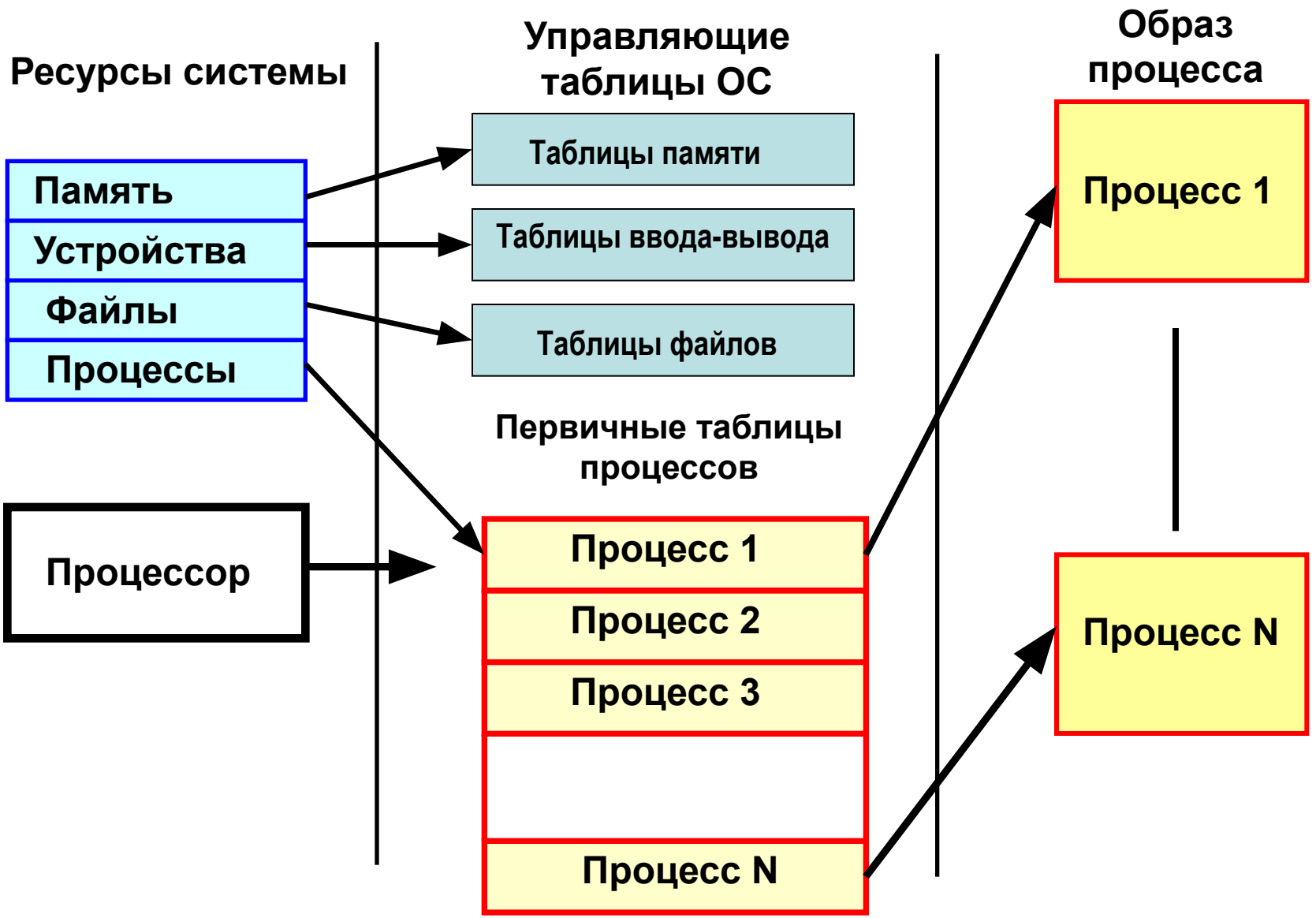
Процессы и потоки

Тема 2

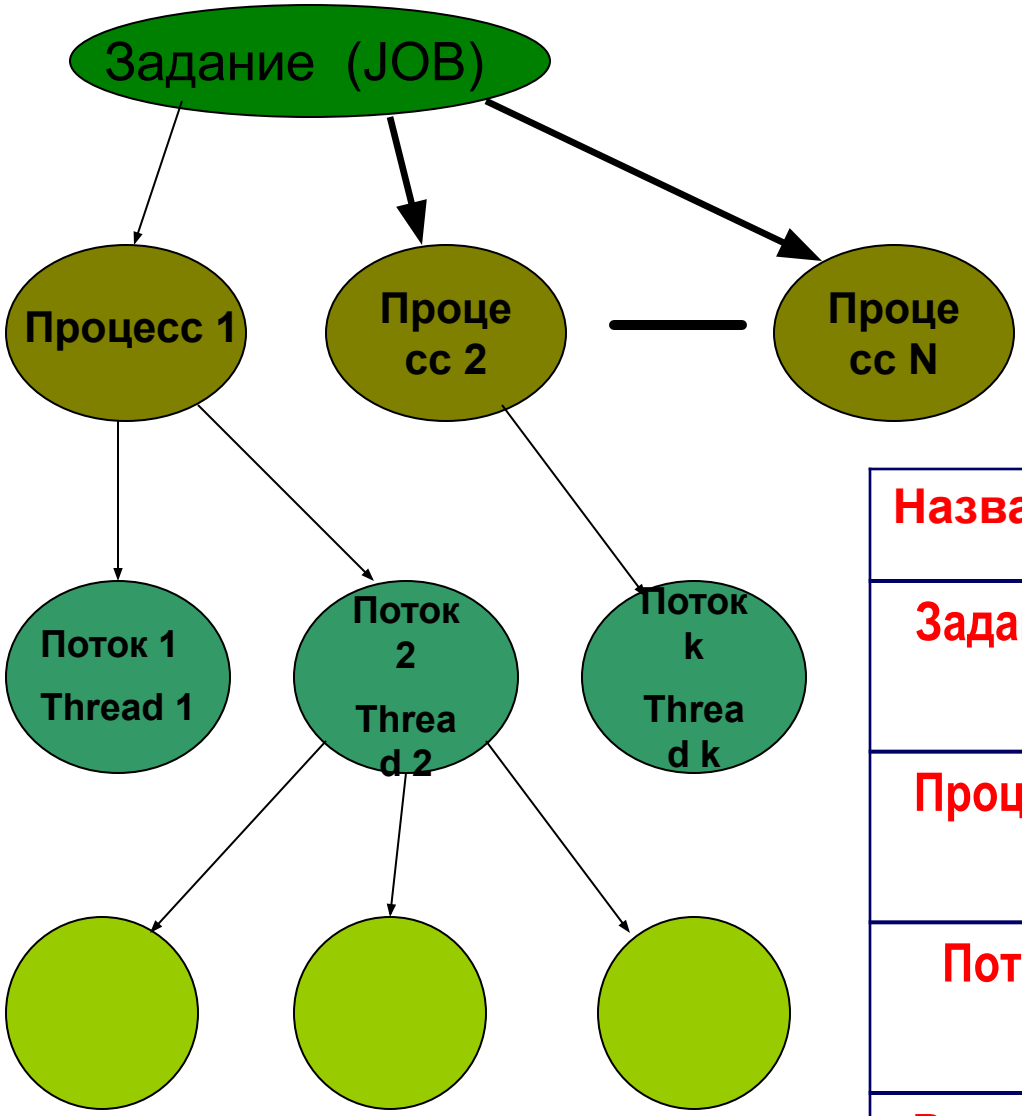

Ресурсы системы
Управляющие таблицы ОС
Образ процесса
Память
Устройства
Файлы
Процессы
Процессор
Таблицы памяти
Таблицы ввода-вывода
Таблицы файлов
Первичные таблицы процессов
Процесс 1
Процесс 2
Процесс 3
Процесс N
Процесс 1
Процесс N

Управление процессами

- *Процесс* (*process) -* **это программа при ее исполнении. Для процесса требуется ряд *ресурсов*, включая время процессора, память, файлы, устройства ввода-вывода, сетевые устройства и др.**
- **Обычно при создании процесса для него создается новое пространство виртуальной памяти**
- **ОС отвечает за следующие действия, связанные с управлением процессами:**
  - **Создание и удаление процессов.**
  - **Приостановка и возобновление процессов.**
  - **Обеспечение механизмов для:**
    - **Синхронизации процессов (семафоры, мониторы и др.)**
    - **Взаимодействия процессов (условные переменные, события и др.)**

Понятие процесса

- ОС исполняет множество классов программ:
  - Пакетная система (batch system) – задания (jobs)
  - Система с разделением времени – пользовательские программы (задачи – tasks)
- Во многих учебниках термины “задание” и “процесс” – почти синонимы
- Процесс – программа при ее выполнении; он должен выполняться последовательно
- Процесс включает:
  - Счетчик команд (program counter)
  - Стек (stack)
  - Секцию данных (data section)

