Основы параллельного программирования с использованием MPI

Лекция 3

Немнюгин Сергей Андреевич

Санкт-Петербургский государственный университет

физический факультет

кафедра вычислительной физики

Интернет-Университет
Суперкомпьютерных Технологий

High-Performance Computing University

Лекция 3

Аннотация

В третьей лекции обсуждаются средства организации двухточечных обменов. Рассматриваются различные режимы блокирующих двухсторонних обменов, их реализация в MPI, приводятся примеры использования. Основное внимание уделяется стандартным обменам, обменам с буферизацией и «по готовности». Рассматриваются функции «приёмопередачи», а также возможные проблемы при организации двухточечных обменов.

План лекции

- Настройка рабочего места для выполнения практических заданий.
- Блокирующие двухточечные обмены.
- Двухточечные обмены с буферизацией, другие типы двухточечных обменов.
- Совмещённые операции приёма и передачи.

Настройка рабочего места

Настройка рабочего места

Рассмотрим настройку рабочего места в среде Microsoft Windows

MPICH2 для MS Windows можно скачать по адресу:
http://www.mcs.anl.gov/research/projects/mpich2/

Выбираем операционную систему, затем переходим по ссылке [http]
и скачиваем **msi**-файл (размер около 9 Мбайт), например:
mpich2-1.0.8-win-ia32.msi

Затем следует щелкнуть по имени файла, тогда запустится программа-установщик. Для установки потребуются права Администратора системы (запуск MPI-программ выполняется соответствующей службой).

Настройка рабочего места

После щелчка будет произведена установка диспетчера процессов spmd (spmd process manager). Необходимо указать «секретное» слово. По умолчанию это:
behappy

В меню «Пуск»->«Все программы» появится раздел MPICH2 с пунктами:
jumpshot
wmpiconfig.exe
wmpiexec.exe
wmpiregister.exe

Настройка рабочего места

В папке MPICH2 располагаются папки:
bin
examples
include
jumpshot
lib
Динамически подключаемые (dll) библиотеки копируются в папку windows/system32.

Диспетчер процессов smpd автоматически запускается сразу же после установки MPICH2, а также после перезагрузки системы.

Настройка рабочего места

Библиотеки в папке **lib** скомпилированы с помощью компиляторов MS Visual C++.NET и Intel ® Fortran 8.1.

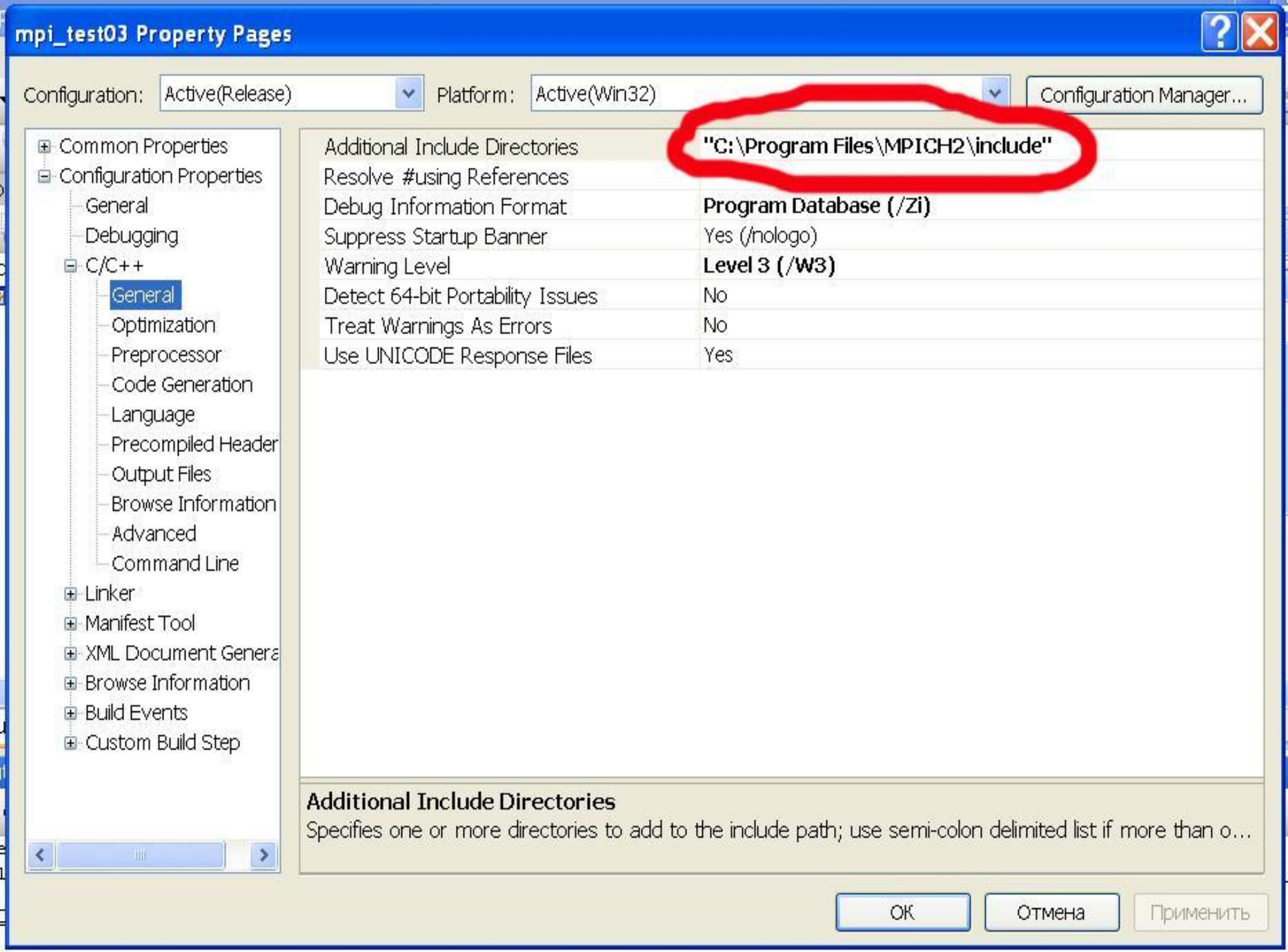
Запустим MS Visual Studio 2008 и создадим новый проект **Win32 Console Application** (Консольное приложение Win32), в конфигурации **Release**.

В окно проекта копируется исходный текст программы, если он уже существует или программа набирается заново.

Затем необходимо настроить пути и ссылки на библиотеку MPICH2.

