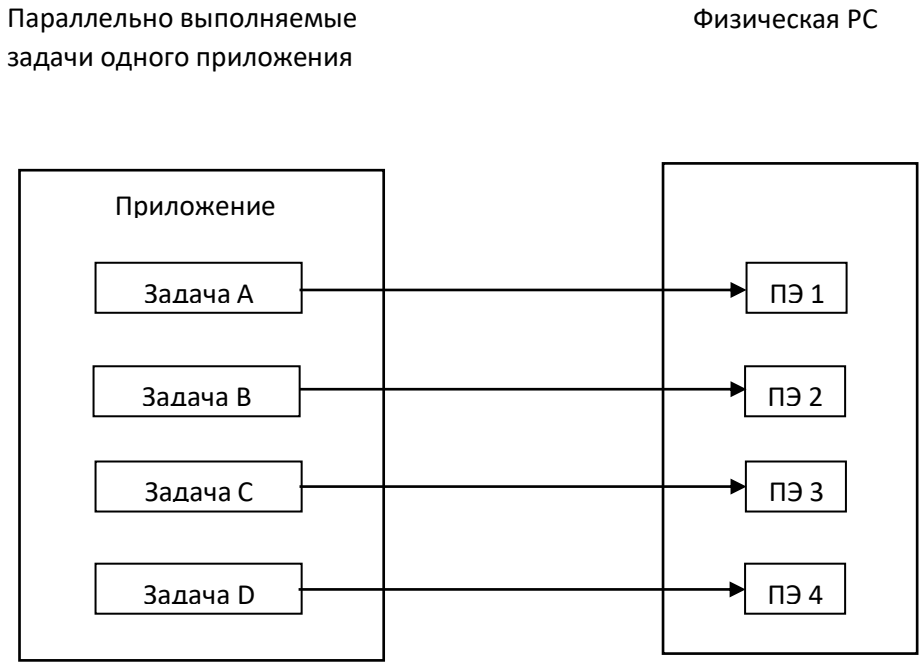
# Лекция 1. СПОСОБЫ РЕАЛИЗАЦИИ РАСПАРАЛЛЕЛИВАНИЯ. ОРГАНИЗАЦИЯ ПАРАЛЛЕЛЬНЫХ/РАСПРЕДЕЛЕННЫХ ВЫЧИСЛЕНИЙ

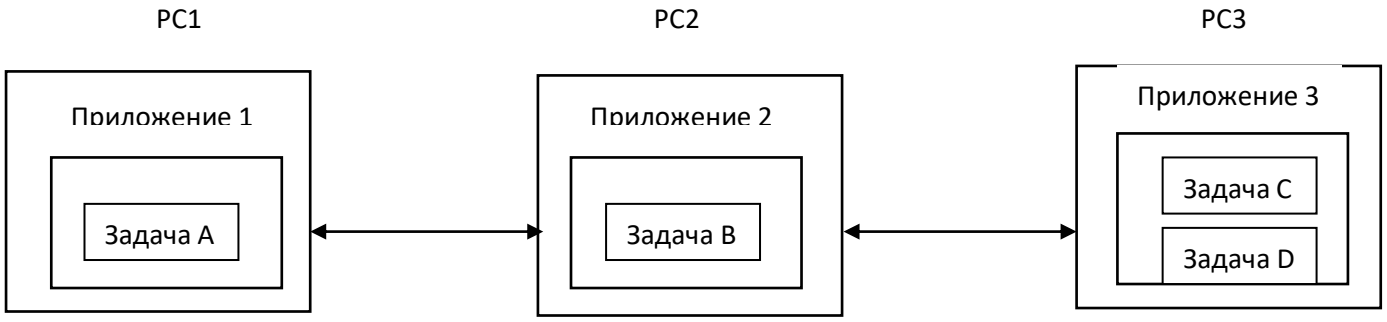
## Способы реализации распараллеливания

Методы параллельного программирования – распределение программы между N процессорными элементами (ПЭ) в рамках одной РС. Методы распределенного программирования – распределение программы между N процессами (задачами), который реализуются на разных рабочих станциях (РС).

При чистом параллелизме одновременно выполняются части одной и той же программы. При распределенном программировании параллельно выполняются отдельные программы, являющиеся частями распределенного приложения.



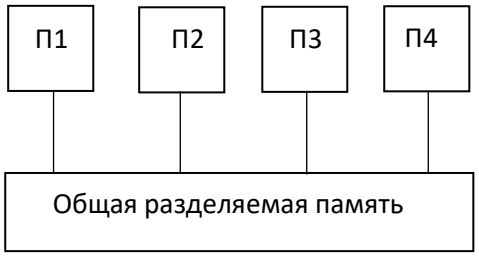

## Распределенное приложение



Приложение 1 реализует задачу A, приложение 2 реализует задачу B, приложение 3 реализует задачи C и D

## Организация параллельных/распределенных вычислений с точки зрения модели взаимодействия процессов

1. Реализация принципа разделения памяти (память – ресурс, к которому реализуют доступ процессы). Т.о. процессы имеют доступ к любому участку общей памяти. Взаимодействие процессов реализуется через общую (разделяемую) память.

