERRED_SIZE)

        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)

        .addGroup(layout.createSequentialGroup())

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

        .addComponent(jLabel4)

        .addComponent(jLabel3))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

        .addComponent(button1, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

        .addComponent(jLabel7)

        .addComponent(jTextField3,
javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)))

        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
24, Short.MAX_VALUE)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

        .addComponent(jLabel2)

        .addComponent(jLabel6))

        .addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)

        .addComponent(jLabel8)

        .addComponent(jLabel9))

        .addGap(30, 30, 30))

    );

    pack();
} // </editor-fold>

private void button1ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:

```

```

Socket client = null;

int clients = 0;

double step = 0.000002;

boolean disconnect = true;

for (int i = 0; i < list_socket.size(); i++) {
    try {
        client = (Socket) list_socket.get(i);

        BufferedWriter writer1 = new BufferedWriter(new
OutputStreamWriter(client.getOutputStream()));

        writer1.write(String.valueOf("1/"));

        writer1.newLine();

        writer1.flush();

    } catch (IOException ex) {
        list_socket.remove(client);

        disconnect = false;
    }

    if (disconnect) {
        clients++;
    }
}

try{

if (clients > 0 && step<=10) {
    jLabel9.setText(clients+"");

    int lower = Integer.parseInt(jTextField1.getText()); //нижний порог
    int upper = Integer.parseInt(jTextField2.getText()); //верхний порог

    double n1 = (upper - lower) / step;

    int n = (int) n1; // количество шагов
    allSteps = n;

    System.err.println(n);

    int k = n % clients;

    int q = n / clients;

    jProgressBar1.setMinimum(-n);

    jProgressBar1.setMaximum(0);

```

```

System.err.println(q);

System.err.println(k);

String task = null;

int clientNumber = 0; //номер клиента
int threadNumber = 0; //номер потока

double processorLoad = Double.parseDouble(jTextField3.getText());

sum = 0;

for (double i = 0; i < n; i = i + q) {
    if (i + q + k != n) {
        task = lower + "/" + step + "/" + i + "/" + (i + q) + "/" +
processorLoad;

        System.err.println(clientNumber + " " + task);
        threads[threadNumber] = new MyThread(clientNumber, task);
        threads[threadNumber].start();
        threadNumber++;
        clientNumber++;
    } else {
        task = lower + "/" + step + "/" + i + "/" + (i + q + k) + "/" +
processorLoad;

        System.err.println(clientNumber + " " + task);
        threads[threadNumber] = new MyThread(clientNumber, task);
        threads[threadNumber].start();
        threadNumber++;
        break;
    }
}

while (true) {
    while (true) {
        try {
            boolean exit = true;

            for (int i = 0; i <= threadNumber; i++) {

                if (threads[i] != null) {
                    exit = false;
                }
            }

            jLabel3.setText(time + "");

```

```

        jProgressBar1.setValue(-allSteps);
        super.update(this.getGraphics());
        Thread.sleep(1000);

        if (exit) {
            break;
        }

    } catch (InterruptedException ex) {
        Logger.getLogger(Server.class.getName()).log(Level.SEVERE,
null, ex);
    }
}

if (povtor.size() == 0) {
    break;
} else {
    int o = 0;
    if (list_socket.size() > povtor.size()) {
        o = povtor.size();
    } else {
        o = list_socket.size();
    }
    for (int i = 0; i < o; i++) {
        String zad = (String) povtor.get(i);
        threads[i] = new MyThread(i, zad);
        threads[i].start();
        povtor.remove(zad);
    }
}

}

//-----
-----

double prom = ((upper - lower) / n1) * sum;
DecimalFormat df = new DecimalFormat("###.#####");
jProgressBar1.setValue(0);
jLabel6.setText(df.format(prom));
}

```

```

        else
        {
            error_clients();
        }
    } catch (NumberFormatException ex) {
        error_clients();
    }
}

private void jTextField2ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
}

public static void error_clients() {

    serwer_kursach_fedunin.Error_clients.main();

}

class MyThread extends Thread {

    private int to;

    private String task;

    public MyThread(int to, String zadanie) {
        this.to = to;

        this.task = zadanie;
    }

    @Override
    public void run() {
        int stepsPassed = 0;

        Socket client = (Socket) list_socket.get(to);