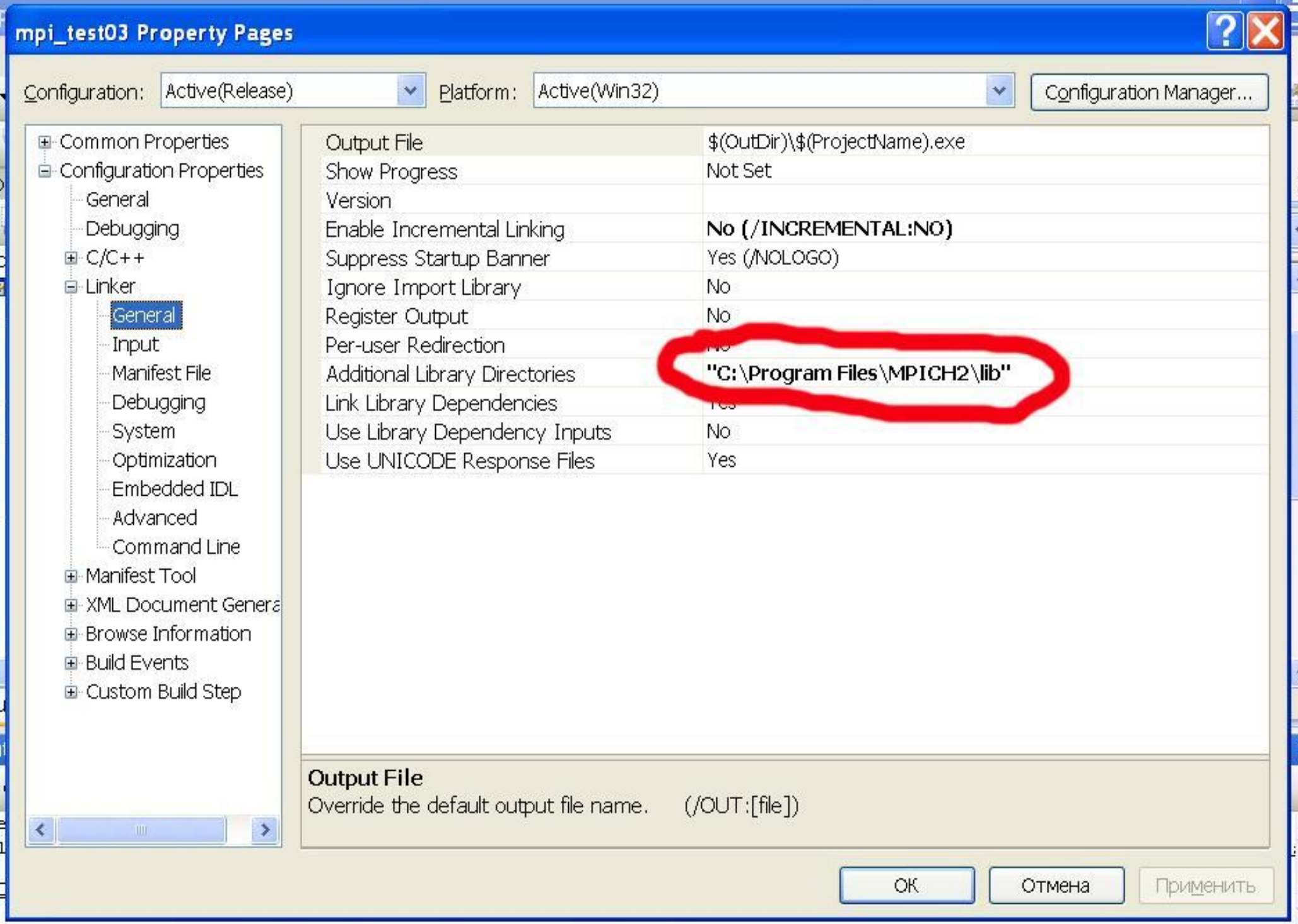
Настройка рабочего места

В строке
Project -> <Project_name> properties -> C++->General->Additional Include Directories
вводится путь к каталогу **include**.


mpi_test03 Property Pages
Configuration: Active(Release)
Platform: Active(Win32)
Configuration Manager...
Common Properties
Configuration Properties
General
Debugging
C/C++
General
Optimization
Preprocessor
Code Generation
Language
Precompiled Header
Output Files
Browse Information
Advanced
Command Line
Linker
Manifest Tool
XML Document Genera
Browse Information
Build Events
Custom Build Step
Additional Include Directories
"C:\Program Files\MPICH2\include"
Resolve #using References
Debug Information Format
Program Database (/Zi)
Suppress Startup Banner
Yes (/nologo)
Warning Level
Level 3 (/W3)
Detect 64-bit Portability Issues
No
Treat Warnings As Errors
No
Use UNICODE Response Files
Yes
Additional Include Directories
Specifies one or more directories to add to the include path; use semi-colon delimited list if more than o...
OK
Отмена
Применить

