Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

ОТЧЁТ

по лабораторной работе №6

по курсу «Программирование на языке Java»

на тему «Сетевое взаимодействие в Java»

Выполнили:

Студенты группы 22ВВП1

Матросов А.М.

Пяткин М.Э.

Приняли:

Юрова О. В.

Карамышева Н. С.

Пенза 2025

### Цель работы

Научиться создавать клиент-серверные приложения c использованием стандартных классов Java.

### Лабораторное задание:

**Вариант 10**:

Модифицировать приложение из предыдущей лабораторной работы, реализовав клиент-серверную архитектуру, обеспечивающую распределенное вычисление определенного интеграла на нескольких вычислительных узлах (клиентах) при этом каждый узел использует несколько нитей, как в предыдущей работе. Сервер не занимается вычислениями, а лишь реализует взаимодействие с пользователем и агрегацию результатов вычислений от клиентов. Нечетные варианты используют протокол UDP, а четные TCP. Оформление лабораторной работы должно быть выполнено в соответствии с требованиями, приведенными в Приложении 2.

**Ход работы**

1. **Базовая структура**

Создали два проекта: **IntegralServer** для сервера и **IntegralClient** для клиента. Определили архитектура: сервер управляет задачами и агрегирует результаты, клиенты выполняют вычисления.

1. **Реализация сервера**

Использовли **ServerSocket** для TCP-соединения на порту 12345.Реализовли поток для принятия подключений клиентов с использованием Socket и списка **ArrayList<ClientHandler>**.

Затем добавили консольный ввод параметров интеграла (a, b, dx) через Scanner.

1. **Логика распределения задач и агрегация результатов**

Сервер отправляет клиентам параметры (a, b, dx, numClients) через PrintWriter с flush() для немедленной передачи.

Метод **addPartialResult** суммирует частичные результаты от клиентов, он синхронизирован с помощью synchronized.

Сервер ожидает результатов от числа клиентов (numClients), используя wait() и notifyAll().

1. **Реализация клиента**

Добавили TCP-подключение к серверу через Socket с использованием **BufferedReade**r и **PrintWriter**.

Клиент парсит входные параметры и определяет свои подинтервалы на основе clientId % numClients. Каждый клиент делит свои подинтервалы на потоки (до 10 в зависимости от задач), используя Thread[].

Результаты суммируются в **addPartialResult** и отправляются серверу одним сообщением после завершения всех потоков.

**Таблица распределения задач**

Предположим, что входные данные:

* NUM\_THREADS = 10 — всего задач
* numClients = 3 — клиентов
* tasksPerClient = 10 / 3 = 3
* extraTasks = 10 % 3 = 1 — 1 задача осталась лишней

|  |  |  |  |
| --- | --- | --- | --- |
| **ClientId** | **clientIndex = clientId % numClients** | **Входит в extraTasks?** | **Задач (tasksForThisClient)** |
| 0 | 0 | Да (0 < 1) | 3 + 1 = 4 задачи |
| 1 | 1 | Нет (1 ≥ 1) | 3 + 0 = 3 задачи |
| 2 | 2 | Нет (2 ≥ 1) | 3 + 0 = 3 задачи |