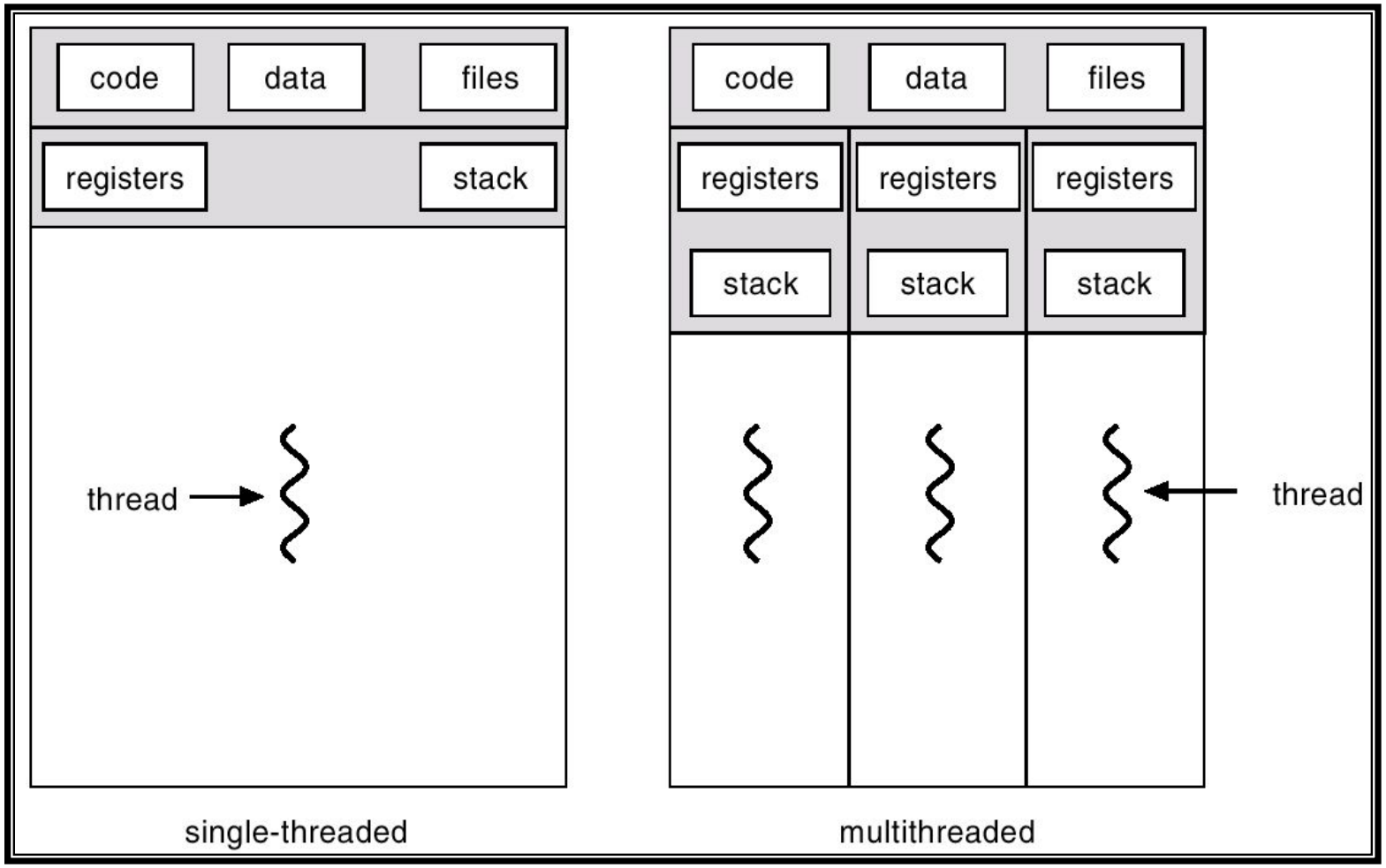


Волокна (Fibers)

Объекты

Название	Описание
Задание	Набор процессов с общими квотами и лимитами
Процесс	Контейнер для ресурсов и потоков
Поток	Исполнение кода в процессе
Волокно	Облегченный поток, полностью управляемый в пространстве пользователя

Однопоточные и многопоточные процессы

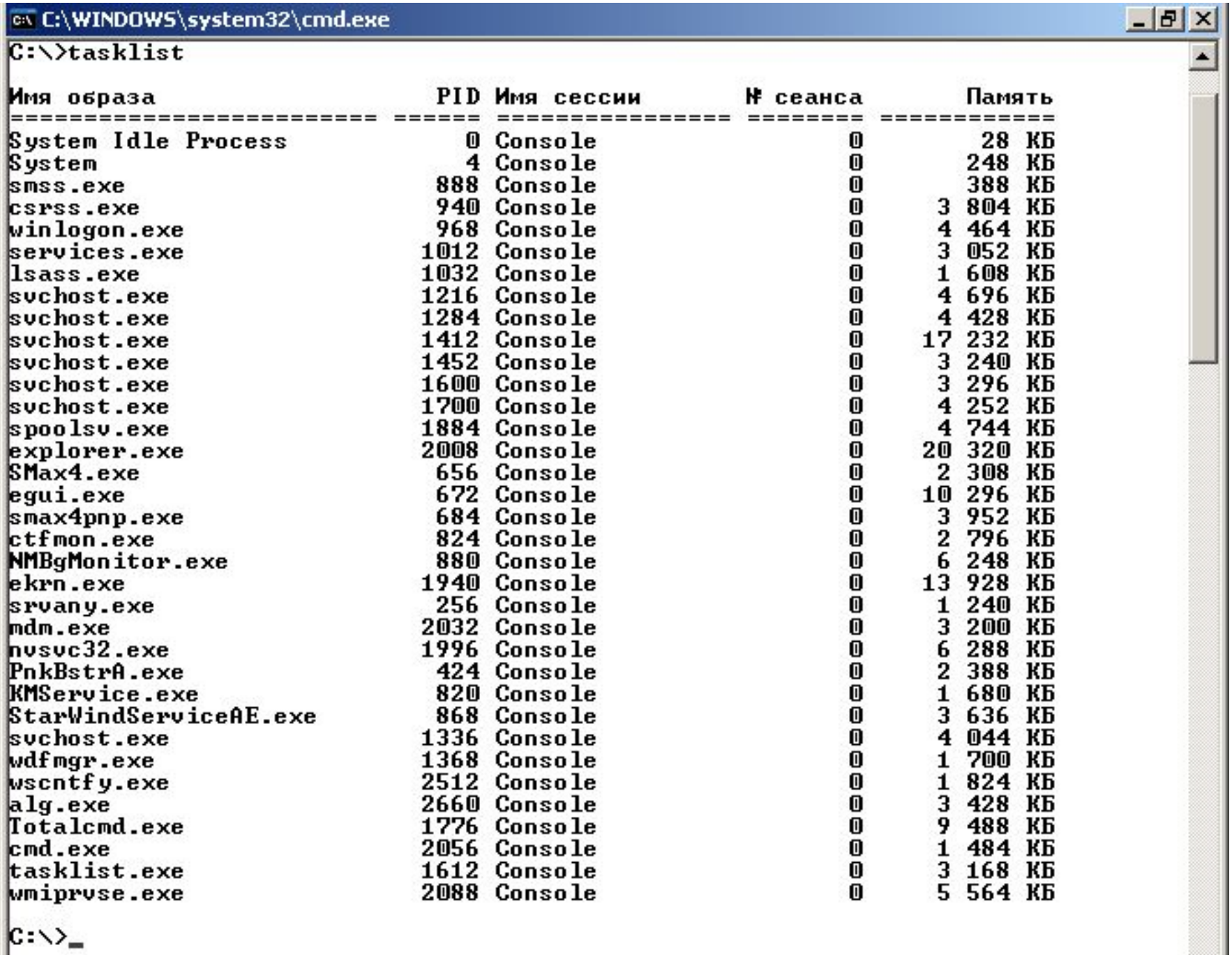


Образ процесса: программа, данные, стек и атрибуты процесса

Информация	Описание
Данные пользователя	**Изменяемая часть пользовательского адресного пространства (данные программы, пользовательский стек и модифицируемый код)**
Пользовательская программа	**Программа, которую нужно выполнить**
Системный стек	**Один или несколько системных стеков для хранения параметров и адресов вызова процедур и системных служб**
Управляющий блок процесса	**Данные, необходимые ОС для управления процессом: 1) дескриптор процесса, 2) контекст процесса**

Дескриптор процесса содержит:

1. Информацию по идентификации процесса (идентификатор процесса, идентификатор пользователя, идентификаторы родительского и дочерних процессов).
2. Информацию по состоянию процесса
3. Информацию, используемую для управления процессом


C:\WINDOWS\system32\cmd.exe
C:\>tasklist
Имя образа PID Имя сессии № сеанса Память
System Idle Process 0 Console 0 28 КБ
System 4 Console 0 248 КБ
smss.exe 888 Console 0 388 КБ
csrss.exe 940 Console 0 3 804 КБ
winlogon.exe 968 Console 0 4 464 КБ
services.exe 1012 Console 0 3 052 КБ
lsass.exe 1032 Console 0 1 608 КБ
svchost.exe 1216 Console 0 4 696 КБ
svchost.exe 1284 Console 0 4 428 КБ
svchost.exe 1412 Console 0 17 232 КБ
svchost.exe 1452 Console 0 3 240 КБ
svchost.exe 1600 Console 0 3 296 КБ
svchost.exe 1700 Console 0 4 252 КБ
spoolsv.exe 1884 Console 0 4 744 КБ
explorer.exe 2008 Console 0 20 320 КБ
SMax4.exe 656 Console 0 2 308 КБ
egui.exe 672 Console 0 10 296 КБ
smax4pnp.exe 684 Console 0 3 952 КБ
ctfmon.exe 824 Console 0 2 796 КБ
NMBgMonitor.exe 880 Console 0 6 248 КБ
ekrn.exe 1940 Console 0 13 928 КБ
srvany.exe 256 Console 0 1 240 КБ
mdm.exe 2032 Console 0 3 200 КБ
nvsvc32.exe 1996 Console 0 6 288 КБ
PnkBstrA.exe 424 Console 0 2 388 КБ
KMService.exe 820 Console 0 1 680 КБ
StarWindServiceAE.exe 868 Console 0 3 636 КБ
svchost.exe 1336 Console 0 4 044 КБ
wdfmgr.exe 1368 Console 0 1 700 КБ
wscntfy.exe 2512 Console 0 1 824 КБ
alg.exe 2660 Console 0 3 428 КБ
Totalcmd.exe 1776 Console 0 9 488 КБ
cmd.exe 2056 Console 0 1 484 КБ
tasklist.exe 1612 Console 0 3 168 КБ
wmiprvse.exe 2088 Console 0 5 564 КБ
C:\>

C:\WINDOWS\system32\cmd.exe

```
C:\>tasklist /?

TASKLIST [/S <система> [/U <имя пользователя> [/P [<пароль>]]]]
         [/M [<модуль>] | /SVC | /V] [/FI <фильтр>] [/FO <формат>] [/NH]

Описание:
    Отображает список приложений и связанные с ними задачи/процессы, которые
    исполняются в текущий момент на локальном или удаленном компьютере.

Список параметров:
   /S     <система>        Подключаемый удаленный компьютер.

   /U  [<домен>\]<пользователь>  Пользовательский контекст, в котором
                                 должна выполняться эта команда.

   /P     [<пароль>]       Пароль для этого пользовательского контекста.
                           Запрашивает ввод пароля, если он не задан.

   /M     [<модуль>]       Отображение всех задач, которые загрузили модули
                           DLL, отвечающие указанному критерию.
                           Если имя модуля не указано, отображаются
                           все модули загруженные каждой задачей.

   /SVC                    Отображение служб для каждого процесса.

   /V                      Указывает, что должна отображаться
                           подробная информация.

   /FI    <фильтр>         Отображение списка задач, которые отвечают
                           указанному в фильтре критерию.

   /FO    <формат>         Описание формата выходного файла.
                           Допустимые значения: "TABLE", "LIST", "CSV".

   /NH                     Отключение отображения заголовка "Column Header"
                           в выходных данных.
                           Допустимо для форматов "TABLE" и "CSV".

   /?                      Вывод справки по использованию.

Фильтры:
    Имя фильтра     Допустимые операторы           Допустимые значения
    -----------     --------------------           -------------------
    STATUS          eq, ne                         RUNNING | NOT RESPONDING
    IMAGENAME       eq, ne                         Имя образа
    PID             eq, ne, gt, lt, ge, le         Значение PID
    SESSION         eq, ne, gt, lt, ge, le         Номер сессии
    SESSIONNAME     eq, ne                         Имя сессии
    CPUTIME         eq, ne, gt, lt, ge, le         Время CPU в формате
                                                   hh:mm:ss.
                                                   hh - часы,
                                                   mm - минуты, ss - секунды
    MEMUSAGE        eq, ne, gt, lt, ge, le         Использование памяти в КБ
    USERNAME        eq, ne                         Имя пользователя в формате
                                                   [<домен>\<пользователь>]
    SERVICES        eq, ne                         Имя службы
    WINDOWTITLE     eq, ne                         Название окна
    MODULES         eq, ne                         Имя DLL

Примеры:
    TASKLIST
    TASKLIST /M
```