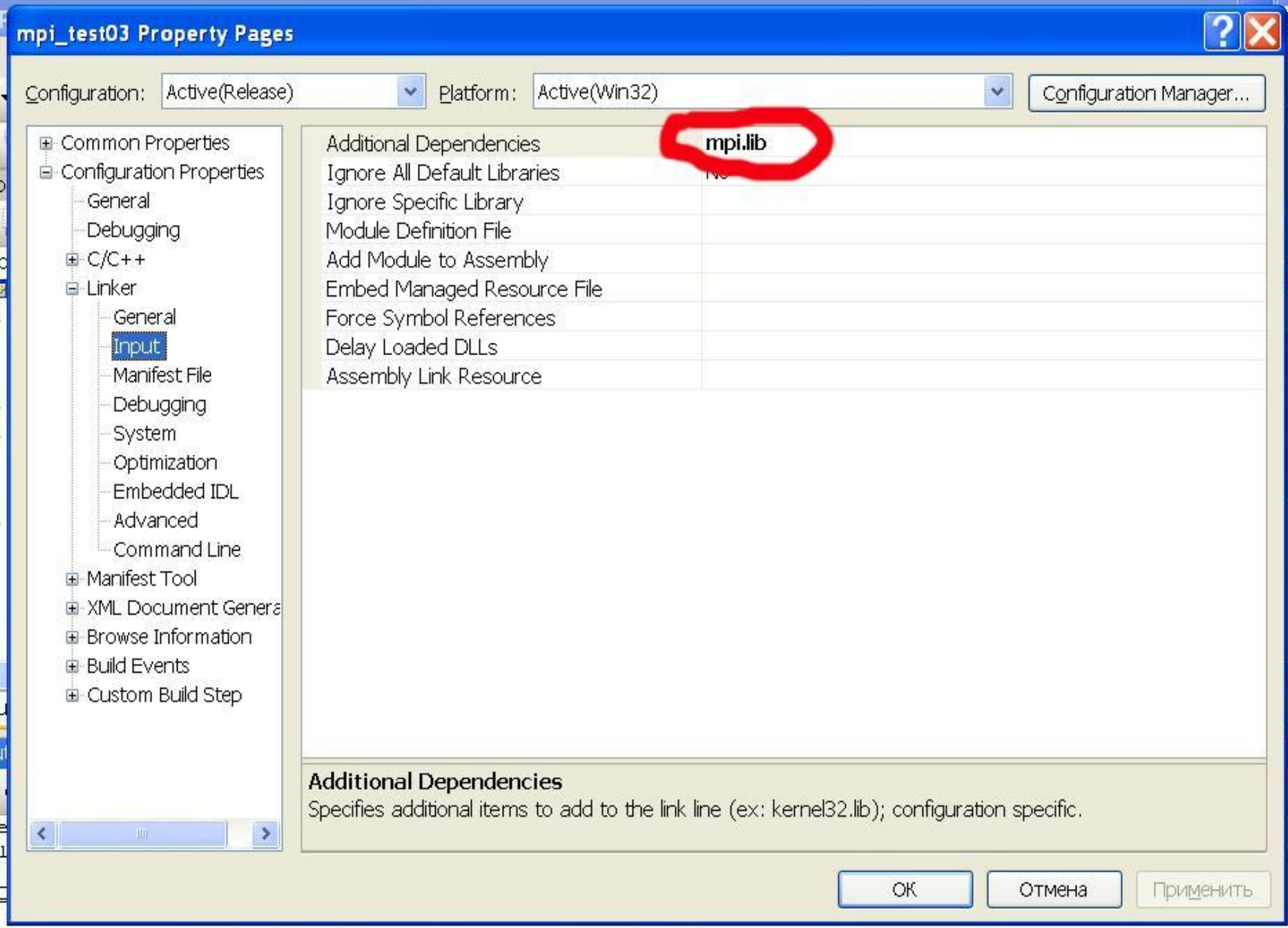
Настройка рабочего места

В строке
Project -> <Project_name> properties ->
C++->Linker->General->Additional Library Directories
вводится путь к каталогу **lib**.


mpi_test03 Property Pages
Configuration: Active(Release)
Platform: Active(Win32)
Configuration Manager...
Common Properties
Configuration Properties
General
Debugging
C/C++
Linker
General
Input
Manifest File
Debugging
System
Optimization
Embedded IDL
Advanced
Command Line
Manifest Tool
XML Document Genera
Browse Information
Build Events
Custom Build Step
Output File $(OutDir)\$(ProjectName).exe
Show Progress Not Set
Version
Enable Incremental Linking No (/INCREMENTAL:NO)
Suppress Startup Banner Yes (/NOLOGO)
Ignore Import Library No
Register Output No
Per-user Redirection
Additional Library Directories "C:\Program Files\MPICH2\lib"
Link Library Dependencies
Use Library Dependency Inputs No
Use UNICODE Response Files Yes
Output File
Override the default output file name. (/OUT:[file])
OK
Отмена
Применить

Настройка рабочего места

В строке
Project -> <Project_name> properties ->
C++->Linker->Input->Additional Dependencies
вводится имя библиотеки **mpi.lib**.


mpi_test03 Property Pages
Configuration: Active(Release)
Platform: Active(Win32)
Configuration Manager...
Common Properties
Configuration Properties
General
Debugging
C/C++
Linker
General
Input
Manifest File
Debugging
System
Optimization
Embedded IDL
Advanced
Command Line
Manifest Tool
XML Document Genera
Browse Information
Build Events
Custom Build Step
Additional Dependencies
mpi.lib
Ignore All Default Libraries
Ignore Specific Library
Module Definition File
Add Module to Assembly
Embed Managed Resource File
Force Symbol References
Delay Loaded DLLs
Assembly Link Resource
Additional Dependencies
Specifies additional items to add to the link line (ex: kernel32.lib); configuration specific.
OK
Отмена
Применить

Настройка рабочего места

Теперь проект настроен на использование MPICH2.

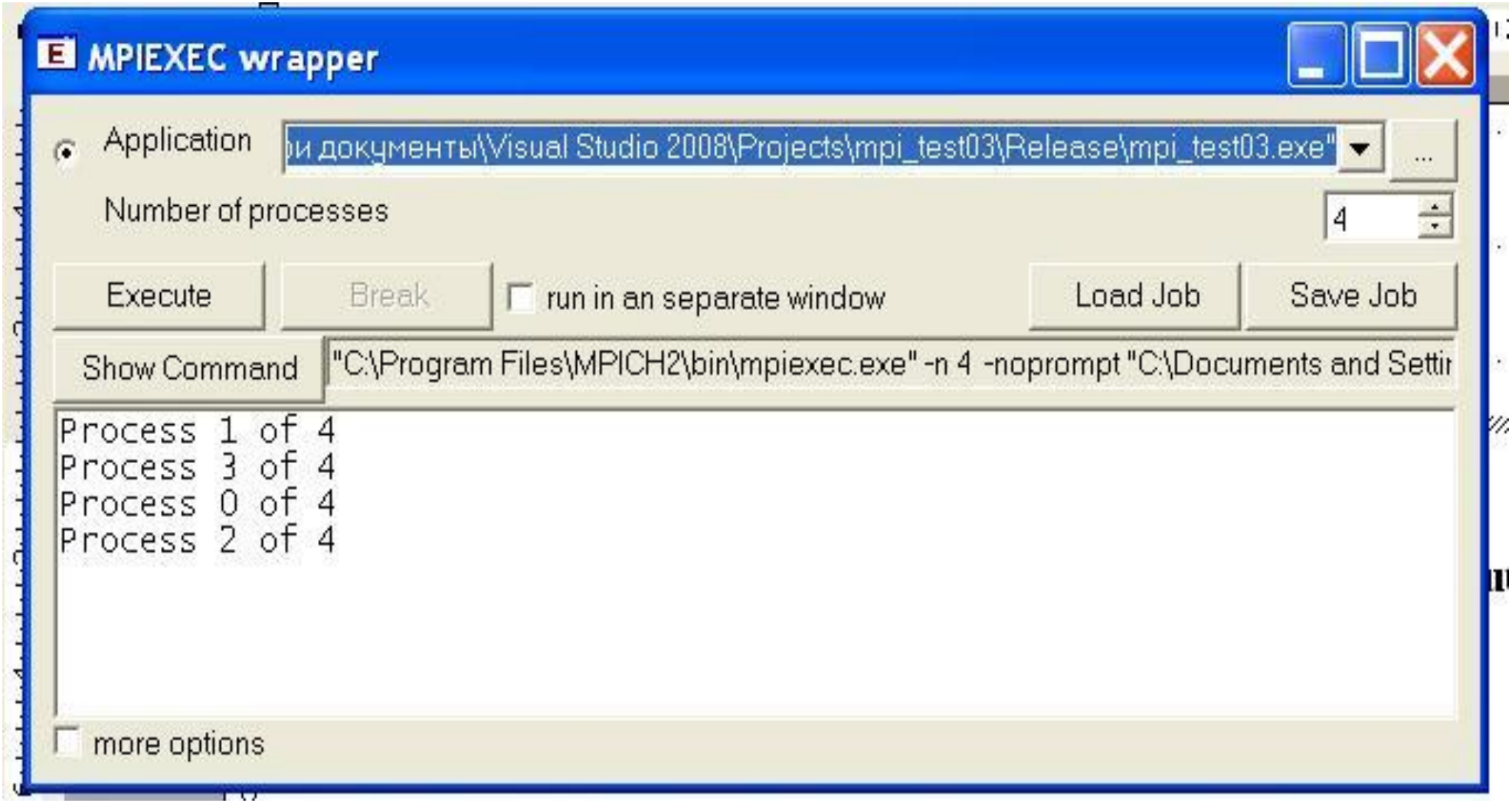
Выполним компиляцию программы (пункт **Build -> Rebuild Solution**).