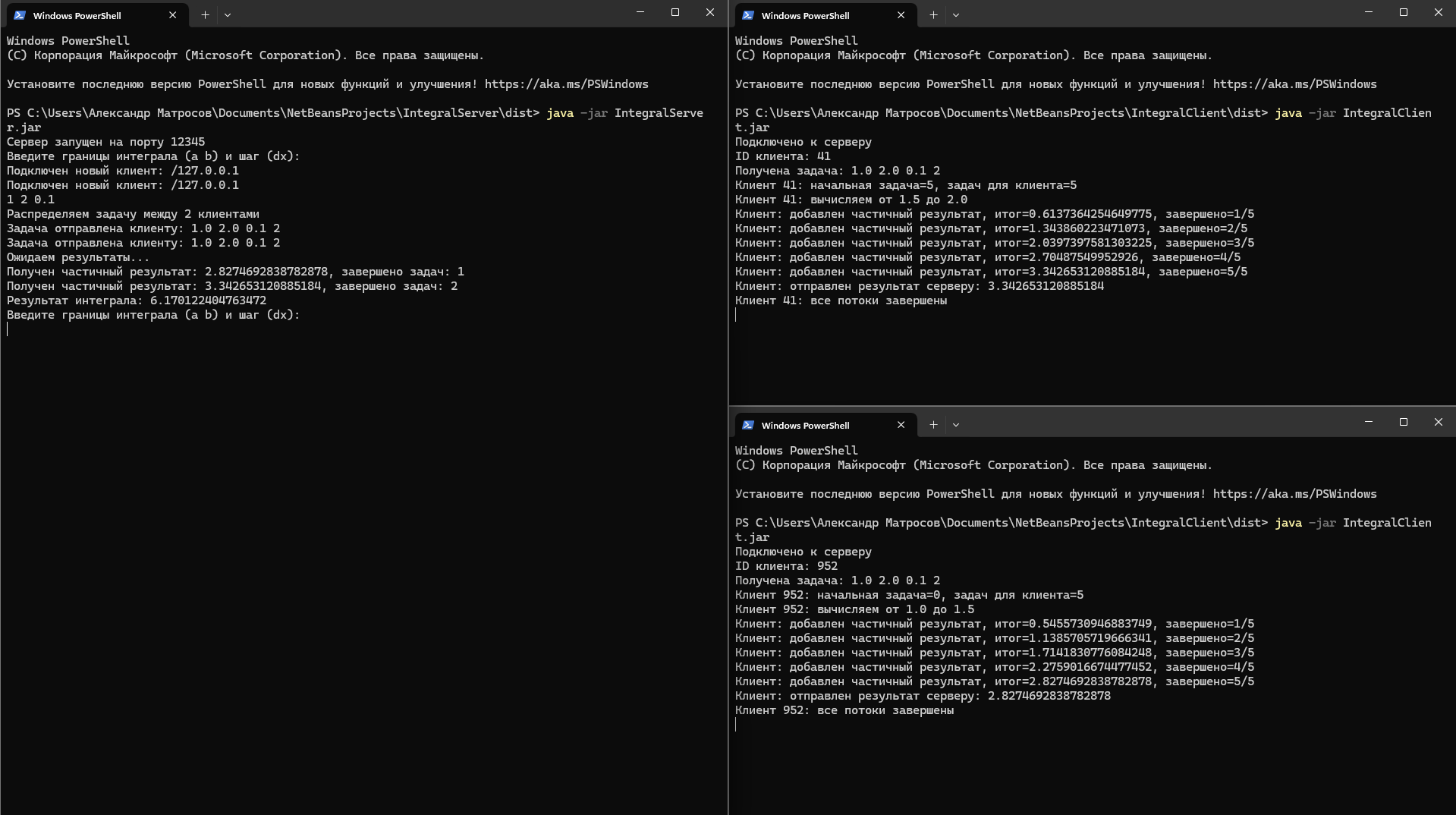
**int startTask = clientId % numClients \* tasksPerClient + Math.min(clientId % numClients, extraTasks);**

* clientId % numClients — остаток от деления clientId на количество клиентов, чтобы распределить лишние задачи.
* tasksPerClient — количество задач, которое должен получить клиент по умолчанию.
* Math.min(clientId % numClients, extraTasks) — добавляем 1 задачу, если клиент попадает в список, получающий лишние задачи.

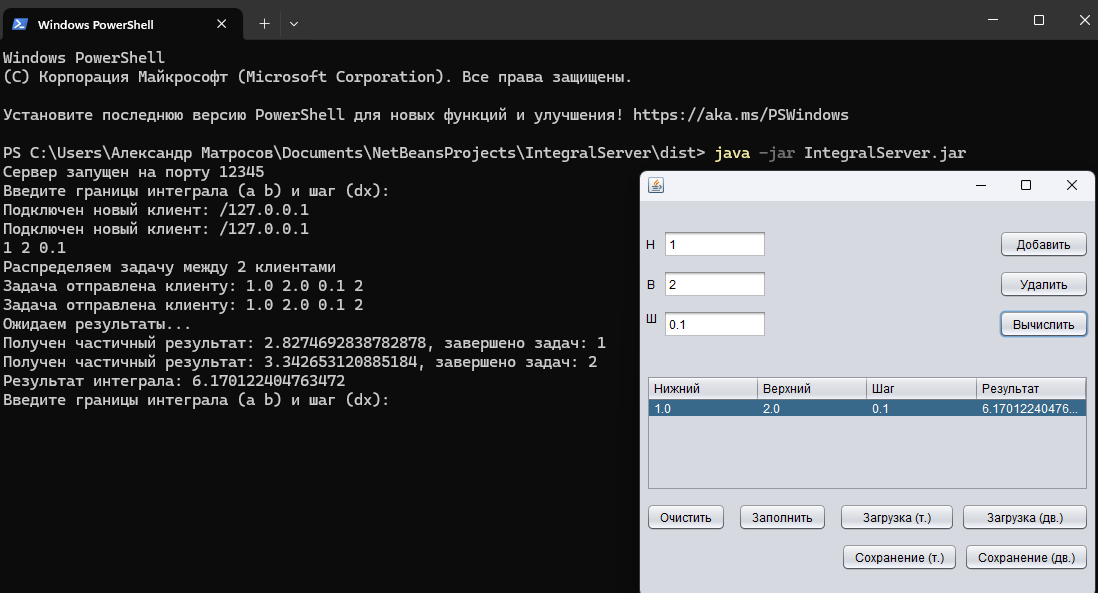
**tasksForThisClient = tasksPerClient + (clientId % numClients < extraTasks ? 1 : 0);**

