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).

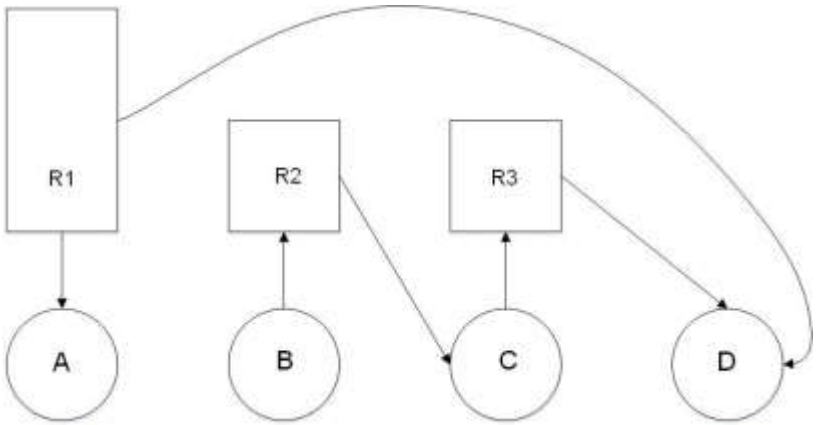
- информационные ресурсы – к ним относятся данные (объекты, переменные), системные данные (файлы), глобальные переменные (семафоры, мутексы).

- 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

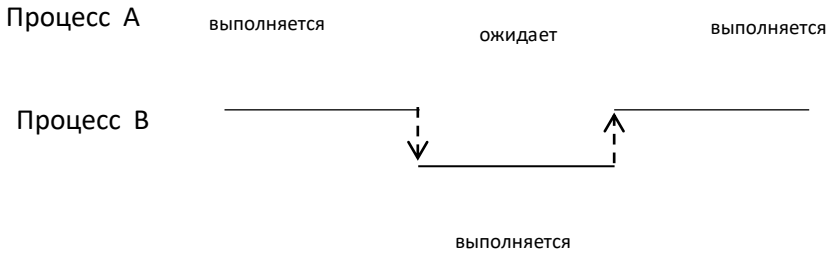
ожидание

Особенности асинхронных процессов:

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

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

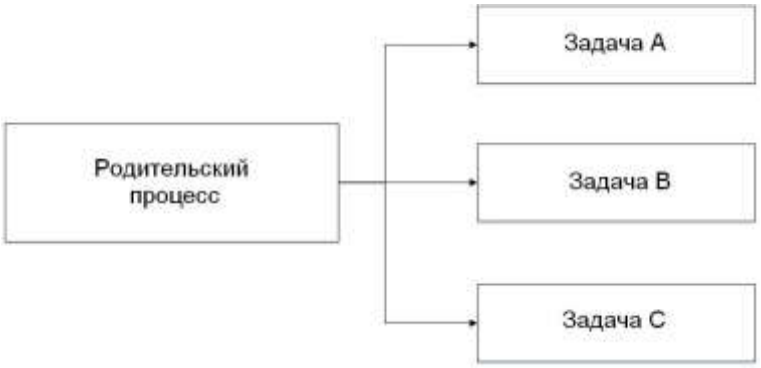
Синхронные процессы – процессы с чередующимся выполнением (например, блокирование процесса А до окончания выполнения процесса В).



