**MPI. Терминология и обозначения**

*MPI - message passing interface* - библиотека функций, предназначенная для поддержки работы параллельных процессов в терминах передачи сообщений.

*Номер процесса* - целое неотрицательное число, являющееся уникальным атрибутом каждого процесса.

*Атрибуты сообщения* - номер процесса-отправителя, номер процесса-получателя и идентификатор сообщения. Для них заведена структура *MPI\_Status*, содержащая три поля: *MPI\_Source* (номер процесса отправителя), *MPI\_Tag* (идентификатор сообщения), *MPI\_Error* (код ошибки); могут быть и добавочные поля.

*Идентификатор сообщения (msgtag)* - атрибут сообщения, являющийся целым неотрицательным числом, лежащим в диапазоне от 0 до 32767.   
Процессы объединяются в *группы*, могут быть вложенные группы. Внутри группы все процессы перенумерованы. С каждой группой ассоциирован свой *коммуникатор*. Поэтому при осуществлении пересылки необходимо указать идентификатор группы, внутри которой производится эта пересылка. Все процессы содержатся в группе с предопределенным идентификатором *MPI\_COMM\_WORLD*.

При описании процедур MPI будем пользоваться словом OUT для обозначения "выходных" параметров, т.е. таких параметров, через которые процедура возвращает результаты.

**Общие процедуры MPI**

**int MPI\_Init( int\* argc, char\*\*\* argv)**

*MPI\_Init* - инициализация параллельной части приложения. Реальная инициализация для каждого приложения выполняется не более одного раза, а если MPI уже был инициализирован, то никакие действия не выполняются и происходит немедленный возврат из подпрограммы. Все оставшиеся MPI-процедуры могут быть вызваны только после вызова *MPI\_Init*.

Возвращает: в случае успешного выполнения - *MPI\_SUCCESS*, иначе - код ошибки. (То же самое возвращают и все остальные функции, рассматриваемые в данном руководстве.)

**int MPI\_Finalize( void )**

*MPI\_Finalize* - завершение параллельной части приложения. Все последующие обращения к любым MPI-процедурам, в том числе к *MPI\_Init*, запрещены. К моменту вызова *MPI\_Finalize* некоторым процессом все действия, требующие его участия в обмене сообщениями, должны быть завершены.   
Сложный тип аргументов *MPI\_Init* предусмотрен для того, чтобы передавать всем процессам аргументы *main*:

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

{

MPI\_Init(&argc, &argv);

...

MPI\_Finalize();

}

**int MPI\_Comm\_size( MPI\_Comm comm, int\* size)**

Определение общего числа параллельных процессов в группе *comm*.

* *comm* - идентификатор группы
* OUT *size* - размер группы

**int MPI\_Comm\_rank( MPI\_Comm comm, int\* rank)**

Определение номера процесса в группе *comm*. Значение, возвращаемое по адресу *&rank*, лежит в диапазоне от 0 до *size\_of\_group-1*.

* *comm* - идентификатор группы
* OUT *rank* - номер вызывающего процесса в группе *comm*

**double MPI\_Wtime(void)**

Функция возвращает астрономическое время в секундах (вещественное число), прошедшее с некоторого момента в прошлом. Гарантируется, что этот момент не будет изменен за время существования процесса.

**Прием/передача сообщений между отдельными процессами**

**Прием/передача сообщений с блокировкой**

**int MPI\_Send(void\* buf, int count, MPI\_Datatype datatype, int dest, int msgtag, MPI\_Comm comm)**

* *buf* - адрес начала буфера посылки сообщения
* *count* - число передаваемых элементов в сообщении
* *datatype* - тип передаваемых элементов
* *dest* - номер процесса-получателя
* *msgtag* - идентификатор сообщения
* *comm* - идентификатор группы

Блокирующая посылка сообщения с идентификатором *msgtag*, состоящего из *count* элементов типа *datatype*, процессу с номером *dest*. Все элементы сообщения расположены подряд в буфере *buf*. Значение *count* может быть нулем. Тип передаваемых элементов *datatype* должен указываться с помощью предопределенных констант типа. Разрешается передавать сообщение самому себе.

