**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МОЭВМ**

отчет

**по лабораторной работе №2**

**по дисциплине «Параллельные алгоритмы»**

ТЕМА: **ИСПОЛЬЗОВАНИЕ АРГУМЕНТОВ-ДЖОКЕРОВ.**

|  |  |  |
| --- | --- | --- |
| Студент |  | Степаненко Д. В. |
| Преподаватель |  | Татаринов Ю. С. |

Санкт-Петербург

2023 г.

## Цель

Отработать на практике применение аргументов-джокеров в функциях библиотеки MPI.

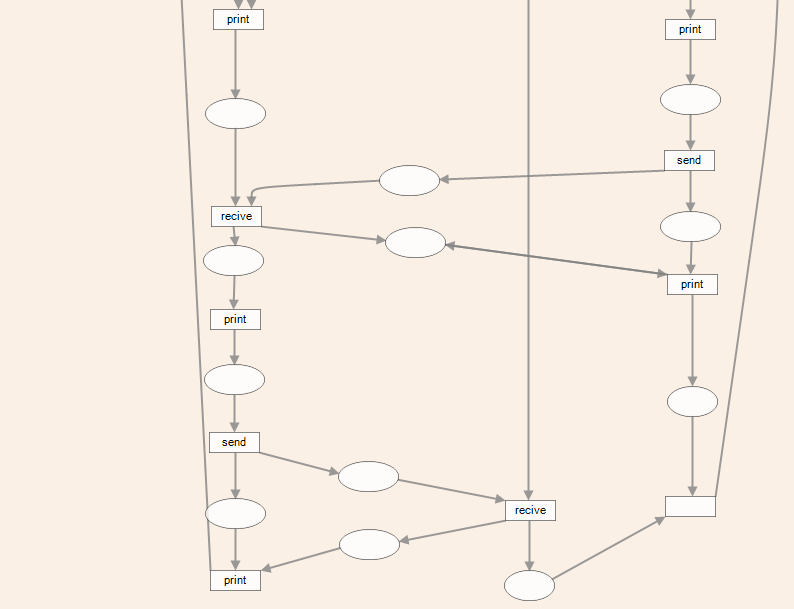
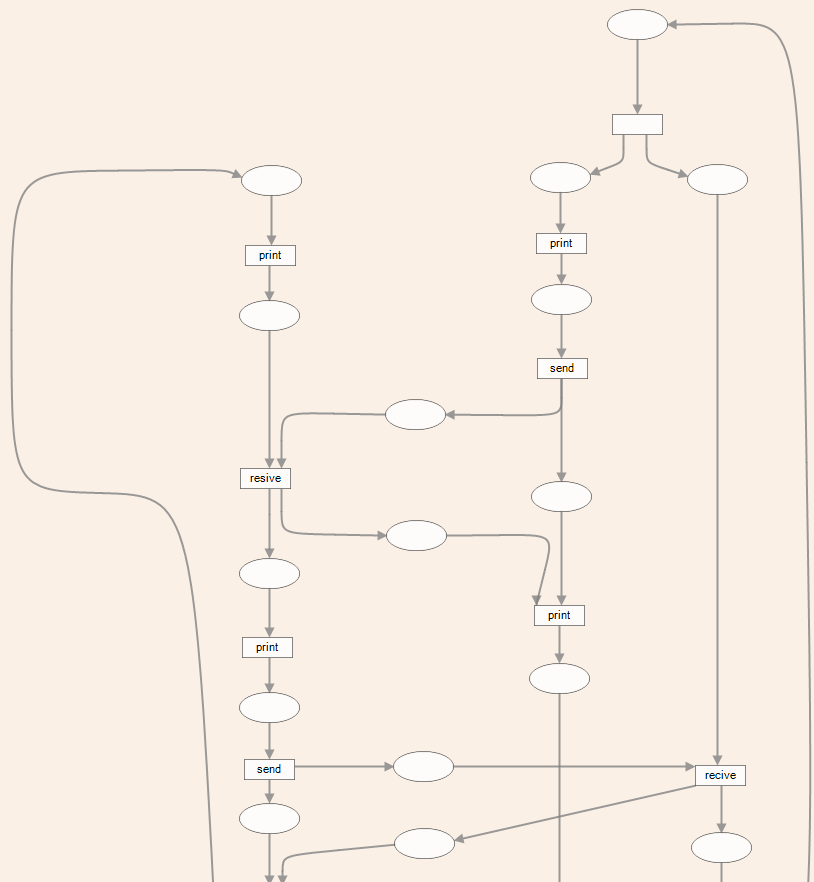
## Постановка задачи (вариант 3)

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.

## Выполнение работы

Программа создает несколько процессов, считывает ранг каждого и общее количество процессов. Если процесс не 0 ранга, то он посылает сообщение (адрес назначения и данные) процессу с рангом 0. Ранг 0 получает несколько таких сообщений и поочередно их обрабатывает: пересылает сообщение на указанный в пакете адрес, (после получения сообщения адресатом) отправляет сообщение процессу адресанту об успешном завершении операции.

