

## **Лекция 4. ВЗАИМОДЕЙСТВИЕ РАСПРЕДЕЛЕННЫХ ПРОЦЕССОВ ПОСРЕДСТВОМ ПЕРЕДАЧИ СООБЩЕНИЙ**

### **Взаимоотношение между синхронизируемыми задачами**

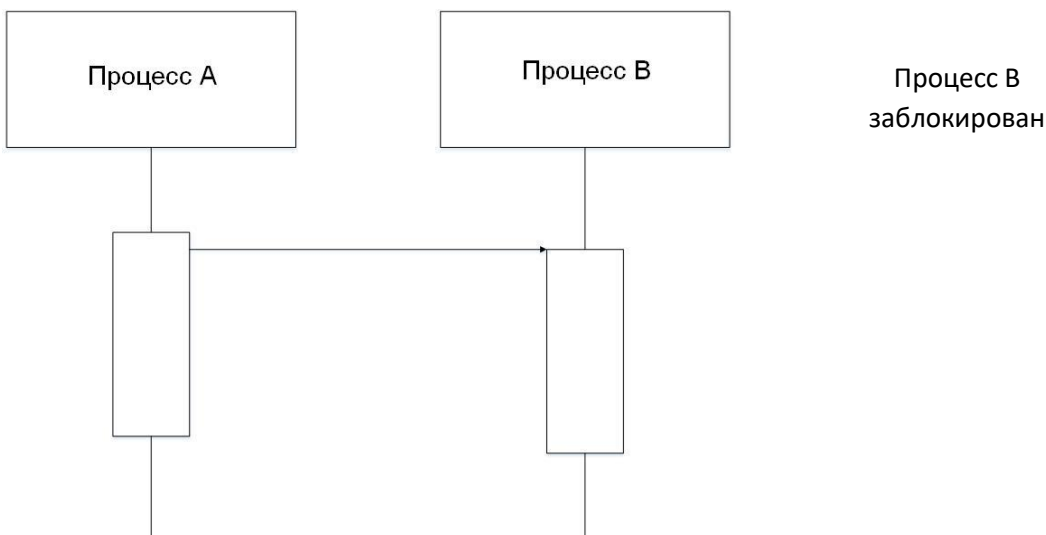
Четыре основных типа соотношений синхронизации между процессами (потоками) (2 потока в одном процессе, либо 2 процесса в одном приложении):

- **старт – старт;**
- **финиш – старт;**
- **старт – финиш;**
- **финиш – финиш.**

### **Взаимодействие «старт – старт»**

Процесс В активизируется (начинает выполнение) после активизации процесса А.

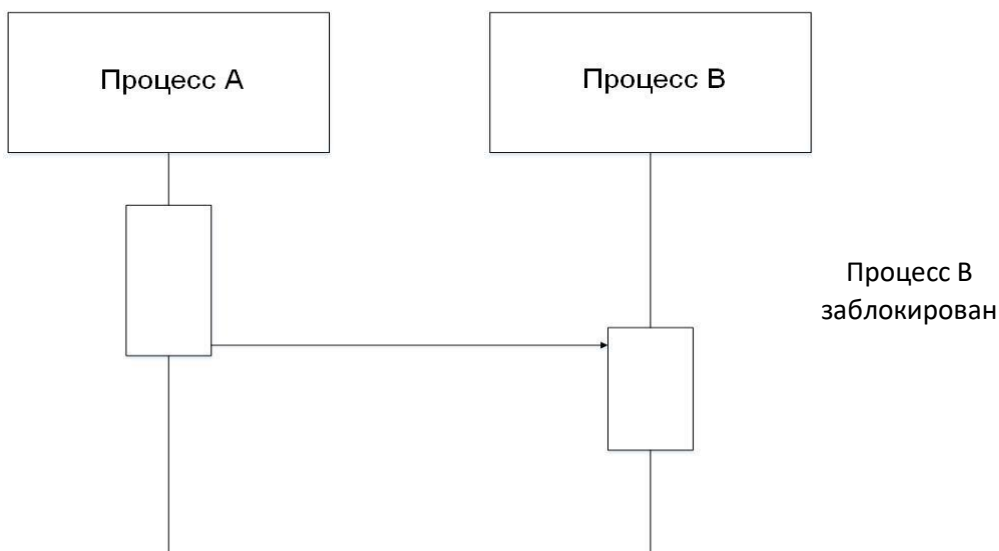
Данная схема синхронизации предполагает параллельное выполнение процессов.



### **Отношение синхронизации типа «финиш – старт»**

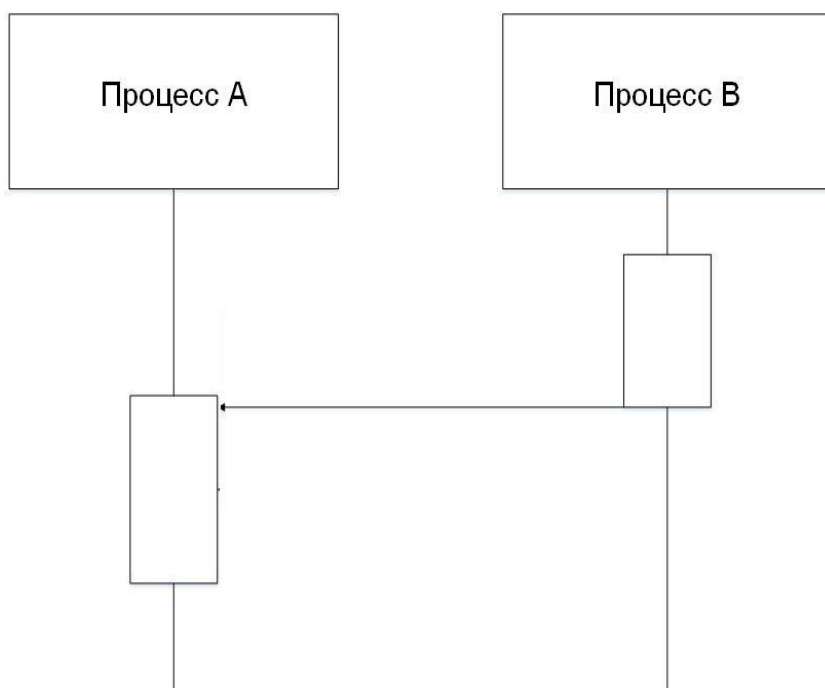
Процесс А не может завершиться до тех пор, пока не начнется процесс В (предшествующий процесс А (родитель) – потомок – процесс В). Т.о.