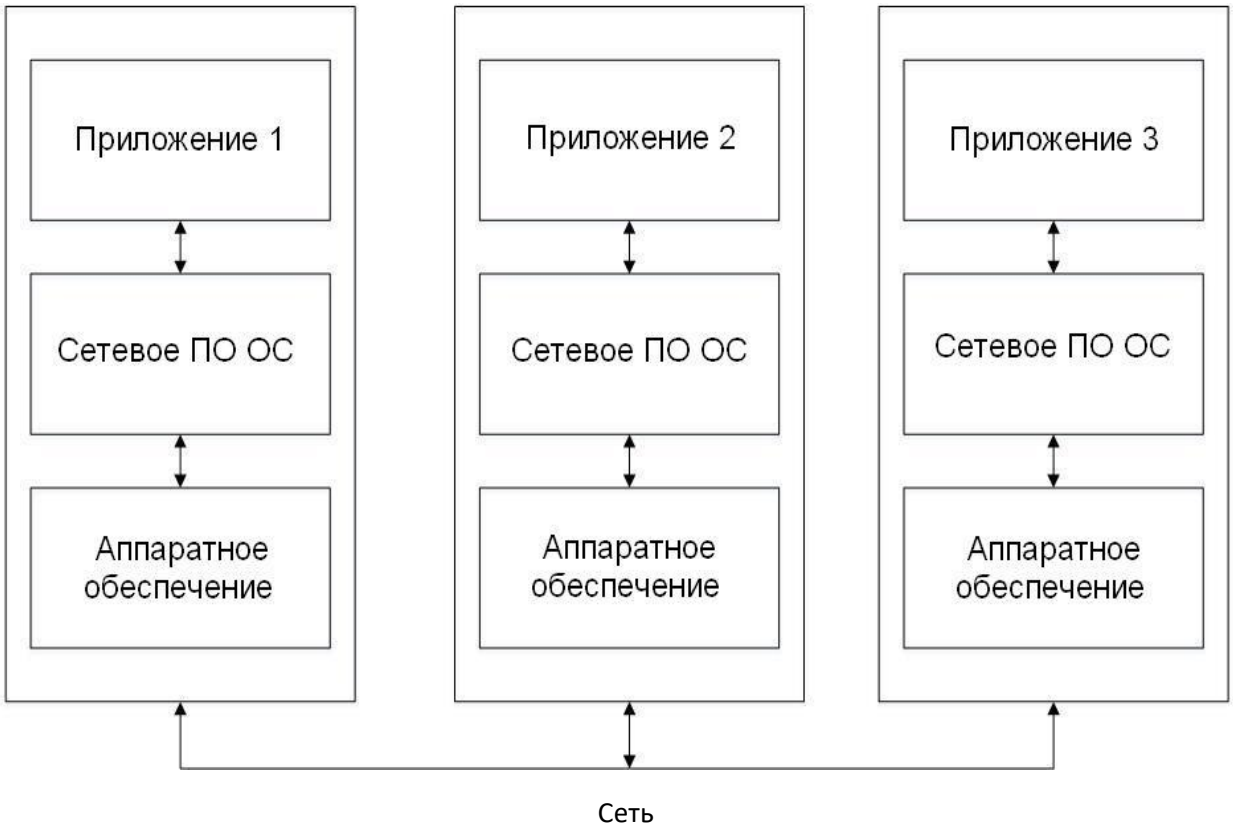



Пi – i-ый процесс, реализуемый на i-ом ПЭ

2. Реализация принципа (модели) распределенной памяти. Каждый процесс реализует доступ к локальной памяти. Доступ к памяти других устройств (PC) невозможен. Взаимодействие процессов реализуется через коммуникационную среду (сеть). Т.е. взаимодействие реализуется посредством передачи сообщений.

Пi – i-е приложение

Схема непосредственного взаимодействия компонент (элементов) распределенного приложения с использованием средств ОС



## Классификация схем параллелизма

Классификация осуществляется с точки зрения анализа количества программ, одновременно воздействующих на один или несколько потоков данных.

1) MPSD (Multi Program Simple Data) – предполагает организацию конвейера программ (макроконвейера). Каждые вычислительные устройства, входящие в состав конвейера, реализует один из этапов обработки потока данных. Таким образом, на различных вычислительных устройствах одновременно выполняются преобразования одного потока данных.

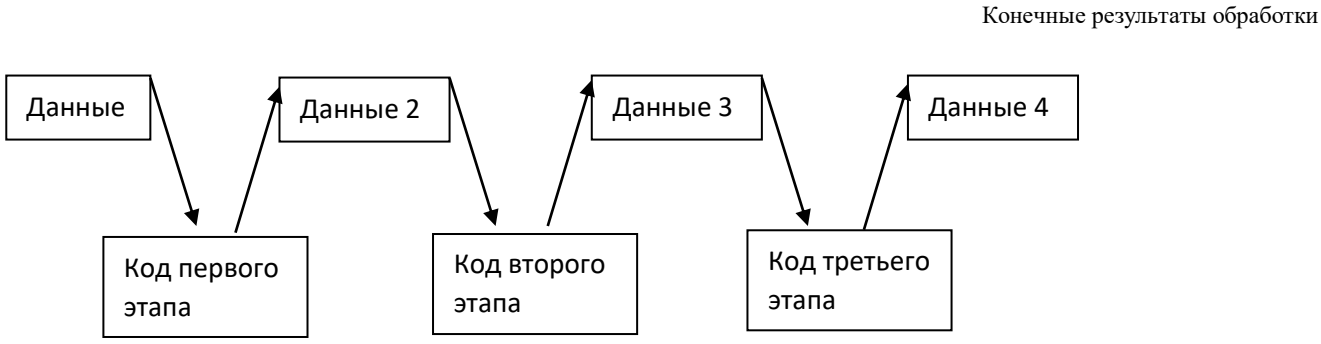
Предпосылки к конвейерной обработке данных – возможность представления алгоритма в виде последовательности этапов (стадий) обработки. Результаты обработки данных предшествующего этапа являются исходными (входными) данными последующего этапа. Особенности организации конвейерной обработки:

- каждый сегмент конвейера реализует хранение кода этапов обработки, который на нем интерпретируется;

- обмен данными между вычислительными устройствами различных этапов реализуется при готовности результатов обработки на предыдущем этапе;

- обрабатываемые данные хранятся на сегменте конвейера только в течение интервала времени, равного длительности их обработки.

## Схема конвейерной обработки потока данных



2) SPMD (Simple Program Multi Data) – одна программа – несколько потов данных.

Схема SPMD предполагает N копий кода (где N – количество вычислительных устройств), каждые ВУ интерпретирует свою копию кода обрабатывающую строго «свои» данные (т.е. реализуется одновременная обработка N блоков данных N копиями одной программы).