Информация по состоянию и управлению процессом

- Состояние процесса, определяющее его готовность к выполнению (выполняющийся, готовый к выполнению, ожидающий события, приостановленный);
- Данные о приоритете (текущий, по умолчанию, максимально возможный);
- Информация о событиях – идентификация события, наступление которого позволит продолжить выполнение процесса;
- Указатели, позволяющие определить расположение образа процесса в оперативной памяти и на диске;
- Указатели на другие процессы (находящиеся в очереди на выполнение);
- Флаги,сигналы и сообщения, имеющие отношение к обмену информацией между двумя независимыми процессами;
- Данные о привилегиях, определяющие прав доступа к определенной области памяти или возможности выполнять определенные виды команд, использовать системные утилиты и службы;
- Указатели на ресурсы, которыми управляет процесс;
- Сведения по использованию ресурсов и процессора;
- Информация, связанная с планированием.

КОНТЕКСТ ПРОЦЕССА

- **Содержимое регистров процессора, доступных пользователю (обычно 8 – 32 регистра и до 100 регистров в RISC – процессорах);**
- **Содержимое счетчика команд;**
- **Состояние управляющих регистров и регистров состояния;**
- **Коды условия, отражающие результат выполнения последней арифметической или логической операции (например, равенство нулю,переполнение);**
- **Указатели стеков,хранящие параметры и адреса вызова процедур и системных служб.**

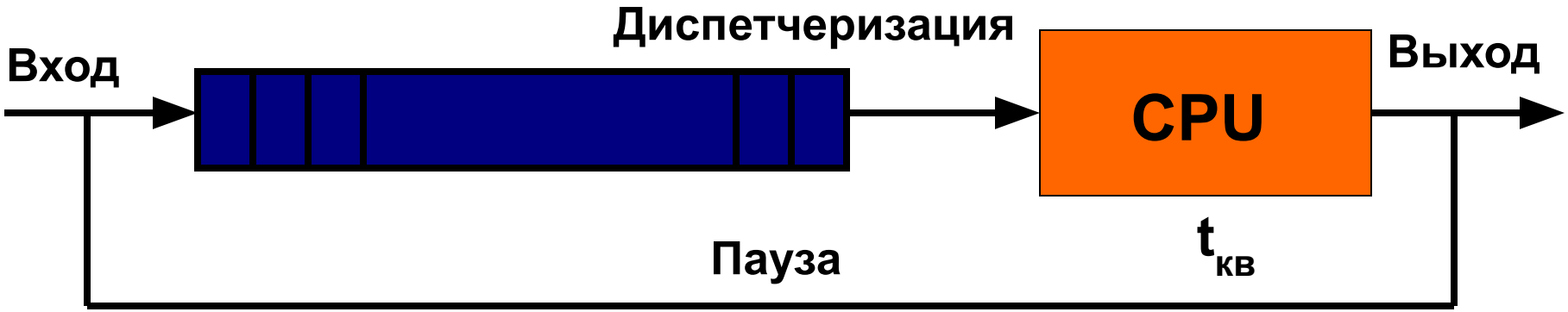
Значительная часть этой информации фиксируется в виде слова состояния программы PSW (program status word – EFLAGS в процессоре Pentium).

Переключение контекста процесса (context switch)

- Когда процессор переключается на другой процесс, система должна сохранить состояние старого процесса и загрузить сохраненное состояние для нового процесса
- Переключение контекста относится к накладным расходам (overhead); система не выполняет никаких полезных действий при переключении с одного процесса на другой
- Время зависит от аппаратной поддержки.

Простейшая модель процесса

Граф состояний и переходов



Потоки и их модели

Описатель потока: блок управления потоком и контекст потока (в многопоточной системе процессы контекстов не имеют).

Способы реализации пакета потоков:

- в пространстве пользователя (user – level threads – ULT);
- в ядре (kernel – level threads – KLT).

Типичный граф состояния потока

- В мультипрограммной системе поток может находиться в одном из трех основных состояний:
- *выполнение* − активное состояние потока, во время которого поток обладает всеми необходимыми ресурсами и непосредственно выполняется процессором;
- *ожидание* − пассивное состояние потока, находясь в котором поток заблокирован по своим внутренним причинам (ждет осуществления некоторого события, например завершения операции ввода-вывода, получения сообщения от другого потока или освобождения какого-либо необходимого ему ресурса);
- *готовность* − также пассивное состояние потока, но в этом случае поток заблокирован в связи с внешним по отношению к нему обстоятельством (имеет все требуемые для него ресурсы, готов выполняться, однако процессор занят выполнением другого потока).

- Переход от выполнения одного потока к другому осуществляется в результате планирования и диспетчеризации. Планирование потоков включает в себя решение двух задач:
- определение момента времени для смены текущего активного потока;
- выбор для выполнения потока из очереди готовых потоков.

Диспетчеризация сводится к следующему:

- *сохранение* контекста текущего потока, который требуется сменить;
- *загрузка* контекста нового потока, выбранного в результате планирования;
- *запуск* нового потока на выполнение.

Алгоритмы планирования потоков

1. **Невытесняющие (non-preemptive)**

- **планирование распределяется между ОС и прикладными программами;**
- **необходимость частых передач управлений ОС, в противном случае возможна монополизация процессора приложением;**
- **зависания приложений могут привести к краху системы**

2. Вытесняющие (preemptive)

- **функции планирования сосредоточены в ОС;**
- **планирование на основе квантования процессорного времени;**
- **планирование на основе приоритетов потоков: статических, динамических, абсолютных, относительных, смешанных;**

- В большинстве операционных систем универсального назначения планирование осуществляется *динамически* (on-line), то есть решения принимаются во время работы системы на основе анализа текущей ситуации.
- Планировщик называется *статическим*, если он принимает решения о планировании не во время работы системы, а заранее (off-line).

Критерии диспетчеризации

- **Использование процессора – поддержание его в режиме занятости, насколько это возможно**
- **Пропускная способность (throughput) – число процессов, завершающих свое выполнение за единицу времени**
- **Время обработки (turnaround time) – время, необходимое для исполнения какого-либо процесса**
- **Время ожидания (waiting time)– время, которое процесс ждет в очереди процессов, готовых к выполнению**
- **Время ответа (response time) – время, требуемое от момента первого запроса до первого ответа (для среды разделения времени)**

Алгоритмы планирования, основанные на квантовании

Каждому потоку поочередно для выполнения предоставляется ограниченный непрерывный период процессорного времени – *квант.*

Смена активного потока происходит, если:

- поток завершился и покинул систему;
- произошла ошибка;
- поток перешел в состояние ожидания;
- исчерпан квант процессорного времени, отведенный данному потоку.

- Поток, который исчерпал свой квант, переводится в состояние готовности и ожидает, когда ему будет предоставлен новый квант процессорного времени, а на выполнение в соответствии с определенным правилом выбирается новый поток из очереди готовых.
- Кванты, выделяемые потокам, могут быть одинаковыми для всех потоков или различными.

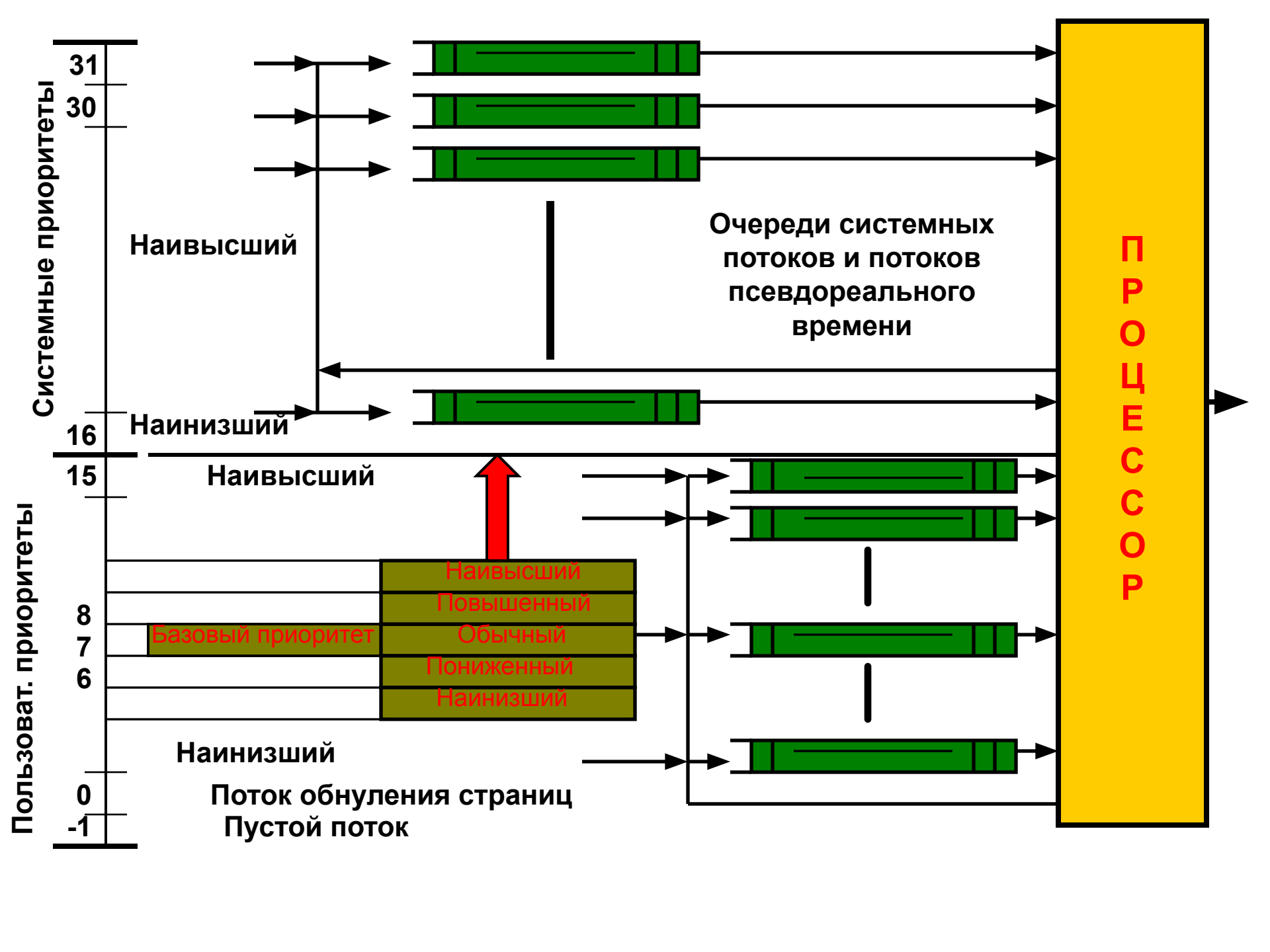
Стратегия Round Robin (RR) – “круговая система”

