МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

ЛАБОРАТОРНАЯ РАБОТА №2

по дисциплине «Параллельные алгоритмы и системы»

Тема: Передача данных по процессам

Студент гр. 1307 _____ Угрюмов М.М.

Преподаватель _____

Санкт-Петербург

2025

Введение

Тема работы: запуск параллельной программы и передача данных по процессам.

Цель работы: освоить функции передачи данных между процессами.

Лабораторная работа №2

Задание №1:

1) Запустить 4 процесса.

2) На каждом процессе создать переменные: ai,bi,ci, где I – номер процесса. Инициализировать переменные. Вывести данные на печать.

3) Передать данные на другой процесс. Напечатать номера процессов и поступившие данные. Найти: c0=a1+b2; c1=a3+b0; c2=a0+b3; c3=a2+b1.

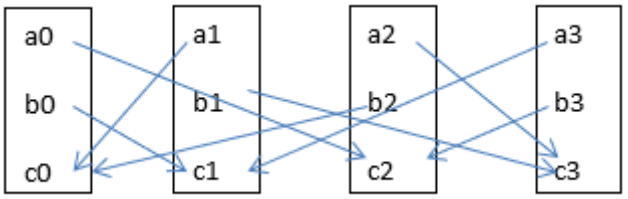

Рис.1 – Задание №1

Задание №2 (Вариант 6):

Запустить n процессов и найти количество четных положительных элементов.

Ход работы

Задание №1

Каждый процесс (0, 1, 2, 3) имеет заранее определенные отправляющие и принимающие процессы: send_a_to и send_b_to — массивы, которые содержат индексы процессов, на которые текущий процесс отправляет свои значения a и b. recv_a_from и recv_b_from — массивы, которые содержат индексы процессов, от которых текущий процесс будет получать значения a и b.

Каждый процесс вызывает метод `sendData(a, b, send_a_to, send_b_to)`, чтобы отправить свои значения переменных a и b на другие процессы.

`MPI.COMM_WORLD.Send(new int[]{a}, 0, 1, MPI.INT, send_a_to, 99)` отправляет значение a на процесс, указанный в send_a_to. `MPI.COMM_WORLD.Send(new int[]{b}, 0, 1, MPI.INT, send_b_to, 99)` отправляет значение b на процесс, указанный в send_b_to.

После того как процесс отправил свои данные, он получает данные от других процессов, вызывая метод `receiveData(received_a, received_b, recv_a_from, recv_b_from)`. `MPI.COMM_WORLD.Recv(received_a, 0, 1, MPI.INT, recv_a_from, 99)` получает значение a от процесса, указанного в recv_a_from. `MPI.COMM_WORLD.Recv(received_b, 0, 1, MPI.INT, recv_b_from, 99)` получает значение b от процесса, указанного в recv_b_from.

После того как процесс получил данные от других процессов, он выводит полученные значения и вычисляет результат: сумму полученных значений a и b.