Блокировка гарантирует корректность повторного использования всех параметров после возврата из подпрограммы. Выбор способа осуществления этой гарантии: копирование в промежуточный буфер или непосредственная передача процессу *dest*, остается за MPI. Следует специально отметить, что возврат из подпрограммы *MPI\_Send* не означает ни того, что сообщение уже передано процессу *dest*, ни того, что сообщение покинуло процессорный элемент, на котором выполняется процесс, выполнивший *MPI\_Send*.

**int MPI\_Recv(void\* buf, int count, MPI\_Datatype datatype, int source, int msgtag, MPI\_Comm comm, MPI\_Status \*status)**

* OUT *buf* - адрес начала буфера приема сообщения
* *count* - максимальное число элементов в принимаемом сообщении
* *datatype* - тип элементов принимаемого сообщения
* *source* - номер процесса-отправителя
* *msgtag* - идентификатор принимаемого сообщения
* *comm* - идентификатор группы
* OUT *status* - параметры принятого сообщения

Прием сообщения с идентификатором *msgtag* от процесса *source* с блокировкой. Число элементов в принимаемом сообщении не должно превосходить значения *count*. Если число принятых элементов меньше значения *count*, то гарантируется, что в буфере *buf* изменятся только элементы, соответствующие элементам принятого сообщения. Если нужно узнать точное число элементов в сообщении, то можно воспользоваться подпрограммой *MPI\_Probe*.

Блокировка гарантирует, что после возврата из подпрограммы все элементы сообщения приняты и расположены в буфере *buf*.

В качестве номера процесса-отправителя можно указать предопределенную константу *MPI\_ANY\_SOURCE* - признак того, что подходит сообщение от любого процесса. В качестве идентификатора принимаемого сообщения можно указать константу *MPI\_ANY\_TAG* - признак того, что подходит сообщение с любым идентификатором.

Если процесс посылает два сообщения другому процессу и оба эти сообщения соответствуют одному и тому же вызову *MPI\_Recv*, то первым будет принято то сообщение, которое было отправлено раньше.

**int MPI\_Get\_count( MPI\_Status \*status, MPI\_Datatype datatype, int \*count)**

* *status* - параметры принятого сообщения
* *datatype* - тип элементов принятого сообщения
* OUT *count* - число элементов сообщения

По значению параметра *status* данная подпрограмма определяет число уже принятых (после обращения к *MPI\_Recv*) или принимаемых (после обращения к *MPI\_Probe* или *MPI\_Iprobe*) элементов сообщения типа *datatype*.

**int MPI\_Probe( int source, int msgtag, MPI\_Comm comm, MPI\_Status \*status)**

* *source* - номер процесса-отправителя или *MPI\_ANY\_SOURCE*
* *msgtag* - идентификатор ожидаемого сообщения или *MPI\_ANY\_TAG*
* *comm* - идентификатор группы
* OUT *status* - параметры обнаруженного сообщения

Получение информации о структуре ожидаемого сообщения с блокировкой. Возврата из подпрограммы не произойдет до тех пор, пока сообщение с подходящим идентификатором и номером процесса-отправителя не будет доступно для получения. Атрибуты доступного сообщения можно определить обычным образом с помощью параметра *status*. Следует обратить внимание, что подпрограмма определяет только факт прихода сообщения, но реально его не принимает.

**Прием/передача сообщений без блокировки**

**int MPI\_Isend(void \*buf, int count, MPI\_Datatype datatype, int dest, int msgtag, MPI\_Comm comm, MPI\_Request \*request)**

* *buf* - адрес начала буфера посылки сообщения
* *count* - число передаваемых элементов в сообщении
* *datatype* - тип передаваемых элементов
* *dest* - номер процесса-получателя
* *msgtag* - идентификатор сообщения
* *comm* - идентификатор группы
* OUT *request* - идентификатор асинхронной передачи

Передача сообщения, аналогичная *MPI\_Send*, однако возврат из подпрограммы происходит сразу после инициализации процесса передачи без ожидания обработки всего сообщения, находящегося в буфере *buf*. Это означает, что нельзя повторно использовать данный буфер для других целей без получения дополнительной информации о завершении данной посылки. Окончание процесса передачи (т.е. того момента, когда можно переиспользовать буфер *buf* без опасения испортить передаваемое сообщение) можно определить с помощью параметра *request* и процедур *MPI\_Wait* и *MPI\_Test*.   
Сообщение, отправленное любой из процедур *MPI\_Send* и *MPI\_Isend*, может быть принято любой из процедур *MPI\_Recv* и *MPI\_Irecv*.

