Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**ЛАБОРАТОРНАЯ РАБОТА №3**

**Дисциплина: Распределенные задачи и алгоритмы**

Работу выполнила: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Д. Н. Баева

Направление подготовки: 02.03.02 Фундаментальная информатика и информационные технологии

Преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В. И. Шиян

**Тема работы:** Неблокирующие обмены (пробники).

**Ход работы:**

В первом задании лабораторной работы необходимо было дополнить данную программу с пробниками и выполнить её. Исходная программа написана на языке программирования Java.

По предоставленному фрагменту кода можно понять, что в программе происходит отправка и получение данных между тремя процессами.

Сперва программа инициализирует среду MPI с помощью MPI.Init(args) (необходимо для успешного использования MPI). Затем она получает ранг и размер текущего процесса с помощью стандартных методов MPI.COMM\_WORLD.Rank() и MPI.COMM\_WORLD.Size() соответственно. Так как по условию задачи сообщения передаются между тремя процессами, предварительно необходимо их количество. Фрагмент кода, выполняющий это, представлен на рисунке.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 1 – Проверка количества рангов.

Условный оператор if проверяет значение переменной ранга процесса и если он не равен 3, то программа прервёт свою работу, в противном случае – продолжит выполнение по тексту. Изменить текущее количество процессов можно в настройках интерпретатора среды разработки.

Затем следует раздел с инициализацией: объявляются и инициализируются некоторые переменные, включая массив data с одним элементом и массив buf с тремя элементами.

После этого выполняется главная часть в программе – отправка, проверка и принятие сообщений. Логика передвижения сообщений заключается в следующем:

- если ранг процесса равен 0, то он отправляет данные из массива данных data[0] процессу с рангом 2 с использованием функции MPI.COMM\_WORLD.Send() и перед отправкой данных на экран выводится сообщение “Rank 0 отправляет данные: [значение]”;

- если ранг процесса равен 1, то он отправляет буфер buf процессу с рангом 2 с использованием аналогичной функции MPI.COMM\_WORLD.Send() и перед отправкой буфера на экран выводится сообщение “Rank 1 отправляет буфер: [значение]”;

- если ранг процесса равен 2, то он проверяет входящие сообщения от процессов с рангами 0 и 1 с использованием функции MPI.COMM\_WORLD.Iprobe(), когда сообщение обнаружено, выполняется прием данных в соответствующий массив back\_buf или back\_buf2 с использованием функции MPI.COMM\_WORLD.Recv()и полученные данные выводятся на экран в формате “Rank = [номер\_процесса], Принятые данные: [значение1] [значение2] …”.

Именно в применении функции MPI.COMM\_WORLD.Iprobe(), состоит работа с пробниками. Этот метод выполняет неблокирующую проверку наличия сообщения, которое может быть получено и соответствует шаблону, указанному в аргументах source, tag и comm. Функция возвращает значение flag = true, если сообщение, соответствующее указанному шаблону, может быть получено. Если сообщение обнаружено, то оно может быть получено с использованием функции MPI.COMM\_WORLD.Recv(). Проверка наличия сообщения в коде используется именно при работе с процессом с рангом 2, так как он является принимающим сообщения от процессов с рангами 0 и 1. Фрагмент кода, выполняющий это, и результат работы программы представлены на рисунке 2 и 3 соответственно.

Изображение выглядит как текст, снимок экрана

Автоматически созданное описаниеРисунок 2 – Логика передвижения сообщений.

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

Рисунок 3 – Результат выполнения программы.

Во втором задании лабораторной работы необходимо было реализовать так называемую задачу фильтрации с использованием вновь неблокирующих обменов, пробников и метода Waitall().

MPI.COMM\_WORLD.Waitall() — это функция библиотеки MPI, которая блокирует вызывающий процесс до тех пор, пока все операции, связанные с активными запросами в массиве array\_of\_requests, не будут завершены. Она принимает три аргумента: count, array\_of\_requests и array\_of\_statuses.

Сначала процессы с рангами от 3 до size – 1, используя коллекцию Random. генерируют случайное число и сохраняют его в соответствующий элемент массива randNums. Затем они асинхронно отправляют сгенерированные числа процессам с рангами 1 и 2 с использованием функций MPI.COMM\_WORLD.Isend().