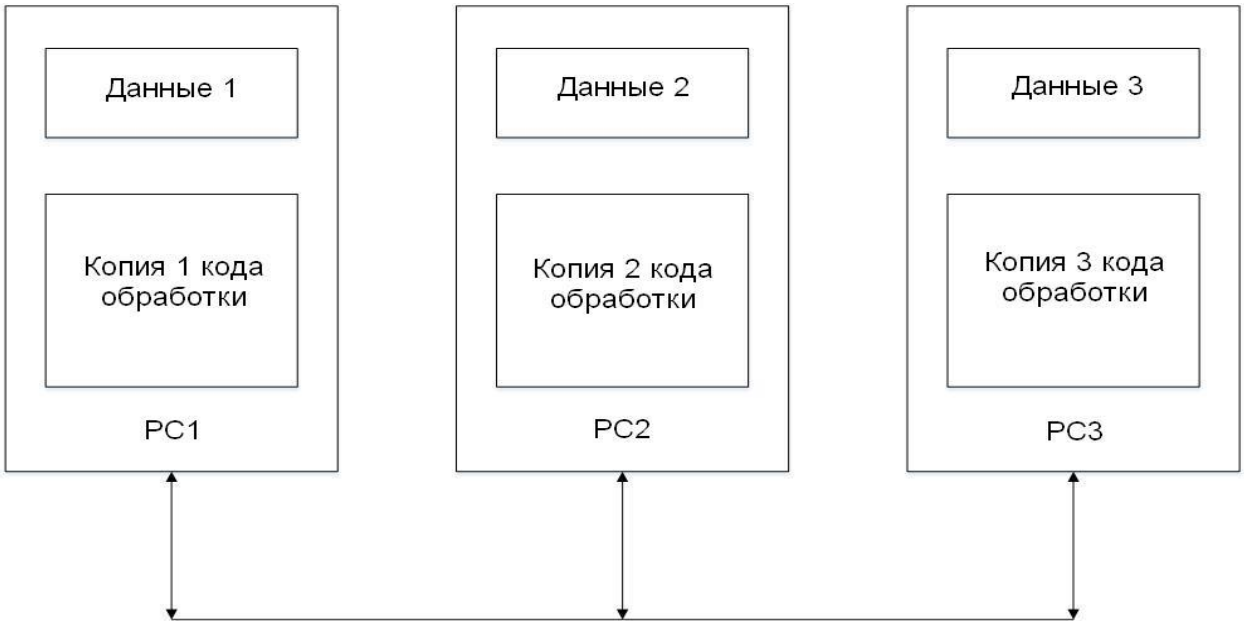
Особенность реализации обработки в системе SPMD:

- обрабатываемые данные являются однотипными (на разных ВУ).

- на разных ВУ предусмотрен одинаковый способ хранения обрабатываемых данных;

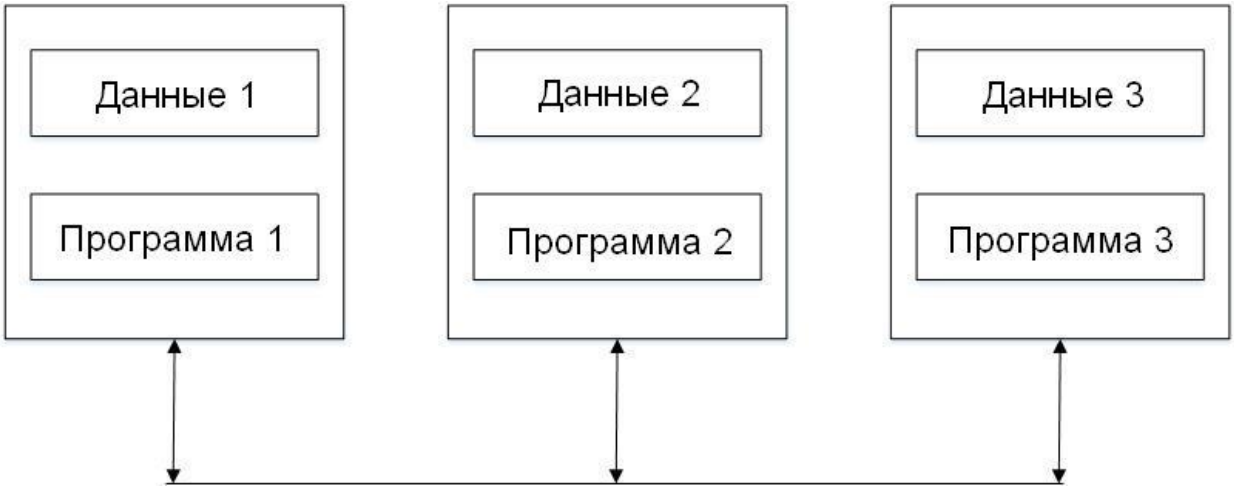
- различные данные, хранимые на разных ВУ подвергаются абсолютно одинаковой обработке

**Схема системы с одинаковой обработкой при разделении данных по ВУ (SMPD)**



3) MPMD (Multi Program Multi Data) – каждые ВУ интерпретирует отличную от других программу, обрабатывая при этом соответствующие этой программные данные.

**Схема организации MPMD систем**




**Архитектура аппаратных средств**

1) **Параллельные вычисления, реализуемые в системах с общей памятью** (вычисления, предусматривающие взаимодействие процессов с использованием общей памяти / модель взаимодействия с использованием общей памяти). Вычисления реализуются в SMP-системах (Symmetric Multi-Processing). В SMP-системах взаимодействие процессов реализуется посредством обращения к общим переменным, находящихся в общедоступной памяти.

Особенности аппаратной организации SMP-систем:

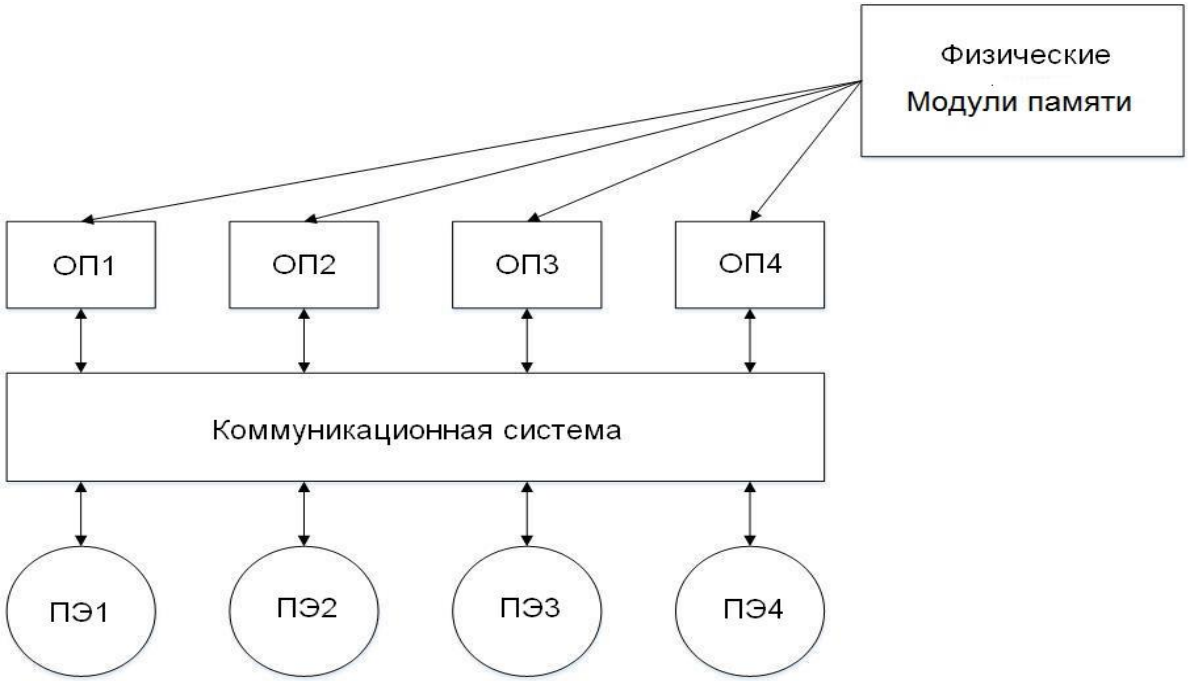
- общая память (общее поле памяти) состоит из отдельных модулей, к любой ячейке каждого модуля ПЭ могут осуществлять доступ;