- Каждый процесс получает небольшой квант процессорного времени, обычно – 10-100 миллисекунд. После того, как это время закончено, процесс прерывается и помещается в конец очереди готовых процессов.
- Если всего *n* процессов в очереди готовых к выполнению, и квант времени - *q*, то каждый процесс получает 1/*n* процессорного времени порциями самое большее по *q* единиц за один раз. Ни один процесс не ждет больше, чем (*n*-1)*q* единиц времени.
- Производительность
  - *q* велико ⇒ FIFO
  - *q* мало ⇒ *q* должно быть бо́льшим, по сравнению со временем контекстного переключения, иначе слишком велики накладные расходы

Алгоритмы планирования, основанные на приоритетах

- *Приоритет* – это число, характеризующее степень привилегированности потока при использовании ресурсов вычислительной машины, в частности, процессорного времени: чем выше приоритет, тем выше привилегии, тем меньше времени будет проводить поток в очередях.

- Во многих ОС предусматривается возможность изменения приоритетов в течение жизни потока.
- В этом случае приоритеты называются *динамическими* в отличие от неизменяемых, *фиксированных,* приоритетов.


Системные приоритеты
31
30
Наивысший
Наинизший
16
Очереди системных потоков и потоков псевдореального времени
ПРОЦЕССОР
Пользоват. приоритеты
15
Наивысший
8
7
6
Базовый приоритет
Наивысший
Повышенный
Обычный
Пониженный
Наинизший
Наинизший
0
Поток обнуления страниц
-1
Пустой поток

Изменение базового приоритета потока

Увеличение приоритета

+ 1 – завершение ввода-вывода по диску;
+ 2 – для последовательной линии;
+ 6 – клавиатура;
+ 8 – звуковая карта;
+ 2 – снимается блокировка по семафору (для потока переднего плана);
+ 1 - снимается блокировка по семафору (для потока непереднего плана);
приоритет 15 на 2 кванта процессора, если готовый к выполнению поток простаивает более некоторого директивного времени.

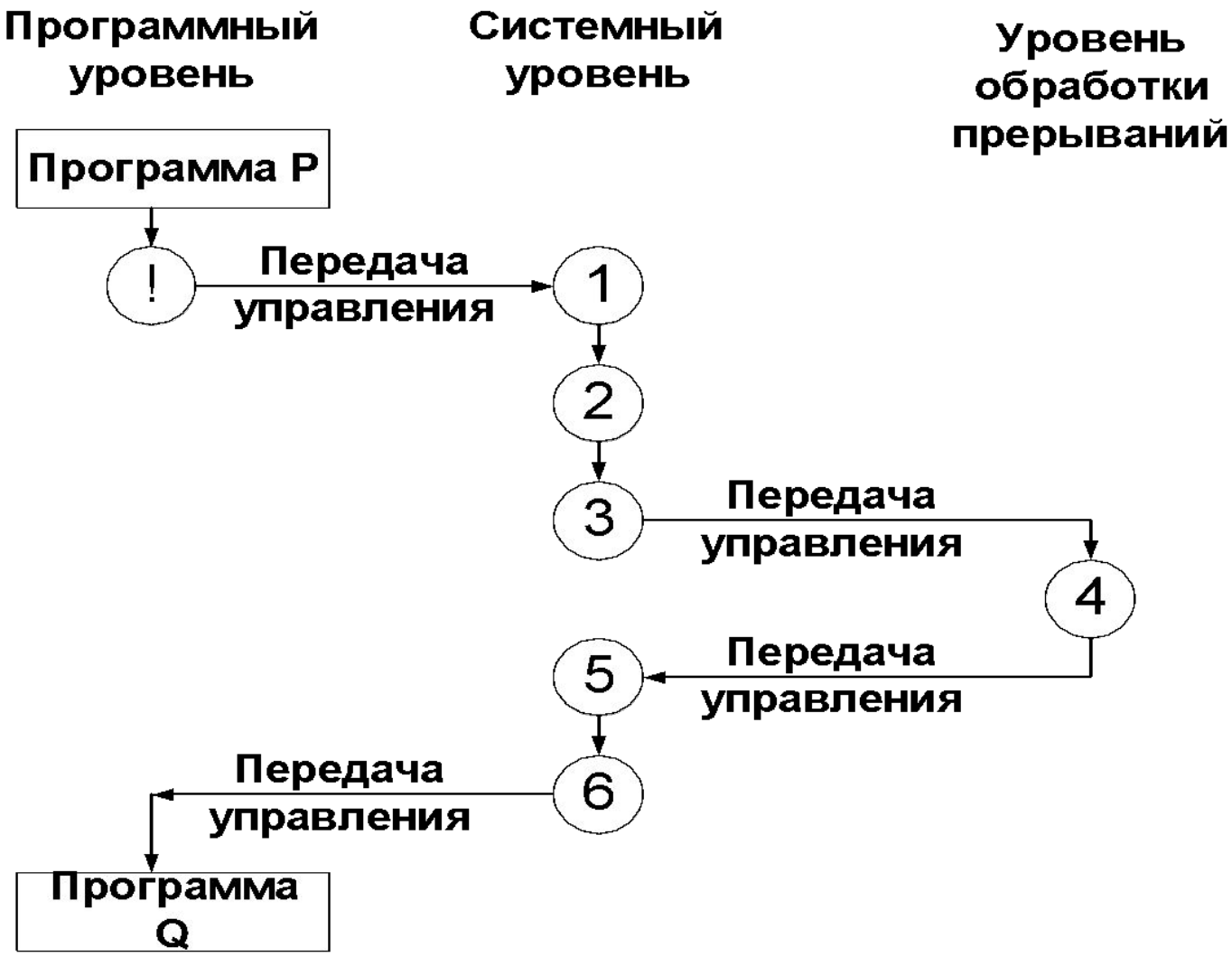
Уменьшение приоритета

- 1 – если полностью использован квант времени процессора (многократно, вплоть до базового приоритета).

Механизм прерываний

- *Прерывание* принудительная передача управления от выполняемой программы к системе, происходящая при возникновении определенного события.

Схема обработки прерываний



Обозначения;

! Прерывание (сигнал – установление факта прерывания)

1 Идентификация прерывания

2 Отключение всех других прерываний

3 Смена контекста_1 (сохранение состояния прерванного процесса из системных регистров, загрузка в системные регистры контекста соответствующего обработчика прерываний)

4 Обработка прерывания, включающая определение программы Q, которую следует запустить;

5 Смена контекста_2 (загрузка в системные регистры контекста определенной на предыдущем шаге программы Q)

6 Установка прежнего режима системы прерываний.

Назначение и типы прерываний

В зависимости от источника прерывания делятся на три больших класса:

- внешние;
- внутренние;
- программные.
- *Внешние* прерывания могут возникать в результате действий пользователя или оператора за терминалом, или же в результате поступления сигналов от аппаратных устройств — сигналов завершения операций ввода-вывода, вырабатываемых контроллерами внешних устройств компьютера . Внешние прерывания называют также *аппаратными.*
- Данный класс прерываний является *асинхронным* по отношению к потоку инструкций прерываемой программы.