родительский процесс не может завершиться, пока не будет сгенерирован процесс-потомок.



### **Отношение синхронизации типа «старт – финиш»**

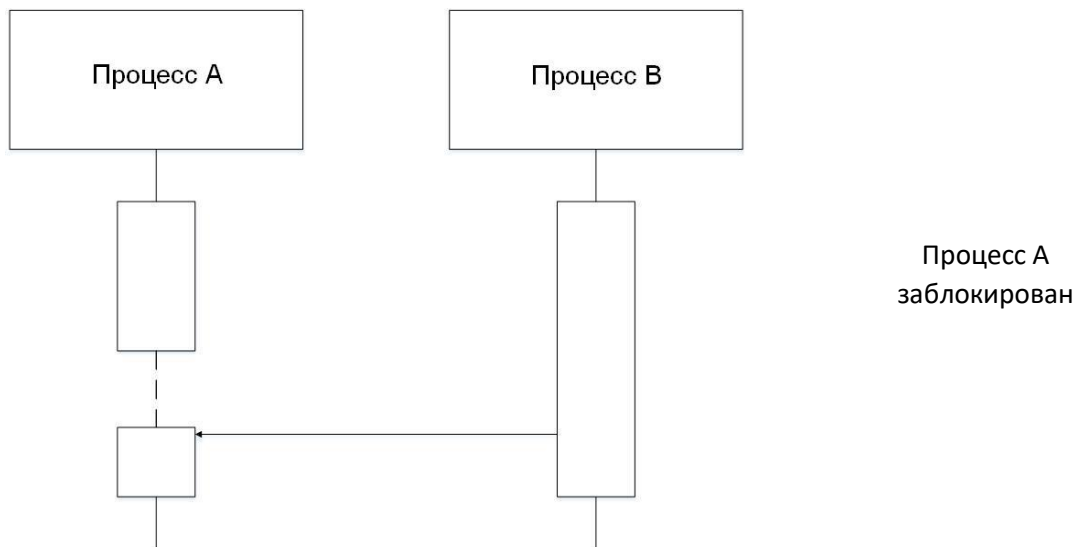
Процесс А не может начать своего выполнения до момента окончания процесса В.



Отношение «старт – финиш» - это отношение обратное «финиш – старт». Обе схемы реализуют взаимодействие типа «производитель – потребитель»

### **Отношение типа «финиш – финиш»**

Одна из задач (задачи А) не может завершаться о тех пор пока не завершится другой процесс (процесс В)



Родительский процесс А ожидает до тех пор, пока не завершатся все процессы потомки и после этого завершается сам. Примером взаимодействия является модель «управляющий-рабочий». «Управляющий» делегирует работу «рабочему» потоку.

### **Примитивы взаимодействия распределенно выполняющихся процессов**

Базовые примитивы – `send ()` и `receive ()`. Параметры примитива `send` в простейшем случае:

- идентификатор процесса – получателя сообщения;
- указатель на буфер с передаваемыми данными в адресном пространстве процесса-отправителя;
- количество передаваемых данных определенного типа.

**Пример** функции отправки данных `send (sendbuf, count, dest);`

Параметры примитива принятия данных `receive();`

- идентификатор процесса – отправителя либо указание идентификатора, позволяющего принимать сообщения от любого процесса;

- указатель на буфер в адресном пространстве процесса получателя, куда следует поместить принимаемые данные;
- количество принимаемых данных

**Пример функции** приема данных `receive (recvbuf , count, source);`

### **Блокирующие операции отправки получения без буферизации**

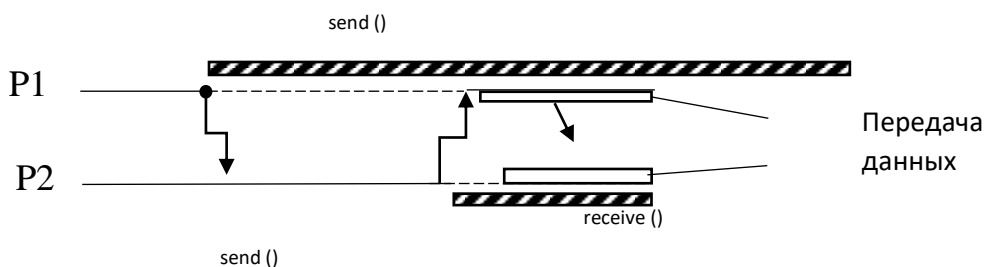
Возврат из вызова `send ()` не осуществляется до тех пор, пока не будет выполнен вызов `receive ()`, соответствующий этому `send ()`, и пока не будут переданы все данные в переменную `recvbuf`.

