

### Задание 13. Коммуникации «точка-точка»: схема «эстафетная палочка»

Напишите MPI-программу, реализующую при помощи блокирующих функций послыки сообщений типа точка-точка схему коммуникации процессов «эстафетная палочка», в которой каждый процесс дожидается сообщения от предыдущего и потом посылает следующему (см. рис. 1). В качестве передаваемого сообщения используйте на процессе 0 его номер, на остальных процессах – инкрементированное полученное сообщение.

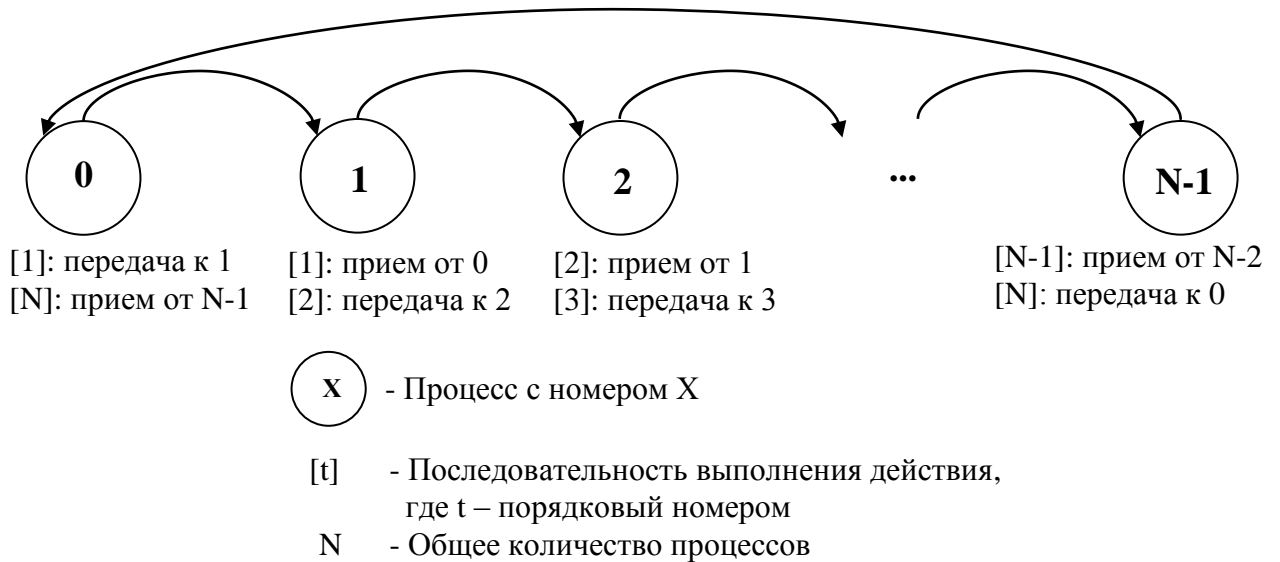


Рис. 1. Схема коммуникации процессов «эстафетная палочка»

