Ковальский Евгений

Лабораторная работа №1

Пример 1

#include <stdio.h>

#include "mpi.h"

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

{

int ProcNum, ProcRank, RecvRank;

MPI\_Status Status;

MPI\_Init(&argc, *&argv*);

MPI\_Comm\_size(MPI\_COMM\_WORLD, *&ProcNum*);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, *&ProcRank*);

*if* (ProcRank == 0)

{

*//* *Действия,* *выполняемые* *только* *процессом* *с* *рангом* *0*

printf ("\n Hello from process %3d", ProcRank);

*for* ( int i=1; i < ProcNum; i++ )

{

MPI\_Recv(*&RecvRank*, 1, MPI\_INT, MPI\_ANY\_SOURCE,

MPI\_ANY\_TAG, MPI\_COMM\_WORLD, *&Status*);

printf("\n Hello from process %3d", RecvRank);

}

} *else* *//* *Сообщение,* *отправляемое* *всеми* *процессами,*

*//* *кроме* *процесса* *с* *рангом* *0*

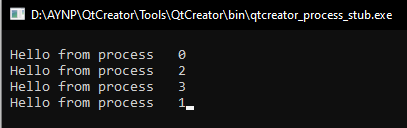
MPI\_Send(&ProcRank,1,MPI\_INT,0,0,MPI\_COMM\_WORLD);

MPI\_Finalize();

*return* 0;

}

Результат выполнения примера:



Задание - Вариант 0.

Реализуйте функцию ring, которая создаёт N процессов и посылает сообщение первому процессу, который посылает сообщение второму, второй - третьему, и так далее. Наконец, процесс N посылает сообщение обратно процессу 1. После того, как сообщение обежало вокруг кольца M раз, все процессы заканчивают работу.

Код программы:

*//--------------------Подключаемые* *библиотеки--------------------//*

#include <headers/mpi.h>

#include <iostream>

#include <cstring>

*using* *namespace* std;

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

*//--------------------Иициализация--------------------//*

int procNum, procRank, recv;

int m = 2;*//количество* *итераций*

MPI\_Status Status;

MPI\_Init(&argc, *&argv*);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &*procNum*);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &*procRank*);

*//--------------------Главные* *процесс--------------------//*

*if* (procRank == 0) {

printf(" Num of processors: %3d", procNum);

printf("\n Main process with rank: %3d", procRank);

int id = 0;

printf("\n Send message to process: %3d", 1);

*//отправляем* *сообщение* *"0"* *процессу* *1*

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

}

*//--------------------Остальные* *процессы--------------------//*

*else* {

*for*(int i = 0; i < m; i++) {

printf("\n Process %3d receiving message... ", procRank);

printf("\n Current time is: %3d", MPI\_Wtime());

*//процесс* *ождидает* *сообщение* *из* *любого* *источника*

*//первый* *попавшийся* *процесс* *перехватит* *управление*

MPI\_Recv(*&recv*, 1, MPI\_INT, MPI\_ANY\_SOURCE,

MPI\_ANY\_TAG, MPI\_COMM\_WORLD, *&Status*);

printf("\n Process recieved massage: %3d", recv);

printf("\n Current time is: %3d", MPI\_Wtime());

*//recv+1* *станет* *отправляемым* *сообщением*

++recv;

*//последний* *процесс* *в* *цепочке*

*if*(procRank == procNum-1) {

*if*(m != i+1){

*//отправляет* *сообщение* *recv* *процессу* *с* *рангом* *на* *1* *больше* *своего*

printf("\n Process sending massage: %3d", recv);

printf("\n Current time is: %3d", MPI\_Wtime());

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

}

}

*//любой* *другой* *процесс*

*else* {

*//отправляет* *сообщение* *recv* *процессу* *с* *рангом* *на* *1* *больше* *своего*

printf("\n Process %3d sending massage: %3d", recv);

printf("\n Current time is: %3d", MPI\_Wtime());

MPI\_Send(&recv, 1, MPI\_INT, procRank+1, 0, MPI\_COMM\_WORLD);

}

}

}

*//--------------------Завершение* *работы* *MPI--------------------//*

MPI\_Finalize();

*return* 0;

}

Вывод программы:

Process 2 receiving message...