        System.out.println(client);

        try {

            System.err.println(client.isBound());

```

```

        BufferedWriter writer = new BufferedWriter(new
OutputStreamWriter(client.getOutputStream()));

        BufferedReader reader = new BufferedReader(new
InputStreamReader(client.getInputStream()));

        writer.write(String.valueOf(task));

        writer.newLine();

        writer.flush();

        String response = "";
        do {
            response = reader.readLine();

            String[] fullString = response.split("/");
            if (fullString[0].equals("hag")) {

                int i = Integer.parseInt(fullString[1]);
                stepsPassed = stepsPassed + i;
                doSomething2(i);

            } else if (fullString[0].equals("disconnect")) {
                doSomething3(task);
                doSomething2(-stepsPassed);
                doSomething4(client);
                break;
            } else {
                response = fullString[1];
                System.err.println(response);
                double p = Double.parseDouble(response);
                doSomething(p);
                break;
            }
        } while (true);

        threads[to] = null;

    } catch (IOException ex) {
        Logger.getLogger(Server.class.getName()).log(Level.SEVERE, null, ex);
    }

    //System.err.println(prohlo);

```

```

    }
}

public synchronized void doSomething(double p) {
    sum = sum + p;
}

public synchronized void doSomething2(int p) {
    if (p < 0) {
        allSteps = allSteps - p;
    }
    if (p != 0 && p > 0) {
        time = allSteps / p;
        time = time / 10;
        allSteps = allSteps - p;
    }
    System.err.println(p);
}

public synchronized void doSomething3(String p) {
    povtor.add(p);
}

public synchronized void doSomething4(Socket client) {
    list_socket.remove(client);
    String myString = jLabel9.getText();
    int i = Integer.parseInt(myString);
    i=i-1;
    jLabel9.setText(i+"");
}

/**
 * @param args the command line arguments
 */
public static void main(String args[]) {
    /* Set the Nimbus look and feel */

```

```

        //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code
(optional) ">

        /* If Nimbus (introduced in Java SE 6) is not available, stay with the default
look and feel.

        * For details see
http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
        */

        try {

            for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {

                if ("Nimbus".equals(info.getName())) {

                    javax.swing.UIManager.setLookAndFeel(info.getClassName());

                    break;

                }

            }

        } catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(Server.class.getName()).log(java.util.logging.Level
.SEVERE, null, ex);

        } catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(Server.class.getName()).log(java.util.logging.Level
.SEVERE, null, ex);

        } catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(Server.class.getName()).log(java.util.logging.Level
.SEVERE, null, ex);

        } catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(Server.class.getName()).log(java.util.logging.Level
.SEVERE, null, ex);

        }

    }

}

//</editor-fold>

/* Create and display the form */

java.awt.EventQueue.invokeLater(new Runnable() {

    public void run() {

        new Server().setVisible(true);

        new Thread(() -> {

            try (ServerSocket server = new ServerSocket(8000);) {

                System.out.println("server started");

                int i = 0;

                while (true) {

```

```

        Socket socket = server.accept();

        list_socket.add(socket);

        //all_socket[i] = socket;

        i++;

        System.out.println(socket.getPort() + " " +
socket.getInetAddress());

    }

    } catch (IOException e) {

        throw new RuntimeException(e);

    }

    }).start();

}

});

}

public Server(Thread[] ts) throws InterruptedException {

    waitFor(Arrays.asList(ts));

}

// Variables declaration - do not modify
private java.awt.Button button1;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel10;
private javax.swing.JLabel jLabel11;
private javax.swing.JLabel jLabel2;
private javax.swing.JLabel jLabel3;
private javax.swing.JLabel jLabel4;
private javax.swing.JLabel jLabel5;
private javax.swing.JLabel jLabel6;
private javax.swing.JLabel jLabel7;
private javax.swing.JLabel jLabel8;
private javax.swing.JLabel jLabel9;
private javax.swing.JProgressBar jProgressBar1;
private javax.swing.JProgressBar jProgressBar2;
private javax.swing.JScrollPane jScrollPane1;

```

```
private javax.swing.JTextArea jTextArea1;  
private javax.swing.JTextField jTextField1;  
private javax.swing.JTextField jTextField2;  
private javax.swing.JTextField jTextField3;  
// End of variables declaration  
}
```