- связь между модулями памяти и ПЭ осуществляется посредством коммуникационной системы (коммутаторов вычислительной системы (ВС));

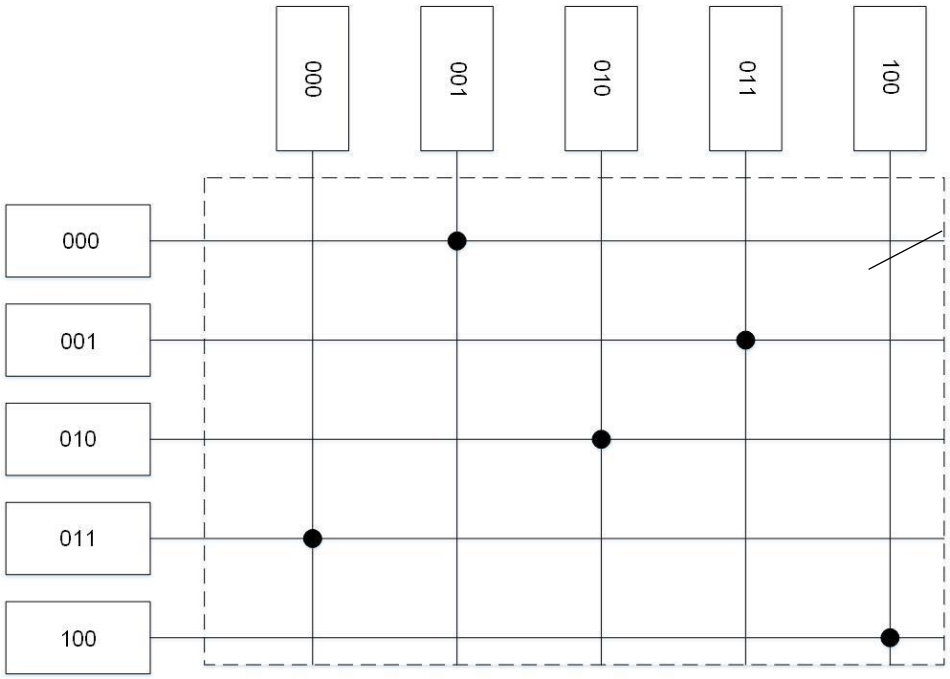
- коммутатор ВС может поддерживать несколько параллельных каналов взаимодействия (чтение / запись) между различными парами ПЭ и блоков ОП;

- в системах реализуется единое логическое адресное пространство ОП, состоящей из отдельных (различных) физических модулей.

**Обобщенная функциональная схема SMP-систем**




Функциональная организация SMP-систем




Коммутатор ВС

Коммутатор поддерживает одновременно 5 сформированных каналов обмена между ПЭ и блоками ОП. На основе адреса ПЭ и адреса блока ОП коммутатором формируется канал (физический) между парой устройств.

**Формат логического адреса ячейки памяти**

| Идентификатор блока памяти | Физический адрес ячейки в блоке |
|---|---|

Формат команды взаимодействия между ПЭ и блоком ОП

| Идентификатор ПЭ | Идентификатор блока ОП | Адрес ячейки | КОП | значение |
|---|---|---|---|---|

Т.о. коммутатор обеспечивает множественность путей (каналов) между ПЭ и блоками ОП.

2) **Распределенные вычисления, реализуемые как SMPD, так и MPMD, выполняются в MPP-системах** (Multi Parallel Processor), либо массовая параллельная архитектура.

Особенности организации MPP-систем:

- Физически распределенная память (модули памяти доступны непосредственно процессорам узлов, которым они принадлежат);

- Модель взаимодействия параллельно (распределено) выполняющихся процессов – посредством обмена сообщениями;

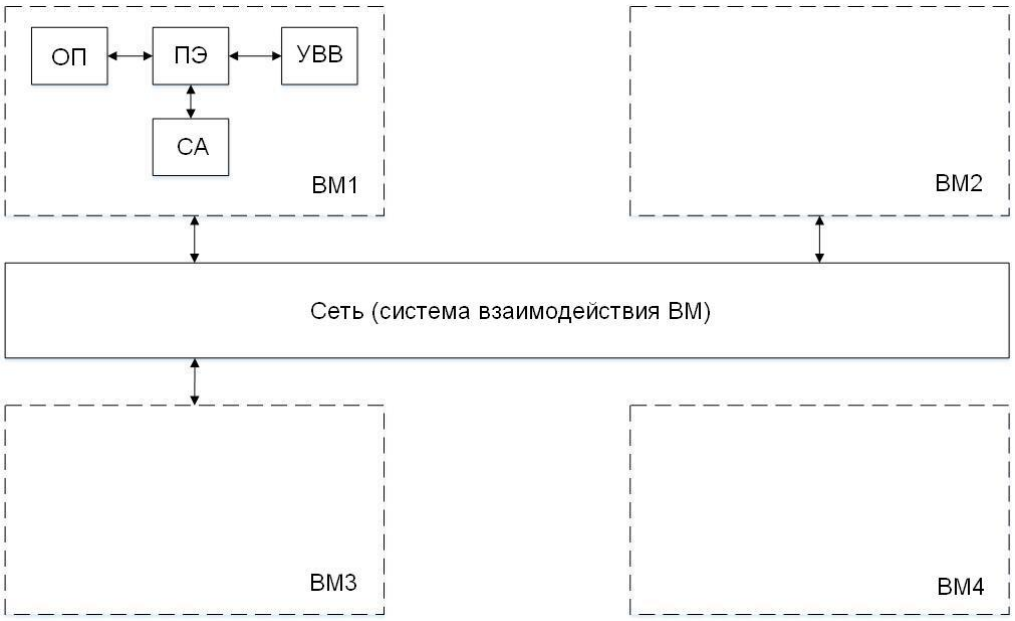
- Вычислительные узлы (PC) объединены сетью с высокой пропускной способностью;