Разделение программы на задачи для параллельного выполнения

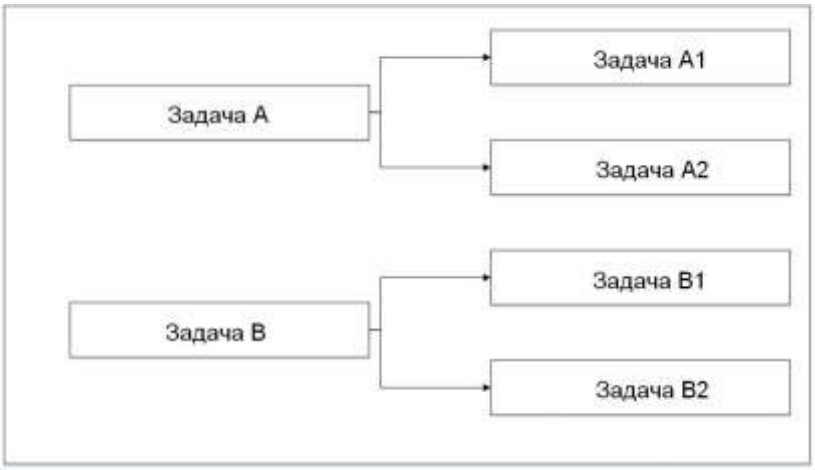
Два уровня параллельной обработки – уровень потоков и уровень процессов. Три способа реализации параллелизма:

- 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;

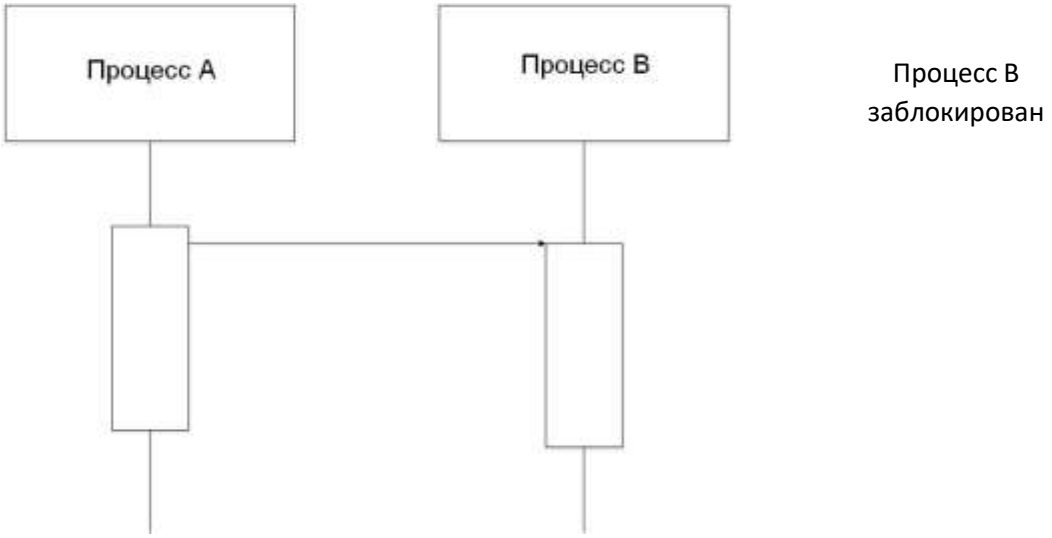
- старт – финиш;

- финиш – финиш.

Взаимодействие «старт – старт»

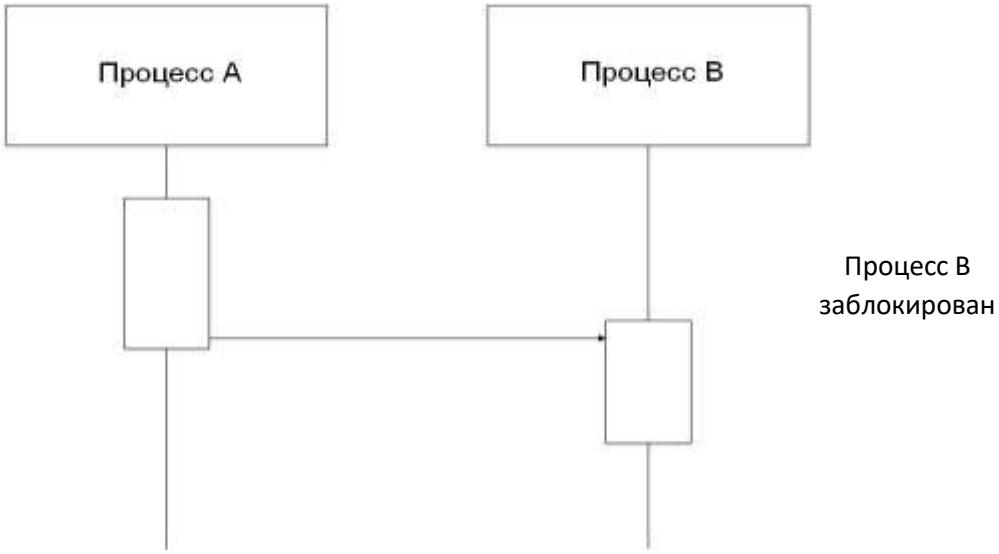
Процесс В активизируется (начинает выполнение) после активизации процесса А.

