

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

ОТЧЕТ
по лабораторной работе №2
по дисциплине «Параллельные алгоритмы»
Тема: Использование аргументов-джокеров

Студент гр. 8303

Почаев Н.А.

Преподаватель

Татаринов Ю.С.

Санкт-Петербург

2020

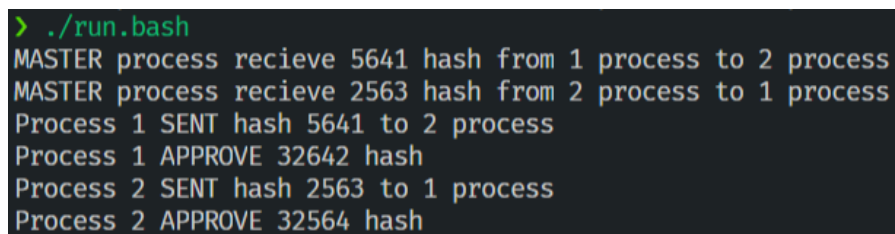
Задание.

Вариант 3 - Имитация топологии «звезда» (процесс с номером 0 реализует функцию центрального узла). Процессы в случайном порядке генерируют пакеты, состоящие из адресной и информационной части и передают их в процесс 0. Маршрутная часть пакета содержит номер процесса-адресата. Процесс 0 переадресовывает пакет адресату. Адресат отчитывается перед процессом 0 в получении. Процесс 0 информирует процесс-источник об успешной доставке.

Описание решения.

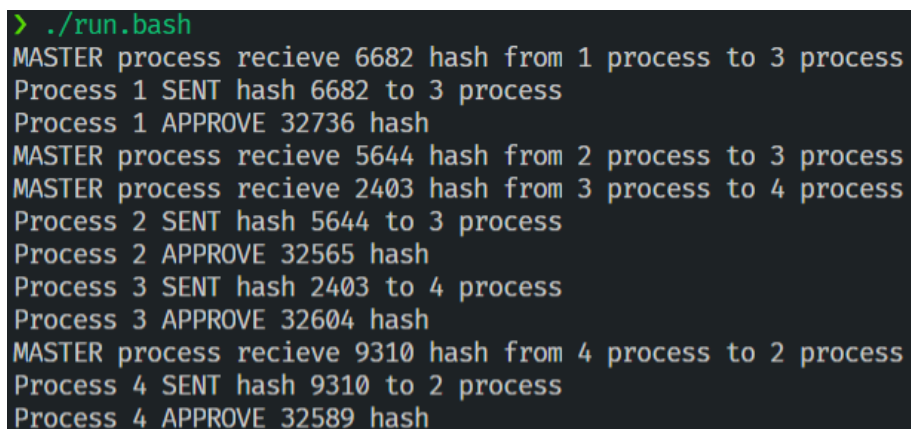
Программа построение на блокирующей посылке пакетов от рабочих процессов в нулевой (master) и неблокирующих ответах от него, при подтверждении удачного получения. В виде информационной нагрузки используется хэш-код, сгенерированный случайно для каждого сеанса общения между процессами. Номера процессов получателей также генерируются случайно. Структура передаваемых данных представлена в первом комментарии в исходном коде программы.

Результаты работы программы представлены на скриншотах 1-3 ниже:



```
> ./run.bash
MASTER process recieve 5641 hash from 1 process to 2 process
MASTER process recieve 2563 hash from 2 process to 1 process
Process 1 SENT hash 5641 to 2 process
Process 1 APPROVE 32642 hash
Process 2 SENT hash 2563 to 1 process
Process 2 APPROVE 32564 hash
```

Рисунок 1 – 3 процесса



```
> ./run.bash
MASTER process recieve 6682 hash from 1 process to 3 process
Process 1 SENT hash 6682 to 3 process
Process 1 APPROVE 32736 hash
MASTER process recieve 5644 hash from 2 process to 3 process
MASTER process recieve 2403 hash from 3 process to 4 process
Process 2 SENT hash 5644 to 3 process
Process 2 APPROVE 32565 hash
Process 3 SENT hash 2403 to 4 process
Process 3 APPROVE 32604 hash
MASTER process recieve 9310 hash from 4 process to 2 process
Process 4 SENT hash 9310 to 2 process
Process 4 APPROVE 32589 hash
```

Рисунок 2 – 5 процессов

```
> ./run.bash
Process 1 SENT hash 4644 to 2 process
Process 1 APPROVE 32644 hash
Process 3 SENT hash 6787 to 4 process
Process 3 APPROVE 32597 hash
Process 6 SENT hash 6507 to 2 process
Process 6 APPROVE 32678 hash
MASTER process recieve 4644 hash from 1 process to 2 process
Process 4 SENT hash 7348 to 2 process
Process 4 APPROVE 32572 hash
Process 2 SENT hash 3311 to 4 process
Process 2 APPROVE 32688 hash
Process 5 SENT hash 1984 to 3 process
Process 5 APPROVE 32557 hash
MASTER process recieve 3311 hash from 2 process to 4 process
MASTER process recieve 6787 hash from 3 process to 4 process
MASTER process recieve 7348 hash from 4 process to 2 process
MASTER process recieve 1984 hash from 5 process to 3 process
MASTER process recieve 6507 hash from 6 process to 2 process
```

Рисунок 3 – 7 процессов

Выводы.

