**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»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 9303 |  | Королёв С.Ю. |
| Преподаватель |  | Татаринов Ю.С. |

Санкт-Петербург

2023

**Цель работы**

Базовое освоение и изучение аргументов-джокеров в библиотеке MPI.

**Формулировка задания**

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.

**Ход работы**

Для изучения аргументов-джокеров в библиотеке MPI в процессе работы был создан скрипт на языке программирования C++ для запуска и обработки результатов запусков процессов.

**Листинг итоговой программы:**

#include <iostream>

#include <vector>

#include <random>

#include "mpi.h"

using namespace std;

using u32 = uint\_least32\_t;

using engine = std::mt19937;

int getRandNumb(int min, int max) {

random\_device os\_seed;

const u32 seed = os\_seed();

engine generator(seed);

uniform\_int\_distribution<u32> distribute(min, max);

return distribute(generator);

}

int main(int argc, char\*\* argv) {

MPI\_Init(&argc, &argv);

int rank, size;

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

MPI\_Status status;

if (size < 2) {

cout << "This program requires at least 2 processes to run." << std::endl;

MPI\_Finalize();

return 1;

}

vector<int> packets\_received;

double start = MPI\_Wtime();

int data = getRandNumb(100, 999);

if (rank == 0) {

while (packets\_received.size() < size - 1) {

int packet[1];

// Принимаем пакеты от других процессов

MPI\_Recv(packet, 1, MPI\_INT, MPI\_ANY\_SOURCE, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &status);

// Переадресуем пакет адресату

MPI\_Send(packet, 1, MPI\_INT, status.MPI\_SOURCE, 0, MPI\_COMM\_WORLD);

cout << "MASTER " << rank << " redirected packet from process " << status.MPI\_SOURCE << " to receiver." << endl;

// Информируем процесс-источник об успешной доставке

MPI\_Send(nullptr, 0, MPI\_INT, status.MPI\_SOURCE, 1, MPI\_COMM\_WORLD);

packets\_received.push\_back(status.MPI\_SOURCE);

}

}

else {

// Процессы-источники генерируют и отправляют пакеты процессу 0

srand(rank); // Для генерации случайных пакетов на разных процессах

int packet[1];

packet[0] = data;

// Генерируем случайный адресат пакета (от 1 до size-1)

int recipient = 1 + rand() % (size - 1);

// Отправляем пакет процессу 0

MPI\_Send(packet, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD);

cout << "Process " << rank << " sent packet to MASTER 0, receiver is: " << recipient << ". Sent data: " << packet[0] << endl;

// Получаем подтверждение доставки от процесса 0

MPI\_Recv(nullptr, 0, MPI\_INT, 0, 1, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

cout << "Process " << rank << " received delivery approval." << " Received data: " << packet[0] << endl;

}

double deltaTime = MPI\_Wtime() - start;

if (rank == 0) {

cout << "Time delta: " << deltaTime;

}

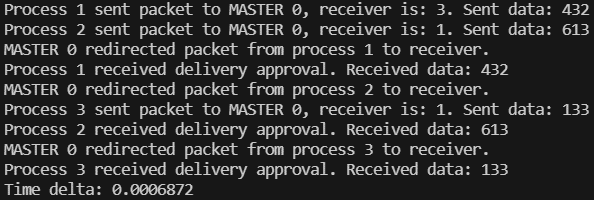
MPI\_Finalize();

return 0;

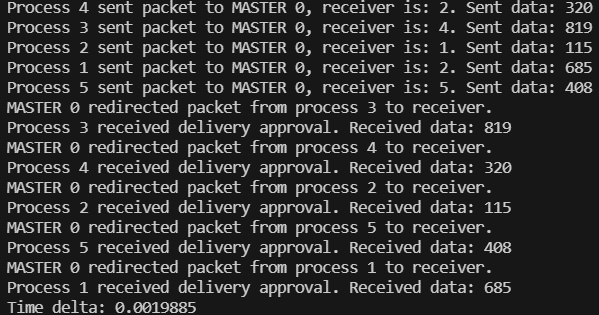
}

**Результаты работы исходной программы на 4, 6, 8, 10 процессах:**

На 4 процессах:



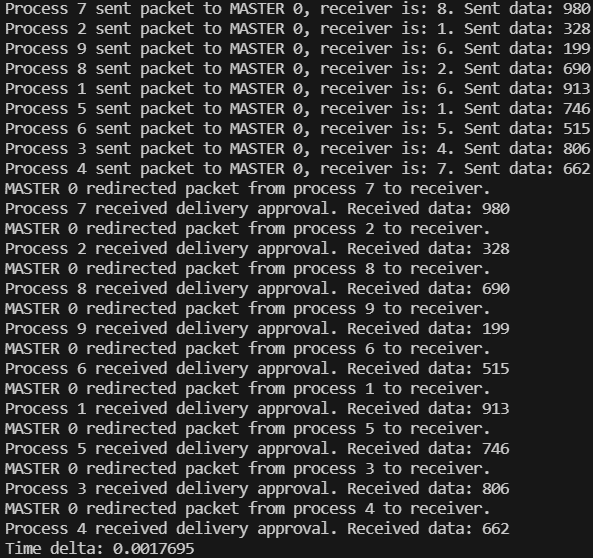
На 6 процессах:



На 8 процессах:



На 10 процессах:

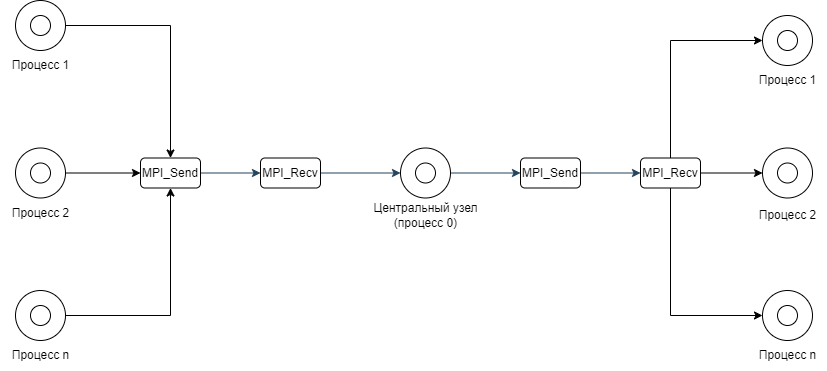


Стоит заметить, что не имеет смысла имитировать данную топологию в том случае, если количество процессов менее 2 (то есть существует только процесс с рангом 0). Это связано с тем, что процесс с рангом 0 не сможет принимать и переадресовывать пакеты другим процессам, потому что других процессов не будет существовать.

**Краткое описание выбранного алгоритма решения**

Процесс с рангом 0 реализует функцию центрального узла в рамках имитации топологии «звезда», который получает сообщения от процессов-отправителей и переадресовывает их процессам-адресатам. Остальные процессы генерируют пакеты со случайными номерами адресатов и отправляют их процессу с рангом 0. Информационной частью сообщения является случайно сгенерированное число от 100 до 999. Процесс-отправитель получает подтверждение доставки пакета от процесса 0.

**Формальное описание выбранного алгоритма решения с использованием аппарата Сетей Петри**



**График зависимости выполнения программы от числа процессов для разного объёма исходных данных**

Таблица 1. Зависимость времени выполнения программы от количества процессов.

|  |  |
| --- | --- |
| Число процессов | Время выполнения для итоговой программы (мс.) |
| 4 | 0,6872 |
| 6 | 1,1877 |
| 8 | 1,9352 |
| 10 | 2,8176 |

**Вывод**

В ходе выполнения работы были получены навыки использования аргументов-джокеров в библиотеке MPI.

Был построен график зависимости времени выполнения от количества процессов. По графику можно сделать вывод, что время выполнения для итоговой программы прямо пропорционально числу процессов. Это объясняется тем, что с увеличением числа процессов увеличивается и количество операций передачи пакетов от процесса-отправителя к центральному узлу (т.е. процессу с рангом 0), а также передачи пакетов от центрального узла к процессу-получателю. Из-за этого в совокупности затрачивается больше времени.