Данная схема синхронизации предполагает параллельное выполнение процессов.



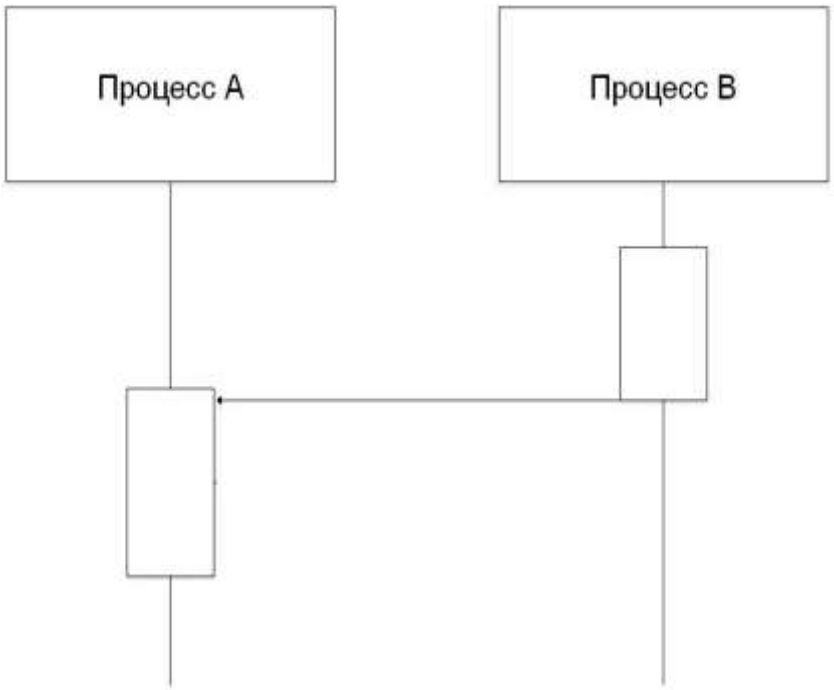
Отношение синхронизации типа «финиш – старт»

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»

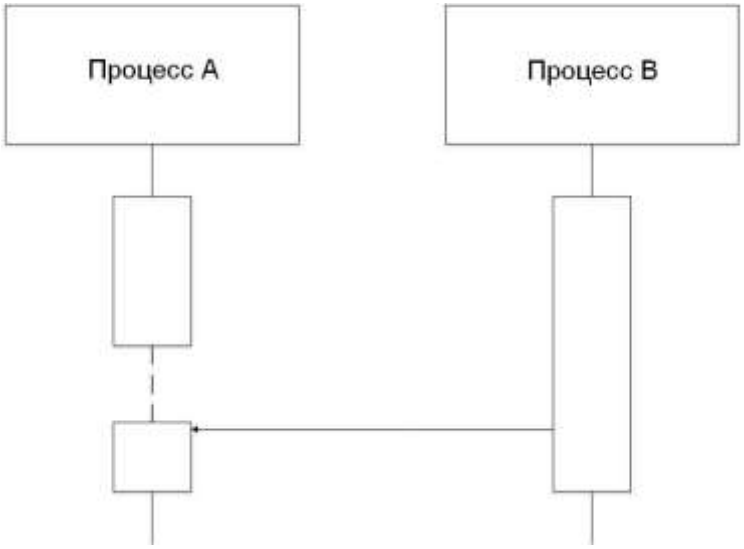
Процесс А не может начать своего выполнения до момента окончания процесса В.