**Входные данные:** нет.

**Выходные данные:** «[<номер\_процесса>]: receive message  
'<сообщение>'».

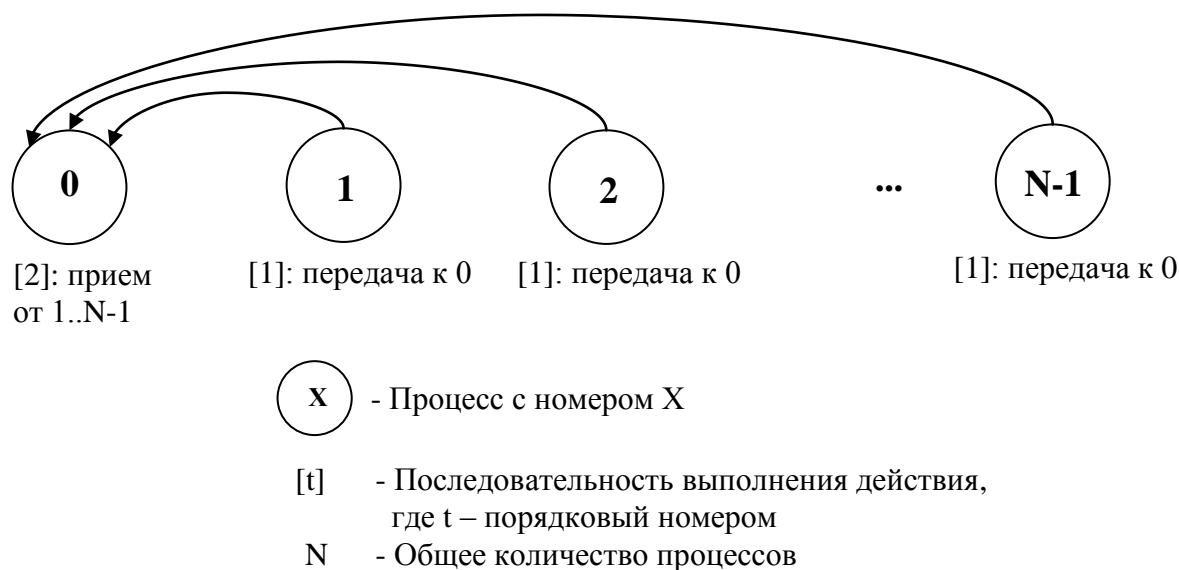
**Пример входных и выходных данных для 4-х процессов**

Входные данные	Выходные данные
	[0]: receive message '3' [1]: receive message '0' [2]: receive message '1' [3]: receive message '2'

### Задание 14. Коммуникации «точка-точка»: схема «мастер-рабочие»

Напишите MPI-программу, реализующую при помощи блокирующих функций послыки сообщений типа точка-точка схему коммуникации процессов «master-slave», в которой один процесс, называемый master<sup>2</sup>, принимает сообщение от остальных процессов, называемых slave (см. рис. 2). В качестве передаваемого сообщения используйте номер процесса. Master-процесс должен вывести на экран все полученные сообщения.

<sup>2</sup> Master-процессом может быть любой произвольный процесс, обычно это процесс с номером 0.



**Рис. 2.** Схема коммуникации процессов «master-slave»

**Входные данные:** нет.

**Выходные данные:** «receive message '<сообщение>' from <номер\_процесса>».

**Пример входных и выходных данных для 4-х процессов**

Входные данные	Выходные данные
	receive message '1'
	receive message '2'
	receive message '3'

### **Задание 15. Коммуникации «точка-точка»: схема «сдвиг по кольцу»**

Напишите MPI-программу, реализующую при помощи блокирующих функций послылки сообщений типа точка-точка схему коммуникации процессов «сдвиг по кольцу», в которой осуществляются одновременные послылка и прием сообщений всеми процессами (см. рис. 3). В качестве передаваемого сообщения используйте номер процесса.

