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

ОТЧЕТ

по лабораторной работе №2
по дисциплине «Параллельный алгоритмы»
Тема: Использование аргументов-джокеров.

Студент гр. 9383 _____ Ноздрин В.Я.

Преподаватель _____ Татаринов Ю.С.

Санкт-Петербург
2021

Задание. Вариант 1. Пляжный волейбол.

Процесс 0 генерирует сообщение и посылает его любому другому процессу группы. Процесс-приемник передает сообщение дальше. Выбор процесса-адресата осуществляется случайным образом.

Выполнение работы.

Была написана программа согласно заданию.

```
ice-jack@ice-pc ~/P/P/lab2 (master)> mpic++ main.cpp && mpirun -n 6 --oversubscribe a.out
0 ---9---> 2
2 ---9---> 1
1 ---9---> 5
5 ---9---> 5
^C⏎
ice-jack@ice-pc ~/P/P/lab2 (master) [1]> mpic++ main.cpp && mpirun -n 6 --oversubscribe a.out
0 ---9---> 0
^C^[[A⏎
ice-jack@ice-pc ~/P/P/lab2 (master) [1]> mpic++ main.cpp && mpirun -n 6 --oversubscribe a.out
0 ---9---> 4
4 ---9---> 5
5 ---9---> 0
^C⏎
ice-jack@ice-pc ~/P/P/lab2 (master) [1]>
```

Рисунок 1. Запуск программы на 6 процессах три раза.

```
ice-jack@ice-pc ~/P/P/lab2 (master)> mpic++ main.cpp && mpirun -n 15 --oversubscribe a.out
0 ---9---> 7
7 ---9---> 8
8 ---9---> 10
10 ---9---> 0
^C⏎
ice-jack@ice-pc ~/P/P/lab2 (master) [1]> mpic++ main.cpp && mpirun -n 15 --oversubscribe a.out
0 ---9---> 1
1 ---9---> 8
8 ---9---> 10
10 ---9---> 10
^C⏎
ice-jack@ice-pc ~/P/P/lab2 (master) [1]> mpic++ main.cpp && mpirun -n 15 --oversubscribe a.out
0 ---9---> 0
^C⏎
```

Рисунок 2. Запуск программы на 15 процессах три раза.