Сеть Петри основной части алгоритма для 3ех процессов:



Листинг программы:

#include <stdio.h>

#include <stdlib.h>

#include <mpi.h>

#define PACKET\_SIZE 2

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

int rank, size;

MPI\_Init(&argc, &argv);

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

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

if (size < 2) {

printf("This program should be run with at least 2 processes\n");

MPI\_Finalize();

return 0;

}

if (rank == 0) {

// Процесс 0 реализует функцию центрального узла

int total\_packets = size - 1; // Количество пакетов для отправки

int i, source, dest, packet[PACKET\_SIZE];

MPI\_Status status;

printf("Process 0 is the central node\n");

for (i = 0; i < total\_packets; i++) {

// Принимаем пакет от других процессов

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

source = status.MPI\_SOURCE;

dest = packet[0];

printf("Process %d received packet from process %d with destination %d\n", rank, source, dest);

if (dest >= 1 && dest < size) {

// Пересылаем пакет адресату

MPI\_Send(&packet, PACKET\_SIZE, MPI\_INT, dest, 0, MPI\_COMM\_WORLD);

printf("Process %d sent packet to process %d\n", rank, dest);

// Получаем подтверждение об отправке

MPI\_Recv(&packet, PACKET\_SIZE, MPI\_INT, dest, 2, MPI\_COMM\_WORLD, &status);

printf("Process 0 received confirmation from process %d\n", dest);

// Посылаем ответ отправителю о доставке

MPI\_Send(&packet, PACKET\_SIZE, MPI\_INT, source, 1, MPI\_COMM\_WORLD);

}

}

} else {

// Процессы 1, 2, ..., size-1 генерируют пакеты и отправляют их процессу 0

int packet[PACKET\_SIZE];

packet[0] = (rank+1)%size; // Адресат

if(packet[0]==0){

packet[0] = 1;

}

packet[1] = rand()/rank; // Информационная часть пакета

printf("Process %d generated packet with destination: %d, information: %d\n", rank, packet[0], packet[1]);

// Отправляем пакет процессу 0

MPI\_Send(&packet, PACKET\_SIZE, MPI\_INT, 0, 0, MPI\_COMM\_WORLD);

printf("Process %d sent packet to process 0\n", rank);

//получаем пакет с информацией от цетрального процесса;

if(MPI\_Recv(&packet, PACKET\_SIZE, MPI\_INT, 0, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE) == MPI\_SUCCESS){

printf("Process %d received information from process 0: %d\n", rank, packet[1]);

//отправляем подтверждение о получении сообщения от цетрального процесса;

MPI\_Send(&packet, PACKET\_SIZE, MPI\_INT, 0, 2, MPI\_COMM\_WORLD);

}

// Получаем подтверждение от процесса 0 о доставке пакета процессу-адресату

MPI\_Recv(&packet, PACKET\_SIZE, MPI\_INT, 0, 1, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

printf("Process %d received delivery confirmation from process 0\n", rank);

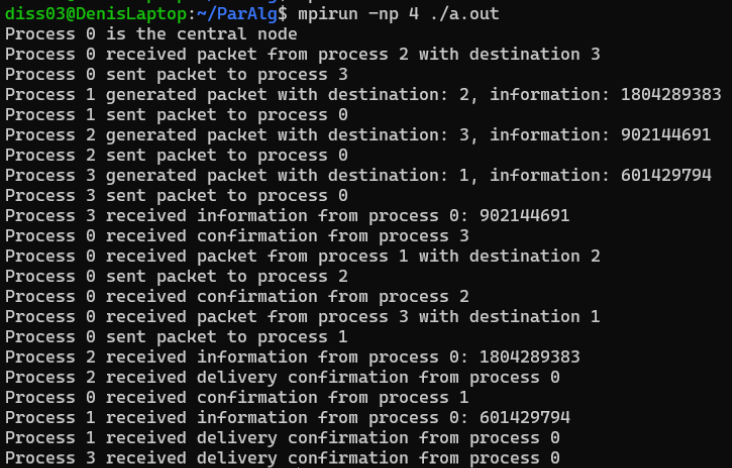
}

MPI\_Finalize();

return 0;

}

Полученный вывод при запуске на 4-ех процессах:



|  |  |
| --- | --- |
| Количество процессов (шт) | Среднее затрачиваемое время (мс) |
| 2 | 0,0722 |
| 4 | 0,0839 |
| 8 | 0,2290 |
| 12 | 0,5993 |
| 13 | 19,9718 |
| 14 | 20,2001 |
| 15 | 59,8388 |
| 16 | 60,0912 |
| 24 | 79,9966 |
| 32 | 89,6237 |

Табл. 1 – Результаты работы программы на разном количестве процессов.

Расчеты ускорения программы выполним по формуле:

|  |  |  |
| --- | --- | --- |
| Количество процессов P (шт) | Среднее затрачиваемое время (мс) | Ускорение |
| 2 | 0.0403 | 1 |
| 4 | 0.0560 | 0,860548 |
| 8 | 0.0670 | 0,315284 |
| 12 | 0.2404 | 0,120474 |
| 13 | 0.4966 | 0,003615 |
| 14 | 1.0935 | 0,003574 |
| 15 | 19.9861 | 0,001207 |
| 16 | 30.4054 | 0,001202 |
| 24 | 38.7814 | 0,000903 |
| 32 | 49.8196 | 0,000806 |

Табл. 2 – Результаты расчетов ускорения программы.

## Вывод

В ходе выполнения лабораторной работы были изучены и использованы аргументы-джокеры: MPI\_ANY\_SOURCE для номера задачи-отправителя и MPI\_ANY\_TAG для идентификатора получаемого сообщения. Их достоинство заключается в том, что приходящие сообщения извлекаются по мере поступления, а не по мере вызова MPI\_Recv с нужными идентификаторами задач/сообщений. Это экономит память и увеличивает скорость работы.

Был написан код, реализующий работу одной из трех топологий взаимодействия процессов – star, в которой основной нулевой процесс взаимодействует с остальными подчиненными.

Скорость работы программы напрямую зависит от количества запущенных процессов, т.к. каждый из них является участником топологии. Тем самым он отправляет и получает 4 сообщения. Получается, что с увеличением числа процессов растет время выполнения программного кода из-за повышения объемов пересылаемых сообщений. Таким образом, скорость выполнения программы уменьшается с увеличением числа процессов.