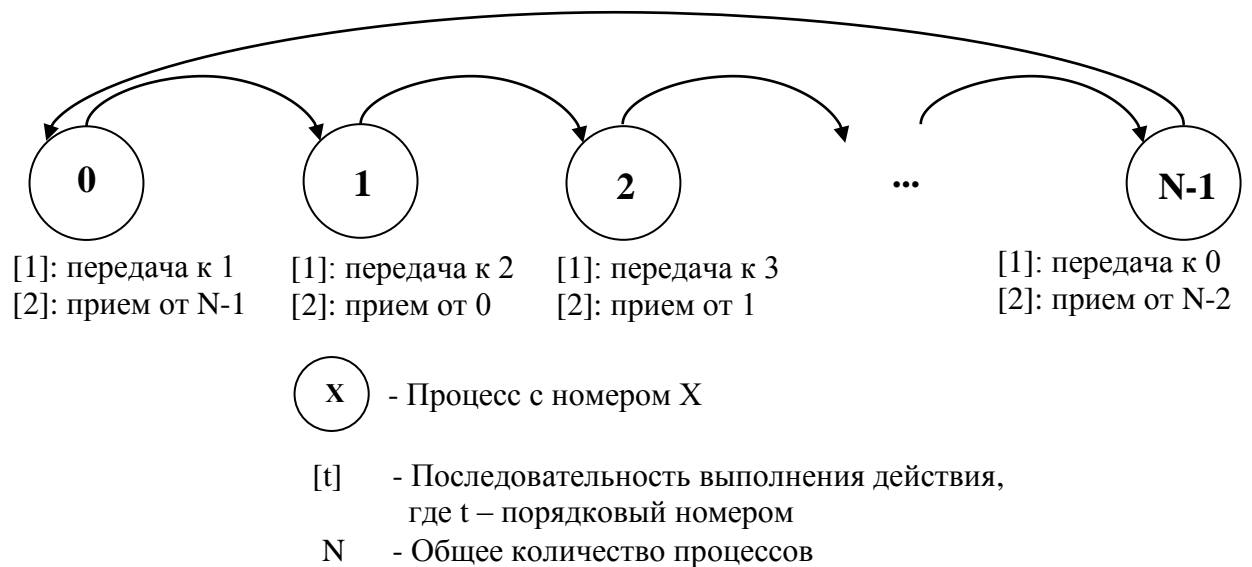


Рис. 3. Схема коммуникации процессов «сдвиг по кольцу»

**Входные данные:** нет.

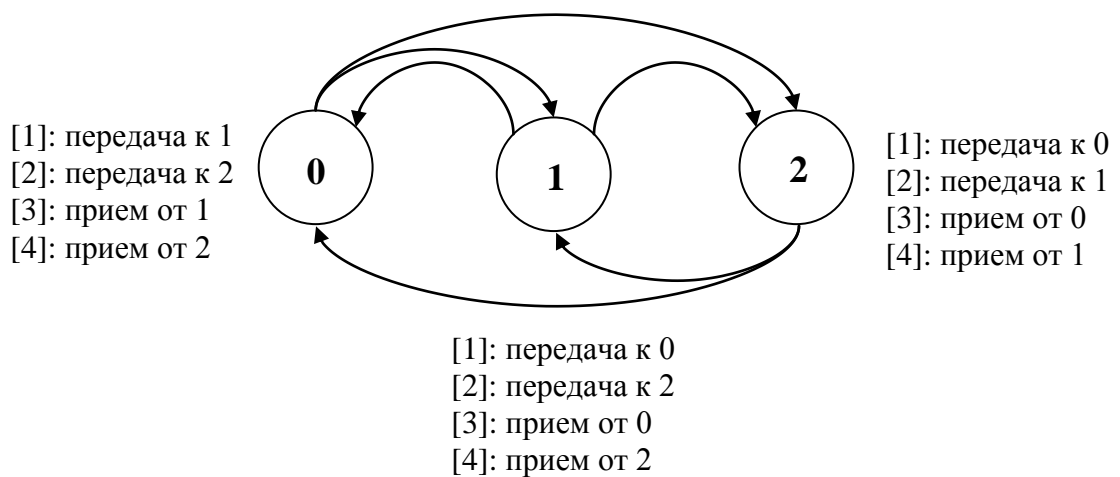
**Выходные данные:** «[<номер\_процесса>]: receive message  
'<сообщение>'».

**Пример входных и выходных данных для 4-х процессов**

Входные данные	Выходные данные
	[0]: receive message '3'
	[1]: receive message '0'
	[2]: receive message '1'
	[3]: receive message '2'

### **Задание 16. Коммуникации «точка-точка»: схема «каждый каждому»**

Напишите MPI-программу, реализующую при помощи блокирующих функций послыки сообщений типа точка-точка схему коммуникации процессов «каждый каждому», в которой осуществляется пересылка сообщения от каждого процесса каждому (см. рис. 4). В качестве передаваемого сообщения используйте номер процесса. Каждый процесс должен вывести на экран все полученные сообщения.