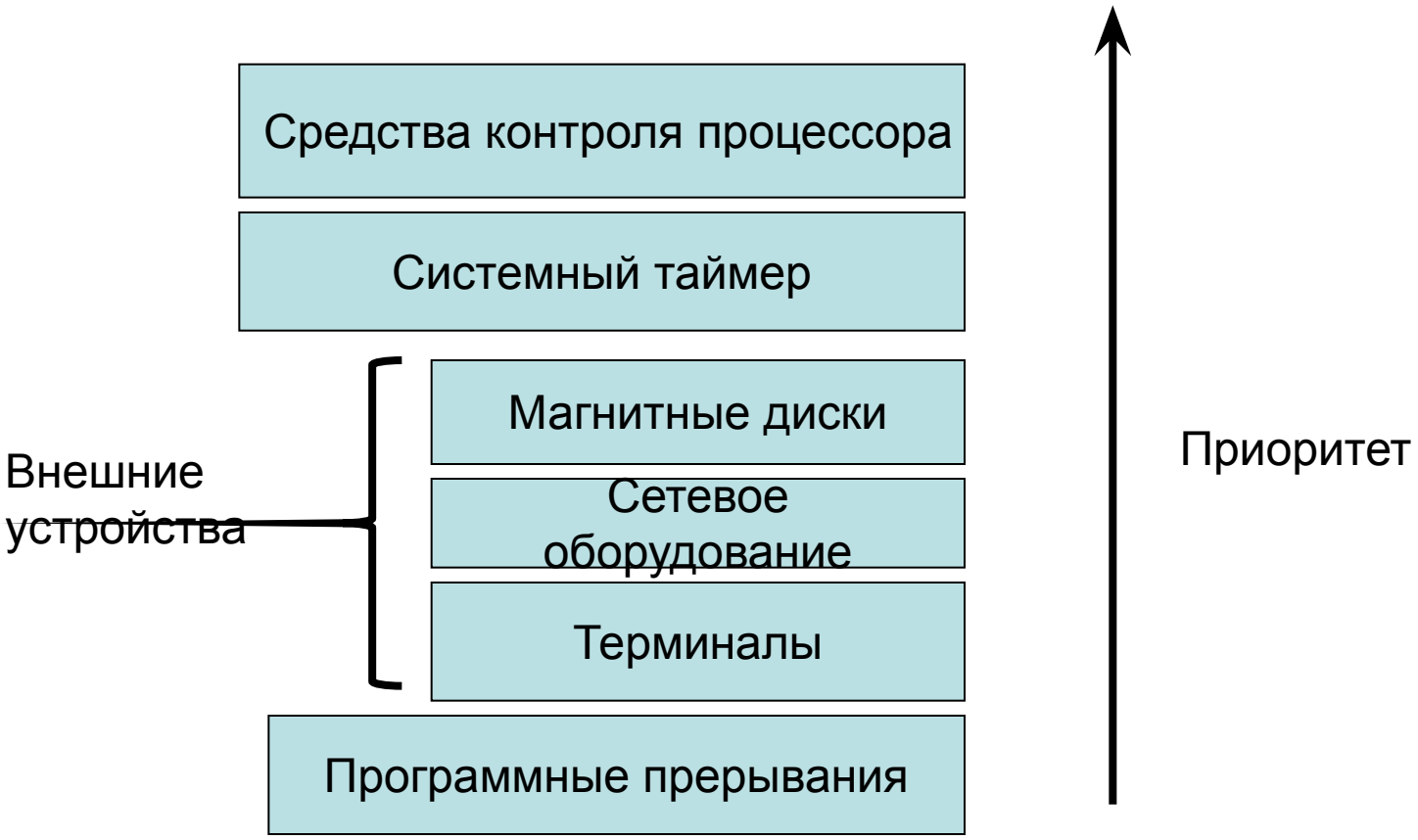
- *Внутренние* прерывания, называемые также *исключениями (exeption),* происходят *синхронно* выполнению программы при появлении аварийной ситуации в ходе исполнения некоторой инструкции программы.
- Примерами исключений являются деление на нуль, ошибки защиты памяти, обращения по несуществующему адресу, попытка выполнить привилегированную инструкцию в пользовательском режиме и т. п.

- *Программные* прерывания отличаются от предыдущих двух классов тем, что они по своей сути не являются «истинными» прерываниями. Программное прерывание возникает при выполнении особой команды процессора, .выполнение которой имитирует прерывание, то есть переход на новую последовательность инструкций.

- Прерываниям приписывается приоритет, с помощью которого они ранжируются по степени важности и срочности. О прерываниях, имеющих одинаковое значение приоритета, говорят, что они относятся к одному *уровню приоритета* прерываний.

- Процедуры, вызываемые по прерываниям, обычно называют *обработчиками прерываний,* или *процедурами обслуживания прерываний (Interrupt Servie Routine, ISR).* Аппаратные прерывания обрабатываются драйверами соответствующих внешних устройств, исключения — специальными модулями ядра, а программные прерывания — процедурами ОС, обслуживающими системные вызовы.
- В операционной системе может находиться так называемый диспетчер прерываний, который координирует работу отдельных обработчиков прерываний.

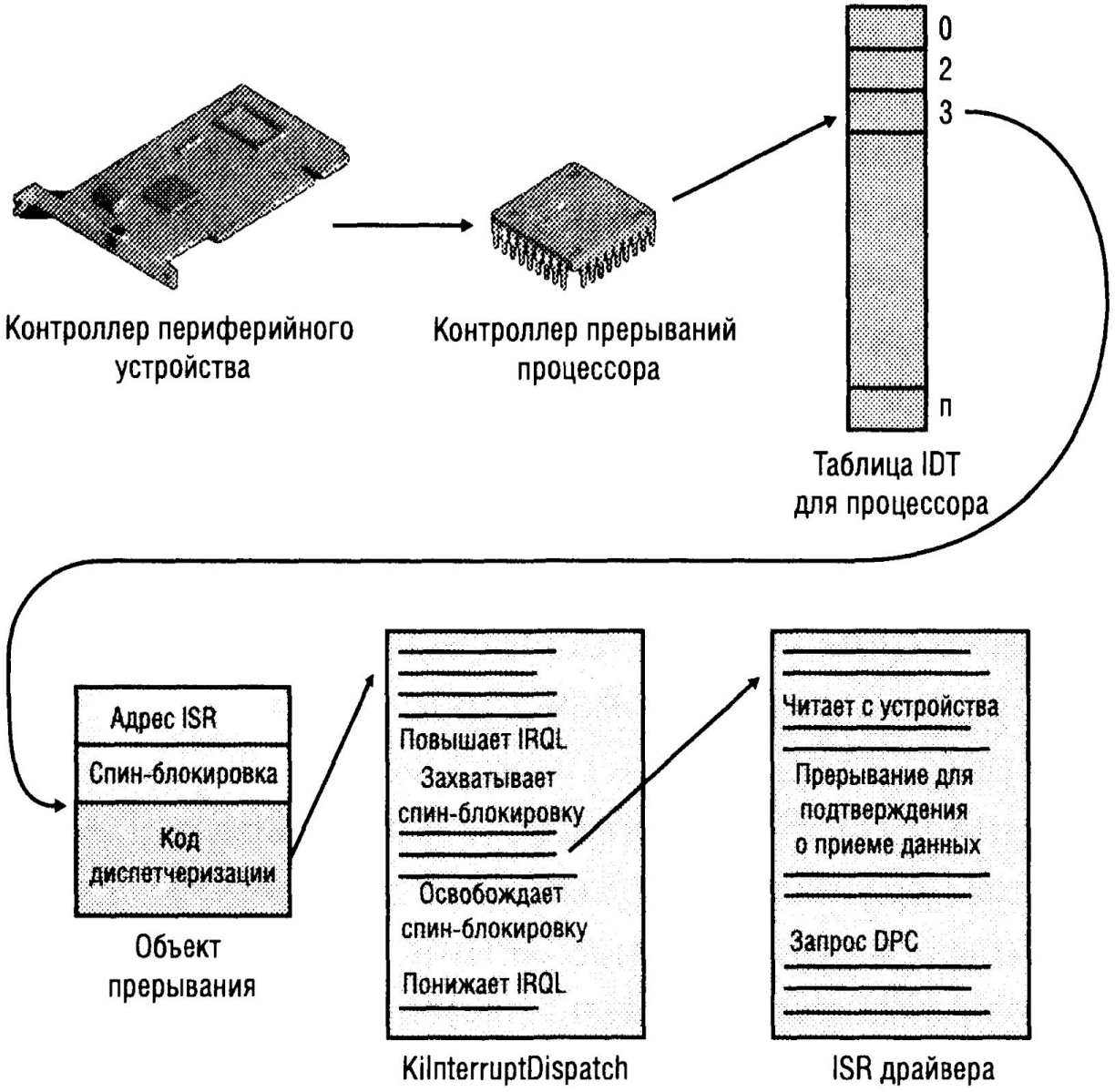
Распределение прерываний по уровням приоритета



- Два основных способа, с помощью которых шины выполняют прерывания: *векторный (vectored)* и *опрашиваемый (polled).* В обоих способах процессору предоставляется информация об уровне приоритета прерывания.
- В случае *векторных* прерываний в процессор передается также информация о начальном адресе программы - *обработчика прерываний.*

- Вектор прерываний, передаваемый в процессор, представляет собой целое число в диапазоне от 0 до 255, указывающее на одну из 256 программ обработки прерываний, адреса которых хранятся в таблице обработчиков прерываний.

- При использовании *опрашиваемых* прерываний процессор получает от устройства только информацию об уровне приоритета прерывания.
- С каждым уровнем прерываний может быть связано несколько устройств и соответственно несколько программ — обработчиков прерываний. процессор должен определить, какое устройство запросило прерывание. Это достигается вызовом всех обработчиков прерываний для данного уровня приоритета, пока один из обработчиков не подтвердит, что прерывание пришло от обслуживаемого им устройства.