**int MPI\_Irecv(void \*buf, int count, MPI\_Datatype datatype, int source, int msgtag, MPI\_Comm comm, MPI\_Request \*request)**

* OUT *buf* - адрес начала буфера приема сообщения
* *count* - максимальное число элементов в принимаемом сообщении
* *datatype* - тип элементов принимаемого сообщения
* *source* - номер процесса-отправителя
* *msgtag* - идентификатор принимаемого сообщения
* *comm* - идентификатор группы
* OUT *request* - идентификатор асинхронного приема сообщения

Прием сообщения, аналогичный *MPI\_Recv*, однако возврат из подпрограммы происходит сразу после инициализации процесса приема без ожидания получения сообщения в буфере *buf*. Окончание процесса приема можно определить с помощью параметра *request* и процедур *MPI\_Wait* и *MPI\_Test*.

**int MPI\_Wait( MPI\_Request \*request, MPI\_Status \*status)**

* *request* - идентификатор асинхронного приема или передачи
* OUT *status* - параметры сообщения

Ожидание завершения асинхронных процедур *MPI\_Isend* или *MPI\_Irecv*, ассоциированных с идентификатором *request*. В случае приема, атрибуты и длину полученного сообщения можно определить обычным образом с помощью параметра *status*.

**int MPI\_Waitall( int count, MPI\_Request \*requests, MPI\_Status \*statuses)**

* *count* - число идентификаторов
* *requests* - массив идентификаторов асинхронного приема или передачи
* OUT *statuses* - параметры сообщений

Выполнение процесса блокируется до тех пор, пока все операции обмена, ассоциированные с указанными идентификаторами, не будут завершены. Если во время одной или нескольких операций обмена возникли ошибки, то поле ошибки в элементах массива *statuses* будет установлено в соответствующее значение.

**int MPI\_Waitany( int count, MPI\_Request \*requests, int \*index, MPI\_Status \*status)**

* *count* - число идентификаторов
* *requests* - массив идентификаторов асинхронного приема или передачи
* OUT *index* - номер завершенной операции обмена
* OUT *status* - параметры сообщений

Выполнение процесса блокируется до тех пор, пока какая-либо операция обмена, ассоциированная с указанными идентификаторами, не будет завершена. Если несколько операций могут быть завершены, то случайным образом выбирается одна из них. Параметр *index* содержит номер элемента в массиве *requests*, содержащего идентификатор завершенной операции.

**int MPI\_Waitsome( int incount, MPI\_Request \*requests, int \*outcount, int \*indexes, MPI\_Status \*statuses)**

* *incount* - число идентификаторов
* *requests* - массив идентификаторов асинхронного приема или передачи
* OUT *outcount* - число идентификаторов завершившихся операций обмена
* OUT *indexes* - массив номеров завершившихся операции обмена
* OUT *statuses* - параметры завершившихся сообщений

Выполнение процесса блокируется до тех пор, пока по крайней мере одна из операций обмена, ассоциированных с указанными идентификаторами, не будет завершена. Параметр *outcount* содержит число завершенных операций, а первые *outcount* элементов массива *indexes* содержат номера элементов массива *requests* с их идентификаторами. Первые *outcount* элементов массива *statuses* содержат параметры завершенных операций.

**int MPI\_Test( MPI\_Request \*request, int \*flag, MPI\_Status \*status)**

* *request* - идентификатор асинхронного приема или передачи
* OUT *flag* - признак завершенности операции обмена
* OUT *status* - параметры сообщения

Проверка завершенности асинхронных процедур *MPI\_Isend* или *MPI\_Irecv*, ассоциированных с идентификатором *request*. В параметре *flag* возвращает значение 1, если соответствующая операция завершена, и значение 0 в противном случае. Если завершена процедура приема, то атрибуты и длину полученного сообщения можно определить обычным образом с помощью параметра *status*.

**int MPI\_Testall( int count, MPI\_Request \*requests, int \*flag, MPI\_Status \*statuses)**

* *count* - число идентификаторов
* *requests* - массив идентификаторов асинхронного приема или передачи
* OUT *flag* - признак завершенности операций обмена
* OUT *statuses* - параметры сообщений

В параметре *flag* возвращает значение *1*, если все операции, ассоциированные с указанными идентификаторами, завершены (с указанием параметров сообщений в массиве *statuses*). В противном случае возвращается *0*, а элементы массива *statuses* неопределены.