Если отсутствуют синтаксические ошибки, программа готова к выполнению.

Работая в среде MS Windows, запустим **wmpiregister.exe** и введем свое регистрационное имя, а также «секретное слово» (**behappy**).
В результате будет запущена служба **spmd**.

Для выполнения MPI-программы необходимо воспользоваться программой **wmpiexec.exe**.

Запуск MPI-программы в среде MS Windows

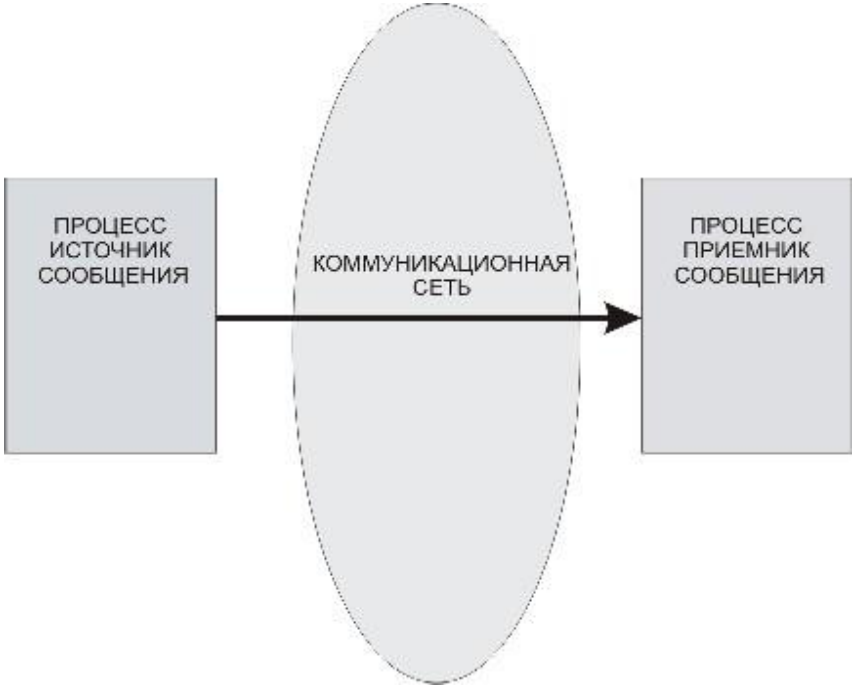


Двухточечные обмены

Двухточечные обмены

Двухточечный (point-to-point, p2p) обмен

В двухточечном обмене участвуют только два процесса, процесс-отправитель и процесс-получатель (источник сообщения и адресат). Двухточечные обмены используются для организации локальных и неструктурированных коммуникаций.



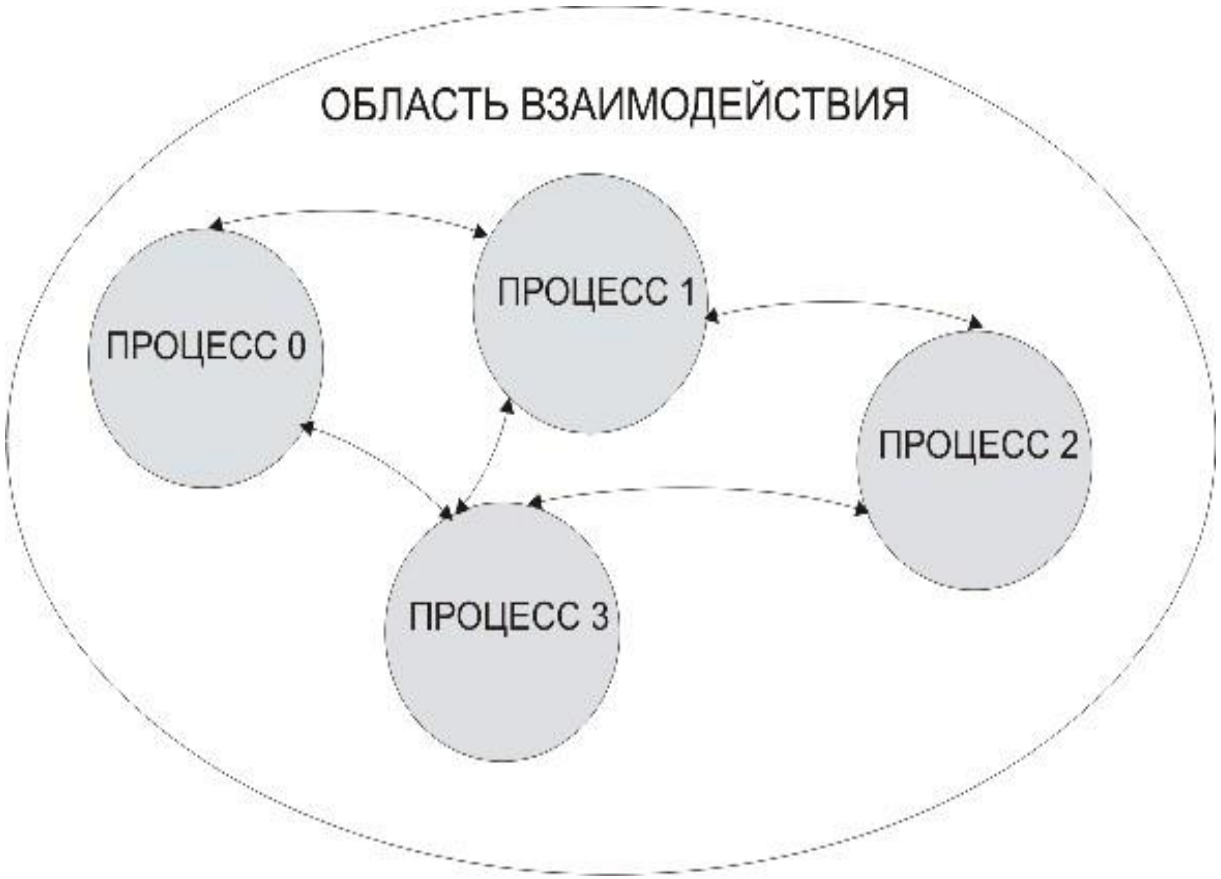
Двухточечные обмены

Имеется несколько разновидностей двухточечного обмена

- *блокирующие* прием/передача, которые приостанавливают выполнение процесса на время приема или передачи сообщения;
- *неблокирующие* прием/передача, при которых выполнение процесса продолжается в фоновом режиме, а программа в нужный момент может запросить подтверждение завершения приема сообщения;
- *синхронный* обмен, который сопровождается уведомлением об окончании приема сообщения;
- *асинхронный* обмен, который таким уведомлением не сопровождается.

Двухточечные обмены

Двухточечный обмен возможен только между процессами, принадлежащими одной области взаимодействия (одному коммуникатору).