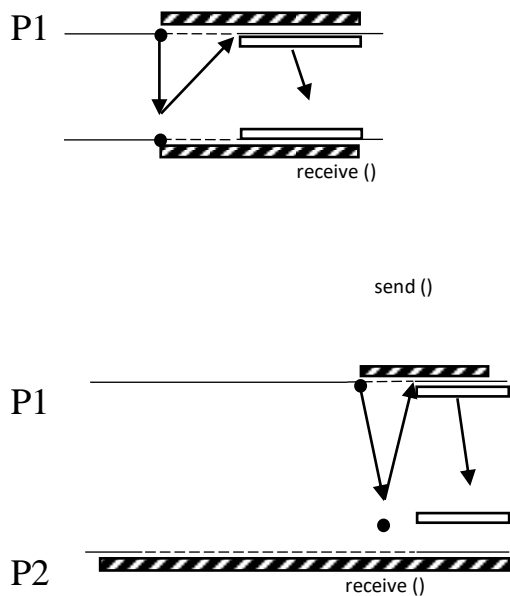
Передача данных предусматривает дополнительный обмен сигналами между производителем и потребителем.


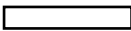
Последовательность передачи сообщений (сигналов) при передаче данных в рассматриваемом механизме взаимодействия:

- при готовности отправителя к передаче данных (вход в вызов функции `send ()`) он отправляет запрос на передачу данных получателю и блокируется в ожидании получения ответа;
- получатель отвечает на запрос после того, как он достигнет состояния готовности к приему данных (вызов `receive ()`).
- передача данных от производителя начинается после получения сигнала о готовности от принимающего процесса.

При передаче не используются дополнительные буферы на стороне отправителя и на стороне получателя.





-  - длительность блокировки процесса
-  - длительность передачи
- - вызов функций `send ()` и `receive ()`

Блокирующая отправка/получение могут быть использованы в случае, если вызов функций `send ()` и `received ()` выполняется приблизительно в одно время.

**Блокирующие отправка/получение могут привести к взаимной блокировке процессов.**

**Пример синтаксиса при взаимной блокировке**

P1	P2
<code>send (&amp;a, 1, 2);</code>	<code>send (&amp;b, 1, 1);</code>
<code>receive (&amp;b, 1, 2);</code>	<code>receive (&amp;a, 1, 1);</code>

### Блокирующие операции буферизированной отправки / получения

Указанный способ передачи предусматривает создание буферов на передающей и приемной сторонах.

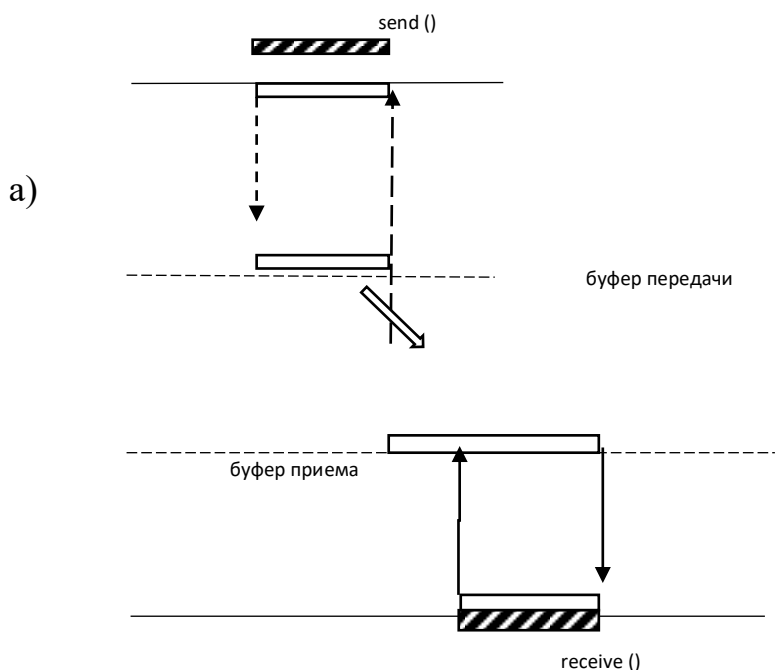
Действия на передающей стороне при реализации вызова `send ()` и на принимающей стороне при вызове `recv ()`:

- создание буфера для передаваемого сообщения (идентификаторы буфера ID процесса получения, ID сообщения);
- копирование данных из адресного пространства процесса отправителя в буфер передачи;
- передача управления из вызова `send ()` в управляющий процесс (управляющий процесс не блокируется);
- независимое от пользовательских процессов копирование данных из буфера на передающей стороне в буфер на приемной стороне;
- при готовности к приему данных (вызов `recv ()`) получатель извлекает данные из буфера приема и размещает их в адресном пространстве процесса.

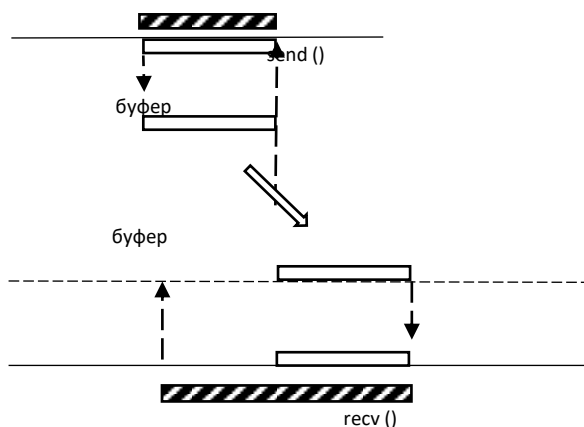
Обмен данными реализуется непосредственно системной распределенной обработки с использованием созданных предварительно буферов (без участия приложений).

#### Схема блокирующего

#### буферизированного взаимодействия



б)



Данная схема требует дополнительных накладных расходов: создание буферов, копирование данных между ними и т.д. Т.о буферизация позволяет исключить ситуации взаимоблокировок.