* tasksPerClient — стандартное количество задач.
* (clientId % numClients < extraTasks ? 1 : 0) — если клиент должен получить лишнюю задачу, добавляем 1 к его задачам.

**Результат работы программы**

******

***Рисунок 1 – Полный цикл клиент-серверного взаимодействия.***

******

***Рисунок 2 – Расчёт совпал с лаб.работой 5.***

**Вывод:** в ходе выполнения лабораторной научились создавать клиент-серверные приложения на языке Java.

**Листинг**

**IntegralServer.java**

/\*

\* Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license

\* Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Main.java to edit this template

\*/

package integralserver;

import java.io.\*;

import java.net.\*;

import java.util.ArrayList;

import java.util.Scanner;

public class IntegralServer {

private static final int PORT = 12345;

private static ArrayList<ClientHandler> clients = new ArrayList<>();

private static double totalResult = 0;

private static int completedTasks = 0;

private static final int NUM\_SUBINTERVALS = 10;

public static void main(String[] args) throws IOException {

ServerSocket serverSocket = new ServerSocket(PORT);

System.out.println("Сервер запущен на порту " + PORT);

new Thread(() -> {

while (true) {

try {

Socket clientSocket = serverSocket.accept();

ClientHandler client = new ClientHandler(clientSocket);

synchronized (clients) {

clients.add(client);

}

System.out.println("Подключен новый клиент: " + clientSocket.getInetAddress());

new Thread(client).start();

} catch (IOException e) {

System.err.println("Ошибка подключения клиента: " + e.getMessage());

}

}

}).start();

Scanner scanner = new Scanner(System.in);

while (true) {

System.out.println("Введите границы интеграла (a b) и шаг (dx):");

String input = scanner.nextLine();

String[] parts = input.split(" ");

if (parts.length != 3) {

System.out.println("Неверный формат ввода. Пример: 1 2 0.01");

continue;

}

try {

double a = Double.parseDouble(parts[0]);

double b = Double.parseDouble(parts[1]);

double dx = Double.parseDouble(parts[2]);

if (a >= b || dx <= 0) {

System.out.println("Некорректные параметры: a должно быть меньше b, dx должен быть положительным");

continue;

}

synchronized (clients) {

if (clients.isEmpty()) {

System.out.println("Нет подключенных клиентов!");

continue;

}

distributeTask(a, b, dx);

}

} catch (NumberFormatException e) {

System.out.println("Ошибка ввода: введите корректные числа!");

}

}

}

private static void distributeTask(double a, double b, double dx) throws IOException {

totalResult = 0;

completedTasks = 0;

int numClients = clients.size();

System.out.println("Распределяем задачу между " + numClients + " клиентами");

for (ClientHandler client : clients) {

client.sendTask(a, b, dx, numClients);

System.out.println("Задача отправлена клиенту: " + a + " " + b + " " + dx + " " + numClients);

}

System.out.println("Ожидаем результаты...");

synchronized (IntegralServer.class) {

while (completedTasks < numClients) {

try {

IntegralServer.class.wait();

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

}

}

}

System.out.println("Результат интеграла: " + totalResult);

}

private static synchronized void addPartialResult(double partialResult) {

totalResult += partialResult;

completedTasks++;

System.out.println("Получен частичный результат: " + partialResult + ", завершено задач: " + completedTasks);

if (completedTasks == clients.size()) {

IntegralServer.class.notifyAll();

}

}

static class ClientHandler implements Runnable {

private Socket socket;

private PrintWriter out;

private BufferedReader in;

public ClientHandler(Socket socket) throws IOException {

this.socket = socket;

out = new PrintWriter(socket.getOutputStream(), true);

in = new BufferedReader(new InputStreamReader(socket.getInputStream()));

}

public void sendTask(double a, double b, double dx, int numClients) throws IOException {

out.println(a + " " + b + " " + dx + " " + numClients);

out.flush();