Двухточечные обмены

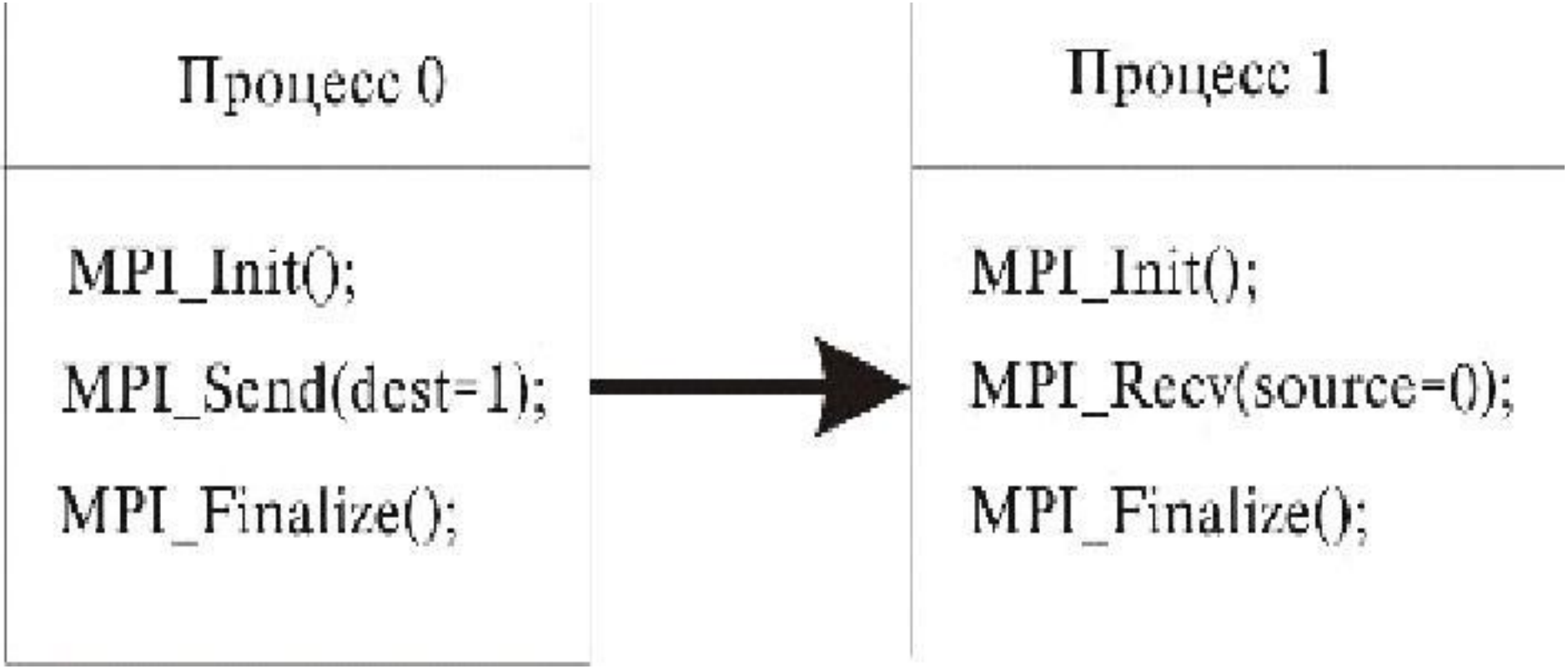
Правильно организованный двухточечный обмен сообщениями должен исключать возможность блокировки или некорректной работы параллельной MPI-программы.

Примеры ошибок в организации двухточечных обменов:

- выполняется передача сообщения, но не выполняется его прием;
- процесс-источник и процесс-получатель одновременно пытаются выполнить блокирующие передачу или прием сообщения.

