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

ОТЧЕТ

по лабораторной работе №4
по дисциплине «Параллельные алгоритмы»
Тема: Использование аргументов-джокеров

Студент гр. 9381 _____ Колованов Р.А.

Преподаватель _____ Татаринов Ю.С.

Санкт-Петербург
2021

Цель работы.

Написание программы, использующую функции обмена библиотеки MPI с применением аргументов-джокеров.

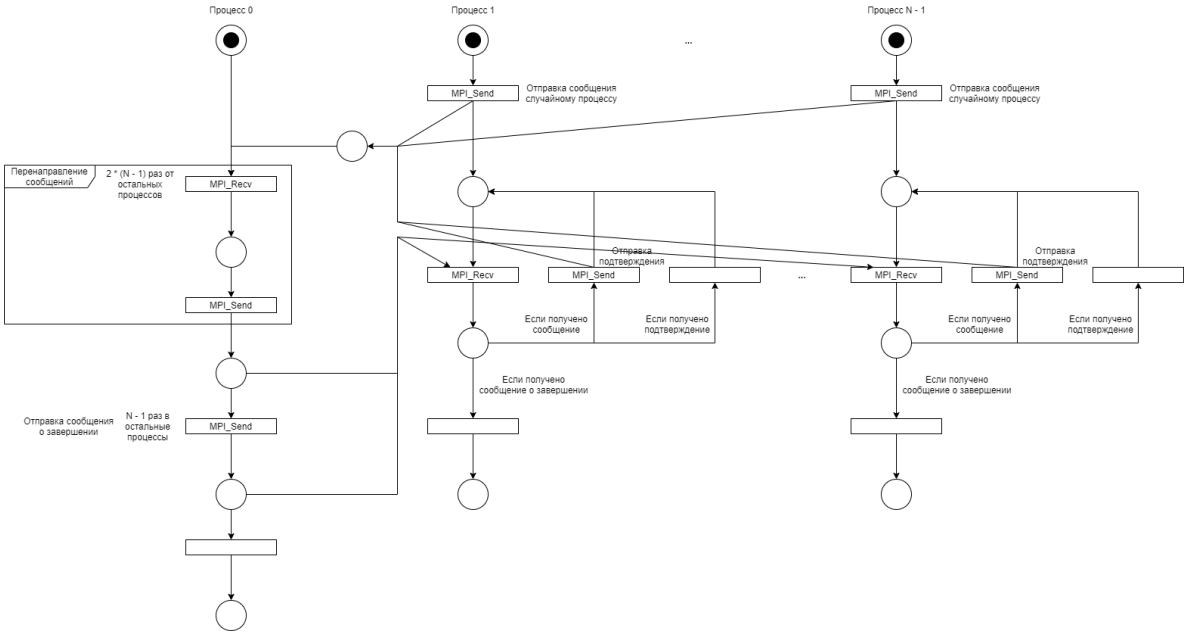
Формулировка задания.

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.

Краткое описание алгоритма.

В самом начале процесс 0 начинает принимать сообщения от других процессов, а другие процессы начинают отправлять сообщения, содержащие маршрутную информацию и данные, процессу 0 (сами пакеты хранят ранг некоторого процесса в качестве места назначения), после чего входят в режим ожидания сообщений от процесса 0. Процесс 0, принимая сообщения от разных процессов, перенаправляет их процессу, указанному в маршрутной информации пакета. Аналогичным способом осуществляется отправка сообщения с подтверждением получения. Когда все сообщения будут отправлены, процесс 0 отправит всем процессом сообщение с указанием завершения работы и заканчивает свою работу. Остальные процессы, получившие сообщения с указанием завершения работы от процесса 0, тоже завершают работу.

Формальное описание алгоритма.



Листинг программы.

Листинг 1. Код программы.