Типичная схема обслуживания прерываний

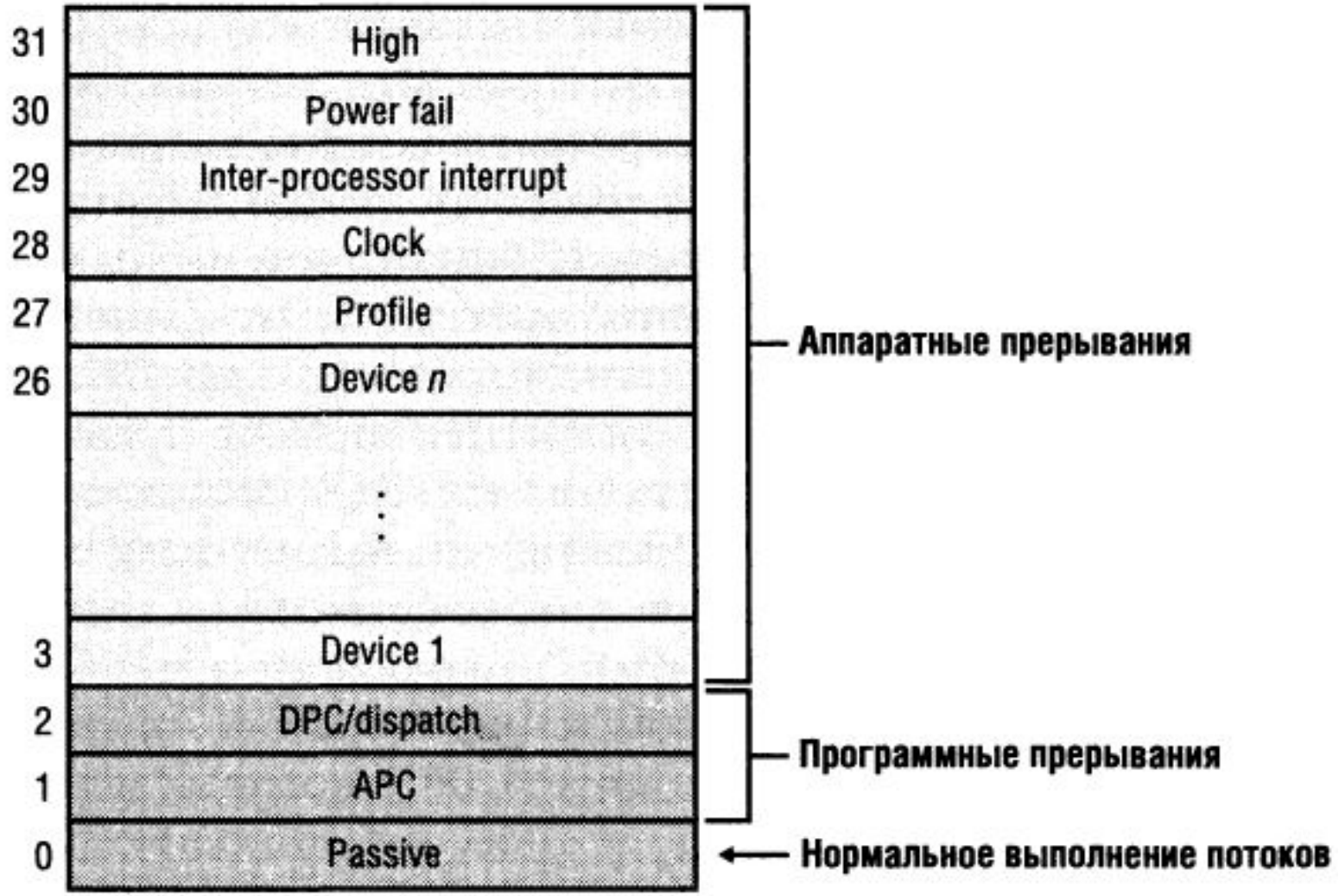


Рис. 3-3. *Уровни запросов прерываний (IRQL) в x86-системах*

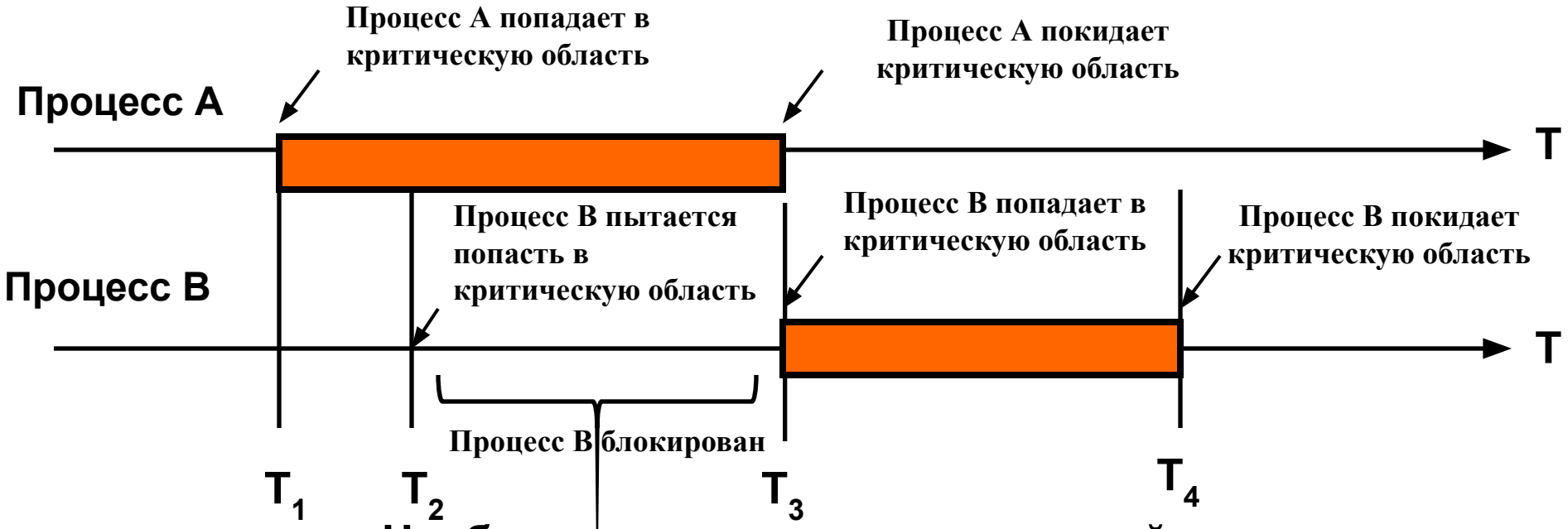
Взаимодействие и синхронизация процессов и потоков

Проблемы взаимодействия и синхронизации

Степень осведомленности	Взаимосвязь	Влияние одного процесса на другой	Потенциальные проблемы
Процессы не осведомлены друг о друге	Конкуренция	• Результат работы одного процесса не зависит от действий других. • Возможно влияние одного про цесса на время работы другого.	•Взаимоисключения •Взаимоблокировки •Голодание
Процессы косвенно осведомлены о наличии друг друга	Сотрудничество с использо-ванием разде-ления	•Результат работы одного процесса может зависеть от информации, полученной от других. •Возможно влияние одного процесса на время работы другого.	•Взаимоисключения •Взаимоблокировки •Голодание •Синхронизация
Процессы непосредст-венно осведомлены о наличии друг друга	Сотрудничество с исполь-зованием связи	•Результат работы одного процесса зависит от информации, полученной от других процессов. •Возможно влияние одного процесса на время работы другого.	•Взаимоблокировки (расходуемые ресурсы) •Голодание

Конкуренция процессов в борьбе за ресурсы

Конкуренция – ситуация, когда два или более процессов требуют доступ к одному и тому же ресурсу (принтеру, файлу и т.п.), называемому критическим. Часть программы, использующая критический ресурс, называется критической секцией.

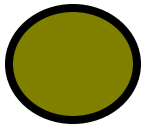


Необходимость взаимоисключений:

1. Процессы не должны одновременно находиться в критических областях.
2. В программе не должно быть предположений о скорости или количестве процессов.
3. Процесс, находящийся вне критической области, не может блокировать другие процессы.
4. Невозможна ситуация, в которой процесс вечно ждет попадания в критическую область.

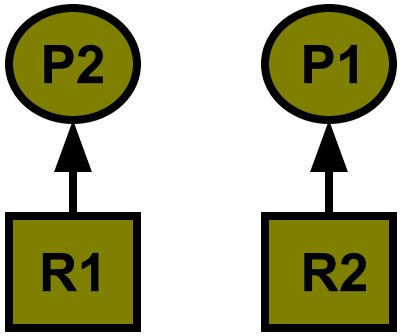
Взаимоблокировки (тупики, deadlock)

Группа процессов находится в тупиковой ситуации, если каждый процесс из группы ожидает события, которое может вызвать только другой процесс из этой же группы

