**Лабораторные задания** (№ варианта = 1)

**Задание.** В соответствии с вариантом задания, написать на C++ программу, реализующую многопроцессность на основе технологии MPI, работающую на основе программа должна работать на основе простой передачи сообщений.

|  |  |
| --- | --- |
| **Вариант** | **Задание: написать программы, демонстрирующие работу следующей функции:** |
| **1** | Реализуйте процесс-«счётчик», (который запускается со значением 0 и  1) если получена -1, то он выводит в текущее значение и заканчивает работу;  2) если получено любое другое сообщение, то значение увеличивается на 1 и выводится сообщение об этом. |

#include <stdio.h>

#include "mpi.h"

#include <iostream>

#include <time.h>

#include <locale>

using namespace std;

int generate\_int(int min\_v, int max\_v) {

return min\_v + rand() % (max\_v - min\_v);

}

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

{

int ProcNum, ProcRank;

int RecvRank = 0;

MPI\_Init(&argc, &argv);

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

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

MPI\_Status Status;

bool flag = true;

int counter = 0;

int rand\_send = 0;

srand(time(0));

if (ProcRank == 0)

{

//printf("\n Beginning if");

while (flag) {

//printf("\n Beginning 0");

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

{

//printf("\n Beginning 1");

MPI\_Recv(&RecvRank, 1, MPI\_INT, i, MPI\_ANY\_TAG, MPI\_COMM\_WORLD, &Status);

if (RecvRank == -1) {

flag = false;

MPI\_Send(&flag, 1, MPI\_C\_BOOL, Status.MPI\_SOURCE, 0, MPI\_COMM\_WORLD);

printf("\n Exit the loop. Counter value: %3d", counter);

}

else // Сообщение, отправляемое всеми процессами, кроме процесса с рангом 0

{

counter++;

printf("\n The counter has been increased: %3d", counter);

MPI\_Send(&flag, 1, MPI\_C\_BOOL, Status.MPI\_SOURCE, 0, MPI\_COMM\_WORLD);

}

}

}

}

else {

//printf("\n Beginning else");

while (flag) {

//printf("\n Beginning 2");

rand\_send = generate\_int(-1,7);

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

printf("\n Random value sent to 0 process: %d", rand\_send);

MPI\_Recv(&flag, 1, MPI\_C\_BOOL, 0, 0, MPI\_COMM\_WORLD, &Status);

printf("\n Flag information accepted");

}

}

MPI\_Finalize();

return 0;

}

**Контрольные вопросы**

1. В чем состоят основы технологии MPI?
2. В чем состоят основные преимущества и недостатки технологии MPI?
3. Что понимается под параллельной программой в рамках технологии MPI?
4. Как происходит инициализация и завершение MPI программ?
5. Как происходит передача и прием сообщений MPI программе?

**1.** 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.

Процессы объединяются в **группы**. Под **коммуникатором** в MPI понимается служебный объект, объединяющий в своем составе группу процессов и ряд дополнительных параметров (**контекст**), используемых при выполнении операций передачи данных. Один и тот же процесс может принадлежать разным группам и коммуникаторам. Все имеющиеся в программе процессы входят в состав создаваемого по умолчанию коммуникатора с идентификатором MPI\_COMM\_WORLD.

При необходимости передачи данных между процессами из разных групп необходимо создавать глобальный коммуникатор (**intercommunicator**).

**2.**

Преимущества MPI:

* Обеспечивает чрезвычайно высокую производительность при грамотной передаче сообщений;
* Старейший стандарт, открытый, имеет множество реализаций для различных языков программирования;
* Имеет множество реализаций для различных архитектур;
* Поддерживаются различные примитивы синхронизации: барьер, мьютекс, монитор.

Недостатки MPI:

* Сложность разработки и отладки
* Большое количество реализаций вызывает проблемы во

взаимодействии между ними за счет недостаточно качественного кода и, де факто, частичной поддержки стандарта;

* Портирование одной из существующих реализаций под новую архитектуру может быть сложным в силу низкоуровневости стандарта;
* Для запуска программы необходима установка соответствующего программного обеспечения на все составляющие программного комплекса

**3.** 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 SPMP**). Все процессы программы последовательно перенумерованы от 0 до **p-1**, где **p** есть общее количество процессов. Номер процесса именуется **рангом** процесса.

**4.**

**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.**

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);

где

- buf – адрес буфера памяти, в котором располагаются данные отправляемого сообщения,

- count – количество элементов данных в сообщении,

- type - тип элементов данных пересылаемого сообщения,

- dest - ранг процесса, которому отправляется сообщение,

- tag - значение-тег, используемое для идентификации сообщений,

- comm - коммуникатор, в рамках которого выполняется передача данных.

- 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.

**Прием сообщений**

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);

где

- 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.