Возможный пример блокирования в стеке с буферизацией:

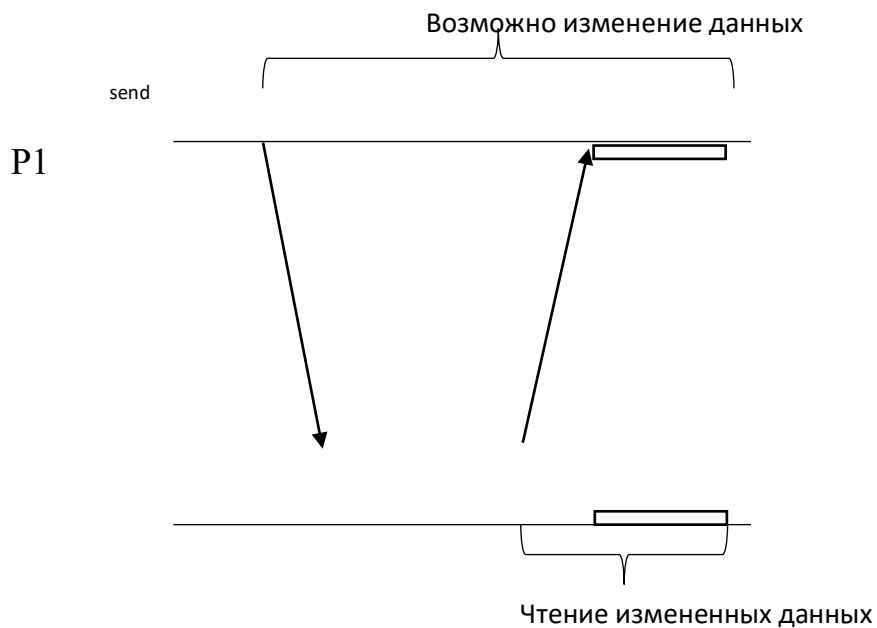
P1	P2
receive (&a, 1, 2);	receive (&b, 1, 1);
send (&b, 1, 2);	send (&a, 1, 1);

### **Неблокирующие операции отправки / получения**

При неблокирующих операциях приема / передачи возврат управления в исполняемый процесс осуществляется сразу после вызова соответствующей функции.

В данном случае процесс – производитель может изменить значение передаваемой переменной и процесс потребитель получит не соответствующее значение. Т.е. вызов send () начинает операцию передачи, но не гарантирует передачи нужных данных.

## Схема неблокирующей передачи



Невозможность изменения данных гарантируется блокированием процесса отправителя на вызове `recv ()`. После извлечения данных процесс-получатель подтверждает прием командой (вызовом) `send ()`.

## **Взаимодействие распределенных процессов посредством передачи сообщений (механизм взаимодействия).**

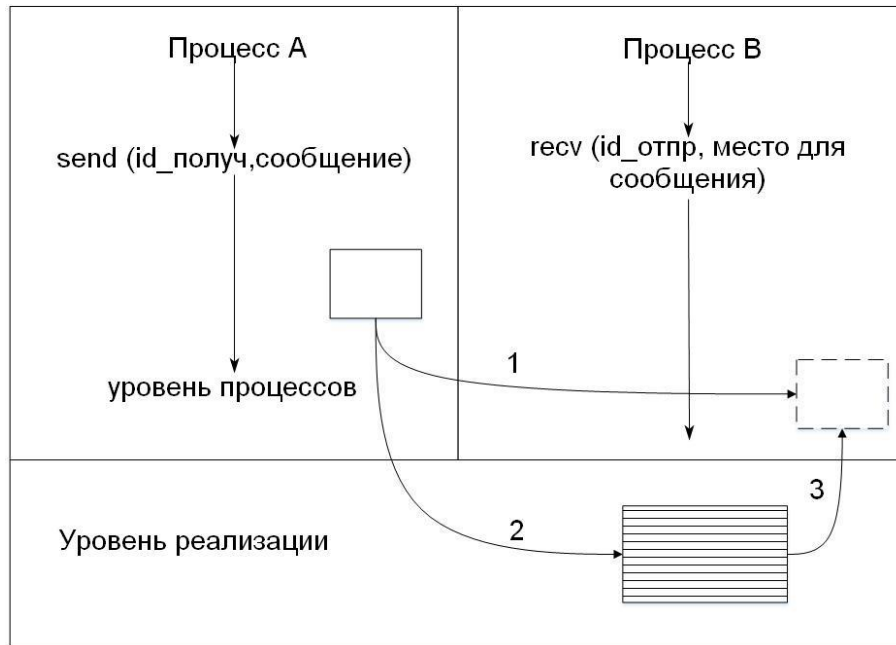
Состав сообщения:

Место назначения	Заголовок (используется механизмом транспортирования)
Источник	
Тип сообщения	
Тип сообщения	Используется взаимодействующими процессами



Поле «тип сообщения» используется для задания его вида: запрос или ответ на запрос. В полях «Источник» и «Место назначения» указывается по одному процессу.

### **Реализация процесса передачи сообщения**



- 1) Сформированное сообщение размещается в адресном пространстве процесса. Если процесс В запросил наличие сообщения, то оно переписывается в адресное пространство процесса В.
- 2) Если процесс В не запросил сообщение, оно из адресного пространства процесса А переписывается в буфер процесса В
- 3) При готовности процесса В к получению сообщения, оно извлекается из буфера в адресное пространство процесса В. Сообщение при передаче копируется дважды: из адресного пространства процесса А в буфер, из буфера в адресное пространство процесса В.

### **ОСОБЕННОСТИ ВЗАИМОДЕЙСТВИЯ ПОСРЕДСТВОМ ПЕРЕДАЧИ СООБЩЕНИЙ**

- Взаимодействующие процессы не указывают идентификаторы друг друга;
- Процесс реализует отправку сообщения нескольким адресатам;
- Требуется различать типы сообщений, которыми обмениваются процессы (запрос, ответ);

В первом случае в сообщении вместо ID-отправителя указывается ID вида «Any».