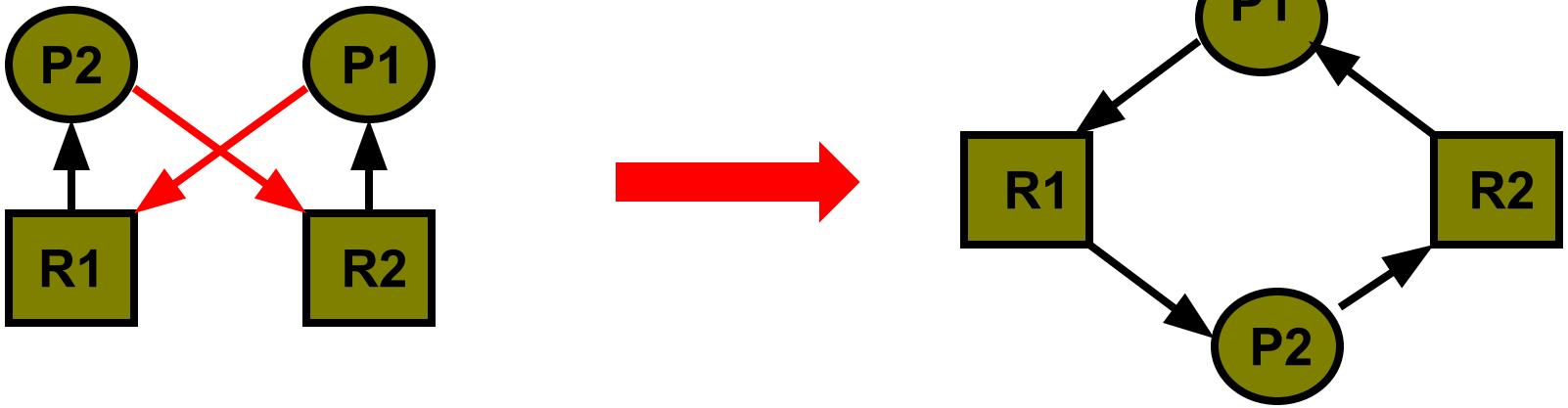


Процесс

Ресурс

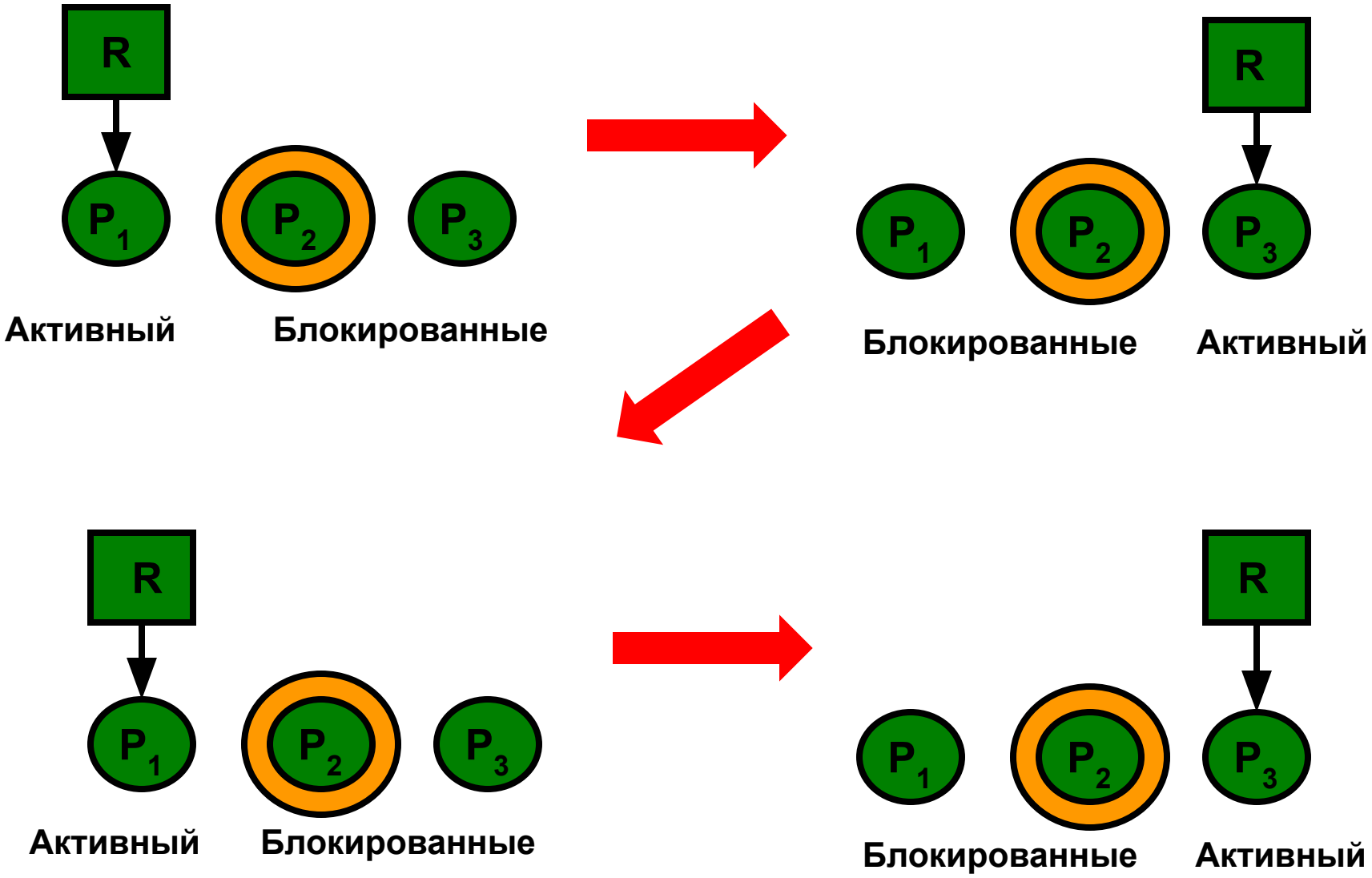


Исходное распределение ресурсов



Тупиковая ситуация

Проблема “голодание”



Сотрудничество с использованием разделения

Процессы, взаимодействующие с другими процессами без наличия явной информации о друг друге, обращаются к разделяемым переменным, к совместно используемым файлам или базам данных.

Проблемы: взаимоисключение, взаимоблокировка, голодание.

Дополнительно: синхронизация процессов для обеспечения согласованности данных

Пример: пусть должно выполняться a = b при начальном значении a = b = 1

1-й вариант: процессы выполняются последовательно

P1: a = a + 1; b = b + 1; P2: b = 2 * b; a = 2 * a;

2-й вариант: процессы прерывают друг друга

P1: a = a + 1; прерывание; P2: b = 2 * b; прерывание;

P1: b = b + 1; прерывание; P2: a = 2 * a;

Согласование нарушено: a = 4, b = 3

Ситуации, в которых два или более процессов обрабатывают разделяемые данные (файлы) и конечный результат зависит от скоростей процессов (потоков), называются гонками.

Методы взаимоисключений

1. **Запрещение прерываний при входе в критическую область и разрешение прерываний после выхода из критической области. Достоинства: простота реализации. Недостатки: монополизация процессора, возможный крах ОС при сбое процесса, невозможность использования в многопроцессорных системах.**

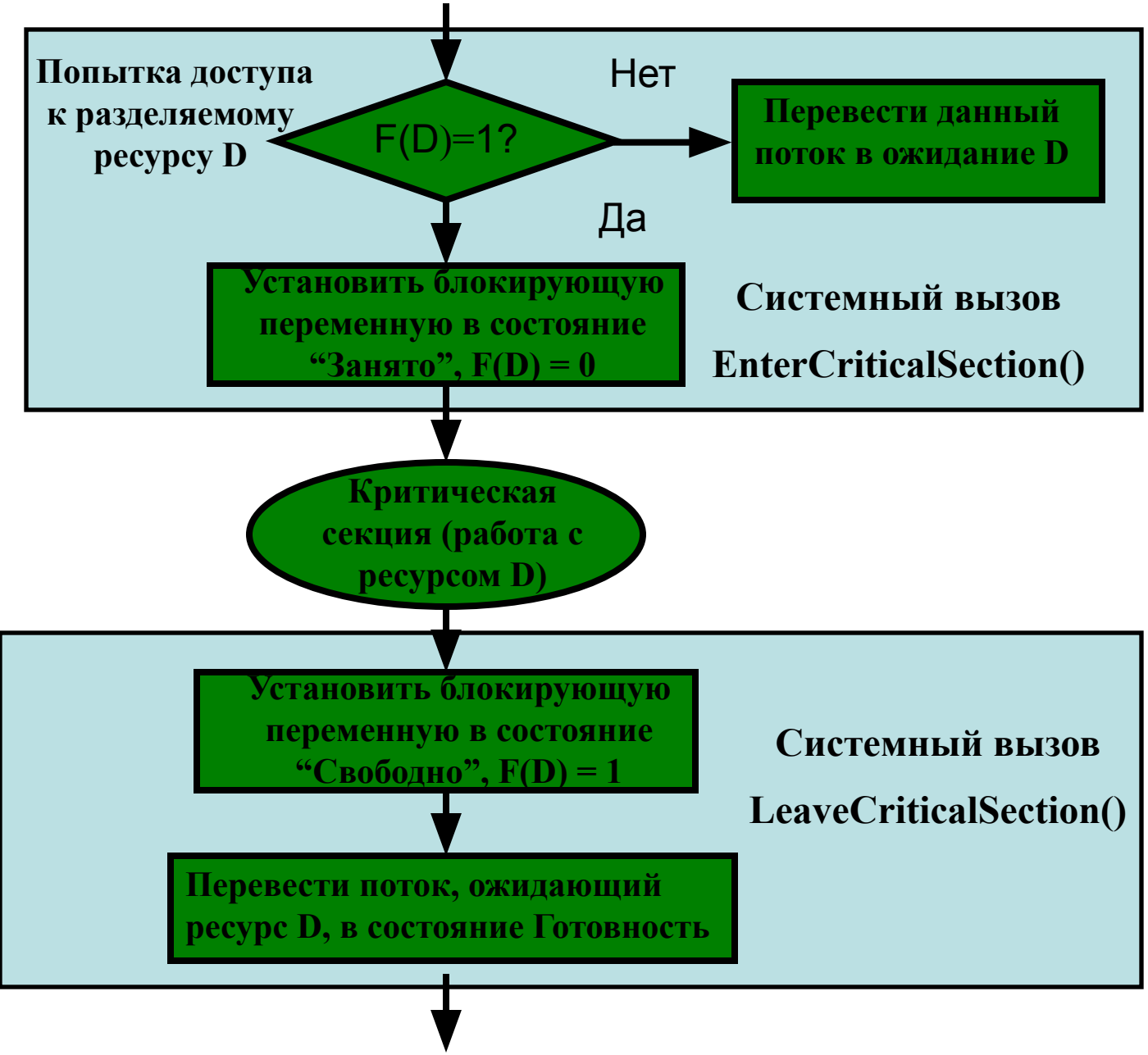
2. **Блокирующие переменные (программный подход)**

Неделимая операция “проверка-установка”

Команды TC, BTR, BTS процессора Pentium (анализ и присвоение значения логической переменной)

Недостатки: необходимость постоянного опроса другими потоками, требующими тот же ресурс, блокирующей переменной; дополнительные затраты процессорного времени.

Использование системных функций входа в критическую секцию



Достоинство: исключается потеря времени процессора на циклическую проверку освобождения занятого ресурса.

Недостаток: растут накладные расходы ОС на по реализации функции входа в критическую секцию и выхода из нее

Семафоры Дийкстры (Dijkstra)

Семафор: переменная S, примитивы P (proberen – проверка; down) и V (verhogen – увеличение, up)

V(S) – переменная S увеличивается на 1 единым действием. Выборка, наращивание и запоминание не могут быть прерваны. К переменной S нет доступа во время выполнения этой операции.

P(S) – переменная S уменьшается на 1, если это возможно, составясь в области неотрицательных значений. Если S уменьшить невозможно, поток, выполняющий операцию P, ждет, пока это уменьшение станет возможным. Операция P неделима.

В частном случае семафор S может принимать двоичные значения 0 и 1, превращаясь в блокирующую переменную (двоичный семафор).

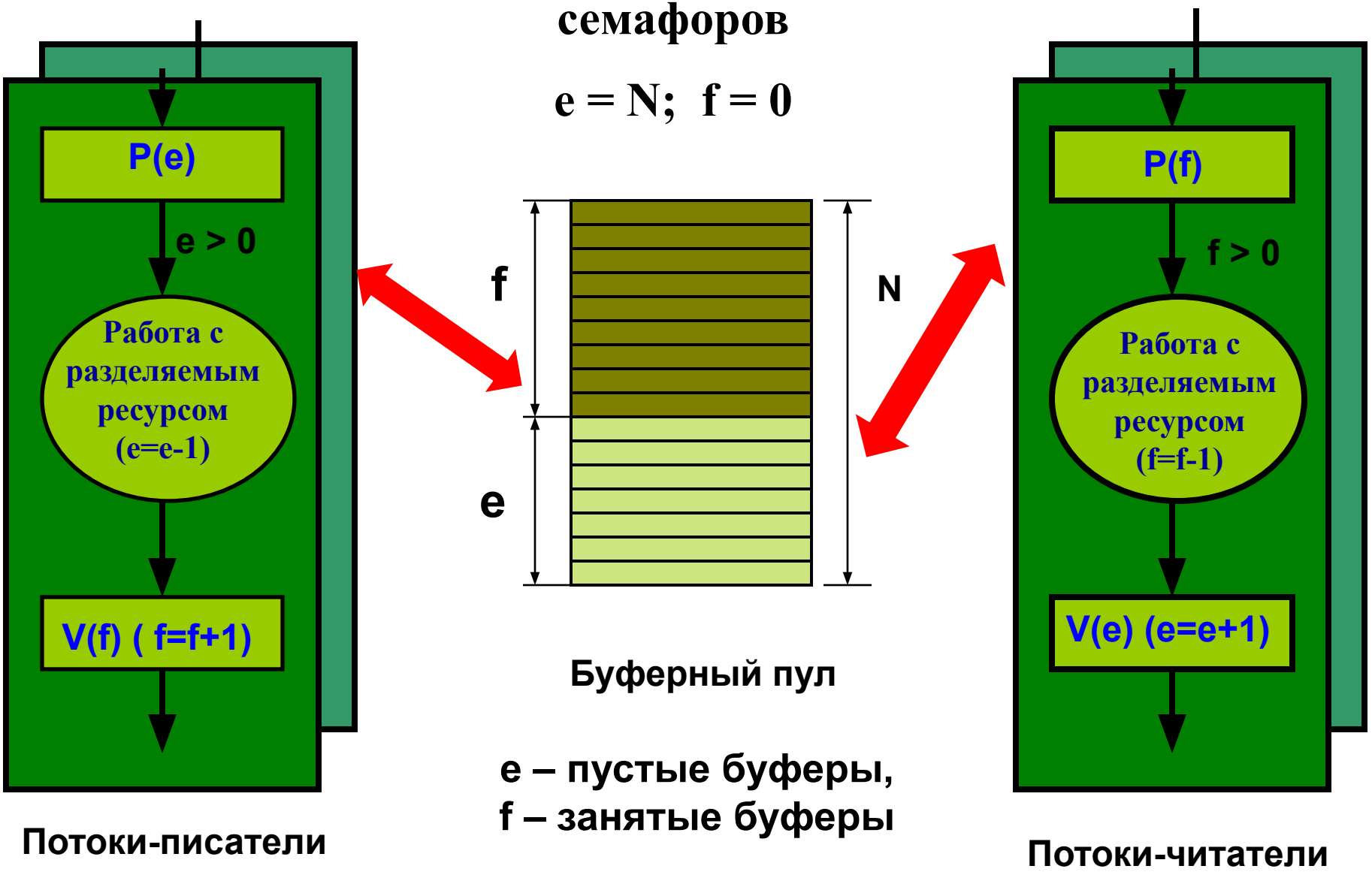
Операция P заключает в себе потенциальную возможность перехода процесса, который ее выполняет, в состояние ожидания (если S = 0).

