

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

**ОТЧЕТ**  
**по лабораторной работе №4**  
**по дисциплине «ПА»**  
**Тема: Коммуникаторы**

Студент гр. 8304

\_\_\_\_\_

Сергеев А.Д.

Преподаватель

\_\_\_\_\_

Татаринов Ю.С.

Санкт-Петербург

2020

### **Задание.**

В каждом процессе дано целое число  $N$ , которое может принимать два значения: 0 и 1 (имеется хотя бы один процесс с  $N = 1$ ). Кроме того, в каждом процессе с  $N = 1$  дано вещественное число  $A$ . Используя функцию `MPI_Comm_split` и одну коллективную операцию пересылки данных, переслать числа  $A$  в первый из процессов с  $N = 1$  и вывести их в порядке возрастания рангов переславших их процессов (включая число, полученное из этого же процесса).

### **Описание алгоритма.**

Функция `MPI_Comm_split` расщепляет исходный коммуникатор на набор коммуникаторов, каждый из которых связывает процессы, входящие в исходный коммуникатор, для которых при расщеплении был задан один и тот же цвет. В последний аргумент, указатель на коммуникатор, кладётся указатель на именно тот коммуникатор, которому принадлежит данный процесс. Для четных рангов выполняется поиск минимума с помощью `MPI_Reduce`.

### **Листинг программы.**

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <mpi.h>

int main(int argc, char* argv[]) {
    int ProcNum, ProcRank;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &ProcNum);
    MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);
    srand(time(NULL) + ProcRank);

    double start_time = MPI_Wtime();

    int a = rand() % 10000;
    int n = ProcRank < 2 ? ProcRank : (rand() % 2);
    printf("Process %d (n = %d): a = %d\n", ProcRank, n, a);

    MPI_Comm low_comm;
    MPI_Comm_split(MPI_COMM_WORLD, n, ProcRank, &low_comm);
```

```

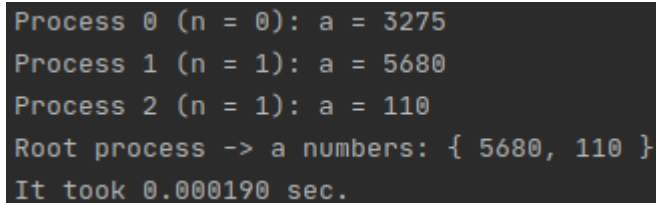
if (n == 1) {
    int NProcNum, NProcRank;
    MPI_Comm_size(low_comm, &NProcNum);
    MPI_Comm_rank(low_comm, &NProcRank);

    int *buffer = malloc(NProcNum * sizeof(int));
    MPI_Gather(&a, 1, MPI_INT, buffer, 1, MPI_INT, 0, low_comm);
    if (NProcRank == 0) {
        printf("Root process -> a numbers: {");
        for (int i = 0; i < NProcNum; ++i) {
            if (i != NProcNum - 1) printf(" %d,", buffer[i]);
            else printf(" %d", buffer[i]);
        }
        printf(" }\nIt took %lf sec.\n", MPI_Wtime() - start_time);
    }

    free(buffer);
}
MPI_Finalize();
return 0;
}

```

### Результаты работы.

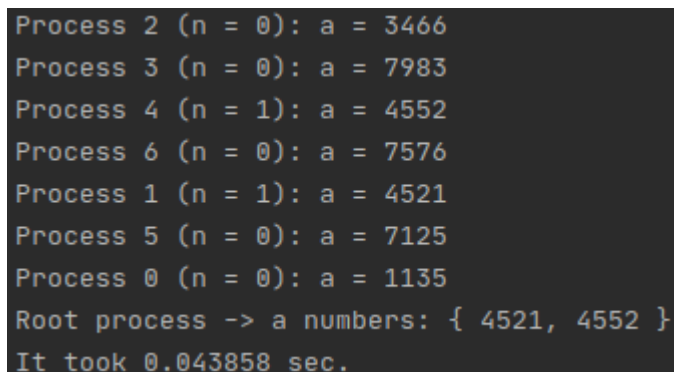


```

Process 0 (n = 0): a = 3275
Process 1 (n = 1): a = 5680
Process 2 (n = 1): a = 110
Root process -> a numbers: { 5680, 110 }
It took 0.000190 sec.

```

Рисунок 1 — Результат работы программы для 3 процессов



```

Process 2 (n = 0): a = 3466
Process 3 (n = 0): a = 7983
Process 4 (n = 1): a = 4552
Process 6 (n = 0): a = 7576
Process 1 (n = 1): a = 4521
Process 5 (n = 0): a = 7125
Process 0 (n = 0): a = 1135
Root process -> a numbers: { 4521, 4552 }
It took 0.043858 sec.

```

Рисунок 2 — Результат работы программы для 7 процессов

```

Process 4 (n = 1): a = 4058
Process 6 (n = 1): a = 5768
Process 14 (n = 0): a = 5327
Process 16 (n = 1): a = 6330
Process 10 (n = 0): a = 2002
Process 0 (n = 0): a = 5743
Process 8 (n = 1): a = 2568
Process 3 (n = 1): a = 5843
Process 5 (n = 0): a = 1105
Process 18 (n = 1): a = 783
Process 7 (n = 1): a = 137
Process 9 (n = 0): a = 3665
Process 15 (n = 0): a = 6035
Process 12 (n = 0): a = 7571
Process 1 (n = 1): a = 3598
Process 2 (n = 1): a = 228
Process 17 (n = 0): a = 2645
Process 11 (n = 1): a = 9621
Process 13 (n = 0): a = 2374
Root process -> a numbers: { 3598, 228, 5843, 4058, 5768, 137, 2568, 9621, 6330, 783 }
It took 0.193028 sec.

```

Рисунок 3 — Результат работы программы для 19 процессов