}

@Override

public void run() {

try {

String line;

while ((line = in.readLine()) != null) {

try {

double result = Double.parseDouble(line);

addPartialResult(result);

} catch (NumberFormatException e) {

System.err.println("Ошибка разбора результата от клиента: " + line);

}

}

} catch (IOException e) {

System.err.println("Клиент отключился: " + e.getMessage());

} finally {

synchronized (clients) {

clients.remove(this);

}

try {

socket.close();

} catch (IOException e) {

System.err.println("Ошибка закрытия сокета: " + e.getMessage());

}

}

}

}

}

**IntegralClient.java**

/\*

\* Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license

\* Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Main.java to edit this template

\*/

package integralclient;

import java.io.\*;

import java.net.\*;

import java.util.Random;

public class IntegralClient {

private static final String SERVER\_ADDRESS = "localhost";

private static final int SERVER\_PORT = 12345;

private static final int NUM\_THREADS = 10;

private double totalResult = 0;

private int completedThreads = 0;

private PrintWriter out;

private int tasksForThisClient;

public static void main(String[] args) {

try (Socket socket = new Socket(SERVER\_ADDRESS, SERVER\_PORT);

BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));

PrintWriter outWriter = new PrintWriter(socket.getOutputStream(), true)) {

IntegralClient client = new IntegralClient(outWriter);

System.out.println("Подключено к серверу");

Random rand = new Random();

int clientId = rand.nextInt(1000);

System.out.println("ID клиента: " + clientId);

while (true) {

String task = in.readLine();

if (task == null) break;

System.out.println("Получена задача: " + task);

String[] parts = task.split(" ");

double a = Double.parseDouble(parts[0]);

double b = Double.parseDouble(parts[1]);

double dx = Double.parseDouble(parts[2]);

int numClients = Integer.parseInt(parts[3]);

client.computeIntegral(a, b, dx, numClients, clientId);

}

} catch (IOException e) {

System.err.println("Ошибка: " + e.getMessage());

}

}

public IntegralClient(PrintWriter out) {

this.out = out;

}

private void computeIntegral(double a, double b, double dx, int numClients, int clientId) {

totalResult = 0;

completedThreads = 0;

double interval = (b - a) / NUM\_THREADS;

int tasksPerClient = NUM\_THREADS / numClients;

int extraTasks = NUM\_THREADS % numClients;

int startTask = clientId % numClients \* tasksPerClient + Math.min(clientId % numClients, extraTasks);

tasksForThisClient = tasksPerClient + (clientId % numClients < extraTasks ? 1 : 0);

System.out.println("Клиент " + clientId + ": начальная задача=" + startTask + ", задач для клиента=" + tasksForThisClient);

if (startTask >= NUM\_THREADS) {

System.out.println("Клиент " + clientId + ": задач не назначено");

return;

}

double clientA = a + startTask \* interval;

double clientB = clientA + tasksForThisClient \* interval;

System.out.println("Клиент " + clientId + ": вычисляем от " + clientA + " до " + clientB);

Thread[] threads = new Thread[tasksForThisClient];

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

double subA = clientA + i \* interval;

double subB = subA + interval;

IntegralCalculator calculator = new IntegralCalculator(subA, subB, dx, this);

threads[i] = new Thread(calculator);

threads[i].start();

}

for (Thread thread : threads) {

try {

thread.join();

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

}

}

System.out.println("Клиент " + clientId + ": все потоки завершены");

}

private double trapezoidalIntegral(double a, double b, double dx) {

int n = (int) Math.ceil((b - a) / dx);

if (n <= 0) return 0;

double sum = (function(a) + function(b)) / 2.0;

for (int i = 1; i < n; i++) {

double x = a + i \* dx;

sum += function(x);

}

return sum \* dx;

}

private double function(double x) {

return Math.exp(x) / x;

}

private synchronized void addPartialResult(double partialResult) {

totalResult += partialResult;

completedThreads++;

System.out.println("Клиент: добавлен частичный результат, итог=" + totalResult + ", завершено=" + completedThreads + "/" + tasksForThisClient);

if (completedThreads == tasksForThisClient) {

out.println(totalResult);

out.flush();

System.out.println("Клиент: отправлен результат серверу: " + totalResult);

}

}

class IntegralCalculator implements Runnable {

private double a, b, step;

private IntegralClient parent;

public IntegralCalculator(double a, double b, double step, IntegralClient parent) {

this.a = a;

this.b = b;

this.step = step;

this.parent = parent;

}

@Override

public void run() {

double result = parent.trapezoidalIntegral(a, b, step);

parent.addPartialResult(result);

}

}

}