В ходе выполнения лабораторной работы лабораторной работы была изучена работа с процессами и приёмы передачи между ними с использованием API MPI. Была изучена работа с функциями неблокирующей передачи данных, а также использование джokers для приёма сообщений от всех возможных процессов.

ПРИЛОЖЕНИЕ А.

ИСХОДНЫЙ КОД ПРОГРАММЫ.

```
/**
 * Array structure:
 * [0]:
 *     0 - initial message
 *     1 - delivery approve message
 * [1]: message sender
 * [2]: message reciever
 * [3]: random number - message
 */

#include "mpi.h"
#include <iostream>
#include <cstdio>
#include <vector>
#include <algorithm>
#include <cctype>
#include <random>

#define PROCESS_COUNT 13
#define MASTER 0

using u32 = uint_least32_t;
using engine = std::mt19937;

/**
 * Generate random number in the specified range (inclusive)
 */
int getRandNumb(int min, int max) {
    std::random_device os_seed;
    const u32 seed = os_seed();

    engine generator(seed);
    std::uniform_int_distribution<u32> distribute(min, max);

    return distribute(generator);
}

/**
 * Get number of process to sent
 * To avoid sending a request to yourself
 */
int getRandreciever(int currentProcess) {
    int res = getRandNumb(1, PROCESS_COUNT - 1);
    while (res == currentProcess) {
        res = getRandNumb(1, PROCESS_COUNT - 1);
    }

    return res;
}
```

```

}

int main(int argc, char *argv[]) {
    int ProcNum;
    int ProcRank;
    int RecvRank;
    MPI_Status Status;
    MPI_Request request;

    // init parallel block
    MPI_Init(&argc, &argv);

    // declare size of processes (group id, group size(return))
    MPI_Comm_size(MPI_COMM_WORLD, &ProcNum);

    // define process rank in group (group id, rank(return))
    MPI_Comm_rank(MPI_COMM_WORLD, &ProcRank);

    if(ProcRank == MASTER) {
        int msgGet[4];
        int msgSent[4];

        for (int i = 1; i < PROCESS_COUNT; ++i) {
            MPI_Recv(msgGet, 4, MPI_INT, i, 0, MPI_COMM_WORLD,
&Status);

            msgSent[1] = msgGet[1];
            msgSent[2] = msgGet[2];
            msgSent[3] = msgGet[3];
            if (msgGet[0] == 0) {
                // receive initial message
                msgSent[0] = 1;
                std::cout << std::flush << "MASTER process receive
" << msgGet[3] << " hash from " << msgGet[1] << " process to " <<
msgGet[2] << " process" << std::endl;
                MPI_Send(msgSent, 4, MPI_INT, msgGet[2], 0,
MPI_COMM_WORLD);
            } else if (msgGet[0] == 1) {
                // send approve message
                msgSent[0] = 0;
                std::cout << std::flush << "MASTER sent approve to
the " << msgSent[1] << " process" << std::endl;
                MPI_Isend(msgSent, 4, MPI_INT, msgGet[1], 0,
MPI_COMM_WORLD, &request);
            }
        }
    } else {
        int msgSent[4];
        int receiverRank = getRandreceiver(ProcRank);
        int hash = getRandNumb(1000, 9999);
        msgSent[0] = 0;
        msgSent[1] = ProcRank;
        msgSent[2] = receiverRank;
    }
}

```

```

        msgSent[3] = hash;
        MPI_Send(msgSent, 4, MPI_INT, 0, 0, MPI_COMM_WORLD);
        std::cout << std::flush << "Process " << ProcRank << " SENT
hash " << hash << " to " << recieverRank << " process" << std::endl;

        int msgApproveGet[4];
        MPI_Irecv(msgApproveGet, 4, MPI_INT, MPI_ANY_SOURCE,
MPI_ANY_TAG, MPI_COMM_WORLD, &request);
        if (msgApproveGet[0] == 1) {
            // get initial message and send approve
            std::cout << std::flush << "Process " << ProcRank << "
GET hash " << msgApproveGet[3] << " from " << msgApproveGet[1] << "
process" << std::endl;
            MPI_Send(msgSent, 4, MPI_INT, 0, 0, MPI_COMM_WORLD);
        } else if (msgApproveGet[0] == 0) {
            std::cout << std::flush << "Process " << ProcRank << "
APPROVE " << msgApproveGet[3] << " hash" << std::endl;
        }
    }

    // finish parallel block
    MPI_Finalize();

    return 0;
}

```