```

Process 16 (n = 1): a = 9747
Process 12 (n = 1): a = 9582
Process 34 (n = 0): a = 4288
Process 41 (n = 0): a = 3044
Process 37 (n = 0): a = 5291
Root process -> a numbers: { 8832, 2397, 1682, 6703, 9490, 9582, 8755, 9747, 9523, 6656, 2870, 8237, 7783, 6550, 5409, 3870, 1954, 6857, 5028, 2194, 243, 5861, 2673, 1641 }
It took 0.897714 sec.

```

Рисунок 4 — Результат работы программы для 57 процессов (некоторые строки опущены)

```

Process 34 (n = 1): a = 2281
Process 68 (n = 1): a = 7228
Process 69 (n = 0): a = 495
Process 48 (n = 0): a = 2060
Process 28 (n = 1): a = 4109
Root process -> a numbers: { 3044, 6857, 5028, 2194, 243, 5861, 2673, 1641, 9797, 5867, 4219, 4642, 5902, 8169, 4488, 4109, 4109, 5094, 2281, 2281, 9954, 5534, 5784, 1497, 5599, 3803, 6709, 3394, 6714, 7184, 8004, 2568, 264, 9685, 7065, 7800, 7228, 8633, 8633, 9231, 8290, 4367, 6804, 9013, 6542, 2261, 4758, 6067 }
It took 3.445792 sec.

```

Рисунок 5 — Результат работы программы для 101 процесса (некоторые строки опущены)

**График зависимости времени выполнения программы от числа процессов.**

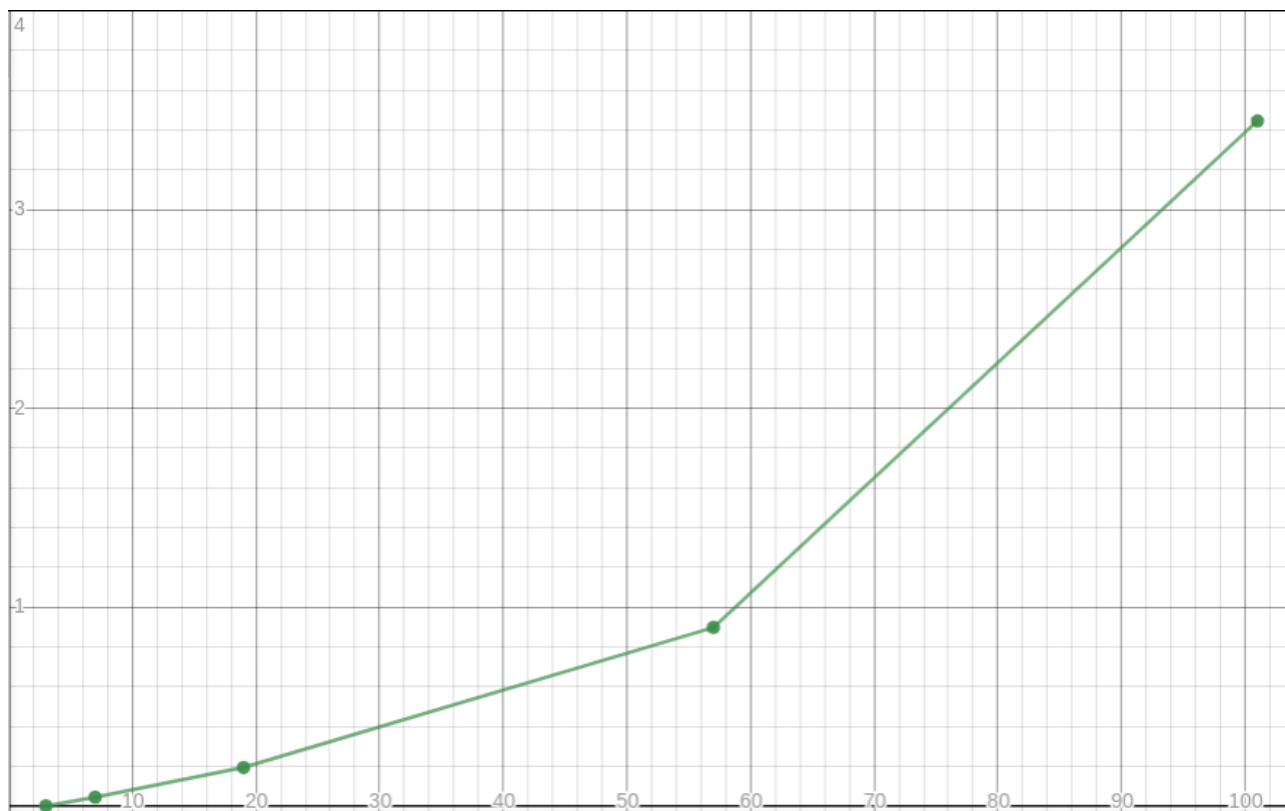


Рисунок 6 — График зависимости времени выполнения программы от числа процессов

### **Выводы.**

Создана программа для поиска минимума среди элементов разных процессов с одним индексом. Изучены принципы работы функции `MPI_Comm_Split`, а также механизм разделения коммутаторов.