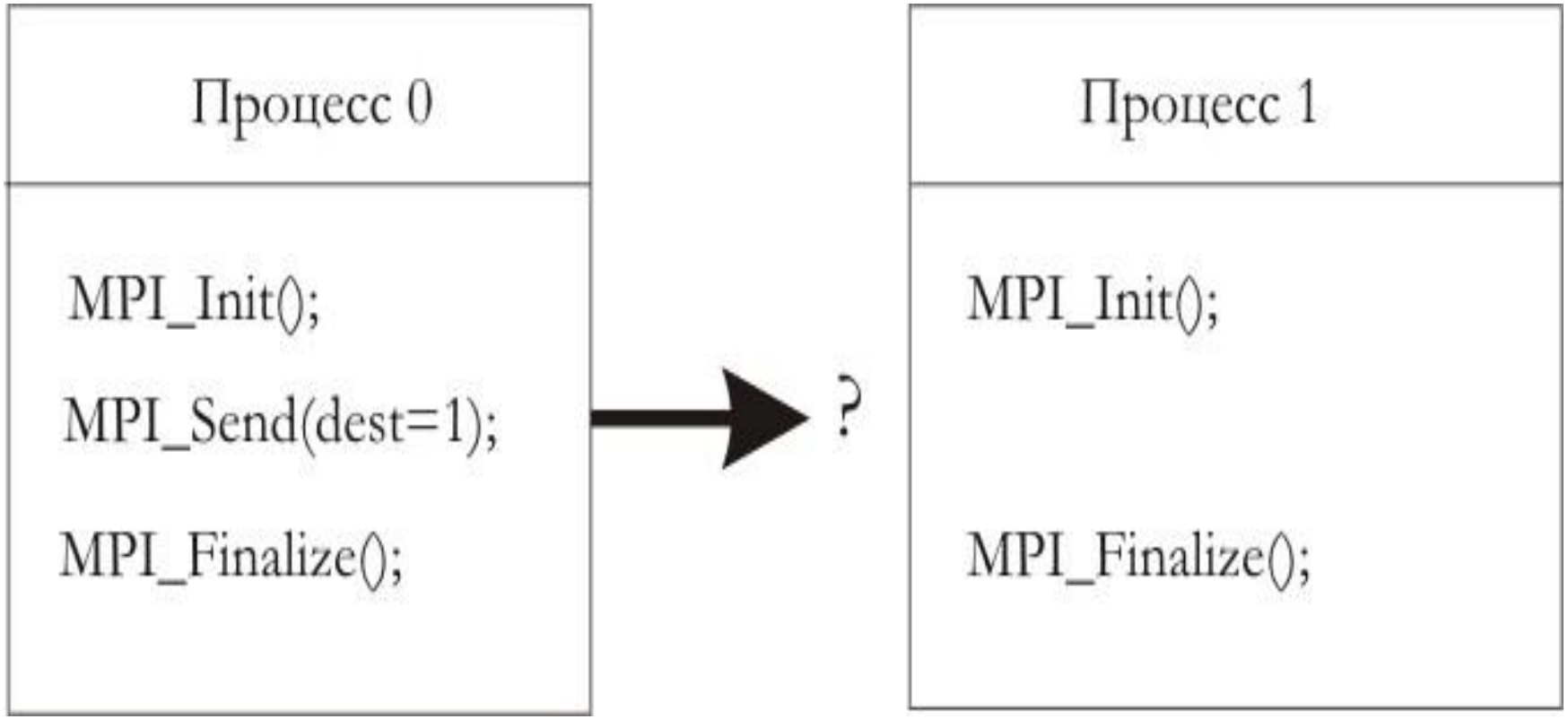
Двухточечные обмены

Правильно



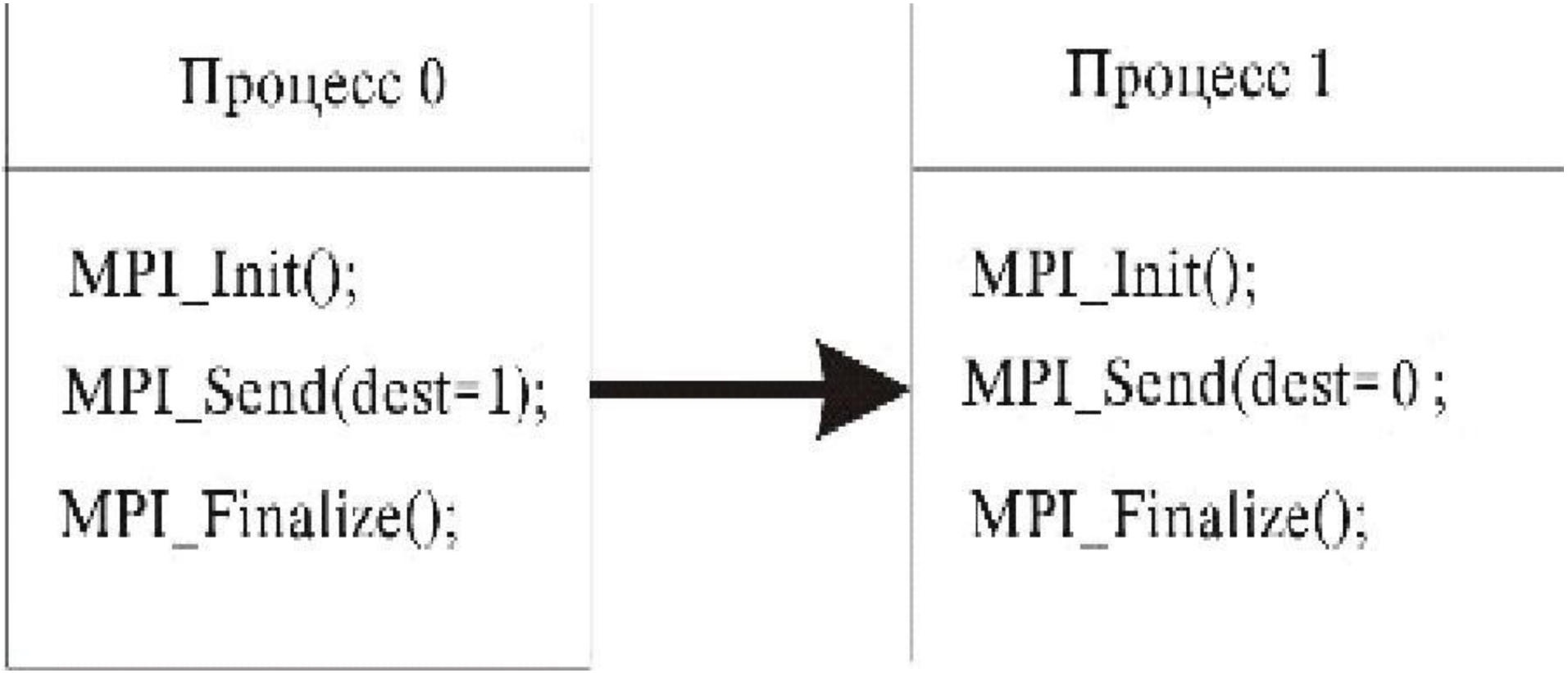
Двухточечные обмены

Неправильно



Двухточечные обмены

Неправильно



Двухточечные обмены

В MPI приняты следующие соглашения об именах подпрограмм двухточечного обмена:
`MPI_[I][R, S, B]Send`
здесь префикс [`I`] (Immediate) обозначает неблокирующий режим.
Один из префиксов [`R, S, B`] обозначает режим обмена:
по готовности, синхронный и буферизованный.
Отсутствие префикса обозначает подпрограмму стандартного обмена.
Имеется 8 разновидностей операции передачи сообщений.
Для подпрограмм приема:
`MPI_[I]Recv`
то есть всего 2 разновидности приема.
Подпрограмма `MPI_Irsend`, например, выполняет передачу «по готовности» в неблокирующем режиме, `MPI_Bsend` буферизованную передачу с блокировкой, а `MPI_Recv` выполняет блокирующий прием сообщений. Подпрограмма приема любого типа может принять сообщения от любой подпрограммы передачи.

Стандартный блокирующий двухточечный обмен

Двухточечные обмены

Стандартная блокирующая передача

```
int MPI_Send(void *buf, int count, MPI_Datatype datatype,int
dest, int tag, MPI_Comm comm)

MPI_Send(buf, count, datatype, dest, tag, comm, ierr)
```

- `buf` - адрес первого элемента в буфере передачи;
- `count` - количество элементов в буфере передачи (допускается `count` = 0);
- `datatype` - тип MPI каждого пересылаемого элемента;
- `dest` - ранг процесса-получателя сообщения (целое число от 0 до n – 1, где n число процессов в области взаимодействия);
- `tag` - тег сообщения;
- `comm` - коммуникатор;
- `ierr` - код завершения.

Двухточечные обмены

При стандартной блокирующей передаче после завершения вызова (после возврата из функции/процедуры передачи) можно использовать любые переменные, использовавшиеся в списке параметров. Такое использование не повлияет на корректность обмена.
Дальнейшая «судьба» сообщения зависит от реализации MPI.
Сообщение может быть сразу передано процессу-получателю или может быть скопировано в буфер передачи.

Завершение вызова не гарантирует доставки сообщения по назначению. Такая гарантия предоставляется при использовании других разновидностей двухточечного обмена (см. далее материал этой лекции).

Двухточечные обмены

Стандартный блокирующий прием

```
int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int
source, int tag, MPI_Comm comm, MPI_Status *status)

MPI_Recv(buf, count, datatype, dest, tag, comm, status, ierr)
```

- `buf` - адрес первого элемента в буфере приёма;
- `count` - количество элементов в буфере приёма;
- `datatype` - тип MPI каждого пересылаемого элемента;
- `source` - ранг процесса-отправителя сообщения (целое число от 0 до n – 1, где n число процессов в области взаимодействия);
- `tag` - тег сообщения;
- `comm` - коммуникатор;
- `status` - статус обмена;
- `ierr` - код завершения.

Двухточечные обмены

Значение параметра `count` может оказаться больше, чем количество элементов в принятом сообщении. В этом случае после выполнения приёма в буфере изменится значение только тех элементов, которые соответствуют элементам фактически принятого сообщения.