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


enum MessageType {
	Data,
	Confirmation,
	Finish
};

struct Message {
	MessageType type;
	int source;
	int destination;
	char data[64];
};

std::ostream& operator<<(std::ostream& os, const Message& message) {
	os << "{data:" << message.data << ", to:" << message.destination << ",
from: " << message.source << ", type: " << message.type << "}";
	return os;
}


int main(int argc, char** argv) {
	int processNumber, processRank;
	MPI_Status status;

	MPI_Init(&argc, &argv);
	MPI_Comm_size(MPI_COMM_WORLD, &processNumber);
	MPI_Comm_rank(MPI_COMM_WORLD, &processRank);

	srand(time(nullptr) + static_cast<time_t>(processRank) * 1000);
```

```
	if (processNumber < 2) {
		std::cerr << "At least two processes are required to work.\n";
		MPI_Finalize();
		return 0;
	}

	if (processRank == 0) {
		Message message;

		// Перенаправляем пакеты
		for (int i = 1; i <=  (processNumber - 1) * 2; ++i) {
			MPI_Recv(&message, sizeof(Message), MPI_BYTE, MPI_ANY_SOURCE,
MPI_ANY_TAG, MPI_COMM_WORLD, &status);
			std::cout << "[0] Received a message from the process [" <<
message.source << "]: " << message << "\n";
			MPI_Send(&message,	sizeof(Message),	MPI_BYTE,
message.destination, message.type, MPI_COMM_WORLD);
		}

		message = { MessageType::Finish, 0, 0, "Finish" };

		// Отправляем пакеты с запросом на завершение
		for (int i = 1; i < processNumber; ++i) {
			message.destination = i;
			MPI_Send(&message,	sizeof(Message),	MPI_BYTE,
message.destination, message.type, MPI_COMM_WORLD);
		}
	}
	else {
		int destinationProcess = 1 + rand() % (processNumber - 1);
		Message	message	=	{MessageType::Data,	processRank,
destinationProcess, "Hello!"};

		// Отправляем пакет с данными случайному процессу
		MPI_Send(&message, sizeof(Message), MPI_BYTE, 0, MessageType::Data,
MPI_COMM_WORLD);

		// Принимаем пакеты от 0 процесса и обрабатываем их
		do {
			MPI_Recv(&message, sizeof(Message), MPI_BYTE, 0, MPI_ANY_TAG,
MPI_COMM_WORLD, &status);
			std::cout << "[" << processRank << "] Received a message from
the process [0]: " << message << "\n";

			if (message.type == MessageType::Data) {
				message	=	{	MessageType::Confirmation,	processRank,
message.source, "Confirmation" };
				MPI_Send(&message,	sizeof(Message),	MPI_BYTE,	0,
MessageType::Confirmation, MPI_COMM_WORLD);
			}

