ЛР-1

[Подзаголовок документа]

Тюльников Михаил ПИН-32

[Год]

# Контрольные вопросы

1. В чем состоят основы технологии MPI?

***Ответ:*** Основу MPI составляют операции передачи сообщений. Среди предусмотренных в составе MPI функций различаются парные (point-to-point) операции между двумя процессами и коллективные (collective) коммуникационные действия для одновременного взаимодействия нескольких процессов.

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

***Ответ:* Преимущества:**

1. Разделение параллельных программ между разными компьютерными системами.
2. MPI облегчает процесс написания параллельных программ, когда при разработке используются библиотеки программных модулей, написанных с использованием этого интерфейса.
3. MPI позволяет создавать хорошо масштабируемые параллельные программы.

**Недостатки:**

1. MPI является низкоуровневым инструментом программиста.
2. Не существует реализаций MPI, в полной мере обеспечивающих совмещение обменов с вычислениями.
3. Отсутствуют механизмы задания начального размещения процессов по процессорам.
4. Отладка программ затруднительна вследствие одновременного исполнения нескольких программных ветвей.

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

***Ответ:***Под параллельной программой в рамках MPI понимается множество одновременно выполняемых процессов. Процессы могут выполняться как на разных процессорах, так и на одном. Каждый процесс параллельной программы порождается на основе копии одного и того же программного кода (модель SPMD). Все процессы программы последовательно перенумерованы от 0 до p-1, где p есть общее количество процессов. Номер процесса именуется рангом процесса.

4. Как происходит инициализация и завершение MPI программ?

***Ответ:* 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?

***Ответ:*** Для **передачи сообщения** процесс-отправитель должен выполнить функцию

int MPI\_Send(void\* buf, int count, MPI\_Datatype type, int dest, int tag, MPI\_Comm comm);

– отправляемое сообщение определяется через указание буфера памяти, в котором это сообщение располагается;

– используемая для указания буфера триада ( buf, count, type ) входит в состав параметров практически всех функций передачи данных;

– процессы, между которыми выполняется передача данных, должны принадлежать коммуникатору, указываемому в функции MPI\_Send.

Сразу же после завершения функции MPI\_Send процесс-отправитель может начать повторно использовать буфер памяти, в котором располагалось отправляемое сообщение. Вместе с этим, следует понимать, что в момент завершения функции MPI\_Send состояние самого пересылаемого сообщения может быть совершенно различным - сообщение может:

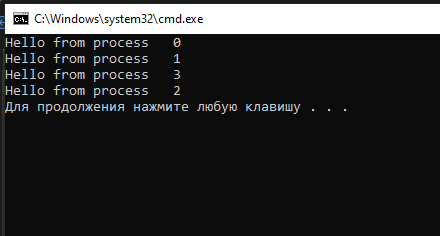
* располагаться в процессе-отправителе,
* находиться в процессе передачи, - храниться в процессе-получателе
* или же может быть принято процессом-получателем при помощи функции MPI\_Recv.

Завершение функции MPI\_Send означает лишь, что операция передачи начала выполняться.

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);

# Пример 1.



# Лабораторное задание Вариант 1

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include "mpi.h"

#include <iostream>

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

{

int M = 4;

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);

int msg = 1, recvMsg = -1;

for (int j = 0; j < M; j++) {

if (ProcRank == 0)

{

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

{

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

printf("Send %d to process %3d\n", msg, i);

}

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

{

MPI\_Recv(&recvMsg, 1, MPI\_INT, MPI\_ANY\_SOURCE, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &Status);

printf("Receive %d from process %3d\n", recvMsg, i);

}

}

else

{

MPI\_Recv(&recvMsg, 1, MPI\_INT, MPI\_ANY\_SOURCE, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &Status);

printf("Receive %d in process %3d\n", recvMsg, ProcRank);

msg = recvMsg \* ProcRank;

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

printf("Send %d to process %3d\n", msg, 0);

}

}

MPI\_Finalize();

return 0;

}

