Министерство образования и науки Российской Федерации

Государственное образовательное учреждение высшего профессионального образования

«Московский Государственный Технический Университет им. Н.Э. Баумана»

Факультет «Аэрокосмический»

Кафедра «Вычислительные машины, комплексы, системы и сети»

Отчёт по лабораторной работе №8

«Написание программ с использованием библиотеки MPI»

по дисциплине

«Вычислительные системы»

Выполнили: \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Федоренков Р.В.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Гаджиев Г.П.

Группа: АК5-91

Москва 2014г.

**Цель работы** – изучение структуры MPI-программ и практическая реализация программ.

**Ответы на вопросы для самопроверки**

**1. Какую структура имеет простая MPI-программа?**

Основная схема программы MPI подчиняется следующим общим шагам:

1. Инициализация для коммуникаций
2. Коммуникации распределения данных по процессам
3. Выход "чистым" способом из системы передачи сообщений по завершении коммуникаций

**2. Какие основные функции реализует библиотека MPI?**

MPI имеет свыше 125 **функций**. Тем не менее, начинающий программист обычно может иметь дело только с шестью функциями, которые перечислены ниже:

* Инициализация для коммуникаций

*MPI\_Init* инициализирует окружение MPI

*MPI\_Comm\_size* возвращает число процессов

*MPI\_Comm\_rank* возвращает номер текущего процесса (ранг = номер по-порядку)

* Коммуникации распределения данных по процессами

*MPI\_Send* отправляет сообщение

*MPI\_Recv* получает сообщение

* Выход из системы передачи сообщений

*MPI\_Finalize*

**3. Как организовать передачу сообщений между процессами?**

С помощью вышеописанных функций, **MPI\_Send, MPI\_Recv.**

**Сообщения** MPI состоят из двух основных частей: отправляемые/получаемые данные, и сопроводительная информация (записи на конверте /оболочке/), которая помогает отправить данные по определенному маршруту. Обычно существуют три вызываемых параметра в вызовах передачи сообщений MPI, которые описывают данные и три других параметра, которые определяют маршрут:

*Сообщение = данные (3 параметра) + оболочка (3 параметра)*

Данные определяют информацию, которая будет отослана или получена. Оболочка (конверт) используется в маршрутизации сообщения к получателю и связывает вызовы отправки с вызовами получения.

**4. Для чего используются коммуникаторы?**

**Коммуникаторы** гарантируют уникальные пространства сообщений. В соединении с группами процессов их можно использовать, чтобы ограничить коммуникацию к подмножеству процессов.

**Практическая часть**

**Задание:** Измените данную программу так, чтобы нулевой процесс не посылал сообщений и запустите её с числом процессов, равным номеру вашего варианта.

**Программа:**

#include <stdio.h>

#include "mpi.h"

void main(int argc,char \* argv[])

{

int ierr,size,rank,a,b;

struct MPI\_Status status;

ierr=MPI\_Init(&argc,&argv);

// Получить размер коммуникатора MPI\_COMM\_WORLD

// (общее число процессов в рамках задачи)

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

// Получить номер текущего процесса в рамках

// коммуникатора MPI\_COMM\_WORLD

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

a=rank;

b=-1;

if (rank%2 == 0) {

if (rank != 0 && rank+1<size)

ierr=MPI\_Send(&a, 1, MPI\_INT, rank+1, 5, MPI\_COMM\_WORLD);

} else {

if (rank != 1) {

ierr=MPI\_Recv(&b,1,MPI\_INT,rank-1,5,MPI\_COMM\_WORLD,&status);

}

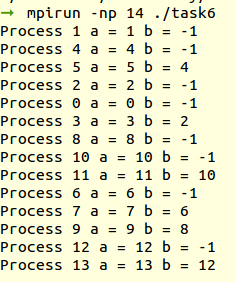
}

printf ("Process %i a = %i b = %i\n",rank,a,b) ;

ierr=MPI\_Finalize ();

}

**Результат работы:**

****

**Задание**: Измените программу так, чтобы при успешном завершении всех передач сообщений процесс 0 посылал остальным процессам сообщение “SENDS COMPLETED” и они, получив получив это сообщение, выводили его.

**Программа:**

#include "mpi.h"

#include <stdio.h>

int main(int argc, char \*argv[])

{

int rank, size;

int i, index[4], count, remaining, a;

int buffer[400];

MPI\_Request request[4];

MPI\_Status status[4];

MPI\_Init(&argc, &argv);

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

if (size > 4)

{

printf("Please run with 4 processes.\n");fflush(stdout);

MPI\_Finalize();

return 1;

}

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

a=0;

// while (a != 1) {

{

if (rank == 0)

{

for (i=0; i<size \* 100; i++)

buffer[i] = i/100;

for (i=0; i<size-1; i++)

{

MPI\_Isend(&buffer[i\*100], 100, MPI\_INT, i+1, 123, MPI\_COMM\_WORLD, &request[i]);

}

remaining = size-1;

while (remaining > 0)

{

MPI\_Waitsome(size-1, request, &count, index, status);

if (count > 0)

{

printf("%d sends completed\n", count);

fflush(stdout);

remaining = remaining - count;

}

if (count == size-1) {

for (i=0; i<size-1; i++) {

buffer[0] = -1;

MPI\_Isend(&buffer[0], 1, MPI\_INT, i+1, 123, MPI\_COMM\_WORLD, &request[i]);

}

}

}

}

else

{

MPI\_Recv(buffer, 100, MPI\_INT, 0, 123, MPI\_COMM\_WORLD, &status[0]);

printf("%d: buffer[0] = %d\n", rank, buffer[0]);

fflush(stdout);

MPI\_Recv(buffer, 1, MPI\_INT, 0, 123, MPI\_COMM\_WORLD, &status[0]);

if (buffer[0] == -1) {

printf("%d: SENDS COMPLITED\n", rank, buffer[0]);

fflush(stdout);

}

}

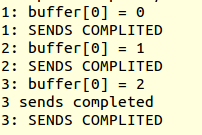
}

MPI\_Finalize();

return 0;

}

**Результат работы:**

****

**Вывод:**

В результате лабораторной работы была изучение структуры MPI-программ и практически реализованы 2 программы.