		} while (message.type != MessageType::Finish);
	}

	MPI_Finalize();
	return 0;
}
```

Результаты работы программы на различном количестве процессов.

№ п/п	Количество процессоров	Результаты работы программы
1.	1	At least two processes are required to work.
2.	2	[0] Received a message from the process [1]: {data:Hello!, to:1, from: 1, type: 0} [1] Received a message from the process [0]: {data:Hello!, to:1, from: 1, type: 0} [0] Received a message from the process [1]: {data:Confirmation, to:1, from: 1, type: 1} [1] Received a message from the process [0]: {data:Confirmation, to:1, from: 1, type: 1} [1] Received a message from the process [0]: {data:Finish, to:1, from: 0, type: 2}
3.	4	[0] Received a message from the process [2]: {data:Hello!, to:1, from: 2, type: 0} [0] Received a message from the process [3]: {data:Hello!, to:2, from: 3, type: 0} [0] Received a message from the process [1]: {data:Hello!, to:1, from: 1, type: 0} [1] Received a message from the process [0]: {data:Hello!, to:1, from: 2, type: 0} [1] Received a message from the process [0]: {data:Hello!, to:1, from: 1, type: 0} [2] Received a message from the process [0]: {data:Hello!, to:2, from: 3, type: 0} [0] Received a message from the process [1]: {data:Confirmation, to:2, from: 1, type: 1} [0] Received a message from the process [1]: {data:Confirmation, to:1, from: 1, type: 1} [0] Received a message from the process [2]: {data:Confirmation, to:3, from: 2, type: 1} [3] Received a message from the process [0]: {data:Confirmation, to:3, from: 2, type: 1} [1] Received a message from the process [0]: {data:Confirmation, to:1, from: 1, type: 1} [2] Received a message from the process [0]: {data:Confirmation, to:2, from: 1, type: 1} [2] Received a message from the process [0]: {data:Finish, to:2, from: 0, type: 2} [3] Received a message from the process [0]: {data:Finish, to:3, from: 0, type: 2} [1] Received a message from the process [0]: {data:Finish, to:1, from: 0, type: 2}

5.	8	[0] Received a message from the process [4]: {data:Hello!, to:2, from: 4, type: 0} [0] Received a message from the process [5]: {data:Hello!, to:5, from: 5, type: 0} [0] Received a message from the process [7]: {data:Hello!, to:5, from: 7, type: 0} [0] Received a message from the process [6]: {data:Hello!, to:2, from: 6, type: 0} [0] Received a message from the process [3]: {data:Hello!, to:5, from: 3, type: 0} [0] Received a message from the process [1]: {data:Hello!, to:6, from: 1, type: 0} [0] Received a message from the process [2]: {data:Hello!, to:1, from: 2, type: 0} [6] Received a message from the process [0]: {data:Hello!, to:6, from: 1, type: 0} [2] Received a message from the process [0]: {data:Hello!, to:2, from: 4, type: 0} [2] Received a message from the process [0]: {data:Hello!, to:2, from: 6, type: 0} [5] Received a message from the process [0]: {data:Hello!, to:5, from: 5, type: 0} [5] Received a message from the process [0]: {data:Hello!, to:5, from: 7, type: 0} [1] Received a message from the process [0]: {data:Hello!, to:1, from: 2, type: 0} [5] Received a message from the process [0]: {data:Hello!, to:5, from: 3, type: 0} [0] Received a message from the process [5]: {data:Confirmation, to:5, from: 5, type: 1} [0] Received a message from the process [5]: {data:Confirmation, to:7, from: 5, type: 1} [0] Received a message from the process [5]: {data:Confirmation, to:3, from: 5, type: 1} [0] Received a message from the process [2]: {data:Confirmation, to:4, from: 2, type: 1} [0] Received a message from the process [2]: {data:Confirmation, to:6, from: 2, type: 1} [0] Received a message from the process [6]: {data:Confirmation, to:1, from: 6, type: 1} [0] Received a message from the process [1]: {data:Confirmation, to:2, from: 1, type: 1} [2] Received a message from the process [0]: {data:Confirmation, to:2, from: 1, type: 1}

		[5] Received a message from the process [0]: {data:Confirmation, to:5, from: 5, type: 1} [1] Received a message from the process [0]: {data:Confirmation, to:1, from: 6, type: 1} [3] Received a message from the process [0]: {data:Confirmation, to:3, from: 5, type: 1} [7] Received a message from the process [0]: {data:Confirmation, to:7, from: 5, type: 1} [6] Received a message from the process [0]: {data:Confirmation, to:6, from: 2, type: 1} [4] Received a message from the process [0]: {data:Confirmation, to:4, from: 2, type: 1} [4] Received a message from the process [0]: {data:Finish, to:4, from: 0, type: 2} [2] Received a message from the process [0]: {data:Finish, to:2, from: 0, type: 2} [5] Received a message from the process [0]: {data:Finish, to:5, from: 0, type: 2} [1] Received a message from the process [0]: {data:Finish, to:1, from: 0, type: 2} [1] Received a message from the process [0]: {data:Finish, to:1, from: 0, type: 2} [7] Received a message from the process [0]: {data:Finish, to:7, from: 0, type: 2} [3] Received a message from the process [0]: {data:Finish, to:3, from: 0, type: 2}

Выводы по работе.

Была написана программа, осуществляющая обмен сообщениями между процессами через центральный процесс 0. При получении сообщений процессы отправляют отправителю подтверждение.

Очевидно, что с увеличение числа процессов (а соответственно и количество отправляемых сообщений) и размеров сообщений время работы программы будет линейно увеличиваться, а эффективность уменьшаться. Исходя из этого было принято решение не строить графики зависимости для данной программы.