- Каждому процессу и обрабатываемым им данным выделяется адресное пространство в блоке ОП соответствующего узла (PC);

- Модель вычислений – совокупность независимо выполняющихся процессов, обрабатывающих свои данные;

- Наличие главного процесса, интерпретируемого на одном из узлов, который выполняет планирование задач и активизацию приложений на других узлах (PC) в MPP-системе (главный процесс планировщик заданий, подчиненные процессы реализуют вычислительные задачи).

Функциональная схема MPP – систем




СА – сетевой адаптер, ВМ – вычислительный модуль.

Достоинство MPP-систем – возможность масштабирования вычислительных модулей.

## Понятие вычислительной модели (модели вычислений) и понятие процесса.

Вычислительная модель – это способ (протокол) активизации вычислительных действий. Т.е. вычислительная модель определяет порядок реализации вычислительных действий (запуск задач, синхронизация, обмен данными и т.д.)

Процесс – часть (единица) работы, предполагающая выполнение действий с данными. Интерпретация программы осуществляется посредством активизации и выполнения процесса. Понятия, связанные с выполнением процесса:

- идентификатор и статус процесса (состояние процесса);

- адресное пространство (в т.ч. стек) процесса;

- ресурсы (системные ресурсы), используемые при реализации процесса;

Виды процессов – пользовательские и системные (распределение памяти, планирование выполнения вычислений, выделение памяти пользовательским процессам для хранения данных)

## Ресурсы процессов

Три типа ресурсов:

- аппаратные ресурсы – физические устройства, которые могут совместно использоваться несколькими процессами. Аппаратные ресурсы могут быть дискретными (страничная организация ОЗУ) либо неделимыми (выделяемые процессам целиком).

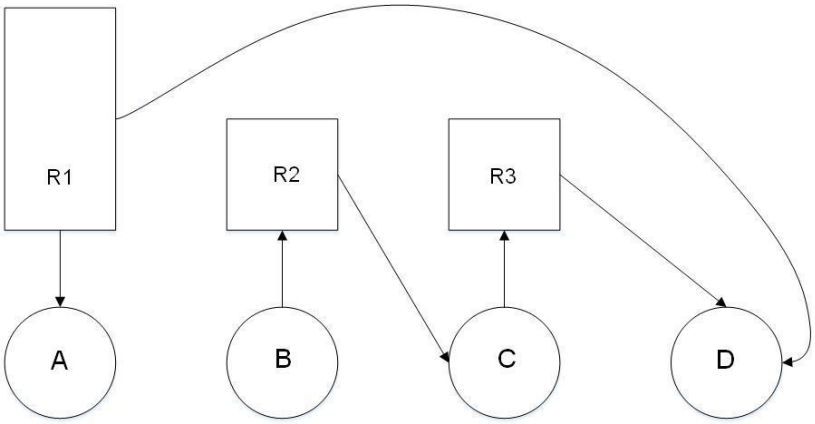
- информационные ресурсы – к ним относятся данные (объекты, переменные), системные данные (файлы), глобальные переменные (семафоры, мутексы).

- программные ресурсы – общий набор процедур, которые могут использоваться различными процессами.

Виды ресурсов с точки зрения реализации доступа к ним:

- совместно используемые (разделяемые), предполагают возможность параллельного доступа нескольких процессов:

Пример разделения ресурсов




Ресурс R1 выделен процессам A и D

Ресурс B запрашивает ресурс R2, который выделен процессу C

Ресурс R3 выделен ресурсу D, но он запращивается процессом C.

Налицо блокирование процессов B и C в ожидании ресурсов.

- неделимые ресурсы (каждый ресурс выделяется в исключительное пользование одному процессу). Реализуется последовательное выделение ресурса каждому процессу.

## Синхронные и асинхронные процессы

Асинхронные процессы выполняются независимо один от другого. Процесс A может быть родительским по отношению к процессу B (процесс A должен получить статус завершения от процесса B)

Способы выполнения асинхронных процессов: последовательно, параллельно, с перекрытием.

**Примеры** выполнения асинхронных процессов.

1.

| Процесс A |
| --- |
| Процесс B |
| Процесс C |

ожидание

2.

| Процесс A |
| --- |
| Процесс B |

ожидание

3.

| Процесс A |
| --- |
| Процесс B |

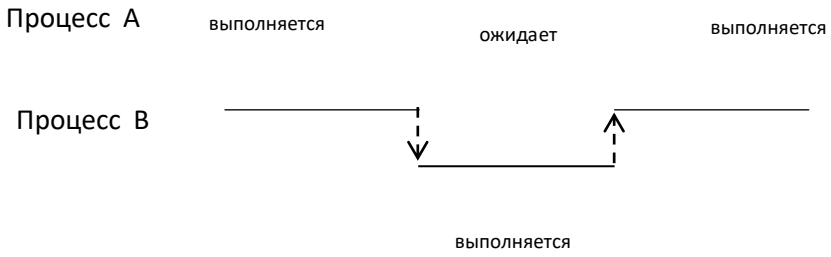
ожидание

Особенности асинхронных процессов:

- совместное использование ресурсов (требуется синхронизация (взаимодействие) при разделении ресурсов);

- совместное использование ресурсов возможно в ситуации, когда асинхронные процессы выполняются параллельно.

**Синхронные процессы** – процессы с чередующимся выполнением (например, блокирование процесса А до окончания выполнения процесса В).