### **Типизация сообщений**

Реализация примитивов с указанием типа сообщения:

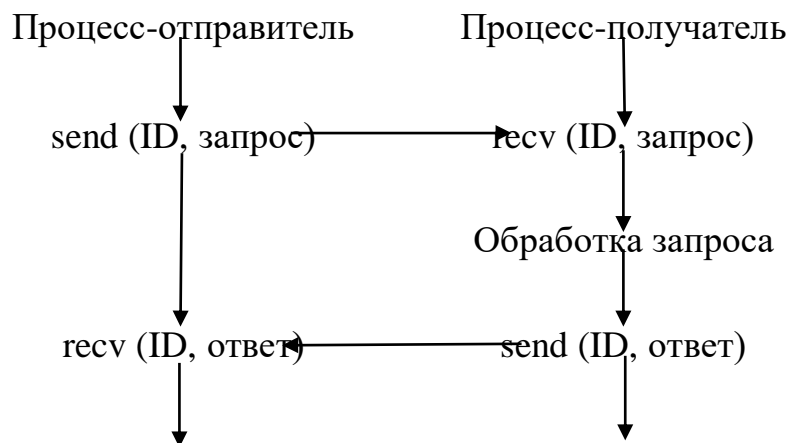
send (ID\_получателя, сообщение-запрос);

recv (ID\_отправителя, сообщение-запрос);

send (ID\_получателя, сообщение-ответ);

recv (ID\_отправителя, сообщение-ответ);

### **Пример взаимодействия между клиентом и сервером**

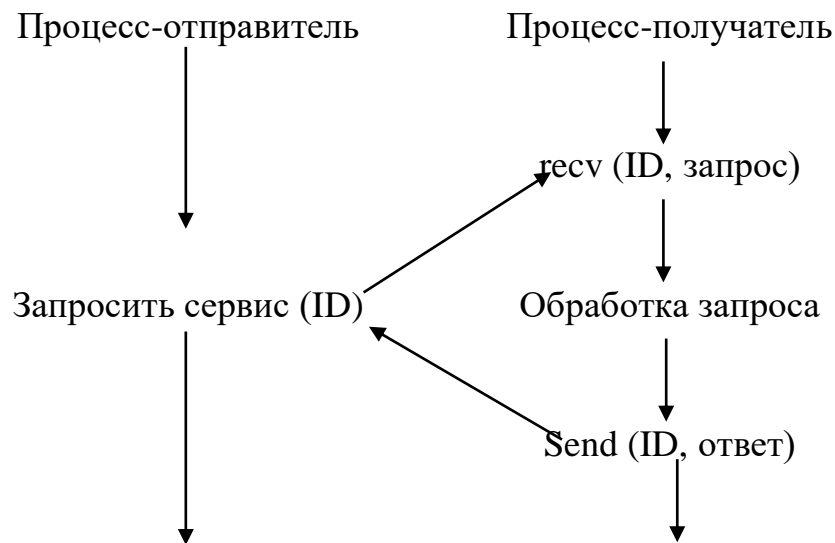


Процесс – отправитель, сформировав запросы далее продолжает свое выполнение

Примитив **Запросить\_сервис()** – это реализация (совместная) пары примитивов

send (ID, запрос) – recv (ID, ответ) (отправитель блокируется до получения ответа).

### **Пример взаимодействия клиента и сервера с блокированием клиента в ситуации ответа**



### Широковещание и мультिवещание сообщений

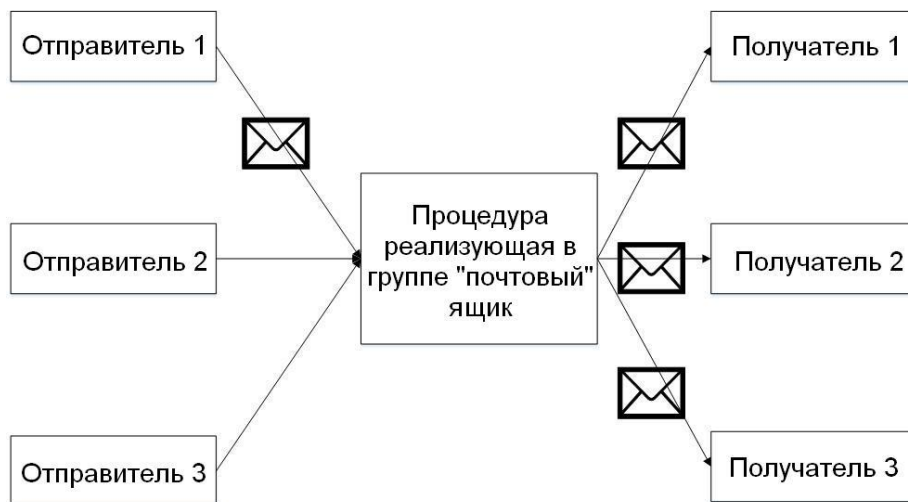
Реализуется отправка сообщения определенной совокупности процессов. Либо реализуется получение сообщений от заданной группы процессов. Перечень процессов, которым рассылаются сообщения либо от которых принимаются сообщения, предварительно д/б сформирован.

Результат – именованная группа процессов (т.е. формируется группа процессов, для которых будет реализовываться широковещание, группа именуется).