⊙  
X - Процесс с номером X

[t] - Последовательность выполнения действия,  
где t – порядковый номер

**Рис. 4.** Схема коммуникации процессов «каждый каждому»  
на примере 3-х процессов

**Входные данные:** нет.

**Выходные данные:** каждый процесс выводит сообщение

«[<номер\_процесса>]: receive message '<сообщение>' from <номер\_процесса>».

**Пример входных и выходных данных для 3-х процессов**

Входные данные	Выходные данные
	[0]: receive message '1' from 1 [0]: receive message '2' from 2 [1]: receive message '0' from 0 [1]: receive message '2' from 2 [2]: receive message '0' from 0 [2]: receive message '1' from 1

### Указания к заданию 13. Коммуникации «точка-точка»: схема «эстафетная палочка»

1. Создайте проект `mpi_baton` в Microsoft Visual Studio 2010 с поддержкой MPI (см. указания к заданию 10).
2. Вставьте в код программы вызовы функций `MPI_Init` и `MPI_Finalize`.
3. Запишите номер каждого процесса в переменную `rank`.
4. Запишите количество параллельных процессов в программе в переменную `size`.
5. Создайте сообщение, объявив целочисленную переменную `buf`.
6. В схеме коммуникации процессов «эстафетная палочка» для обеспечения последовательной от процесса к процессу передачи сообщения («эстафетной палочки»), процесс должен сначала дождаться получения сообщения, а затем пересылать его следующему процессу. Но все процессы не могут начать с вызова операции получения сообщения, т.к. в случае использования блокирующих операций `MPI_Send` и `MPI_Recv` возникнет ситуация **тупика (deadlock)**, при которой все процессы будут простаивать и программа никогда не завершится. Соответственно выделяют процесс, который инициализирует передачу сообщения, т.е. первым действием выполняет операцию отправки сообщения – это процесс с номером 0. Реализовать это можно следующим образом:
  - С помощью оператора `if` выделите в программе две секции кода: для процесса с номером 0 и для остальных процессов:

```
if (rank == 0) {  
    // Код, выполняемый процессом 0  
}  
else {  
    // Код, выполняемый остальными процессами  
}
```
  - В секции для процесса 0 присвойте переменной `buf` значение 0. С помощью `MPI_Send` отправьте переменную `buf` процессу 1. Затем, вызвав функцию `MPI_Recv`, ожидайте сообщение от процесса с номером `size-1`.
  - В секции для остальных процессов, вызвав функцию `MPI_Recv`, ожидайте сообщение от процесса с номером `rank-1`. После получения сообщения увеличьте значение переменной `buf` на единицу и отправьте его следующему процессу: для процесса с номером `size-1` это будет 0, для остальных – `rank+1`.
7. Для всех процессов выведите значение переменной `buf` на экран с помощью `printf`.

8. Скомпилируйте и запустите ваше приложение. Убедитесь, что выводится верный результат.

**Указания к заданию 14. Коммуникации «точка-точка»: схема «мастер-рабочие»**

1. Создайте проект `mpi_masterslave` в Microsoft Visual Studio 2010 с поддержкой MPI (см. указания к заданию 10).
2. Вставьте в код программы вызовы функций `MPI_Init` и `MPI_Finalize`.
3. Запишите номер каждого процесса в переменную `rank`.
4. Запишите количество параллельных процессов в программе в переменную `size`.
5. Создайте сообщение, объявив целочисленную переменную `buf`.
6. С помощью оператора `if` выделите в программе две секции кода: для master-процесса и для остальных процессов:

```
if (rank == 0) {  
    // Код, выполняемый master-процессом  
}  
else {  
    // Код, выполняемый slave-процессами  
}
```

7. В секции для slave-процессов присвойте переменной `buf` значение номера процесса. Отправьте `buf` master-процессу.
8. В секции для master-процесса с помощью оператора `for` создайте цикл со счетчиком `src`, изменяющимся от 1 до `size-1`. В теле цикла с помощью функцию `MPI_Recv` получите сообщение от процесса с номером `src`. Выведите полученное сообщение на экран с помощью `printf`.
9. Скомпилируйте и запустите ваше приложение. Убедитесь, что выводится верный результат.