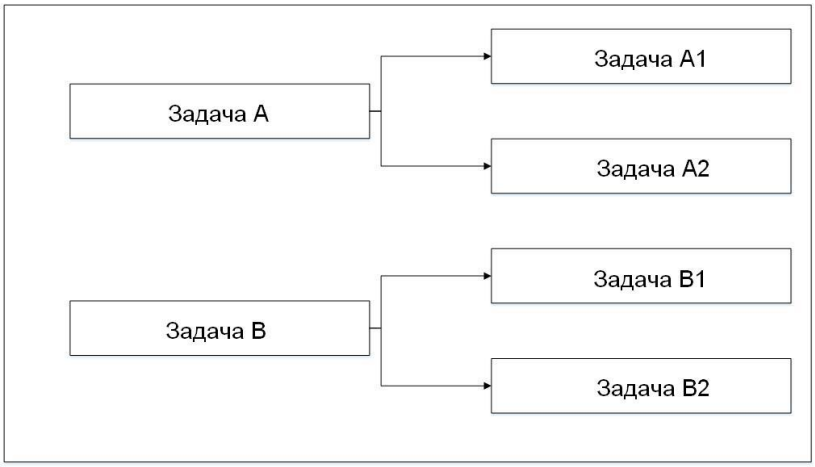
## Разделение программы на задачи для параллельного выполнения

Два уровня параллельной обработки – уровень потоков и уровень процессов. Три способа реализации параллелизма:

- выделение в программе основной задачи, которая инициирует (активизирует) другие задачи;


- разделение программы на множество отдельно выполняемых процессов;


- разделение программы на несколько подзадач, каждая из которых активизирует другие подзадачи;

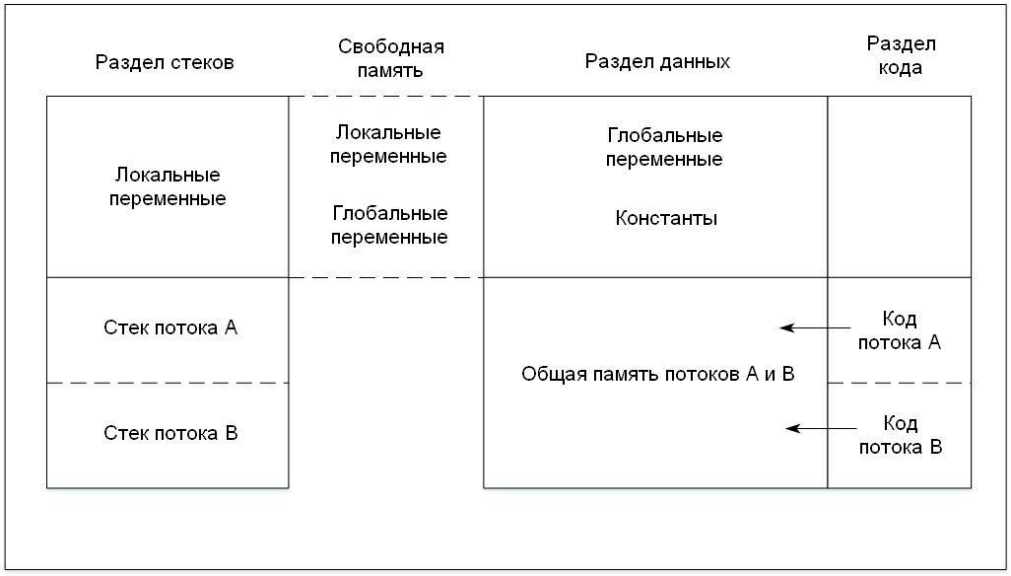


## Различие между процессами и потоками (при реализации модели с общей памятью)

Каждый процесс имеет собственные адресное пространство, потоки содержатся в адресном пространстве процессов. За счет того, что потоки содержатся в адресном пространстве одного процесса, то разделение общих ресурсов (переменных) реализуется достаточно просто.

Адресное пространство процесса А


Адресное пространство процесса А, формирующего потоки.



