

# ANY\_SOURCE и ANY\_TAG

При **ПРИЕМЕ** сообщения вместо номера передающего и идентификатора сообщения можно использовать следующие predefined константы:

- MPI\_ANY\_SOURCE // подходит сообщение от любого процесса
- MPI\_ANY\_TAG // подходит сообщение с любым идентификатором

С помощью констант MPI\_ANY\_SOURCE и MPI\_ANY\_TAG можно принять сообщение от любого процесса с любым идентификатором.

Однако, при посылке необходимо обязательно указывать адресат и номер сообщения!

# Немого о Status и еще кой-о-чем

Status – это структура, содержащая 3 поля:

- Status.MPI\_SOURCE // номер процесса-отправителя
- Status.MPI\_TAG // идентификатор сообщения
- Status.MPI\_ERROR // код ошибки

При пересылке можно использовать специальное значение MPI\_PROC\_NULL для несуществующего процесса. На самом деле приема/пересылки при работе с таким процессом не происходит. Пример:

```
next = myrank + 1; // а если next == size?  
if (next == size) next = MPI_PROC_NULL; // !!!
```

# Не получая сообщение...

Можно, не принимая сообщение, получить информацию о его атрибутах.

```
MPI_Probe (MPI_ANY_SOURCE, tag, MPI_COMM_WORLD,  
&Status);    // записывает в структуру Status параметры  
принимаемого сообщения
```

```
int count;  
MPI_Get_Count (&Status, MPI_INT, &count); // по значению  
полей Status, определяет число принятых (вызов после  
MPI_Recv) или принимаемых (вызов после MPI_Probe)  
элементов соответствующего сообщения
```

**Сообщение не принимается!** Если сразу после MPI\_Probe и вызвать MPI\_Recv, то примется то же самое сообщение

# О коллективных операциях 1

Операция барьерной синхронизации процессов

```
MPI_Barrier (MPI_COMM_WORLD);    // работа процессов  
останавливается, пока все не выполнят эту операцию
```

Пересылка всем от одного:

```
MPI_Bcast (array, 100, MPI_INT, 0, MPI_COMM_WORLD);  
// пересылка от процесса 0 всем остальным (включая  
сам процесс 0) 100 элементов типа int массива array
```

**Этот вызов должен быть во всех процессах!**

# О коллективных операциях 2

```
MPI_Gather (array, 10, MPI_INT, buf, 10, MPI_INT, 0,  
MPI_COMM_WORLD); // сбор информации
```

процесс 0 собирает в массив buf по 10 элементов массивов array из каждого процесса (включая сам процесс 0).  
Записи размещаются в порядке возраст. номеров процессов.

**ЭТОТ ВЫЗОВ ДОЛЖЕН БЫТЬ ВО ВСЕХ ПРОЦЕССАХ!**

```
MPI_Scatter (array, 10, MPI_INT, buf, 10, MPI_INT, 0,  
MPI_COMM_WORLD); // рассылка информации
```

процесс 0 рассылает по 10 элементов массива array в массивы buf всех процессов (включая сам процесс 0).  
Array делится на равные части, i-я часть посылается (i-1)-у процессу

# О коллективных операциях 3

```
MPI_Reduce (array, buf, 10, MPI_INT, MPI_SUM, 0,  
MPI_COMM_WORLD);
```

// процесс 0 собирает в i-ый элемент массива buf результат коллективной операции (MPI\_SUM) над i-ыми элементами массива array в каждом из процессов (включая сам процесс 0)

**Этот вызов должен быть во всех процессах!**

```
MPI_Allreduce(array, buf, 10, MPI_INT, MPI_SUM,  
MPI_COMM_WORLD); // рассылка информации
```

То же самое, что и ф-ия MPI\_Reduce, только результат записывается в массив buf каждого из процессов!

**Этот вызов должен быть во всех процессах!**