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)



Процесс А заблокирован

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.

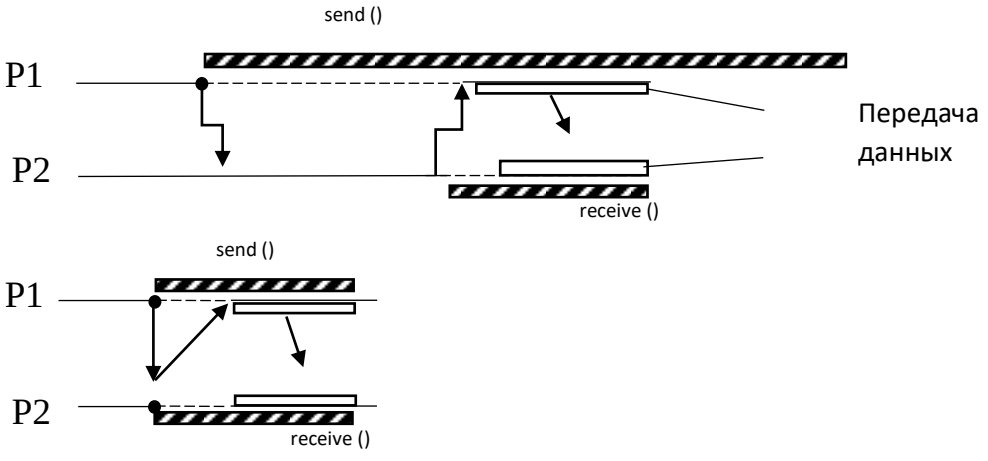
Последовательность передачи сообщений (сигналов) при передаче данных в рассматриваемом механизме взаимодействия:

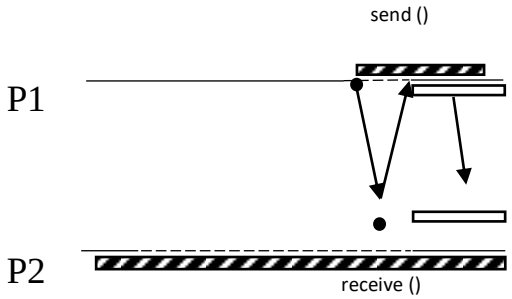
- при готовности отправителя к передаче данных (вход в вызов функции send ()) он отправляет запрос на передачу данных получателю и блокируется в ожидании получения ответа;

- 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);

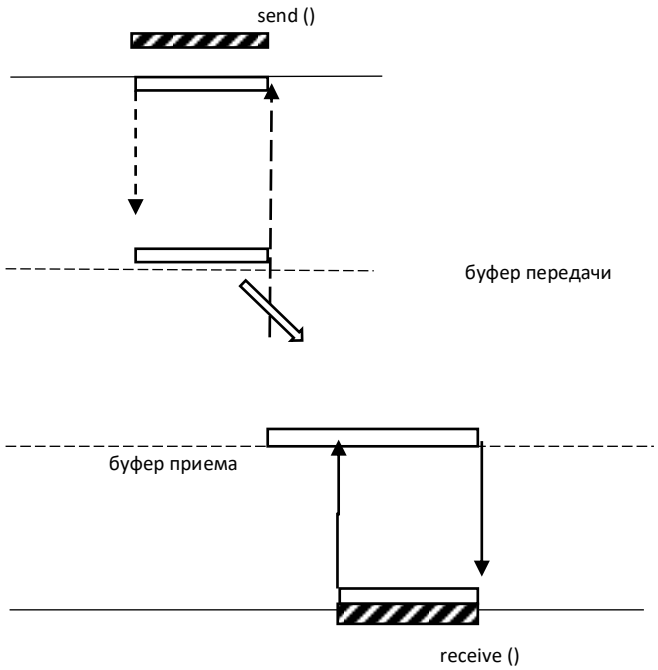
- независимое от пользовательских процессов копирование данных из буфера на передающей стороне в буфер на приемной стороне;