## 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;**

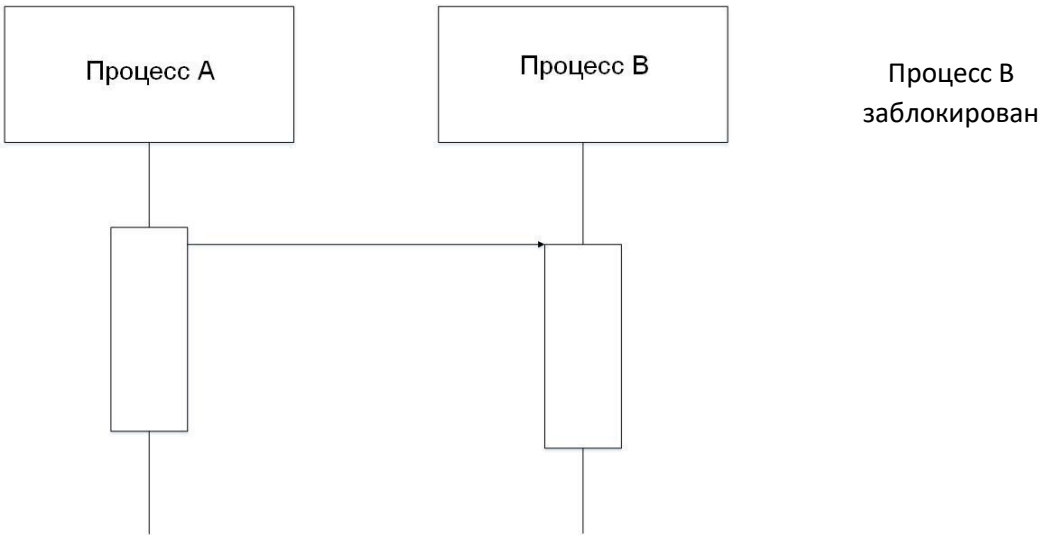
**- старт – финиш;**

**- финиш – финиш.**

## Взаимодействие «старт – старт»

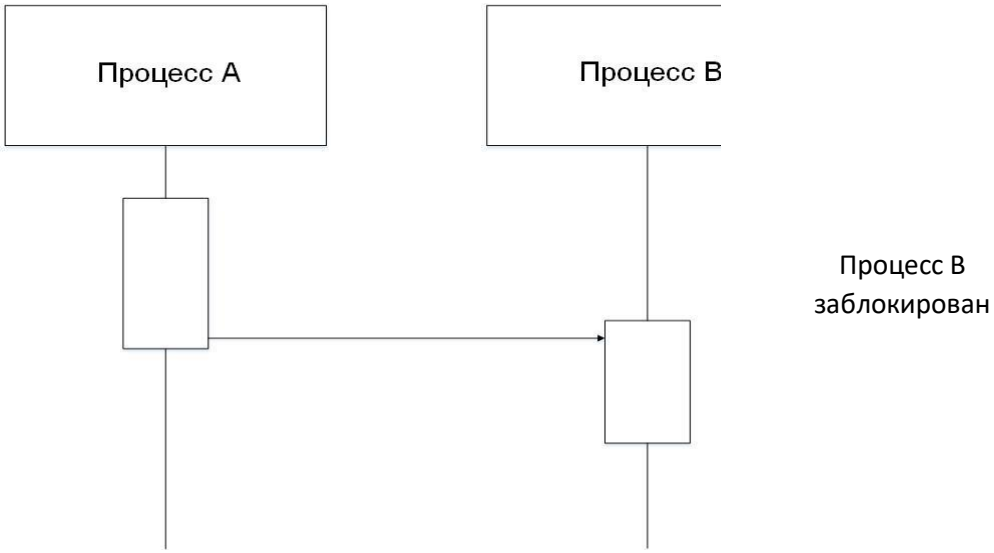
Процесс В активизируется (начинает выполнение) после активизации процесса А.

Данная схема синхронизации предполагает параллельное выполнение процессов.



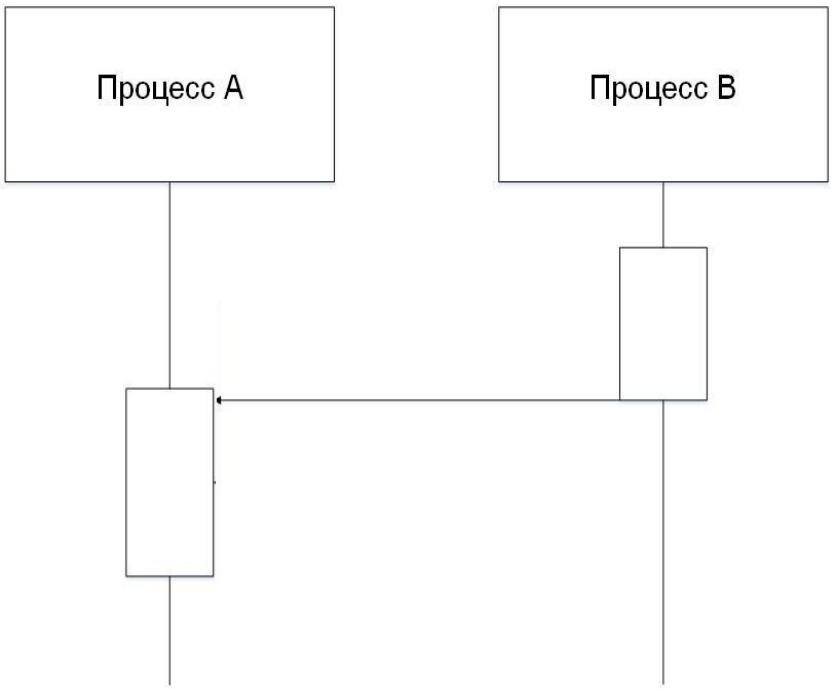

## Отношение синхронизации типа «финиш – старт»

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»**

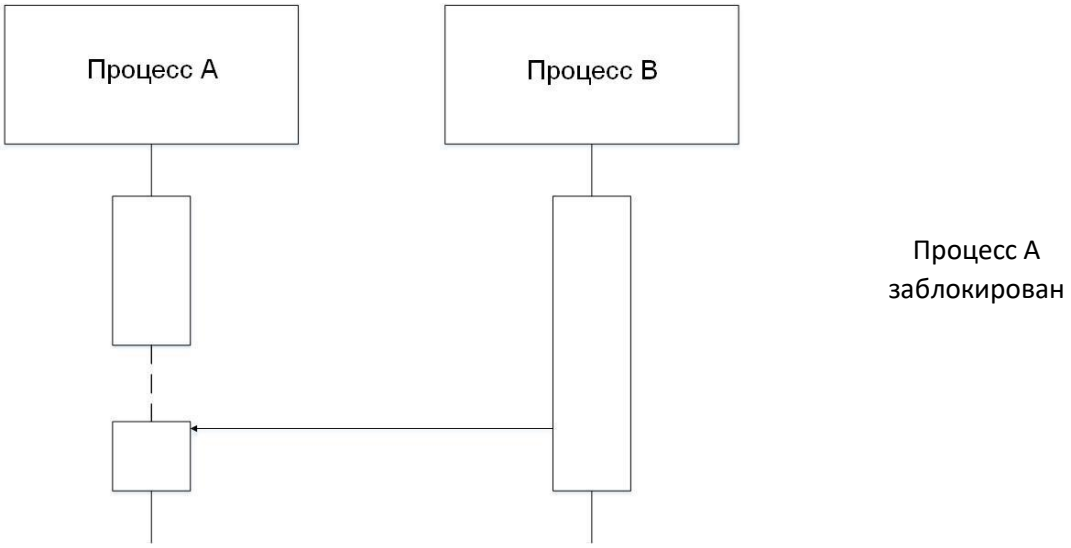
Процесс А не может начать своего выполнения до момента окончания процесса В.




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 распределено 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 – отправиться 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.