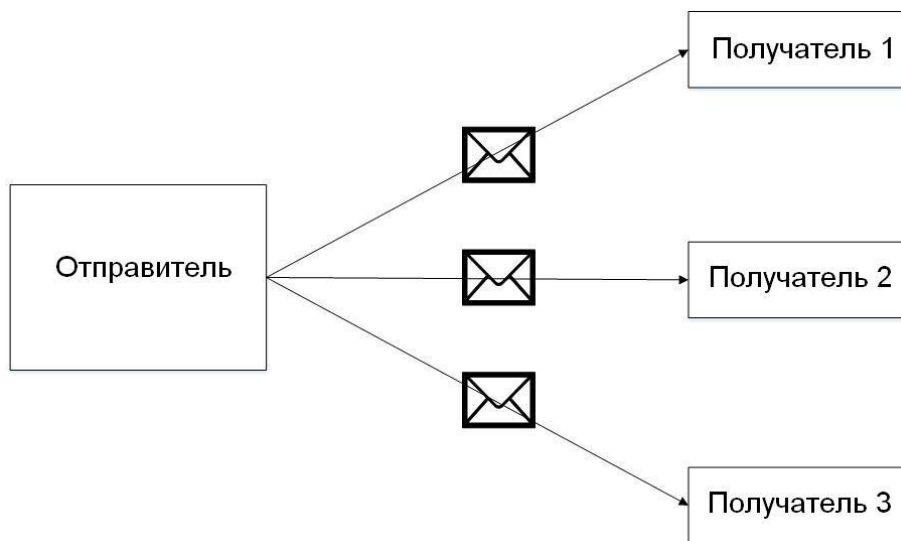
Передача сообщений группе - аналог передачи сообщения отдельному процессу. Формат сообщения, формируемого пользователем:

Имя группы / место назначения
Источник
Тип сообщения
Тело

### Пример широковещания (в пределах групп)



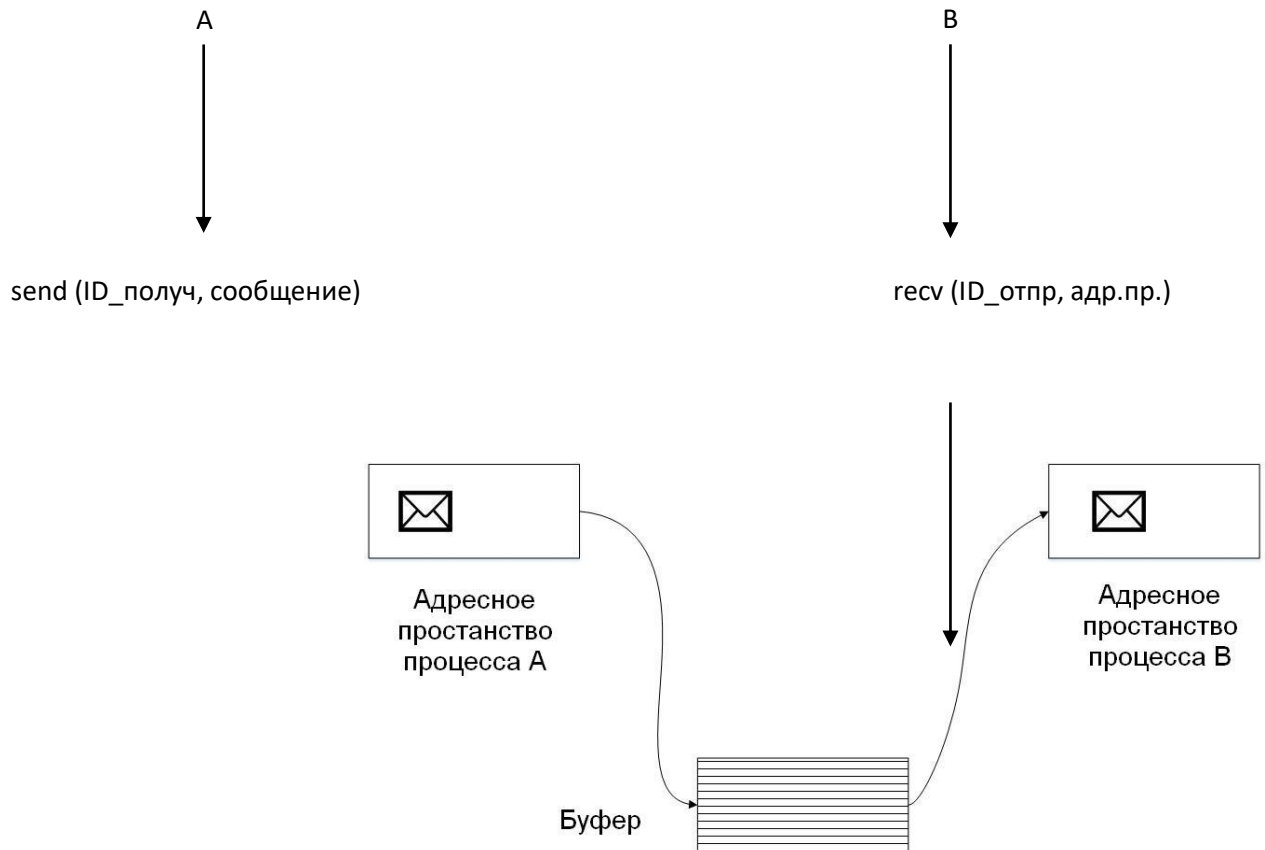
### Пример мультивещания



Т.о. широковещание реализуется в пределах именованной группы процессов.

