**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра вычислительной техники**

**ОТЧЕТ**

**по лабораторной работе №2**

**по дисциплине «Параллельные вычисления»**

**Тема: «Передача данных по процессам»**

| Студентка гр. 1307 |  | Голубев М.А. |
| --- | --- | --- |
| Преподаватель |  | Манжиков Л.П. |

Санкт-Петербург

2025

**Цель работы.**

Освоить функции передачи данных между процессами.

**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; с1=a3+b0; c2=a0+b3; c3=a2+b1.



Текст программы lab2\_1.c.

#include <mpi.h>

#include <stdio.h>

int main(int argc, char \*argv[])

{

int rank, size;

int ai, bi, ci;

int received\_a, received\_b;

MPI\_Init(&argc, &argv);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

if (size != 4)

{

if (rank == 0)

{

printf("Эта программа требует ровно 4 процесса.\n");

}

MPI\_Finalize();

return 1;

}

ai = rank + 1;

bi = (rank + 1) \* 10;

ci = 0;

printf("Процесс %d: ai = %d, bi = %d, ci = %d\n", rank, ai, bi, ci);

if (rank == 0)

{

MPI\_Send(&ai, 1, MPI\_INT, 1, 0, MPI\_COMM\_WORLD);

MPI\_Send(&bi, 1, MPI\_INT, 1, 1, MPI\_COMM\_WORLD);

MPI\_Recv(&received\_a, 1, MPI\_INT, 3, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

MPI\_Recv(&received\_b, 1, MPI\_INT, 3, 1, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

}

else if (rank == 1)

{

MPI\_Send(&ai, 1, MPI\_INT, 2, 0, MPI\_COMM\_WORLD);

MPI\_Send(&bi, 1, MPI\_INT, 2, 1, MPI\_COMM\_WORLD);

MPI\_Recv(&received\_a, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

MPI\_Recv(&received\_b, 1, MPI\_INT, 0, 1, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

}

else if (rank == 2)

{

MPI\_Send(&ai, 1, MPI\_INT, 3, 0, MPI\_COMM\_WORLD);

MPI\_Send(&bi, 1, MPI\_INT, 3, 1, MPI\_COMM\_WORLD);

MPI\_Recv(&received\_a, 1, MPI\_INT, 1, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

MPI\_Recv(&received\_b, 1, MPI\_INT, 1, 1, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

}

else if (rank == 3)

{

MPI\_Send(&ai, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD);

MPI\_Send(&bi, 1, MPI\_INT, 0, 1, MPI\_COMM\_WORLD);

MPI\_Recv(&received\_a, 1, MPI\_INT, 2, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

MPI\_Recv(&received\_b, 1, MPI\_INT, 2, 1, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

}

printf("Процесс %d получил данные: a = %d, b = %d\n", rank, received\_a, received\_b);

if (rank == 0)

{

ci = received\_a + received\_b;

}

else if (rank == 1)

{

ci = received\_a + received\_b;

}

else if (rank == 2)

{

ci = received\_a + received\_b;

}

else if (rank == 3)

{

ci = received\_a + received\_b;

}

printf("Процесс %d: ci = %d\n", rank, ci);

MPI\_Finalize();

return 0;