Далее процессы с рангами 1 и 2 асинхронно получают эти случайные числа от вышеопределённых процессов с помощью соответствующей функции MPI.COMM\_WORLD.Irecv() и Request.Waitall(). Создается массив запросов requests размером size – 3 и массив numbers аналогичного размера, а после для каждого процесса с рангом от 3 до size-1 создается запрос на прием числа с использованием функции MPI.COMM\_WORLD.Irecv(). Выполняется ожидание завершения всех запросов с использованием функции Request.Waitall().Полученные числа сохраняются в массиве numbers.

Полученные сообщения необходимо отсортировать между процессами. Первая половина достаётся процессу с рангом 1, вторая – процессу с рангом 2. Массив randNums, содержащий отсортированные числа, отправляется процессу с рангом 0.

В финальной части программы необходимо объединить отсортированные списки. Для этого сперва создаются переменные list1 и list2, которые будут хранить списки чисел от процессов с рангами 1 и 2 соответственно. С помощью пробников (функции MPI.Iprobe()) в цикле выполняется проверка наличия входящих сообщений от процессов с рангами 1 и 2. Эту проверку выполняет процесс с рангом 0, который ожидает получение этих сообщений. Когда сообщение обнаружено, то выполняется прием данных в массивы list1 и list2 с использованием синхронной функции MPI.COMM\_WORLD.Recv() от соответствующих процессов. Полученные списки чисел объединяются в один под названием mergedList (размерности равный сумме размеров списков list1 и list2) и сохраняются в порядке возрастания с помощью работы с циклом while. Результирующий список выводится в консоль. Результат работы программы на 10 процессах представлен на рисунке 3.

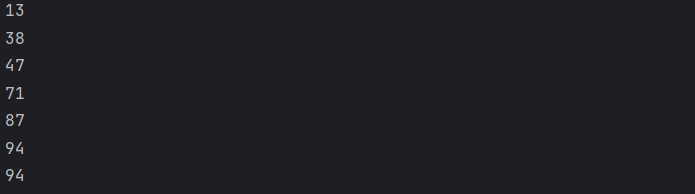


Рисунок 3 – Результат работы задачи фильтрации на 10 процессах.

Аналогично предыдущему заданию, количество процессов, задействованных в программе, можно изменить в конфигурации среды разработки.

**Листинг программ**

Файл LW\_3\_1.java

package LW\_3;  
import mpi.\*;  
import java.util.Arrays;  
  
class LW\_3\_1 {  
 public static void main(String[] args) throws MPIException {  
 MPI.*Init*(args);  
 int rank = MPI.*COMM\_WORLD*.Rank();  
 int size = MPI.*COMM\_WORLD*.Size();  
  
 // проверка количества рангов  
 if (size != 3) {  
 if (rank == 0) {  
 System.*out*.println("Необходимо запустить программу с 3 процессами");  
 }  
 MPI.*Finalize*();  
 return;  
 }  
  
 int data[] = new int[1];  
 int buf[] = {1, 3, 5};  
 int count, TAG = 0;  
 Status st;  
 Request request;  
  
 data[0] = 2016;  
  
  
 if (rank == 0) { // отправка процессу 2  
 System.*out*.println("Rank 0 отправляет данные: " + data[0]);  
 MPI.*COMM\_WORLD*.Send(data, 0, 1, MPI.*INT*, 2, TAG);  
 } else if (rank == 1) { // отправка процессу 2  
 System.*out*.println("Rank 1 отправляет буфер: " + Arrays.*toString*(buf));  
 MPI.*COMM\_WORLD*.Send(buf, 0, buf.length, MPI.*INT*, 2, TAG);  
 } else {  
 //проверка входящих сообщений  
 int source = MPI.*ANY\_SOURCE*;  
 int tag = MPI.*ANY\_TAG*;  
 int[] back\_buf;  
 int[] back\_buf2 = new int[buf.length];  
 // проверка первого сообщения (от процесса 0)  
 while (true) {  
 st = MPI.*COMM\_WORLD*.Iprobe(source, tag); // проверка наличия сообщения  
 if (st != null && st.tag == TAG) {  
 back\_buf = new int[st.Get\_count(MPI.*INT*)]; // массив полученных данных  
 break;  
 }  
 }  
 MPI.*COMM\_WORLD*.Recv(back\_buf, 0, back\_buf.length, MPI.*INT*, 0, TAG); // приём сообщений  
 System.*out*.print("Rank = 0, Принятые данные: "); // вывод данных, которые были приняты  
 for (int i = 0; i < back\_buf.length; i++) {  
 System.*out*.print(back\_buf[i] + " ");  
 }  
 System.*out*.println();  
 // проверка второго сообщения (от процесса 1)  
 while (true) {  
 st = MPI.*COMM\_WORLD*.Iprobe(source, tag);  
 if (st != null && st.tag == TAG) {  
 back\_buf2 = new int[st.Get\_count(MPI.*INT*)];  
 break;  
 }  
 }  
 MPI.*COMM\_WORLD*.Recv(back\_buf2, 0, back\_buf2.length, MPI.*INT*, 1, TAG);  
 System.*out*.print("Rank = 1, Принятые данные: ");  
 for (int i = 0; i < back\_buf2.length; i++) {  
 System.*out*.print(back\_buf2[i] + " ");  
 }  
 System.*out*.println();  
 }  
  