## РЕАЛИЗАЦИЯ СИНХРОНИЗАЦИИ ПРИ ПЕРЕДАЧЕ СООБЩЕНИЙ

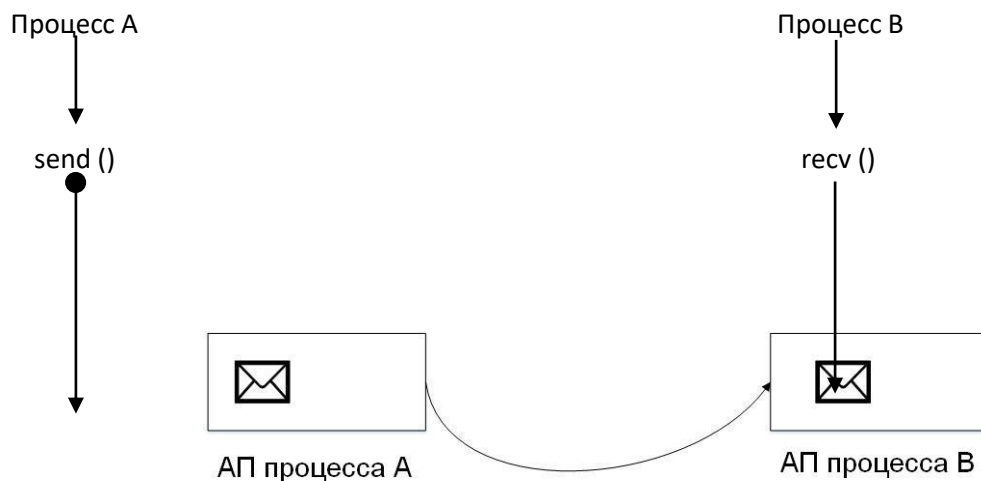
Если реализуется асинхронная передача, то выполняется двойное копирование данных (передающая сторона не блокируется).



Т.о. асинхронная передача связана с необходимостью создания буфера для хранения сообщения в случае, если процесс –приемник не готов получить данные. Также затраты связаны с копированием данных из АП процесса А в буфер и из буфера в АП процесса В.

Снижение затрат ресурсов, связанных с организацией буфера, с организацией копирования в/из буфера возможно посредством синхронной передачи (не предполагает использование буфера). Сообщение копируется от отправителя к получателю в момент синхронизации (отправитель блокируется).

### Схема синхронного взаимодействия посредством передачи сообщений



- - возможное ожидание процесса отправителя

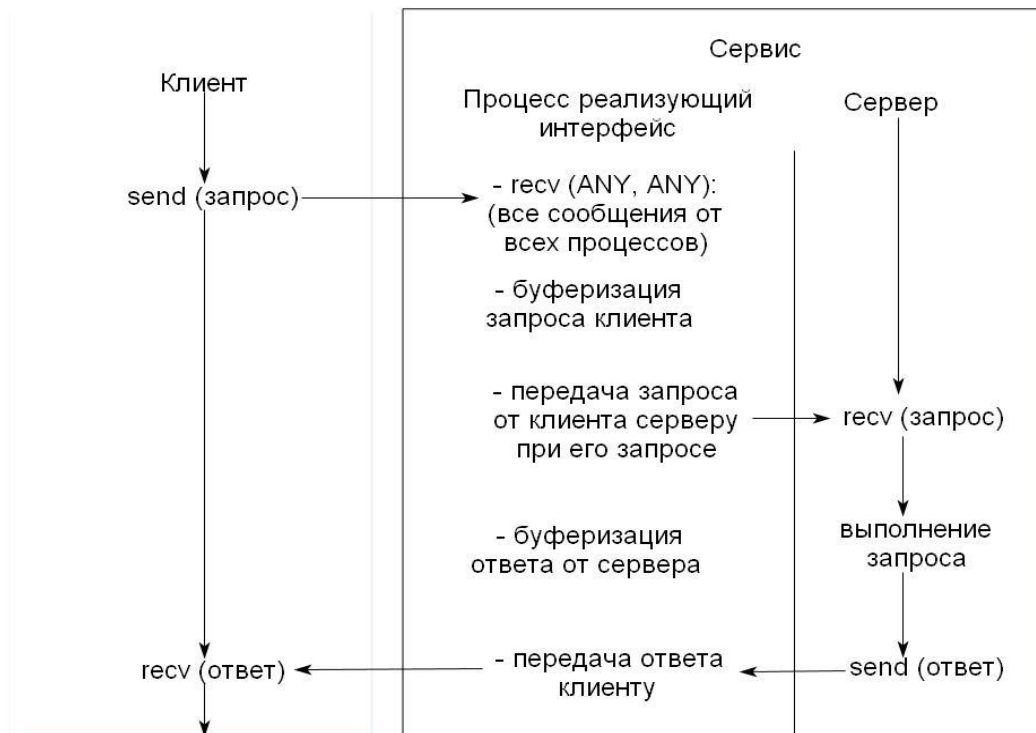
### Случай, когда отправляющий процесс не может быть заблокирован

Процесс является некоторым сервисом (в частности, системным сервисом), который отправил ответ клиенту на его запрос (ожидание ответа клиентам не гарантируется, в случае синхронной передачи сервис д/б заблокирован, однако заблокированным он быть не может).

#### Пример:

Процесс, реализующий ввод данных и передачу их клиентам (процесс, вводящий данные и передающий их клиентам, заблокирован быть не может).

### Схема промежуточной буферизации запросов/ответов, позволяющая исключить блокирование при синхронной передаче



## ВЗАИМОДЕЙСТВИЕ МЕЖДУ ПРОЦЕССАМИ В РАСПРЕДЕЛЕННОЙ СИСТЕМЕ

Виды взаимодействия распределено выполняющихся процессов:

- 1) **Взаимодействие при передаче данных;**
- 2) **Взаимодействие при синхронизации доступа к общим ресурсам;**
- 3) **Удаленный вызов процедур (Remote Procedure Call, RPC)** – вызывающий и вызываемый процессы находятся на разных РС. При удаленном взаимодействии (в соответствии с пунктами 1 и 2) требуется реализовывать синхронную модель.