}

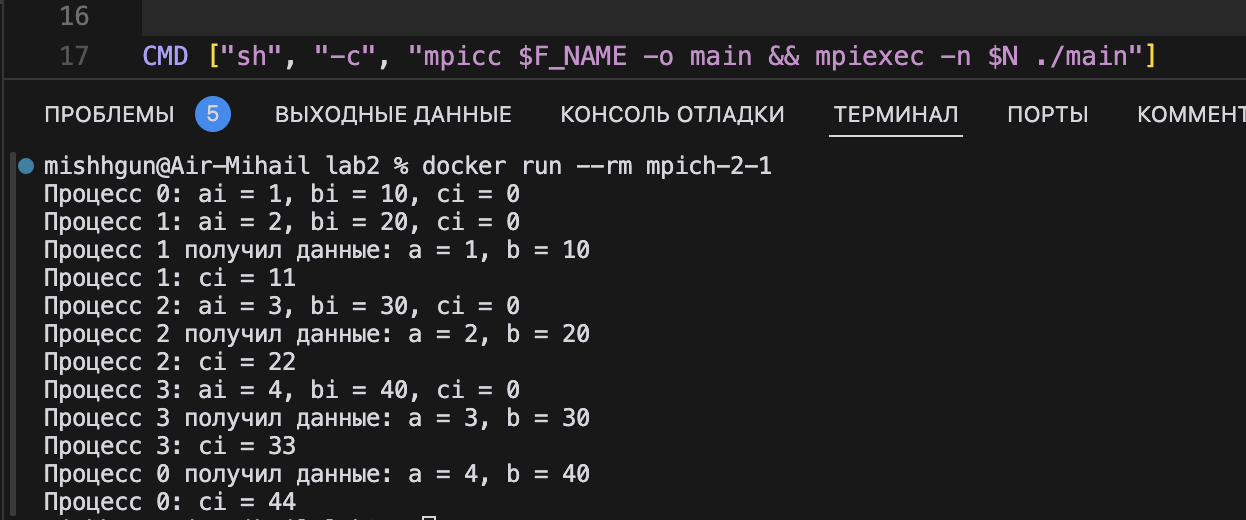


Рисунок 1. Запуск программы на 4-х процессах.

**Задание 2.**

Запустить n процессов и найти по вариантам:

1. Сумму нечетных элементов вектора;
2. Минимальный элемент;
3. Максимальный элемент;
4. Среднее арифметическое элементов вектора;
5. Сумму элементов кратных 3;
6. Количество четных положительных элементов;
7. Максимальный элемент среди отрицательных чисел;
8. **Минимальный элемент среди положительных чисел;**
9. Сумму элементов из заданного пользователем диапазона.

Текст программы lab2\_2.c

#include <mpi.h>

#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

int main(int argc, char \*argv[])

{

int rank, size;

int min\_positive = INT\_MAX;

// Инициализация MPI

MPI\_Init(&argc, &argv);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

srand(rank \* time(NULL));

int random\_number = rand() % 100 + 1;

printf("Процесс %d сгенерировал число: %d\n", rank, random\_number);

if (random\_number > 0 && random\_number < min\_positive)

{

min\_positive = random\_number;

}

// Собираем минимальные значения со всех процессов

int global\_min\_positive;

MPI\_Reduce(&min\_positive, &global\_min\_positive, 1, MPI\_INT, MPI\_MIN, 0, MPI\_COMM\_WORLD);

if (rank == 0)

{

printf("Минимальный положительный элемент: %d\n", global\_min\_positive);

}

MPI\_Finalize();

return 0;