Current time is: 1381941805

Process recieved massage: 1

Current time is: 1418689545

Process 2 sending massage: 1

Current time is: 1418775444

Process 2 receiving message...

Current time is: 1429265472

Process recieved massage: 4

Current time is: 1454299978

Process 5 sending massage: 1

Current time is: 1454382441

Process 1 receiving message...

Current time is: 1381409229

Process recieved massage: 0

Current time is: 1402255282

Process 1 sending massage: 1

Current time is: 1402344617

Process 1 receiving message...

Current time is: 1412745310

Process recieved massage: 3

Current time is: 1453664323

Process 4 sending massage: 1

Current time is: 1453757094

Num of processors: 4

Main process with rank: 0

Send message to process: 1

Process 3 receiving message...

Current time is: 1382330070

Process recieved massage: 2

Current time is: 1435505201

Process sending massage: 3

Current time is: 1435584228

Process 3 receiving message...

Current time is: 1446084564

Process recieved massage: 5

Current time is: 1455045584

Контрольные вопросы:

1. В чем состоят основы технологии MPI?

Основы технологии MPI составляют операции передачи сообщений в параллельной программе. При этом процессы параллельной программы объединяются в группы под управлением коммуникатора.

2. В чем состоят основные преимущества и недостатки технологии MPI?

Основные плюсы:

* высокая масштабируемость решения
* высокий уровень парралельности и портабельность кода

Основные минусы:

* сложность при программировании
* достаточно высокие затраты на синхронизацию и обмен данными

3. Что понимается под параллельной программой в рамках технологии MPI?

Под параллельной программой в рамках MPI понимается множество одновременно выполняемых процессов. Процессы могут выполняться на разных процессорах, но на одном процессоре могут располагаться и несколько процессов

4. Как происходит инициализация и завершение MPI программ?

Первой вызываемой функцией MPI должна быть функция:

int MPI\_Init ( int \*agrc, char \*\*\*argv );

для инициализации среды выполнения MPI-программы. Параметрами функции являются количество аргументов в командной строке и текст самой командной строки.

Последней вызываемой функцией MPI обязательно должна являться функция:

int MPI\_Finalize (void);

5. Как происходит передача и прием сообщений MPI программе?

Для приема сообщения процесс-получатель должен выполнить функцию:

int MPI\_Recv(void \*buf, int count, MPI\_Datatype type, int source,

int tag, MPI\_Comm comm, MPI\_Status \*status),

* buf, count, type – буфер памяти для приема сообщения, назначение каждого отдельного параметра соответствует описанию в MPI\_Send
* source - ранг процесса, от которого должен быть выполнен прием сообщения
* tag - тег сообщения, которое должно быть принято для процесса
* comm - коммуникатор, в рамках которого выполняется передача данных
* status – указатель на структуру данных с информацией о результате выполнения операции приема данных

Следует отметить:

* буфер памяти должен быть достаточным для приема сообщения, а тип элементов передаваемого и принимаемого сообщения должны совпадать; при нехватке памяти часть сообщения будет потеряна и в коде завершения функции будет зафиксирована ошибка переполнения;
* при необходимости приема сообщения от любого процесса-отправителя для параметра source может быть указано значение MPI\_ANY\_SOURCE;
* при необходимости приема сообщения с любым тегом для параметра tag может быть указано значение MPI\_ANY\_TAG;
* параметр status позволяет определить ряд характеристик принятого сообщения: - status.MPI\_SOURCE – ранг процесса-отправителя принятого сообщения, status.MPI\_TAG - тег принятого сообщения

Вызов функции MPI\_Recv не должен согласовываться со временем вызова соответствующей функции передачи сообщения MPI\_Send – прием сообщения может быть инициирован до момента, в момент или после момента начала отправки сообщения.

По завершении функции MPI\_Recv в заданном буфере памяти будет располагаться принятое сообщение. Принципиальный момент здесь состоит в том, что функция MPI\_Recv является блокирующей для процесса-получателя, т.е. его выполнение приостанавливается до завершения работы функции. Таким образом, если ожидаемое для приема сообщение будет отсутствовать, выполнение параллельной программы будет блокировано.