МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНСТИТУТ МАТЕМАТИКИ И ИНФОРМАЦИОННЫХ СИСТЕМ

ФАКУЛЬТЕТ КОМПЬЮТЕРНЫХ И ФИЗИКО-МАТЕМАТИЧЕСКИХ НАУК

КАФЕДРА ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Отчет по лабораторной №6

по дисциплине «Системное программирование»

**Основы MPI. Парные обмены.**

|  |  |  |  |
| --- | --- | --- | --- |
| Выполнил: | студент гр. ФИБ-3301-51-01 |  | / В. Р. Кочкин / |
| Проверил: | к.ф.-м.н. доцент каф. ПМиИ |  | / В. А. Бызов / |

Киров 2022

Цель работы

Получить навыки разработки параллельных программ с использованием попарного взаимодействия процессов в технологии MPI.

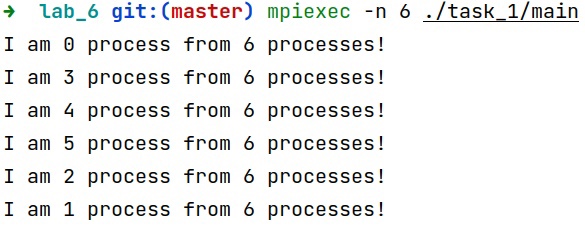
Задания

**Задание 1**

Напишите программу, в которой каждый процесс выводит на экран свой номер и общее количество процессов в формате

I am <Номер процесса > process from < Количество процессов> processes!

**Полученные результаты**



Листинг программы приведен в [приложении А задание 1](#_Задание_1.).

**Задание 2**

Напишите программу, в которой каждый процесс с чётным номером выводит на экран строку

<Номер процесса>: FIRST!

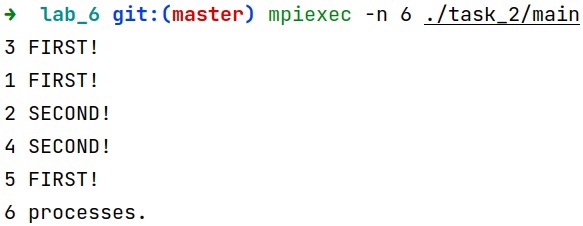
а каждый процесс с нечётным номером строку

<Номер процесса>: SECOND!

Нулевой процесс должен вывести на экран информацию о количестве работающих процессов в формате

<Количество процессов> processes.

**Полученные результаты**

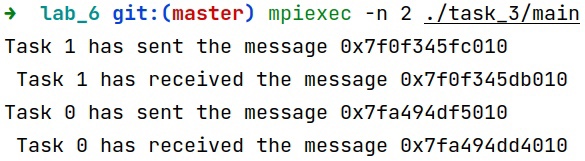


Листинг программы приведен в [приложении А задание 2](#_Задание_2.).

**Задание 3**

Скомпилируйте и запустите на выполнение приведённый ниже код. Поясните, почему возникла тупиковая ситуация? Исправьте программу, заменив блокирующие вызовы на неблокирующие.

**Полученные результаты**

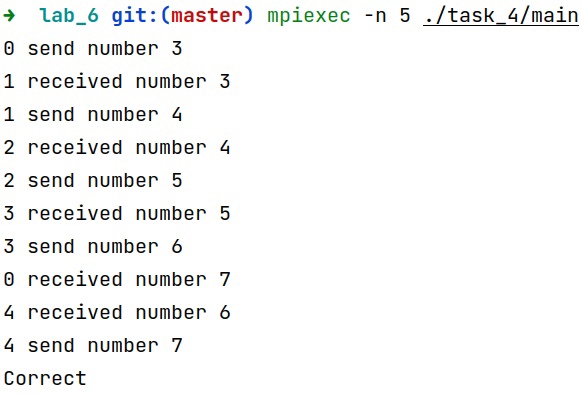


Листинг программы приведен в [приложении А задание 3](#_Задание_3.).

**Задание 4**

Эстафетная палочка. Организовать передачу данных по кругу. Нулевой процесс генерирует случайное целое число и передаёт его первому процессу. Далее каждый процесс получает некоторое целое число, прибавляет к нему 1 и передаёт следующему процессу. Последний процесс передаёт число нулевому. Нулевой процесс выводит «Correct!», если новое число на p – 1 больше исходного или «Error!» в противном случае. Остальные процессы выводят сообщение «<Номер процесса> receive number <число>».

**Полученные результаты**



Листинг программы приведен в [приложении А задание 4](#_Задание_4.).