Для функции `MPI_Recv` гарантируется, что после завершения вызова сообщение принято и размещено в буфере приема.

Двухточечные обмены

Специфические коды завершения:

- `MPI_ERR_COMM` - неправильно указан коммуникатор. Часто возникает при использовании «пустого» коммуникатора;
- `MPI_ERR_COUNT` - неправильное значение аргумента count (количество пересылаемых значений);
- `MPI_ERR_TYPE` - неправильное значение аргумента, задающего тип данных;
- `MPI_ERR_TAG` - неправильно указан тег сообщения;
- `MPI_ERR_RANK` - неправильно указан ранг источника или адресата сообщения;
- `MPI_ERR_ARG` - неправильный аргумент, ошибочное задание которого не попадает ни в один класс ошибок;
- `MPI_ERR_REQUEST` - неправильный запрос на выполнение операции.

Двухточечные обмены

«Джокеры»

В качестве ранга источника сообщения и в качестве тега сообщения можно использовать «джокеры» :

- `MPI_ANY_SOURCE` - любой источник;
- `MPI_ANY_TAG` - любой тег.

При использовании «джокеров» есть опасность приема сообщения, не предназначенного данному процессу

Двухточечные обмены

Подпрограмма `MPI_Recv` может принимать сообщения, отправленные в любом режиме.
Прием может выполняться от произвольного процесса, а в операции передачи должен быть указан вполне определенный адрес. Приемник может использовать «джокеры» для источника и для тега. Процесс может отправить сообщение и самому себе, но следует учитывать, что использование в этом случае блокирующих операций может привести к «тупику».

Двухточечные обмены

Пример использования операции блокирующего двухточечного обмена

```
#include <mpi.h>
#include <stdio.h>
int main (int argc, char *argv[]) {
	int ProcNum, ProcRank, tmp;
	MPI_Status status;
	MPI_Init ( &argc, &argv );
	MPI_Comm_size (MPI_COMM_WORLD, &ProcNum);
	MPI_Comm_rank (MPI_COMM_WORLD, &ProcRank);
	if(ProcRank == 0){
		printf("Hello world from process %i \n", ProcRank);
		for(int i = 1; i < ProcNum; i++){
			MPI_Recv(&tmp,1,MPI_INT,MPI_ANY_SOURCE,0,MPI_COMM_WORL
D, &status);
			printf("Hello world from process %i \n", tmp);
		}
	}else{
		MPI_Send(&ProcRank,1,MPI_INT,0,0,MPI_COMM_WORLD);
	}
	MPI_Finalize();
	return 0;
}
```

Двухточечные обмены

Пример программы, попадающей «в тупик»

```
program main_mpi
include 'mpif.h'
integer rank, tag, cnt, ierr, status(MPI_STATUS_SIZE)
real sndbuf, rcvbuf
tag = 0
sndbuf = 3.14159
cnt = 1
call MPI_Init(ierr)
call MPI_Comm_rank(MPI_COMM_WORLD, rank, ierr)
if (rank.eq.0) then
 call MPI_Recv(rcvbuf, cnt, MPI_REAL, 1, tag, MPI_COMM_WORLD, status, ierr)
 call MPI_Send(sndbuf, cnt, MPI_REAL, 1, tag, MPI_COMM_WORLD, ierr)
else
 call MPI_Recv(rcvbuf, cnt, MPI_REAL, 0, tag, MPI_COMM_WORLD, status, ierr)
 call MPI_Send(sndbuf, cnt, MPI_REAL, 0, tag, MPI_COMM_WORLD, ierr)
end if
call MPI_Finalize(ierr)
stop
end
```

Двухточечные обмены

Размер полученного сообщения (`count`) можно определить с помощью вызова подпрограммы

```
int MPI_Get_count(MPI_Status *status, MPI_Datatype datatype,
int *count)

MPI_Get_count(status, datatype, count, ierr)
```

- `count` - количество элементов в буфере передачи;
- `datatype` - тип MPI каждого пересылаемого элемента;
- `status` - статус обмена;
- `ierr` - код завершения.

Аргумент `datatype` должен соответствовать типу данных, указанному в операции обмена

Двухточечный обмен с буферизацией

Двухточечные обмены

Передача сообщения в буферизованном режиме может быть начата независимо от того, зарегистрирован ли соответствующий прием. Источник копирует сообщение в буфер, а затем передает его в неблокирующем режиме, так же как в стандартном режиме. Эта операция локальна, поскольку ее выполнение не зависит от наличия соответствующего приема. Если объем буфера недостаточен, возникает ошибка. Выделение буфера и его размер контролируются программистом.

Двухточечные обмены

Размер буфера должен превосходить размер сообщения на величину `MPI_BSEND_OVERHEAD`. Это дополнительное пространство используется подпрограммой буферизованной передачи для своих целей.

Если перед выполнением операции буферизованного обмена не выделен буфер, MPI ведет себя так, как если бы с процессом был связан буфер нулевого размера. Работа с таким буфером обычно завершается сбоем программы.

Буферизованный обмен рекомендуется использовать в тех ситуациях, когда программисту требуется больший контроль над распределением памяти. Этот режим удобен и для отладки, поскольку причину переполнения буфера определить легче, чем причину тупика.

Двухточечные обмены

При выполнении буферизованного обмена программист должен заранее создать буфер достаточного размера:

```
int MPI_Buffer_attach(void *buf, size)

MPI_Buffer_attach(buf, size, ierr)
```

В результате вызова создается буфер `buf` размером `size` байтов. В программах на языке Fortran роль буфера может играть массив. За один раз к процессу может быть подключен только один буфер.

Двухточечные обмены

Буферизованная передача завершается сразу, поскольку сообщение немедленно копируется в буфер для последующей передачи. В отличие от стандартного обмена, в этом случае работа источника и адресата не синхронизована:

```
int MPI_Bsend(void *buf, int count, MPI_Datatype datatype,
int   dest, int tag, MPI_Comm comm)

MPI_Bsend(buf, count, datatype, dest, tag, comm, ierr)
```

Двухточечные обмены

После завершения работы с буфером его необходимо отключить:

```
int MPI_Buffer_detach(void *buf, int *size)

MPI_Buffer_detach(buf, size, ierr)
```

Возвращается адрес (`buf`) и размер отключаемого буфера (`size`). Эта операция блокирует работу процесса до тех пор, пока все сообщения, находящиеся в буфере, не будут обработаны. Вызов данной подпрограммы можно использовать для форсированной передачи сообщений. После завершения вызова можно вновь использовать память, которую занимал буфер. В языке С данный вызов не освобождает автоматически память, отведенную для буфера.

