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

ОТЧЕТ

по лабораторной работе №2

по дисциплине «Системы параллельной обработки данных»

Тема: ИСПОЛЬЗОВАНИЕ АРГУМЕНТОВ-ДЖОКЕРОВ

Студентка гр. 5303 _____ Допира В.Е.

Преподаватель _____ Татаринов Ю.С.

Санкт-Петербург

2019

Формулировка задания

Вариант 4. Раздача и сборка массива. Процесс 0 генерирует целочисленный массив и раздает его по частям в остальные процессы; порядок раздачи определяется случайным образом, размер каждого следующего передаваемого фрагмента в 2 раза меньше предыдущего. Процессы-получатели выполняют обработку массива и возвращают результат в процесс 0. Процесс 0 должен собрать массив результатов обработки с сохранением последовательности элементов.

Описание алгоритма с использованием аппарата Сетей Петри

P_0 - начальное состояние. Нулевой процесс разделяет массив на части и передает их из P_0 остальным процессам, тем самым осуществляются переходы t_1, t_2, t_3 к состояниям P_1, P_2, P_3 соответственно. Количество переходов соответствует количеству заданных процессов минус 1. Здесь показано на трех. Далее процессы-получатели делают обработку массива: вытаскивают первый элемент из той части массива, которая им пришла. Результат возвращается процессу 0, срабатывает переход t_4 к состоянию P_4. Нулевой процесс собирает массив из результатов обработки, выводит результат. Программа завершает выполнение.

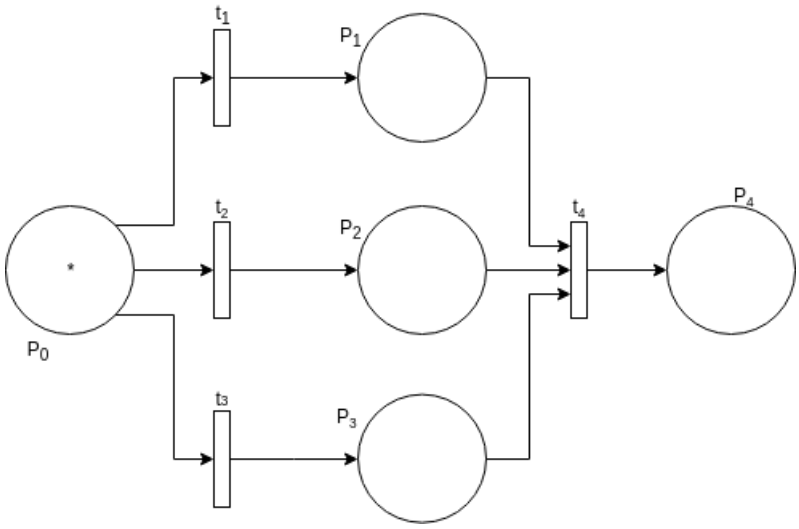

Рисунок 1 — Сеть Петри

Результаты работы программы на 1,2 …. N процессорах

```
$ mpirun -np 2 2.x
P0 sent 9 elements to P1.
Result:
3

$ mpirun -np 3 2.x
P0 sent 6 elements to P1.
P0 sent 3 elements to P2.
Result:
3
5

$ mpirun -np 4 2.x
P0 sent 6 elements to P1.
P0 sent 3 elements to P2.
P0 sent 1 elements to P3.
Result:
3
5
7
```

Вывод

В ходе выполнения лабораторной работы была написана программа с использованием MPI, моделирующая раздачу и сборку массива. В программе был использован джокер.

Листинг программы

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

int handling(int *array, int count)
{
   return array[0];
}

int main(int argc, char *argv[]) {
   int size;
   int rank;
   const int VERY_LARGE_INT = 999999;
   int tag = 1;

   int N = 10;
   int A[N];
   for(int i = 0; i < N; i++){
       A[i] = rand() % 10;
    }
```

```
   MPI_Init(&argc, &argv);
   MPI_Comm_size(MPI_COMM_WORLD, &size);
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);

   int m = 0, k = N - 1;
   int n;

   int *array = (int *)malloc(N * sizeof(int));
   int after_handling = 0;
   int result[size];

   if (rank == 0) {
      if (size == 2) {
         for(int dest = 1; dest < size; ++dest)
         {
            MPI_Send(&A[(dest-1)*k], k, MPI_INT, dest, tag,
MPI_COMM_WORLD);
            printf("P0 sent %d elements to P%d.\n", k, dest);
         }
      }
      else
         for(int dest = 1; dest < size; ++dest)
         {
            n = m + (k-1)/2;
            k = N - n;
            m = n + 1;
            MPI_Send(&A[(dest-1)*k], k, MPI_INT, dest, tag,
MPI_COMM_WORLD);
            printf("P0 sent %d elements to P%d.\n", k, dest);
         }
   }

/* Every other rank is receiving one message: from 0 into array */
   else {
      MPI_Recv(array, k, MPI_INT, MPI_ANY_SOURCE, tag,
MPI_COMM_WORLD, MPI_STATUS_IGNORE);
      after_handling = handling(array, m);
      // printf("Smth from P%d is %d.\n", rank, after_handling);
      MPI_Send(&after_handling, 1, MPI_INT, 0, tag,
MPI_COMM_WORLD);
   }

   MPI_Status Status;
   if(rank == 0)
   {
      int tmp;
      for(int i = 1; i < size; i++){
         MPI_Recv(&tmp, 1, MPI_INT, MPI_ANY_SOURCE, tag,
MPI_COMM_WORLD, &Status);
```

```
            int index = Status.MPI_SOURCE;
            result[index] = tmp;
        }
        printf("Result:\n");
        for(int i = 1; i < size; i++){
            printf("%d\n", result[i]);
        }
    }

    MPI_Finalize();
    return 0;
}
```