**Задание 5**

Концепция master-slave. Нулевой процесс генерирует N массивов целых чисел из M элементов. Далее он распределяет по остальным процессам по одному массиву. Каждый процесс, получив массив, считает сумму его элементов и отсылает обратно нулевому. Нулевой процесс добавляет полученный результат к глобальной сумме и отправляет освободившемуся процессу новую работу. Так происходит до тех пор, пока не вычислена сумма всех элементов всех массивов (сумма элементов матрицы). Реализовать алгоритм двумя способами: с использованием блокирующих и неблокирующих операций. Протестировать для следующих пар N и M: 100 и 1000000, 10000 и 10000, 1000000 и 100. Сравнить быстродействие. Сделать выводы.

**Полученные результаты**

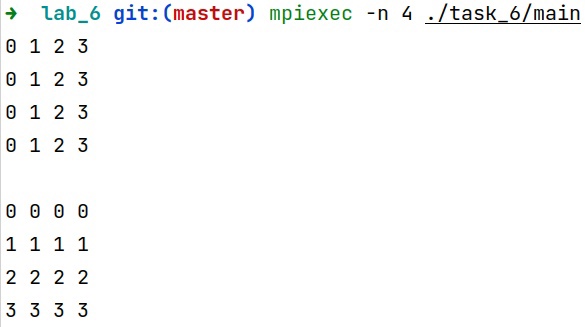


Листинг программы приведен в [приложении А задание 5](#_Задание_5.).

**Задание 6**

Каждый каждому. Целочисленная матрица размера p • p хранится следующим образом: на нулевом процессе – нулевая строка, на первом – первая и т. д. Организовать попарные обмены таким образом, чтобы транспонировать матрицу (на нулевом процессе будет лежать нулевой столбец исходной матрицы, на первом – первый столбец и т. д.)

**Полученные результаты**



Листинг программы приведен в [приложении А задание 6](#_Задание_6.).

Вывод

В ходе лабораторной работы я получил навыки разработки параллельных программ с использованием попарного взаимодействия процессов в технологии MPI.

# **Приложения**

## **Приложение А. Листинги программ**

### Задание 1.

#include <mpich/mpi.h>  
  
int main(int argc, char \*\*argv) {  
 MPI\_Init(&argc, &argv);  
  
 int proc\_num, proc\_rank;  
 MPI\_Comm\_size( MPI\_COMM\_WORLD, &proc\_num);  
 MPI\_Comm\_rank( MPI\_COMM\_WORLD, &proc\_rank);  
  
 printf("I am %d process from %d processes!\n", proc\_rank, proc\_num);  
  
 MPI\_Finalize();  
  
 return 0;  
}

### Задание 2.

#include <mpich/mpi.h>  
  
int main(int argc, char \*\*argv) {  
 MPI\_Init(&argc, &argv);  
  
 int proc\_num, proc\_rank;  
 MPI\_Comm\_size( MPI\_COMM\_WORLD, &proc\_num);  
 MPI\_Comm\_rank( MPI\_COMM\_WORLD, &proc\_rank);  
  
 if (proc\_rank == 0) {  
 printf("%d processes\n", proc\_num);  
 return 0;  
 }  
  
 switch (proc\_rank % 2) {  
 case 0:  
 printf("%d SECOND!\n", proc\_rank);  
 break;  
 case 1:  
 printf("%d FIRST!\n", proc\_rank);  
 break;  
 }  
  
 MPI\_Finalize();  
  
 return 0;  
}

### Задание 3.

#include <iostream>  
#include <mpich/mpi.h>  
#include <vector>  
  
constexpr auto MSGLEN = 32768;  
constexpr auto TAG\_A = 100;  
constexpr auto TAG\_B = 200;  
  
using namespace std;  
  
int main(int argc, char \* argv[]) {  
 vector<float> message1(MSGLEN), message2(MSGLEN);  
 int rank, dest, source, send\_tag, recv\_tag;  
  
 MPI\_Request request;  
  
 MPI\_Init(&argc, &argv);  
 MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);  
  
 for (int i = 0; i < MSGLEN; i++) {  
 message1[i] = 1 - 2 \* rank;  
 }  
  
 if (rank == 0) {  
 dest = 1;  
 source = 1;  
 send\_tag = TAG\_A;  
 recv\_tag = TAG\_B;  
 }  
 else if (rank == 1) {  
 dest = 0;  
 source = 0;  
 send\_tag = TAG\_B;  
 recv\_tag = TAG\_A;  
 } else {  
 return 0;  
 }  
  
 MPI\_Isend(message1.data(), MSGLEN, MPI\_FLOAT, dest, send\_tag,  
 MPI\_COMM\_WORLD, &request);  
 cout << "Task " << rank << " has sent the message " << message1.data() << endl;  
  