Двухточечные обмены

Пример программы, использующей обмен с буферизацией

```
#include "mpi.h"
#include <stdio.h>
int main(int argc,char *argv[])
{
 int *buffer;
 int myrank;
 MPI_Status status;
 int buffsize = 1;
 int TAG = 0;
 MPI_Init(&argc, &argv);
 MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
 if (myrank == 0)
{
 buffer = (int *) malloc(buffsize + MPI_BSEND_OVERHEAD);
 MPI_Buffer_attach(buffer, buffsize + MPI_BSEND_OVERHEAD);
 buffer = (int *) 10;
 MPI_Bsend(&buffer, buffsize, MPI_INT, 1, TAG, MPI_COMM_WORLD);
 MPI_Buffer_detach(&buffer, &buffsize);
}
else
{
 MPI_Recv(&buffer, buffsize, MPI_INT, 0, TAG, MPI_COMM_WORLD, &status);
 printf("received: %i\n", buffer);
}
 MPI_Finalize();
 return 0;}
```

Другие разновидности двухточечного обмена

Двухточечные обмены

Синхронный обмен

Завершение передачи происходит тольк после того, как прием сообщения инициализирован другим процессом. Адресат посылает источнику «квитанцию» - уведомление о завершении приема. После получения этого уведомления обмен считается завершенным и источник "знает", что его сообщение получено:

```
int MPI_Ssend(void *buf, int count, MPI_Datatype
datatype, int dest, int tag, MPI_Comm comm)

MPI_SSEND(BUF, COUNT, DATATYPE, DEST, TAG, COMM, IERR)
```

Двухточечные обмены

Обмен «по готовности»

Передача «по готовности» выполняется с помощью подпрограммы `MPI_Rsend`:

```
 int MPI_Rsend(void *buf, int count, MPI_Datatype
datatype,
 int dest, int tag, MPI_Comm comm)

 MPI_Rsend(buf, count, datatype, dest, tag, comm, ierr)
```

Передача «по готовности» должна начинаться, если уже зарегистрирован соответствующий прием. При несоблюдении этого условия результат выполнения операции не определен.

Завершается она сразу же. Если прием не зарегистрирован, результат выполнения операции не определен. Завершение передачи не зависит от того, вызвана ли другим процессом подпрограмма приема данного сообщения или нет, оно означает только, что буфер передачи можно использовать вновь. Сообщение

Двухточечные обмены

Завершается операция обмена сразу же. Если прием не зарегистрирован, результат выполнения операции не определен. Завершение передачи не зависит от того, вызвана ли другим процессом подпрограмма приема данного сообщения или нет, оно означает только, что буфер передачи можно использовать вновь. Сообщение просто выбрасывается в коммуникационную сеть в надежде, что адресат его получит. Эта надежда может и не сбыться.

Двухточечные обмены

Обмен «по готовности» может увеличить производительность программы, поскольку здесь не используются этапы установки межпроцессных связей, а также буферизация. Все это — операции, требующие времени. С другой стороны, обмен «по готовности» потенциально опасен, кроме того, он усложняет отладку, поэтому его рекомендуется использовать только в том случае, когда правильная работа программы гарантируется ее логической структурой, а выигрыша в быстродействии надо добиться любой ценой.

Двухточечные обмены

Совместные прием и передача

Операции приемопередачи объединяют в едином вызове передачу сообщения одному процессу и прием сообщения от другого процесса. Данный вид обменов может оказаться полезным при выполнении сложных схем обмена сообщениями, например, по цепи процессов.

Подпрограммы приемопередачи могут взаимодействовать с обычными подпрограммами обмена и подпрограммами зондирования.

Двухточечные обмены

Подпрограмма `MPI_Sendrecv` выполняет прием и передачу данных с блокировкой:

```
int MPI_Sendrecv(void *sendbuf, int sendcount,
MPI_Datatype sendtype, int dest, int sendtag, void
*recvbuf, int recvcount, MPI_Datatype recvtype, int
source, int recvtag, MPI_Comm comm, MPI_Status *status)

MPI_Sendrecv(sendbuf, sendcount, sendtype, dest,
sendtag, recvbuf, recvcount, recvtype, source, recvtag,
comm, status, ierr)
```

Имеются разновидности операции приемопередачи.

Двухточечные обмены

Подпрограмма `MPI_Sendrecv_replace` выполняет прием и передачу данных, используя общий буфер для передачи и приёма:

```
int MPI_Sendrecv_replace(void *buf, int count,
MPI_Datatype datatype, int dest, int sendtag, int source,
int recvtag, MPI_Comm comm, MPI_Status *status)

MPI_Sendrecv_replace(BUF, COUNT, DATATYPE, DEST, SENDTAG,
SOURCE, RECVTAG, COMM, STATUS, IERR)
```

Заключение

В этой лекции мы рассмотрели:

- настройку рабочей среды для разработки и запуска MPI-программ в среде Microsoft Windows;
- организацию двухточечных обменов в MPI;
- проблемы, которые могут возникать при использовании двухточечных обменов.

Задания для самостоятельной работы

Выполнить настройку среды на своем рабочем месте для разработки и запуска MPI-программ.

Если в процессе установки будут возникать вопросы, можно обращаться по электронной почте:

parallel-g112@yandex.ru

Задания для самостоятельной работы

В исходном тексте программы на языке С пропущены списки параметров процедур двухточечного обмена MPI. Добавить эти параметры, откомпилировать и запустить программу.

```
#include "mpi.h"
#include <stdio.h>
int main(int argc,char *argv[])
{
 int myid, numprocs;
 char message[20];
 int myrank;
 MPI_Status status;
 int TAG = 0;
 MPI_Init(&argc, &argv);
 MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
 if (myrank == 0)
{
 strcpy(message, "Hi, Second Processor!");
 MPI_Send(...);
}
else
{
 MPI_Recv(...);
 printf("received: %s\n", message);
}
 MPI_Finalize();
 return 0;
}
```

Задания для самостоятельной работы

В исходном тексте программы на языке С предусмотрена некая схема обмена сообщениями между процессами параллельной программы. Определите схему обмена. В тексте пропущены списки параметров процедур двухточечного обмена MPI. Добавить эти параметры, откомпилировать и запустить программу.

```
#include "mpi.h"
#include <stdio.h>
int main(int argc,char *argv[])
{
int myrank, size, message;
int TAG = 0;
MPI_Status status;
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
MPI_Comm_size(MPI_COMM_WORLD, &size);
message = myrank;
if((myrank % 2) == 0)
{
 if((myrank + 1) != size)
 MPI_Send(...);
}
else
{
 if(myrank != 0)
 MPI_Recv(...);
 printf("received :%i\n", message);
}
MPI_Finalize();
return 0;
}
```

Задания для самостоятельной работы

Два вектора **a** и **b** размерности N представлены двумя одномерными массивами, содержащими каждый по N элементов. Напишите параллельную MPI-программу вычисления скалярного произведения этих векторов используя блокирующий двухточечный обмен сообщениями. Программа должна быть организована по схеме master-slave, причем master-процесс должен пересылать подчиненным процессам одинаковые (или почти одинаковые) по количеству элементов фрагменты векторов.

Если у вас имеется доступ к параллельному кластеру и есть возможность запускать на нем параллельные MPI-программы, проведите исследование зависимости ускорения параллельной программы от размера сообщения.

Сделайте то же самое для других вариантов блокирующих обменов.

Тема следующей лекции

Неблокирующие двухточечные обмены в MPI