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

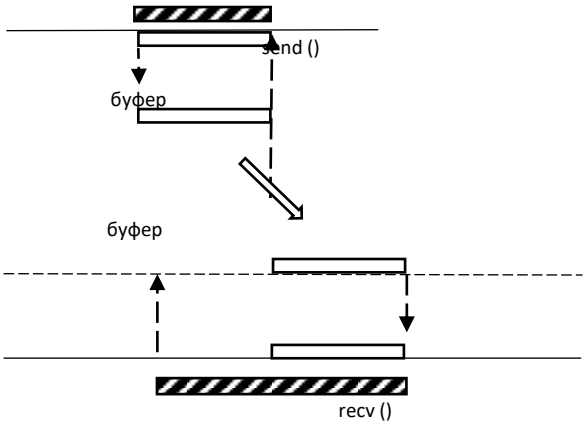
Обмен данными реализуется непосредственно системной распределенной обработки с использованием созданных предварительно буферов (без участия приложений).

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:

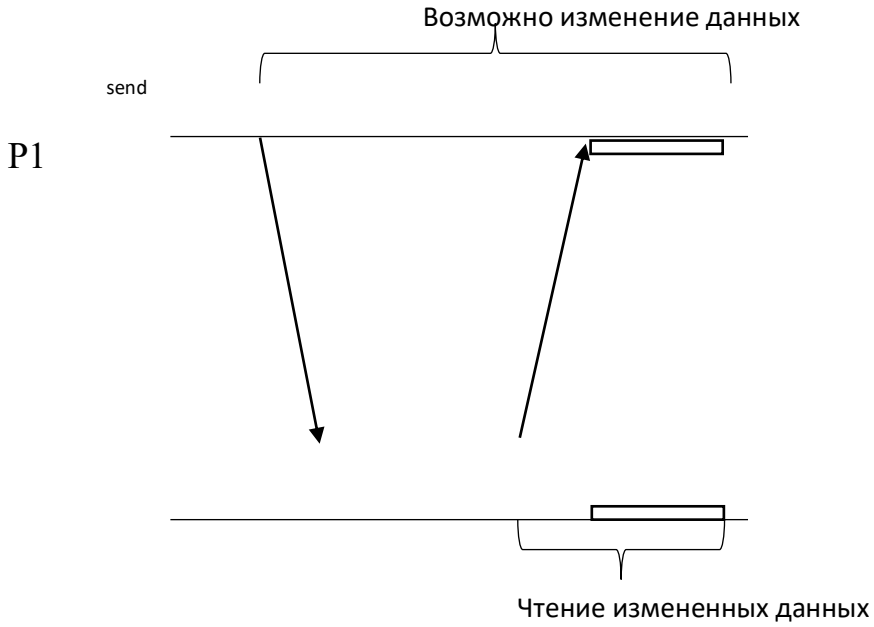
P1	P2
receive (&a, 1, 2);	receive (&b, 1, 1);
send (&b, 1, 2);	send (&a, 1, 1);

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.