  
 MPI\_Irecv(message2.data(), MSGLEN, MPI\_FLOAT, source, recv\_tag,  
 MPI\_COMM\_WORLD, &request);  
 cout << "Task " << rank << " has received the message " << message2.data() << endl;  
  
 MPI\_Finalize();  
  
 return 0;  
}

### Задание 4.

#include <iostream>  
#include <mpich/mpi.h>  
#include <ctime>  
#include <random>  
  
constexpr auto MSGLEN = 1;  
  
int main(int argc, char \* argv[]) {  
 srand(time(nullptr));  
  
 int dest, source;  
  
 MPI\_Status status;  
  
 MPI\_Init(&argc, &argv);  
  
 int proc\_num, proc\_rank;  
 MPI\_Comm\_size( MPI\_COMM\_WORLD, &proc\_num);  
 MPI\_Comm\_rank( MPI\_COMM\_WORLD, &proc\_rank);  
  
 int number = 0;  
 int numberCopy;  
 if (proc\_rank == 0) {  
 number = rand() % 10;  
 numberCopy = number;  
 dest = proc\_rank + 1;  
 source = proc\_num - 1;  
  
 MPI\_Send(&number, MSGLEN, MPI\_INT, dest, 1,  
 MPI\_COMM\_WORLD);  
 printf("%d send number %d\n", proc\_rank, number);  
  
 MPI\_Recv(&number, MSGLEN, MPI\_INT, source, 1,  
 MPI\_COMM\_WORLD, &status);  
 printf("%d received number %d\n", proc\_rank, number);  
  
 if (number - numberCopy == proc\_num - 1) {  
 std::cout << "Correct\n" << std::endl;  
 } else {  
 std::cout << "Error\n" << std::endl;  
 }  
 } else {  
 if (proc\_rank > 0 && proc\_rank < proc\_num - 1) {  
 dest = proc\_rank + 1;  
 source = proc\_rank - 1;  
 } else {  
 dest = 0;  
 source = proc\_rank - 1;  
 }  
  
 MPI\_Recv(&number, MSGLEN, MPI\_INT, source, 1,  
 MPI\_COMM\_WORLD, &status);  
 printf("%d received number %d\n", proc\_rank, number);  
  
 number++;  
  
 MPI\_Send(&number, MSGLEN, MPI\_INT, dest, 1,  
 MPI\_COMM\_WORLD);  
 printf("%d send number %d\n", proc\_rank, number);  
 }  
  
 MPI\_Finalize();  
  
 return 0;  
}

### Задание 5.

#include <iostream>  
#include <mpich/mpi.h>  
#include <vector>  
  
std::vector<std::vector<int>> create\_random\_vectors(int, int);  
std::vector<int> create\_random\_vector(int);  
  
int main(int argc, char \* argv[]) {  
 MPI\_Status status;  
  
 MPI\_Init(&argc, &argv);  
  
 int proc\_num, proc\_rank;  
 MPI\_Comm\_size( MPI\_COMM\_WORLD, &proc\_num);  
 MPI\_Comm\_rank( MPI\_COMM\_WORLD, &proc\_rank);  
  
 if (proc\_rank == 0) {  
 int n = 0, m = 0;  
  
 std::cout << "Enter n: ";  
 std::cin >> n;  
  
 std::cout << "Enter m: ";  
 std::cin >> m;  
  
 std::vector<std::vector<int>> vectors = create\_random\_vectors(n, m);  
 std::vector<int> sums;  
  
 int countRows = n / (proc\_num - 1);  
 int countRowsLast = countRows + n % (proc\_num - 1);  
 for(int i = 1; i < proc\_num; i++) {  
 if (i != proc\_num - 1) {  
 MPI\_Send(&countRows, 1, MPI\_INT, i, 0,MPI\_COMM\_WORLD);  
 } else {  
 MPI\_Send(&countRowsLast, 1, MPI\_INT, i, 0,MPI\_COMM\_WORLD);  
 }  
  