}

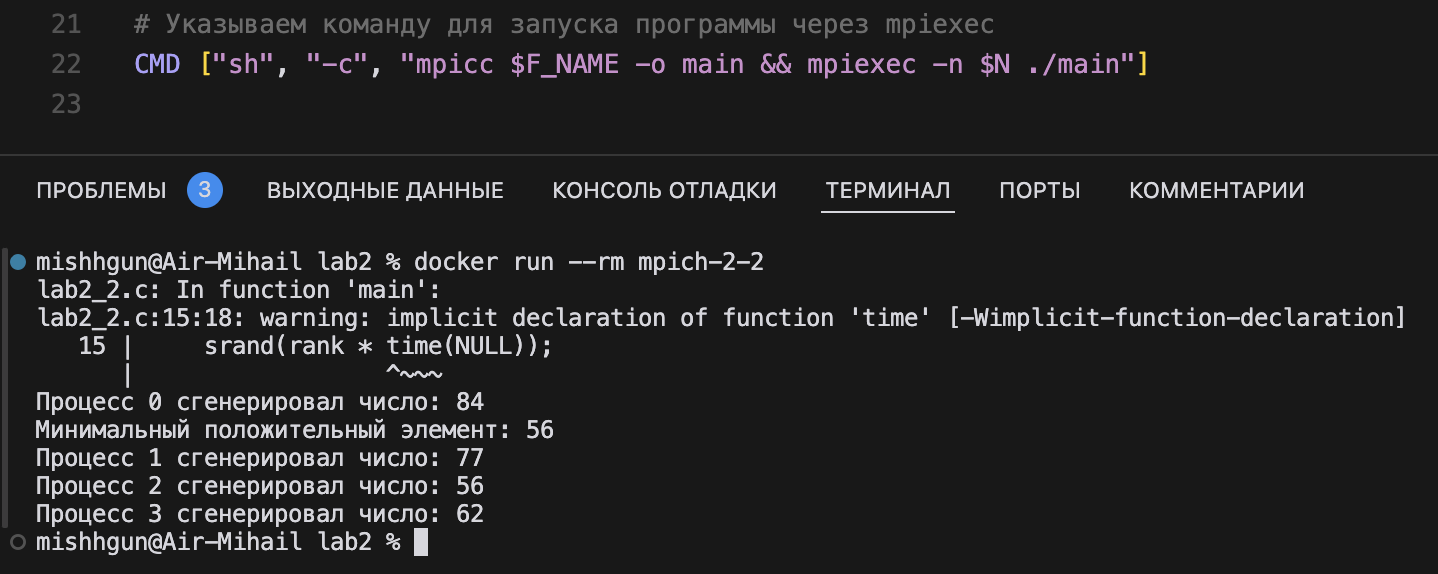


Рисунок 2. Запуск программы на 4-х процессах

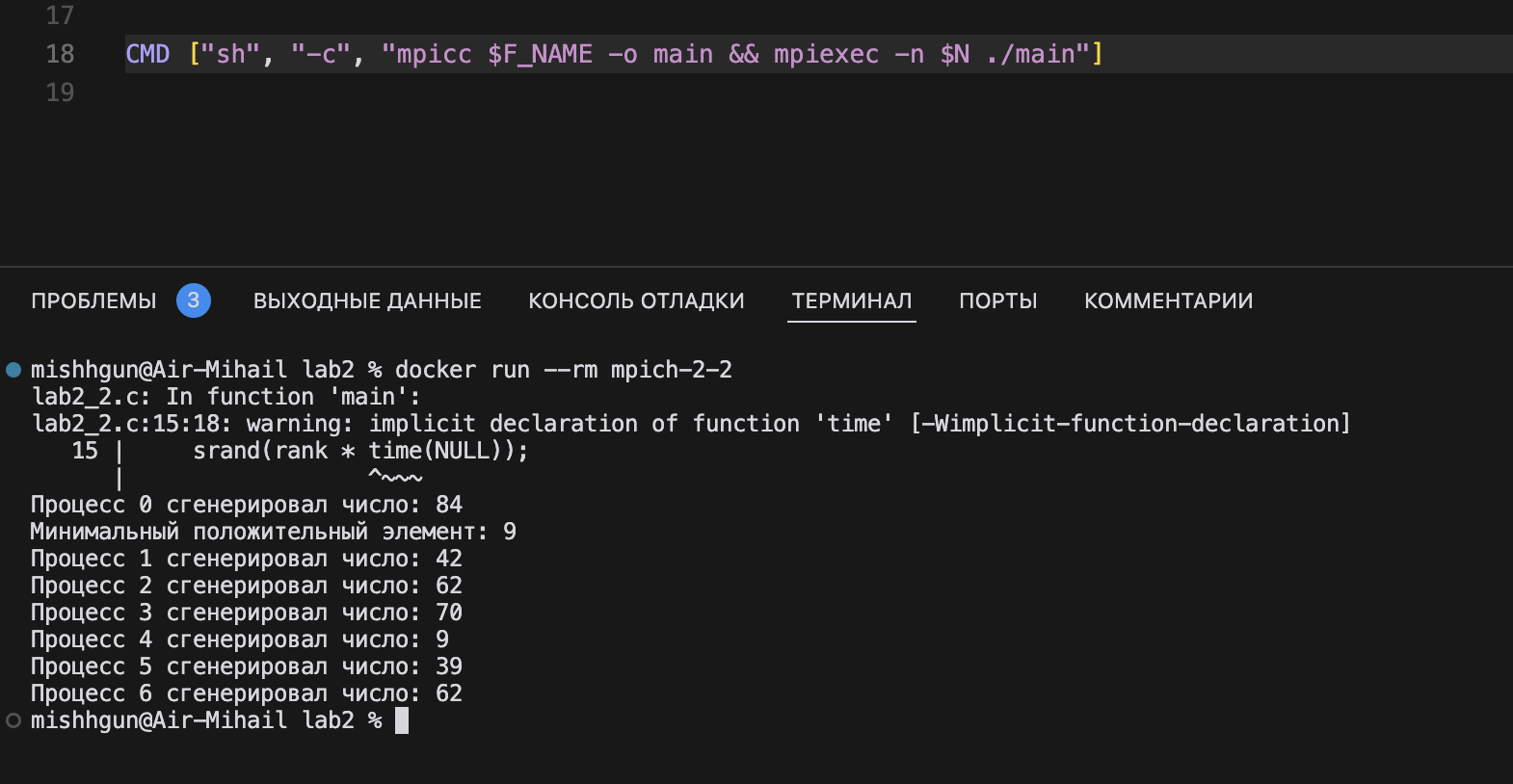


Рисунок 3. Запуск программы на 7-ми процессах

**Выводы.**

В ходе выполнения лабораторной работы были изучены ключевые идеи и возможности MPI, которые требуются для создания параллельных программ. В частности, были освоены процедуры инициализации, определения количества и рангов процессов, передачи и приема данных, распределения вычислений и сбора результатов.

Были получены практические умения работы с базовыми функциями MPI для создания параллельных приложений. Это является основой для разработки более сложных параллельных алгоритмов. Также был получен опыт разработки, тестирования и отладки параллельных программ.