**int MPI\_Testany(int count, MPI\_Request \*requests, int \*index, int \*flag, MPI\_Status \*status)**

* *count* - число идентификаторов
* *requests* - массив идентификаторов асинхронного приема или передачи
* OUT *index* - номер завершенной операции обмена
* OUT *flag* - признак завершенности операции обмена
* OUT *status* - параметры сообщения

Если к моменту вызова подпрограммы хотя бы одна из операций обмена завершилась, то в параметре *flag* возвращается значение *1*, *index* содержит номер соответствующего элемента в массиве *requests*, а *status* - параметры сообщения.

**int MPI\_Testsome( int incount, MPI\_Request \*requests, int \*outcount, int \*indexes, MPI\_Status \*statuses)**

* *incount* - число идентификаторов
* *requests* - массив идентификаторов асинхронного приема или передачи
* OUT *outcount* - число идентификаторов завершившихся операций обмена
* OUT *indexes* - массив номеров завершившихся операции обмена
* OUT *statuses* - параметры завершившихся операций

Данная подпрограмма работает так же, как и *MPI\_Waitsome*, за исключением того, что возврат происходит немедленно. Если ни одна из указанных операций не завершилась, то значение *outcount* будет равно нулю.

**int MPI\_Iprobe( int source, int msgtag, MPI\_Comm comm, int \*flag, MPI\_Status \*status)**

* *source* - номер процесса-отправителя или *MPI\_ANY\_SOURCE*
* *msgtag* - идентификатор ожидаемого сообщения или *MPI\_ANY\_TAG*
* *comm* - идентификатор группы
* OUT *flag* - признак завершенности операции обмена
* OUT *status* - параметры обнаруженного сообщения

Получение информации о поступлении и структуре ожидаемого сообщения без блокировки. В параметре *flag* возвращает значение *1*, если сообщение с подходящими атрибутами уже может быть принято (в этом случае ее действие полностью аналогично *MPI\_Probe*), и значение *0*, если сообщения с указанными атрибутами еще нет.

**Объединение запросов на взаимодействие**

Процедуры данной группы позволяют снизить накладные расходы, возникающие в рамках одного процессора при обработке приема/передачи и перемещении необходимой информации между процессом и сетевым контроллером. Несколько запросов на прием и/или передачу могут объединяться вместе для того, чтобы далее их можно было бы запустить одной командой. Способ приема сообщения никак не зависит от способа его посылки: сообщение, отправленное с помощью объединения запросов либо обычным способом, может быть принято как обычным способом, так и с помощью объединения запросов.

**int MPI\_Send\_init( void \*buf, int count, MPI\_Datatype datatype, int dest, int msgtag, MPI\_Comm comm, MPI\_Request \*request)**

* *buf* - адрес начала буфера посылки сообщения
* *count* - число передаваемых элементов в сообщении
* *datatype* - тип передаваемых элементов
* *dest* - номер процесса-получателя
* *msgtag* - идентификатор сообщения
* *comm* - идентификатор группы
* OUT *request* - идентификатор асинхронной передачи

Формирование запроса на выполнение пересылки данных. Все параметры точно такие же, как и у подпрограммы *MPI\_Isend*, однако в отличие от нее пересылка не начинается до вызова подпрограммы *MPI\_Startall*.

**int MPI\_Recv\_init( void \*buf, int count, MPI\_Datatype datatype, int source, int msgtag, MPI\_Comm comm, MPI\_Request \*request)**

* OUT *buf* - адрес начала буфера приема сообщения
* *count* - число принимаемых элементов в сообщении
* *datatype* - тип принимаемых элементов
* *source* - номер процесса-отправителя
* *msgtag* - идентификатор сообщения
* *comm* - идентификатор группы
* OUT *request* - идентификатор асинхронного приема

Формирование запроса на выполнение приема данных. Все параметры точно такие же, как и у подпрограммы *MPI\_Irecv*, однако в отличие от нее реальный прием не начинается до вызова подпрограммы *MPI\_Startall*.