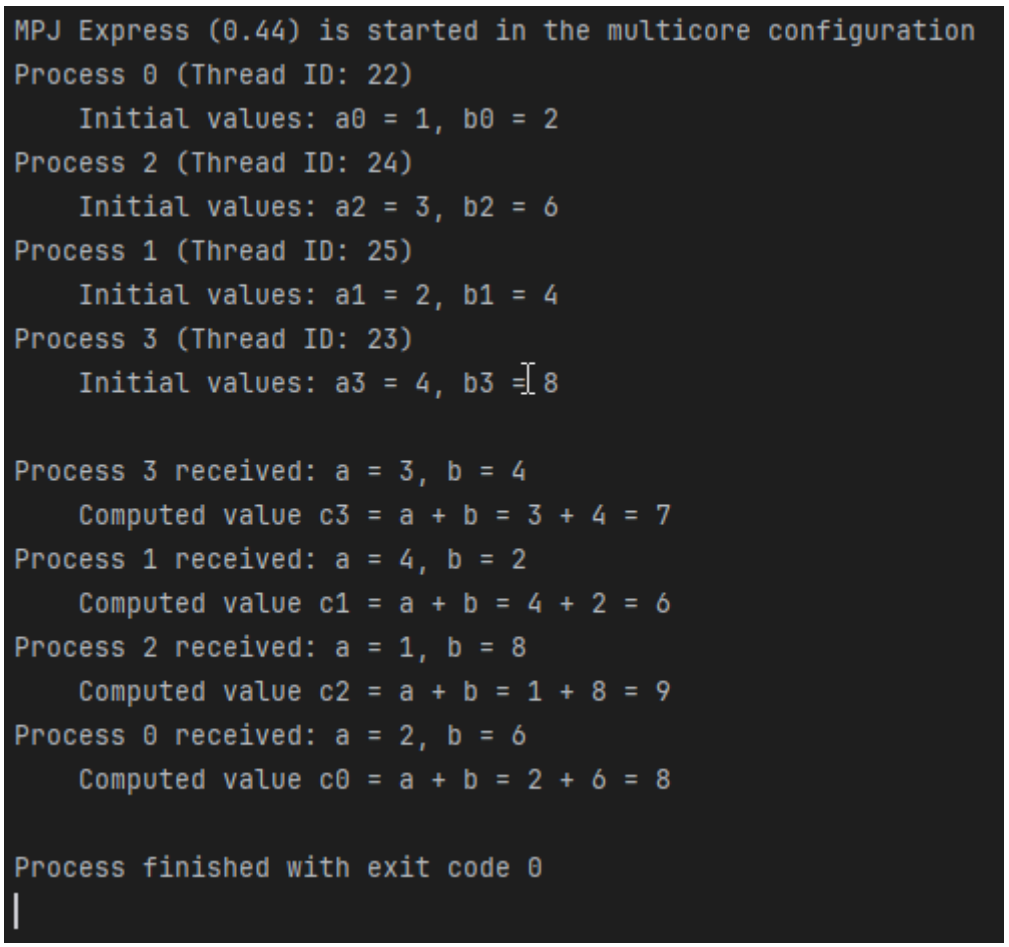

Рис.2 – Результат выполнения задания №1

Задание №2

В качестве примера исходное кол-во процессов – -np = 4.

Каждый процесс генерирует массив из 10 случайных чисел в диапазоне от -100 до 100 с помощью `random.nextInt(201) - 100`.
Подсчёт чётных положительных чисел:

Каждый процесс вызывает метод `countPositive(numbers)`, который считает количество четных положительных чисел в локальном массиве чисел, сгенерированном этим процессом.

Каждый процесс передаёт количество найденных им чётных положительных чисел процессу 0.

Процесс 0 принимает данные от всех процессов и суммирует их для получения общего числа чётных положительных чисел, которое выводится в конце.

```
MPJ Express (0.44) is started in the multicore configuration
Process 1 (Thread ID: 25) numbers: [-68, -55, 34, 62, -15, 26, -73, 6, -73, -98]
Process 1 found 4 even positive numbers.
Process 0 (Thread ID: 22) numbers: [-79, -40, 19, -34, -45, -73, -82, -44, 20, -14]
Process 0 found 1 even positive numbers.
Process 2 (Thread ID: 23) numbers: [67, 85, -79, -33, -49, -35, -86, 48, 82, 26]
Process 2 found 3 even positive numbers.
Process 3 (Thread ID: 24) numbers: [11, 89, 21, -8, 2, 19, -52, 2, 48, -53]
Process 3 found 3 even positive numbers.
Total even positive numbers: 11

Process finished with exit code 0
```