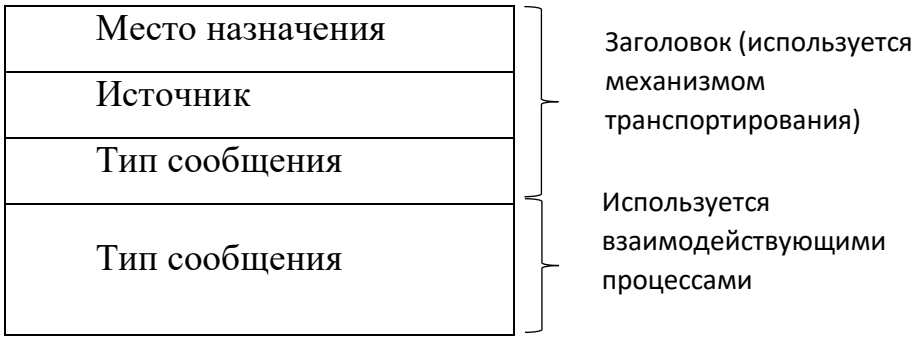
Схема неблокирующей передачи



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

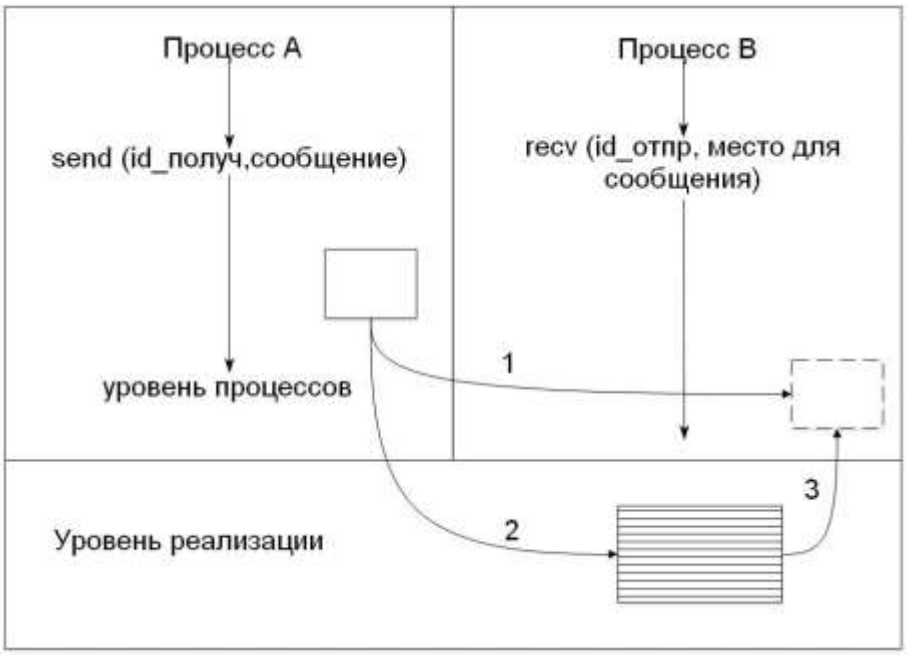
Взаимодействие распределенных процессов посредством передачи сообщений (механизм взаимодействия).

Состав сообщения:



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:

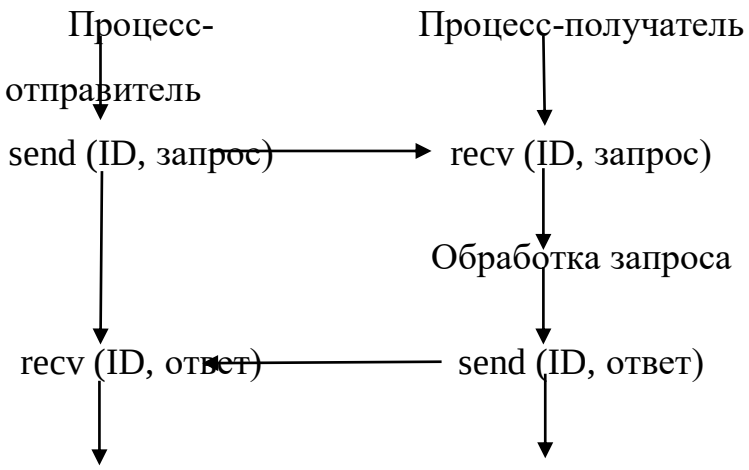
send (ID_получателя, сообщение-запрос);

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