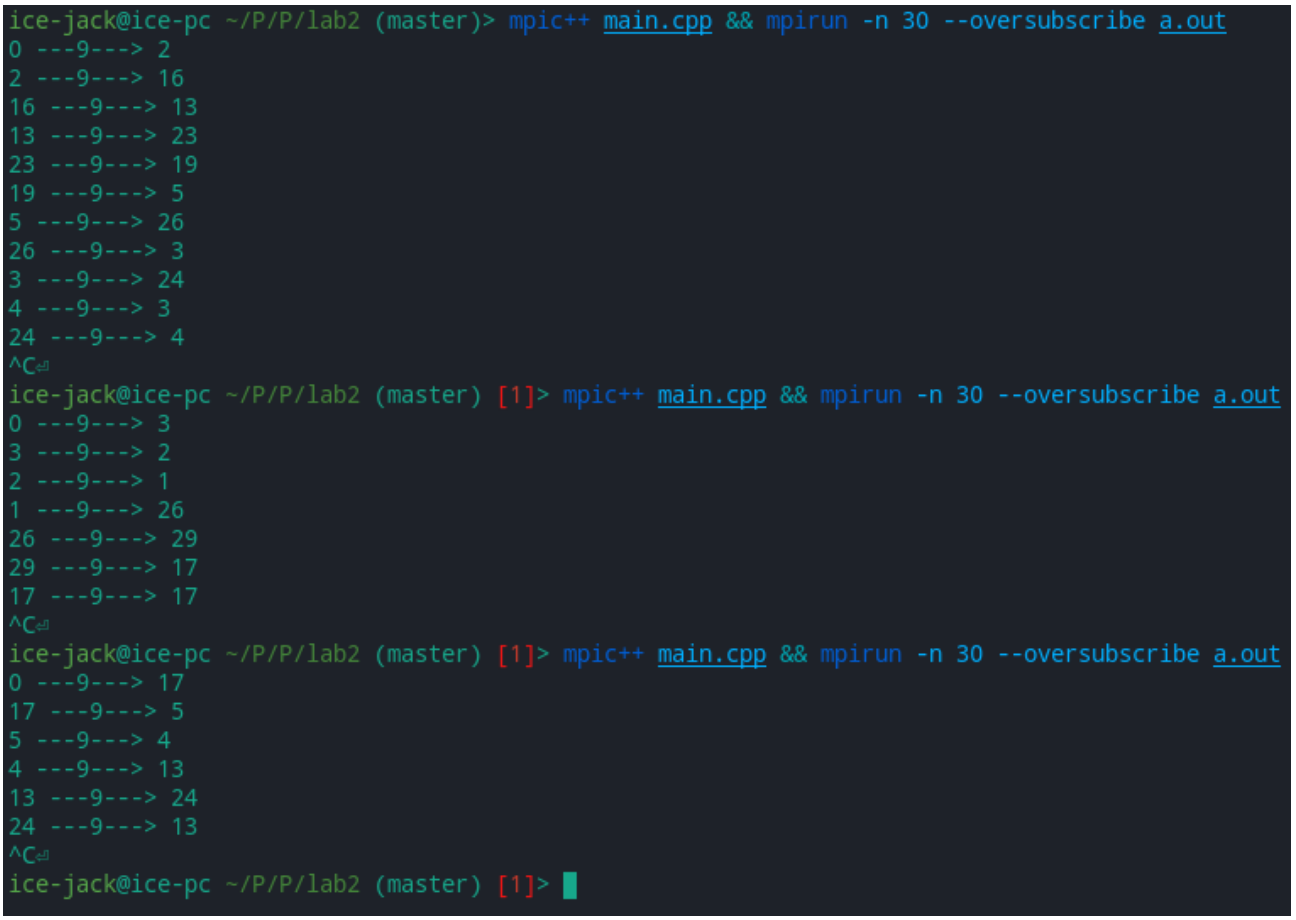

Рисунок 3. Запуск программы на 30 процессах три раза.

Можно заметить, что так как каждый процесс выполняет код один раз, как только сообщение пересылается любому процессу, который уже отправлял сообщение, программу уходит в дедлок и пользователь вынужден ее экстренно завершить. При больших количествах процессов цепочка сообщений выходит длиннее, но при должном “везении” программа может уже на первом сообщении зависнуть. Это видно на рисунке 2 при третьем запуске программы и на рисунке 1 при втором.

Для того, чтобы процессы могли получать сообщения от любых других процессов, в качестве отправителя в методе MPI_Recv подается тэг MPI_ANY_SOURCE.

Выводы.

Получен опыт работы с аргументами-джокерами, которые позволяют получать процессам сообщения от любых других процессов.

ПРИЛОЖЕНИЕ

Файл main.cpp

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

int main(int argc, char* argv[]){
  int size, rank, reciever;
  size_t msg_size = 1;
  int *sendbuf = new int[msg_size];
  int *recvbuf = new int[msg_size];
  MPI_Status Status;
  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  srand(time(NULL) + rank);
  if ( rank == 0 && rank+1 < size){
    reciever = rand()%size;
    *sendbuf = 9;
    std::cout << rank << " ---" << *sendbuf << "---> " << reciever
<< "\n";
    MPI_Send(sendbuf, msg_size, MPI_INT, reciever, 0,
MPI_COMM_WORLD);
    MPI_Recv(recvbuf, msg_size, MPI_INT, MPI_ANY_SOURCE,
MPI_ANY_TAG, MPI_COMM_WORLD, &Status);
  } else {
    MPI_Recv(recvbuf, msg_size, MPI_INT, MPI_ANY_SOURCE,
MPI_ANY_TAG, MPI_COMM_WORLD, &Status);
    reciever = rand()%size;
    std::cout << rank << " ---" << *recvbuf << "---> " << reciever
<< "\n";
```

```
    MPI_Send(recvbuf, msg_size, MPI_INT, reciever, 0,
MPI_COMM_WORLD);
  }
  MPI_Finalize();
  return 0;
}
```