Рис.3 – Результат выполнения задания №2

Листинг программ

Lab2Task1.java

```
import mpi.MPI;
public class Lab2Task1 {
    public static void main(String[] args) {
        MPI.Init(args);
        int rank = MPI.COMM_WORLD.Rank();
        int a = rank + 1;
        int b = (rank + 1) * 2;
        System.out.printf("Process %d (Thread ID: %d)\n", rank,
Thread.currentThread().getId());
        System.out.printf("    Initial values: a%d = %d, b%d = %d\n", rank, a, rank, b);

        int[] received_a = new int[1];
        int[] received_b = new int[1];
        int[] send_a_to = {2, 0, 3, 1};
        int[] send_b_to = {1, 3, 0, 2};
        int[] recv_a_from = {1, 3, 0, 2};
        int[] recv_b_from = {2, 0, 3, 1};
        if (rank == 0 || rank == 3) {
            sendData(a, b, send_a_to[rank], send_b_to[rank]);
            receiveData(received_a, received_b, recv_a_from[rank], recv_b_from[rank]);
        } else {
            receiveData(received_a, received_b, recv_a_from[rank], recv_b_from[rank]);
            sendData(a, b, send_a_to[rank], send_b_to[rank]);
        }

        System.out.println(" \n");
        System.out.printf("Process %d received: a = %d, b = %d\n", rank, received_a[0],
received_b[0]);
        System.out.printf("    Computed value c%d = a + b = %d + %d = %d", rank,
received_a[0], received_b[0], (received_a[0] + received_b[0]));

        MPI.Finalize();
    }

    private static void sendData(int a, int b, int send_a_to, int send_b_to) {
        MPI.COMM_WORLD.Send(new int[]{a}, 0, 1, MPI.INT, send_a_to, 99);
        MPI.COMM_WORLD.Send(new int[]{b}, 0, 1, MPI.INT, send_b_to, 99);
    }

    private static void receiveData(int[] received_a, int[] received_b, int recv_a_from, int
recv_b_from) {
        MPI.COMM_WORLD.Recv(received_a, 0, 1, MPI.INT, recv_a_from, 99);
        MPI.COMM_WORLD.Recv(received_b, 0, 1, MPI.INT, recv_b_from, 99);
    }
}
```

Lab2Task2.java

```
import mpi.MPI;

import java.util.Random;

public class Lab2Task2 {

    public static void main(String[] args) {
        MPI.Init(args);

        int size = MPI.COMM_WORLD.Size();
        int rank = MPI.COMM_WORLD.Rank();

        Random random = new Random();
        int numElements = 10;
        int[] numbers = new int[numElements];
        for (int i = 0; i < numElements; i++) {
            numbers[i] = random.nextInt(201) - 100;
        }

        int localEvenCount = countPositive(numbers);

        System.out.printf("Process %d (Thread ID: %d) numbers: %s\n", rank,
                Thread.currentThread().getId(), arrayToString(numbers));
        System.out.printf("Process %d found %d even positive numbers.\n", rank,
                localEvenCount);

        int totalEvenCount;

        if (rank == 0) {
            totalEvenCount = localEvenCount;
            for (int i = 1; i < size; i++) {
                int[] received = new int[1];
                MPI.COMM_WORLD.Recv(received, 0, 1, MPI.INT, i, 99);
                totalEvenCount += received[0];
            }
            System.out.printf("Total even positive numbers: %d\n", totalEvenCount);
        } else {
            MPI.COMM_WORLD.Send(new int[]{localEvenCount}, 0, 1, MPI.INT, 0, 99);
        }

        MPI.Finalize();
    }


    private static int countPositive(int[] numbers) {
        int count = 0;
        for (int num : numbers) {
            if (num > 0 && num % 2 == 0) {
```

```
                count++;
            }
        }
        return count;
    }

    private static String arrayToString(int[] array) {
        StringBuilder sb = new StringBuilder("[");
        for (int i = 0; i < array.length; i++) {
            sb.append(array[i]);
            if (i < array.length - 1) {
                sb.append(", ");
            }
        }
        sb.append("]");
        return sb.toString();
    }
}
```