### Способы реализации синхронной модели



где send () – блокирующая  
передача

где send () – не блокирующая  
передача (ожидание подтверждения  
синхронизации)

### **Удаленный вызов процедур (RPC) (ключевые слова: «вызов процедур»)**

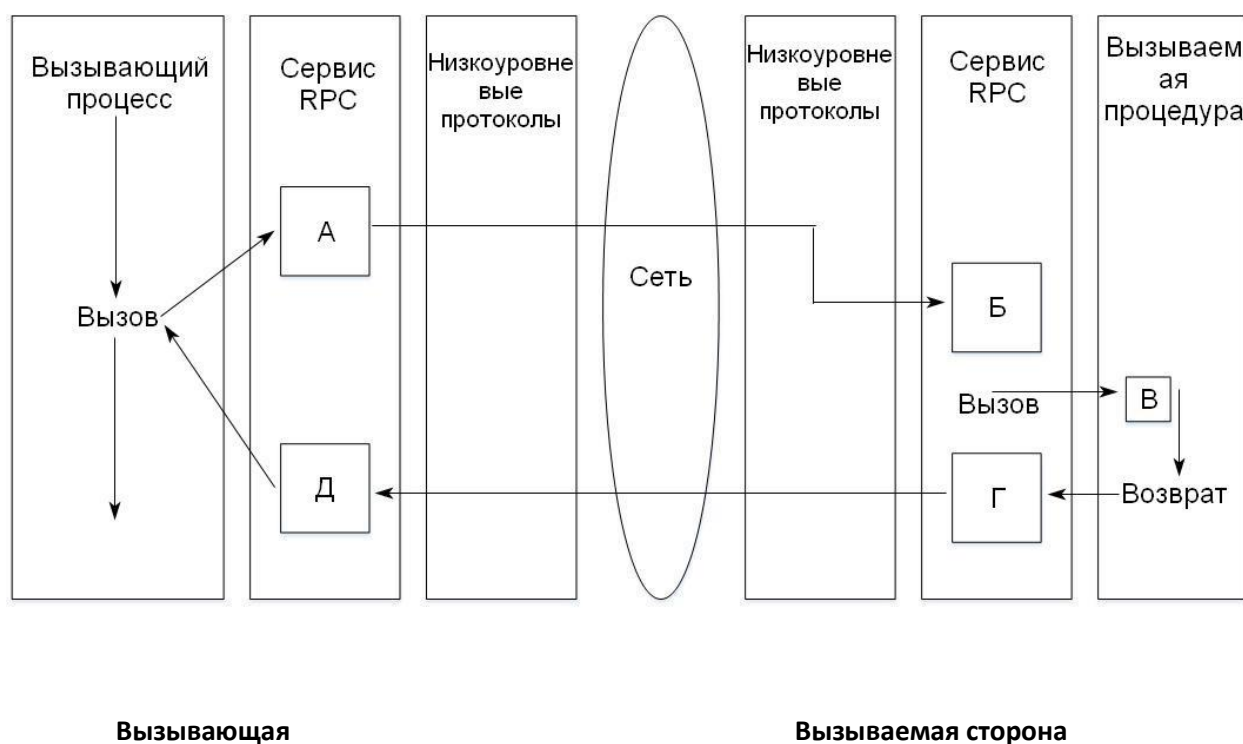
Вызывающий и вызываемый процессы, взаимодействующие в рамках RPC, функционируют в рамках разных АП. Поэтому взаимодействие между ними (в частности, передача аргументов в вызываемую процедуру) реализуется посредством передачи сообщений.

Механизм RPC предполагает, что вызывающий процесс передает удаленной процедуре данные, которые процедура обрабатывает. Результаты возвращаются вызвавшему процессу.

Т.о. механизм RPC реализован в системе, обеспечивающей распределенные вычисления.



## Схема реализации RPC для систем, обеспечивающих распределенные вычисления



Вызывающий процесс реализует обращение к удаленной процедуре (вызов с использованием механизма RPC распознается как удаленный). Аргументы для вызываемой процедуры задаются обычным образом.

Т.к. процедура идентифицирована как удаленная, то управление передается механизму RPC в точку А.

Сервис (механизм) RPC реализует:

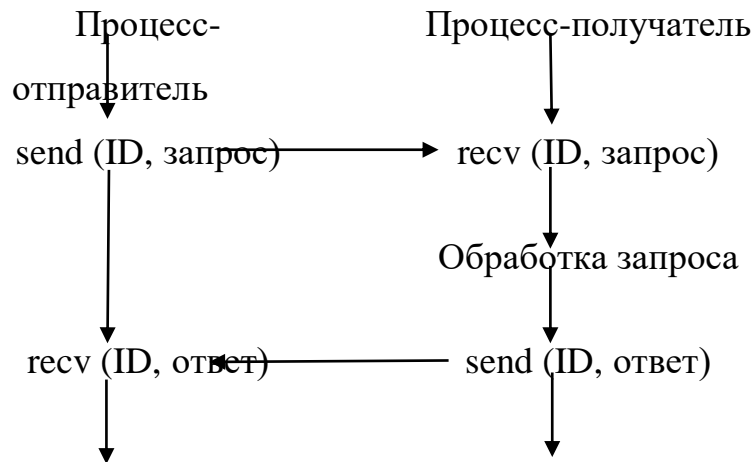
- упаковку аргументов в требуемом виде для передачи по сети;
- формирование идентификатора удаленного вызова;

Сервис RPC на вызываемой стороне реализует:

- распаковку данных в аргументы, которые могут быть переданы в вызываемую процедуру;
- фиксирует идентификатор удаленного вызова;

В соответствии с идентификатором удаленного вызова вызывающая сторона формирует подтверждение в случае получения вызывающим процессом результатов обработки.

### Пример взаимодействия между клиентом и сервером



Процесс – отправитель, сформировав запросы далее продолжает свое выполнение

Примитив запросить – сервис () – это реализация (совместная) пары примитивов

send (ID, запрос) – recv (ID, ответ) (отправитель блокируется до получения ответа).

### Пример взаимодействия клиента и сервера с блокированием клиента в ситуации ответа

