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

# ОТЧЕТ

**по лабораторной работе №4**
**по дисциплине «Параллельный алгоритмы»**
**Тема: Группы процессов и коммуникаторы.**

Студент гр. 9383 _____ Ноздрин В.Я.

Преподаватель _____ Татаринов Ю.С.

Санкт-Петербург
2021

**Задание. Вариант 6.**

В главном процессе дано целое число K и набор из K вещественных чисел, в каждом подчиненном процессе дано целое число N, которое может принимать два значения: 0 и 1 (количество подчиненных процессов с N = 1 равно K). Используя функцию MPI_Comm_split и одну коллективную операцию пересылки данных, переслать по одному вещественному числу из главного процесса в каждый подчиненный процесс с N = 1 и вывести в этих подчиненных процессах полученные числа.

Указание. При вызове функции MPI_Comm_split в процессах, которые не требуется включать в новый коммуникатор, в качестве параметра color следует указывать константу MPI_UNDEFINED.

**Выполнение работы.**

Была написана программа, выполняемая на двух или более процессах. Процесс с рангом 0 генерирует K чисел а также набор чисел N так, чтобы было ровно K единиц, а остальные значения - нули. Дальше нулевой процесс рассылает всем остальным процессам их значения N. Процессы с ненулевыми рангами принимают сообщение от процесса с рангом ноль, сообщение содержит число N равное 0 или 1. Далее все процессы вызывают функцию MPI_Comm_split с соответствующими параметрами. Если N = 1, color задается равным 0, в противном случае используется метка MPI_UNDEFINED. MPI_Comm_split принимает на вход MPI_Comm new_comm, в который записывается коммуникатор для групп процессов. Если в color передавалась метка MPI_UNDEFINED, то коммуникатор в данном процессе равен MPI_COMM_NULL. Выполняется проверка, что коммуникатор не пустой и вызывается коллективная функция MPI_Bcast, рассылающая из нулевого процесса сгенерированные K чисел. Каждый процесс, получивший сообщение печатает его на экран с указанием своего ранга. Процесс с нулевым рангом имеет color = 0, потому в коммутаторе получается K+1 процессов, нулевой

процесс не печатает ничего и ничего не получает, он отправляет всем процессам сообщение, содержащее цифры от 0 до K-1.

Параметр K передается в приложение как аргумент командной строки.

```
ice-jack@ice-pc ~/P/P/lab4 (master)> mpic++ main.cpp && mpirun -n 5 --oversubscribe a.out 3
1 0 1 1 1
[Process 2] recieved 0
[Process 4] recieved 2
[Process 3] recieved 1
ice-jack@ice-pc ~/P/P/lab4 (master)> mpic++ main.cpp && mpirun -n 5 --oversubscribe a.out 3
1 1 1 0 1
[Process 2] recieved 1
[Process 1] recieved 0
[Process 4] recieved 2
ice-jack@ice-pc ~/P/P/lab4 (master)> mpic++ main.cpp && mpirun -n 5 --oversubscribe a.out 3
1 0 1 1 1
[Process 2] recieved 0
[Process 3] recieved 1
[Process 4] recieved 2
```

Рисунок 1. Запуск программы на 3 процессах три раза при K=3.

Видно, что при разных запусках разные процессы получают сообщения от главного процесса. Программа печатает сначала значения массива, задающего N для всех процессов, а затем каждый процесс печатает, когда получает сообщение от главного процесса.

```
ice-jack@ice-pc ~/P/P/lab4 (master)> mpic++ main.cpp && mpirun -n 10 --oversubscribe a.out 5
1 0 0 0 1 1 0 1 1 1
[Process 5] recieved 1
[Process 8] recieved 3
[Process 9] recieved 4
[Process 4] recieved 0
[Process 7] recieved 2
ice-jack@ice-pc ~/P/P/lab4 (master)> mpic++ main.cpp && mpirun -n 10 --oversubscribe a.out 5
1 0 1 0 1 0 0 1 1 1
[Process 4] recieved 1
[Process 8] recieved 3
[Process 2] recieved 0
[Process 7] recieved 2
[Process 9] recieved 4
ice-jack@ice-pc ~/P/P/lab4 (master)> mpic++ main.cpp && mpirun -n 10 --oversubscribe a.out 5
1 0 1 1 0 1 1 0 1 0
[Process 2] recieved 0
[Process 3] recieved 1
[Process 8] recieved 4
[Process 6] recieved 3
[Process 5] recieved 2
```

Рисунок 2. Запуск программы на 10 процессах три раза при K=5.

**Выводы.**

Получен опыт работы с методом MPI_Comm_split, разбивающим процессы на группы. Спроектирована и написана программа, выполняющая задачу согласно условию.

## ПРИЛОЖЕНИЕ

**Файл main.cpp**

```
#include <iostream>
#include "mpi.h"

int main(int argc, char* argv[]){
  int size, rank, K, N, color, key;;
  MPI_Init(&argc, &argv);
  MPI_Status Status;
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  srand(time(NULL) + rank);
  if (size < 2 || argc != 2) {
    if (rank == 0)
      std::cout << "This application is meant to be run with at
least 2 MPI processes and 1 comand line argument.\n";
    MPI_Abort(MPI_COMM_WORLD, EXIT_FAILURE);
  }
  K = std::stoi(argv[1]);
  int sendbuf[K];
  int valuesN[size] = {0};
  if (rank == 0) {
    // generate K numbers
    for (int i = 0; i < K; i++)
      sendbuf[i] = i;
    // generate N's
    int counter = K;
    valuesN[0] = 1;
    for (int i = 1; i < size; i++) {
      if (counter <= 0)
```

```
        break;
      if (size-i==counter) {
        for (int j=i; j < size; j++)
          valuesN[j] = 1;
        counter = 0;
      }
      if ((rand()%2) && (counter > 0)) {
        valuesN[i] = 1;
        counter--;
      }
    }
    for (int i = 0; i < size; i++) {
      MPI_Send(&valuesN, size, MPI_INT, i, 0, MPI_COMM_WORLD);
      std::cout << valuesN[i] << " ";
    }
    std::cout << "\n";
  } else {
    MPI_Recv(&valuesN, size, MPI_INT, 0, MPI_ANY_TAG,
MPI_COMM_WORLD, &Status);
  }
  N = valuesN[rank];
  if (N == 1) {
    color = 0;
    key = rank;
  } else {
    color = MPI_UNDEFINED;
    key = rank;
  }
  MPI_Comm new_comm;
  MPI_Comm_split(MPI_COMM_WORLD, color, key, &new_comm);
```

```
    if (new_comm != MPI_COMM_NULL) {
    MPI_Bcast(&sendbuf, K, MPI_INT, 0, new_comm);
    int new_rank;
    MPI_Comm_rank(new_comm, &new_rank);
    if (rank != 0)
      std::cout << "[Process " << rank << "] recieved " <<
sendbuf[new_rank-1] << "\n";
  }
  MPI_Finalize();
  return EXIT_SUCCESS;
}
```