 MPI.*Finalize*();  
 }  
}

Файл LW\_3\_2.java

package LW\_3;  
  
import mpi.\*;  
import java.util.Arrays;  
import java.util.Random;  
  
class LW\_3\_2 {  
 public static void main(String[] args) throws MPIException {  
 MPI.*Init*(args);  
 int rank = MPI.*COMM\_WORLD*.Rank();  
 int size = MPI.*COMM\_WORLD*.Size();  
  
 // генерация ранга от 3 до size - 1  
 Random rand = new Random();  
 int[] randNums = new int[size - 3];  
 if (rank >= 3) {  
 randNums[rank - 3] = rand.nextInt(100);  
 // ассинхронная отправка случайных чисел в процесс 1 и 2  
 MPI.*COMM\_WORLD*.Isend(randNums, rank - 3, 1, MPI.*INT*, 1, 0);  
 MPI.*COMM\_WORLD*.Isend(randNums, rank - 3, 1, MPI.*INT*, 2, 0);  
 }  
  
 // ассинхронное получение случайных чисел  
 if (rank == 1 || rank == 2) {  
 Request[] requests = new Request[size - 3];  
 int[] numbers = new int[size - 3];  
 for (int i = 3; i < size; i++) {  
 requests[i - 3] = MPI.*COMM\_WORLD*.Irecv(numbers, i - 3, 1, MPI.*INT*, i, 0);  
 }  
 Request.*Waitall*(requests);  
  
  
 // сортировка и ассинхронная отправка отсортированных чисел  
 Arrays.*sort*(numbers);  
 if (rank == 1) {  
 // ранг 1 - первая половина  
 randNums = Arrays.*copyOfRange*(numbers, 0, numbers.length / 2);  
 } else {  
 // ранг 2 - вторая половина  
 randNums = Arrays.*copyOfRange*(numbers, numbers.length / 2, numbers.length);  
 }  
 MPI.*COMM\_WORLD*.Isend(randNums, 0, randNums.length, MPI.*INT*, 0, 0);  
 }  
  
 // объединение отсортированных списков  
 if (rank == 0) {  
 int[] list1 = null, list2 = null;  
 Status status;  
 while (true) {  
 status = MPI.*COMM\_WORLD*.Iprobe(1, 0); // проверка наличия сообщения в ранге 1  
 if (status != null) {  
 list1 = new int[status.Get\_count(MPI.*INT*)];  
 MPI.*COMM\_WORLD*.Recv(list1,0, list1.length ,MPI.*INT*, 1, 0); // принимаем список сообщений  
 break;  
 }  
 }  
 while (true) {  
 status = MPI.*COMM\_WORLD*.Iprobe(2, 0); // проверка наличия сообщения в ранге 2  
 if (status != null) {  
 list2 = new int[status.Get\_count(MPI.*INT*)];  
 MPI.*COMM\_WORLD*.Recv(list2,0, list2.length, MPI.*INT*, 2, 0); // принимаем список сообщений  
 break;  
 }  
 }  
  
 // объединяем списки  
 int[] mergedList = new int[list1.length + list2.length];  
 int i = 0, j = 0, k = 0;  
 while (i < list1.length && j < list2.length) {  
 if (list1[i] <= list2[j]) {  
 mergedList[k++] = list1[i++];  
 } else {  
 mergedList[k++] = list2[j++];  
 }  
 }  
 while (i < list1.length) {  
 mergedList[k++] = list1[i++];  
 }  
 while (j < list2.length) {  
 mergedList[k++] = list2[j++];  
 }  
  
 // вывод результирующего списка  
 for (int num : mergedList) {  
 System.*out*.println(num);  
 }  
 }  
  
 MPI.*Finalize*();  
 }  
}