**MPI\_Startall( int count, MPI\_Request \*requests)**

* *count* - число запросов на взаимодействие
* OUT *requests* - массив идентификаторов приема/передачи

Запуск всех отложенных взаимодействий, ассоциированных вызовами подпрограмм *MPI\_Send\_init* и *MPI\_Recv\_init* с элементами массива запросов *requests*. Все взаимодействия запускаются в режиме без блокировки, а их завершение можно определить обычным образом с помощью процедур *MPI\_Wait* и *MPI\_Test*.

**Совмещенные прием/передача сообщений**

**int MPI\_Sendrecv( void \*sbuf, int scount, MPI\_Datatype stype, int dest, int stag, void \*rbuf, int rcount, MPI\_Datatype rtype, int source, MPI\_Datatype rtag, MPI\_Comm comm, MPI\_Status \*status)**

* *sbuf* - адрес начала буфера посылки сообщения
* *scount* - число передаваемых элементов в сообщении
* *stype* - тип передаваемых элементов
* *dest* - номер процесса-получателя
* *stag* - идентификатор посылаемого сообщения
* OUT *rbuf* - адрес начала буфера приема сообщения
* *rcount* - число принимаемых элементов сообщения
* *rtype* - тип принимаемых элементов
* *source* - номер процесса-отправителя
* *rtag* - идентификатор принимаемого сообщения
* *comm* - идентификатор группы
* OUT *status* - параметры принятого сообщения

Данная операция объединяет в едином запросе посылку и прием сообщений. Принимающий и отправляющий процессы могут являться одним и тем же процессом. Сообщение, отправленное операцией *MPI\_Sendrecv*, может быть принято обычным образом, и точно также операция *MPI\_Sendrecv* может принять сообщение, отправленное обычной операцией *MPI\_Send*. Буфера приема и посылки обязательно должны быть различными.

**Коллективные взаимодействия процессов**

В операциях коллективного взаимодействия процессов участвуют все процессы коммуникатора. Соответствующая процедура должна быть вызвана каждым процессом, быть может, со своим набором параметров. Возврат из процедуры коллективного взаимодействия может произойти в тот момент, когда участие процесса в данной операции уже закончено. Как и для блокирующих процедур, возврат означает то, что разрешен свободный доступ к буферу приема или посылки, но не означает ни того, что операция завершена другими процессами, ни даже того, что она ими начата (если это возможно по смыслу операции).

**int MPI\_Bcast(void \*buf, int count, MPI\_Datatype datatype, int source, MPI\_Comm comm)**

* OUT *buf* - адрес начала буфера посылки сообщения
* *count* - число передаваемых элементов в сообщении
* *datatype* - тип передаваемых элементов
* *source* - номер рассылающего процесса
* *comm* - идентификатор группы

Рассылка сообщения от процесса *source* всем процессам, включая рассылающий процесс. При возврате из процедуры содержимое буфера *buf* процесса *source* будет скопировано в локальный буфер процесса. Значения параметров *count*, *datatype* и *source* должны быть одинаковыми у всех процессов.

**int MPI\_Gather( void \*sbuf, int scount, MPI\_Datatype stype, void \*rbuf, int rcount, MPI\_Datatype rtype, int dest, MPI\_Comm comm)**

* *sbuf* - адрес начала буфера посылки
* *scount* - число элементов в посылаемом сообщении
* *stype* - тип элементов отсылаемого сообщения
* OUT *rbuf* - адрес начала буфера сборки данных
* *rcount* - число элементов в принимаемом сообщении
* *rtype* - тип элементов принимаемого сообщения
* *dest* - номер процесса, на котором происходит сборка данных
* *comm* - идентификатор группы
* OUT *ierror* - код ошибки

Сборка данных со всех процессов в буфере *rbuf* процесса *dest*. Каждый процесс, включая *dest*, посылает содержимое своего буфера *sbuf* процессу *dest*. Собирающий процесс сохраняет данные в буфере *rbuf*, располагая их в порядке возрастания номеров процессов. Параметр *rbuf* имеет значение только на собирающем процессе и на остальных игнорируется, значения параметров *count*, *datatype* и *dest* должны быть одинаковыми у всех процессов.