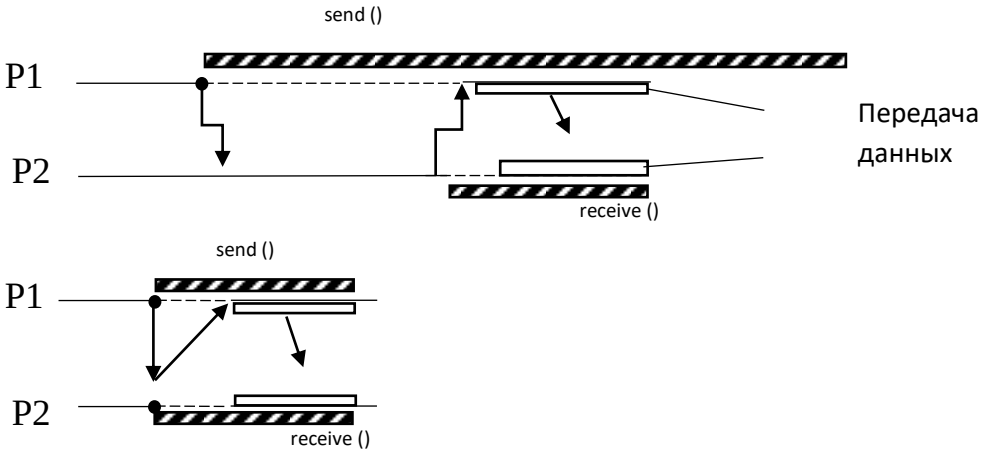
Последовательность передачи сообщений (сигналов) при передаче данных в рассматриваемом механизме взаимодействия:

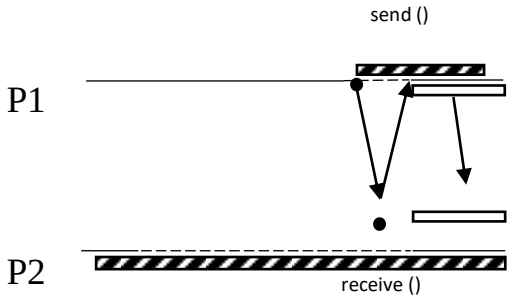
- при готовности отправителя к передаче данных (вход в вызов функции send ()) он отправляет запрос на передачу данных получателю и блокируется в ожидании получения ответа;

- 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 используется 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 () восполняется приблизительно в одно время.

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

**Пример синтаксиса при взаимной блокировке**

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

## 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);

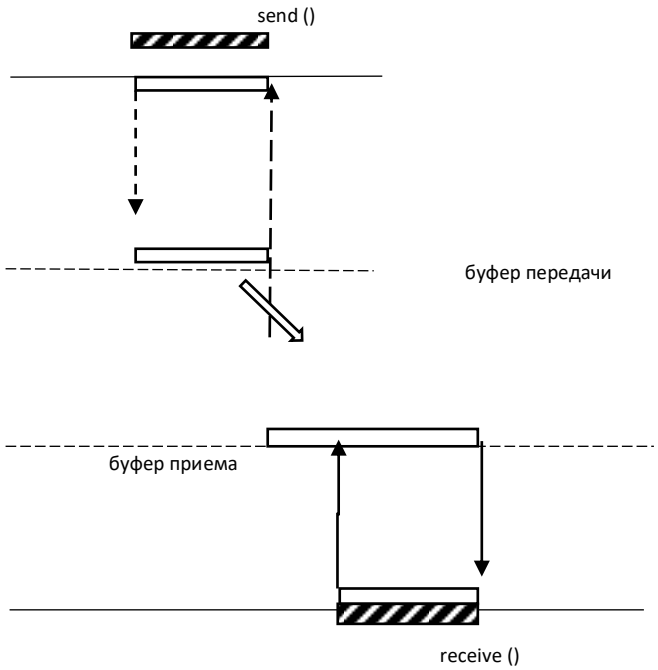
- независимое от пользовательских процессов копирование данных из буфера на передающей стороне в буфер на приемной стороне;

- при готовности к приему данных (вызов recv ()) получатель извлекает данные из буфера приема и размещает их в адресном пространстве процесса.

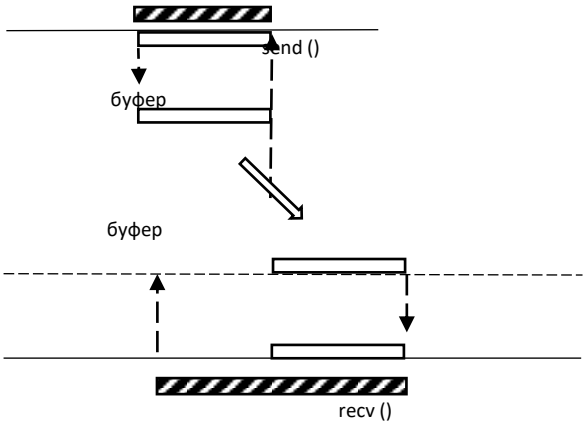
Обмен данными реализуется непосредственно системной распределенной обработки с использованием созданных предварительно буферов (без участия приложений).

Схема блокирующего буферизированного взаимодействия

а)



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:

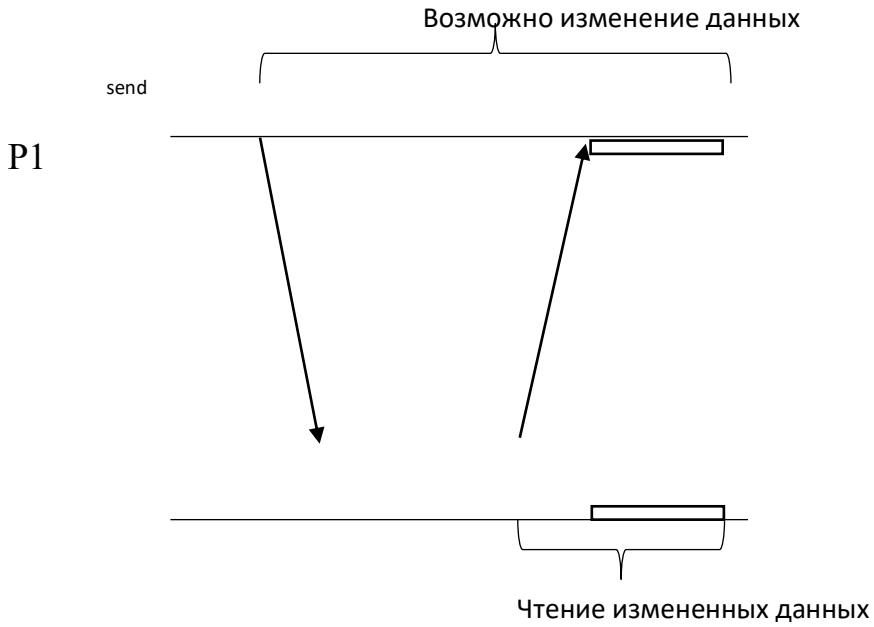
| P1 | P2 |
|---|---|
| receive (&a, 1, 2); | receive (&b, 1, 1); |
| send (&b, 1, 2); | send (&a, 1, 1); |

## 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 извлечение данных процесс-получатель подтверждает прием командой (вызовом) send ().