 MPI\_Send(&m, 1, MPI\_INT, i, 1,MPI\_COMM\_WORLD);  
 }  
  
 for(int i = 0; i < vectors.size(); i++) {  
 int targetThread = i % (proc\_num - 1) + 1;  
  
 if (i >= countRowsLast - 1) {  
 targetThread = proc\_num - 1;  
 }  
  
 MPI\_Send(vectors[i].data(), m, MPI\_INT, targetThread, targetThread,MPI\_COMM\_WORLD);  
 printf("%d send row to %d\n", proc\_rank, targetThread);  
  
 int sum = 0;  
 MPI\_Recv(&sum, 1, MPI\_INT, targetThread, targetThread,MPI\_COMM\_WORLD, &status);  
 printf("%d received sum from %d\n", proc\_rank, targetThread);  
  
 sums.push\_back(sum);  
 }  
  
 int resultSum = 0;  
 for (int sum : sums) {  
 resultSum += sum;  
 }  
 printf("%d result sum %d\n", proc\_rank, resultSum);  
 } else {  
 int n;  
 MPI\_Recv(&n, 1, MPI\_INT, 0, 0,MPI\_COMM\_WORLD, &status);  
 printf("received n=%d\n", n);  
  
 int m;  
 MPI\_Recv(&m, 1, MPI\_INT, 0, 1,MPI\_COMM\_WORLD, &status);  
  
 for(int i = 0; i < n; i++) {  
 std::vector<int> vector(m);  
  
 MPI\_Recv(vector.data(), m, MPI\_INT, 0, proc\_rank, MPI\_COMM\_WORLD, &status);  
  
 int sum = 0;  
 for (int item : vector) {  
 sum += item;  
 }  
  
 MPI\_Send(&sum, 1, MPI\_INT, 0, proc\_rank,MPI\_COMM\_WORLD);  
 }  
 }  
  
 MPI\_Finalize();  
  
 return 0;  
}  
  
std::vector<std::vector<int>> create\_random\_vectors(int count, int size) {  
 std::vector<std::vector<int>> vectors;  
  
 for (int i = 0; i < count; ++i) {  
 std::vector<int> vector = create\_random\_vector(size);  
 vectors.push\_back(vector);  
 }  
  
 return vectors;  
}  
  
std::vector<int> create\_random\_vector(int size) {  
 std::vector<int> vector;  
  
 for (int i = 0; i < size; ++i) {  
 vector.push\_back(rand() % 10);  
 }  
  
 return vector;  
}

### Задание 6.

#include <iostream>  
#include <mpich/mpi.h>  
#include <vector>  
#include <ctime>  
#include <random>  
  
std::vector<std::vector<int>> create\_random\_vectors(int, int);  
std::vector<int> create\_random\_vector(int);  
void proc\_print\_vector(int, int, const std::vector<int>&);  
  
int main(int argc, char \* argv[]) {  
 srand(time(nullptr));  
 MPI\_Status status;  
  
 MPI\_Init(&argc, &argv);  
  
 int proc\_num, proc\_rank;  
 MPI\_Comm\_size( MPI\_COMM\_WORLD, &proc\_num);  
 MPI\_Comm\_rank( MPI\_COMM\_WORLD, &proc\_rank);  
  
 int num;  
 std::vector<int> row = create\_random\_vector(proc\_num);  
  
 proc\_print\_vector(proc\_rank, proc\_num, row);  
  
 MPI\_Barrier(MPI\_COMM\_WORLD);  
  
 for(int i = 0; i < proc\_num; i++) {  
 if (i == proc\_rank) continue;  
 MPI\_Send(&row[i], 1, MPI\_INT, i, i, MPI\_COMM\_WORLD);  
 MPI\_Recv(&num, 1, MPI\_INT, i, proc\_rank,MPI\_COMM\_WORLD, &status);  
 row[i] = num;  
 }  
  
 proc\_print\_vector(proc\_rank, proc\_num, row);  
  
 MPI\_Finalize();  
  
 return 0;  
}  
  
std::vector<std::vector<int>> create\_random\_vectors(int count, int size) {  
 std::vector<std::vector<int>> vectors;  
  
 for (int i = 0; i < count; ++i) {  
 std::vector<int> vector = create\_random\_vector(size);  
 vectors.push\_back(vector);  
 }  
  
 return vectors;  
}  
  
std::vector<int> create\_random\_vector(int size) {  
 std::vector<int> vector;  
  
 for (int i = 0; i < size; ++i) {  
 vector.push\_back(i);  
 }  
  
 return vector;  
}  
  
void proc\_print\_vector(int proc\_rank, int proc\_num, const std::vector<int>& vector) {  
 int buf = 0;  
 MPI\_Status status;  
  
 if (proc\_rank != 0) {  
 MPI\_Recv(&buf, 1, MPI\_INT, proc\_rank - 1, MPI\_ANY\_TAG,MPI\_COMM\_WORLD, &status);  
 }  
 for(int element : vector) {  
 printf("%d ", element);  
 }  
 printf("\n");  
 MPI\_Send(&buf, 1, MPI\_INT, proc\_rank == proc\_num - 1 ? 0 : proc\_rank + 1, 0, MPI\_COMM\_WORLD);  
  
 if (proc\_rank == proc\_num - 1) {  
 printf("\n");  
 }  
}