**int MPI\_Allreduce( void \*sbuf, void \*rbuf, int count, MPI\_Datatype datatype, MPI\_Op op, MPI\_Comm comm)**

* *sbuf* - адрес начала буфера для аргументов
* OUT *rbuf* - адрес начала буфера для результата
* *count* - число аргументов у каждого процесса
* *datatype* - тип аргументов
* *op* - идентификатор глобальной операции
* *comm* - идентификатор группы

Выполнение *count* глобальных операций *op* с возвратом *count* результатов во всех процессах в буфере *rbuf*. Операция выполняется независимо над соответствующими аргументами всех процессов. Значения параметров *count* и *datatype* у всех процессов должны быть одинаковыми. Из соображений эффективности реализации предполагается, что операция *op* обладает свойствами ассоциативности и коммутативности.

**int MPI\_Reduce( void \*sbuf, void \*rbuf, int count, MPI\_Datatype datatype, MPI\_Op op, int root, MPI\_Comm comm)**

* *sbuf* - адрес начала буфера для аргументов
* OUT *rbuf* - адрес начала буфера для результата
* *count* - число аргументов у каждого процесса
* *datatype* - тип аргументов
* *op* - идентификатор глобальной операции
* *root* - процесс-получатель результата
* *comm* - идентификатор группы

Функция аналогична предыдущей, но результат будет записан в буфер *rbuf* только у процесса *root*.

**Синхронизация процессов**

**int MPI\_Barrier( MPI\_Comm comm)**

* *comm* - идентификатор группы

Блокирует работу процессов, вызвавших данную процедуру, до тех пор, пока все оставшиеся процессы группы *comm* также не выполнят эту процедуру.

**Работа с группами процессов**

**int MPI\_Comm\_split( MPI\_Comm comm, int color, int key, MPI\_Comm \*newcomm)**

* *comm* - идентификатор группы
* *color* - признак разделения на группы
* *key* - параметр, определяющий нумерацию в новых группах
* OUT *newcomm* - идентификатор новой группы

Данная процедура разбивает все множество процессов, входящих в группу *comm*, на непересекающиеся подгруппы - одну подгруппу на каждое значение параметра *color* (неотрицательное число). Каждая новая подгруппа содержит все процессы одного цвета. Если в качестве *color* указано значение *MPI\_UNDEFINED*, то в *newcomm* будет возвращено значение *MPI\_COMM\_NULL*.

**int MPI\_Comm\_free( MPI\_Comm comm)**

* OUT *comm* - идентификатор группы

Уничтожает группу, ассоциированную с идентификатором *comm*, который после возвращения устанавливается в *MPI\_COMM\_NULL*.

**Предопределенные константы**

**Предопределенные константы типа элементов сообщений**

|  |
| --- |
|  |
| Константы MPI | Тип в C |
|  |  |
| *MPI\_CHAR* | signed char |
| *MPI\_SHORT* | signed int |
| *MPI\_INT* | signed int |
| *MPI\_LONG* | signed long int |
| *MPI\_UNSIGNED\_CHAR* | unsigned char |
| *MPI\_UNSIGNED\_SHORT* | unsigned int |
| *MPI\_UNSIGNED* | unsigned int |
| *MPI\_UNSIGNED\_LONG* | unsigned long int |
| *MPI\_FLOAT* | float |
| *MPI\_DOUBLE* | double |
| *MPI\_LONG\_DOUBLE* | long double |

**Другие предопределенные типы**

*MPI\_Status* - структура; атрибуты сообщений; содержит три обязательных поля:

* *MPI\_Source* (номер процесса отправителя)
* *MPI\_Tag* (идентификатор сообщения)
* *MPI\_Error* (код ошибки)

*MPI\_Request* - системный тип; идентификатор операции посылки-приема сообщения

*MPI\_Comm* - системный тип; идентификатор группы (коммуникатора)

* *MPI\_COMM\_WORLD* - зарезервированный идентификатор группы, состоящей их всех процессов приложения

**Константы-пустышки**

* *MPI\_COMM\_NULL*
* *MPI\_DATATYPE\_NULL*
* *MPI\_REQUEST\_NULL*

**Константа неопределенного значения**

* *MPI\_UNDEFINED*

**Глобальные операции**

*MPI\_MAX*

*MPI\_MIN*

*MPI\_SUM*

*MPI\_PROD*

**Любой процесс/идентификатор**

*MPI\_ANY\_SOURCE*

*MPI\_ANY\_TAG*

**Код успешного завершения процедуры**

*MPI\_SUCCESS*