**Указания к заданию 15. Коммуникации «точка-точка»: простые неблокирующие обмены : схема «сдвиг по кольцу»**

1. Создайте проект `mpi_ring` в Microsoft Visual Studio 2010 с поддержкой MPI (см. указания к заданию 10).
2. Ознакомьтесь с основными функциями неблокирующей передачи сообщений библиотеки MPI:

**`int MPI_Isend (void *buf, int count, MPI_Datatype datatype, int dest, int msgtag, MPI_Comm comm, MPI_Request *request)`** – 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 по-

вторно использовать данный буфер для других целей без получения дополнительной информации о завершении данной посылки. Окончание процесса передачи (т.е. тот момент, 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)** – 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)** – 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.

3. Запишите номер каждого процесса в переменную rank.
4. Запишите количество параллельных процессов в программе в переменную size.
5. Создайте сообщение, объявив целочисленную переменную buf, присвойте ей значение rank.
6. В схеме коммуникации процессов «сдвиг по кольцу» все процессы выполняют одни и те же действия: отправляют сообщение следующему процессу, затем получают сообщение от предыдущего процесса. Реализовать это можно следующим образом:
  - Определите номер процесса, которому будет отправляться сообщение. Вызовите неблокирующую MPI- функцию отправки сообщения.
  - Определите номер процесса, от которого будет приниматься сообщение. Вызовите неблокирующую MPI- функцию приема сообщения.
  - Дождитесь завершения операций обмена с помощью двух вызовов функции MPI\_Wait, либо объединенной функции MPI\_Waitall.

**int MPI\_Waitall (int count, MPI\_Request \*array\_of\_requests, MPI\_Status \*array\_of\_statuses)** блокирует работу, пока все операции обмена, связанные с активными дескрипторами в списке, не завершатся, и возвращает статус всех операций. Оба массива имеют одинаковое количество элементов. Элемент с номером i в array\_of\_statuses устанавливается в возвращаемый статус i-ой операции.

7. Для всех процессов выведите значение переменной buf на экран с помощью printf.
8. Скомпилируйте и запустите ваше приложение. Убедитесь, что выводится верный результат.

**Указания к заданию 16. Коммуникации «точка-точка»: схема «каждый каждому»**

1. Создайте проект mpi\_all в Microsoft Visual Studio 2010 с поддержкой MPI (см. указания к заданию 10).
2. Вставьте в код программы вызовы функций MPI\_Init и MPI\_Finalize.
3. Запишите номер каждого процесса в переменную rank.
4. Запишите количество параллельных процессов в программе в переменную size.
5. Создайте сообщение, объявив целочисленную переменную buf, присвойте ей значение rank.
6. В схеме коммуникации процессов «каждый каждому» с n процессами каждый процесс отправляет всем другим процессам по одному сообщению (всего n-1 сообщение), и принимает по одному сообщению от n-1 процесса. Реализовать это можно следующим образом:
  - Создайте цикл от 0 до n-1. Вызовите в теле этого цикла неблокирующую MPI- функцию отправки сообщения, в качестве номера процесса-получателя укажите счетчик цикла. С помощью оператора if исключите возможность отправки сообщения процессом самому себе.
  - После завершения цикла. Вызовите функцию ожидания завершения асинхронных операций MPI\_Waitall.
  - Создайте еще один цикл от 0 до n-1. Вызовите в теле этого цикла неблокирующую MPI- функцию приема сообщения, в качестве номера процесса-получателя укажите счетчик цикла. С помощью оператора if исключите возможность приема сообщения процессом от самого себя.
  - После завершения цикла. Вызовите функцию ожидания завершения асинхронных операций MPI\_Waitall.
7. Для всех процессов выведите значение переменной buf на экран с помощью printf.
8. Скомпилируйте и запустите ваше приложение. Убедитесь, что выводится верный результат.