Операция V может при некоторых обстоятельствах активизировать процесс, приостановленный операцией P.

Для хранения процессов, ожидающих семафоры, используется очередь, работающая по принципу FIFO.

Начальные значения семафоров

$e = N;\ f = 0$



Потоки-писатели

Буферный пул

e – пустые буферы,
f – занятые буферы

Потоки-читатели

Взаимоблокировки (тупики)

Условия возникновения взаимоблокировки (тупиковой) ситуации:

1. **Взаимное исключение. Каждый ресурс в данный момент или отдан ровно одному процессу, или недоступен.**
2. **Условие удержания и ожидания. Процессы, в данный момент удерживающие полученные ранее ресурсы, могут запрашивать новые ресурсы.**
3. **Отсутствие принудительной выгрузки ресурсов. У процесса нельзя забрать принудительно ранее полученные ресурсы.**
4. **Условие циклического ожидания. Существует круговая последовательность из двух и более процессов, каждый из которых ждет доступа к ресурсу, удерживаемому следующим членом последовательности.**

Стратегии борьбы с взаимоблокировками:

1. Пренебрежение проблемой в целом.
2. Обнаружение и устранение взаимоблокировок (восстановление).
3. Недопущение тупиковых ситуаций с помощью аккуратного распределения ресурсов.
4. Предотвращать с помощью структурного опровержения одного из четырех условий, необходимых для взаимоблокировки

Методы обнаружения взаимоблокировок

1. **В системе один ресурс каждого типа.**

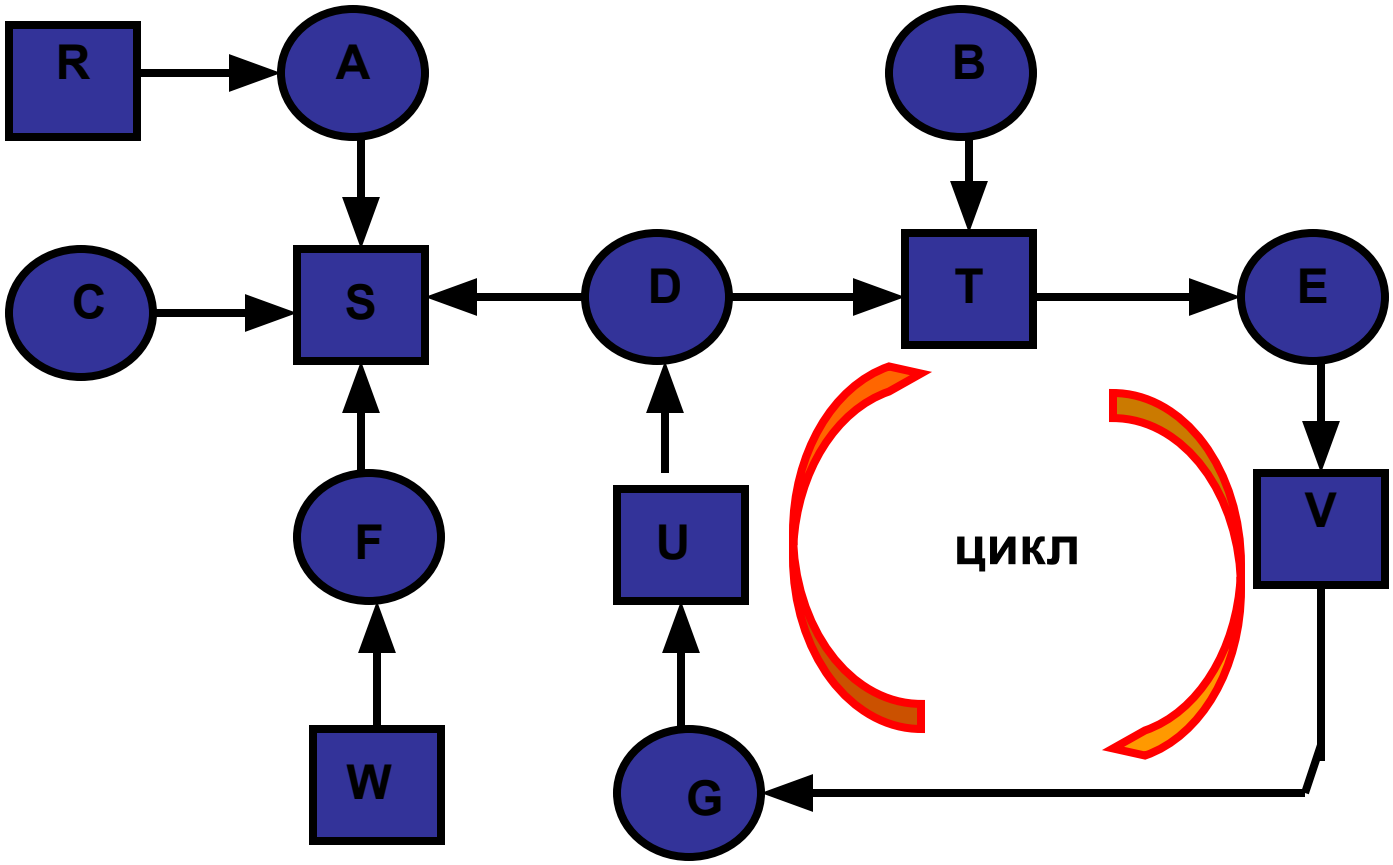
Например, пусть система из семи процессов (A, B, C, D, E, F, G) и шести ресурсов (R, S, T, V, W, U) в некоторый момент соответствует следующему списку:

- **Процесс A занимает ресурс R и хочет получить ресурс S.**
- **Процесс B ничего не использует, но хочет получить ресурс T.**
- **Процесс C ничего не использует, но хочет получить ресурс S.**
- **Процесс D занимает ресурс U и хочет получить ресурсы S и T.**
- **Процесс E занимает ресурс T и хочет получить ресурс V.**
- **Процесс F занимает ресурс W и хочет получить ресурс S.**
- **Процесс G занимает ресурс V и хочет получить ресурс U.**

ВОПРОС: Заблокирована ли эта система и если да, то какие процессы в этом участвуют?

ОТВЕТ МОЖНО ПОЛУЧИТЬ, ПОСТРОИВ ГРАФ РЕСУРСОВ И ПРОЦЕССОВ.

Граф ресурсов и процессов



2. В системе несколько ресурсов каждого типа.

$P = \{P_1, P_2, \ldots, P_n\}$ – множество процессов, n – число процессов;

$E = \{E_1, E_2, \ldots, E_m\}$ – множество ресурсов, m – число типов ресурсов;

$A = \{A_1, A_2, \ldots, A_m\}$ – вектор свободных ресурсов; $A_J <= E_J$, $j = \overline{1, m}$;

$C = \{c_{I\,J} \mid i = \overline{1, n};\ j = \overline{1, m}\}$ – матрица текущего распределения ресурсов;

$R = \{r_{i\,j} \mid i = \overline{1, n};\ j = \overline{1, m}\}$ – матрица запрашиваемых ресурсов.

Существующие ресурсы

$$E = \{E_1, E_2, \ldots, E_m\}$$

$$\begin{vmatrix} c_{11} & c_{12} & \ldots & c_{1m} \\ c_{21} & c_{21} & \ldots & c_{2m} \\ . & . & \ldots & . \\ c_{n1} & c_{n2} & & c_{nm} \end{vmatrix}$$

Доступные ресурсы

$$A = \{A_1, A_2, \ldots, A_m\}$$

$$\begin{vmatrix} r_{11} & r_{12} & \ldots & r_{1m} \\ r_{21} & r_{22} & \ldots & r_{2m} \\ . & . & \ldots & . \\ r_{n1} & r_{n2} & \ldots & R_{nm} \end{vmatrix}$$

$$\sum_{I=1}^{n} c_{i\,j} + A_j = E_j, \quad j = \overline{1, m}$$

Алгоритм обнаружения тупиков

Основан на сравнении векторов ресурсов. В исходном состоянии все процессы не маркированы (не отмечены). По мере реализации алгоритма на процессы будет ставиться отметка, обозначающая, что они могут закончить свою работу, т. е. не находятся в тупике. После завершения алгоритма любой немаркированный процесс находится в тупиковой ситуации.

1. Ищется процесс P_i , для которого i – я строка матрицы R меньше вектора A, т. е. $R_i <= A$ или $r_{ij} <= A_j$, $j = \overline{1, m}$.
2. Если такой процесс найден, он маркируется, и далее прибавляется I - я строка матрицы C к вектору A, т.е. $A_j := A_j + c_{ij}$, $j = \overline{1, m}$.

   Возврат к шагу 1.
3. Если таких процессов не существует, работа алгоритма заканчивается. Если есть немаркированные процессы, то они попали в тупик.

Методы устранения тупиков

1. **Принудительная выгрузка ресурсов. Изъятие ресурса у процесса, передача его другому процессу, а затем возврат ресурса таким образом, что исходный процесс этого “ не замечает” (сложно и чаще всего невозможно).**
2. **Восстановление через “откат”. Процессы периодически создают контрольные точки, позволяющие запустить процесс с предыстории. При возникновении тупика процесс, занимающий необходимый ресурс “откатывается” к контрольной точке, после которой он получил ресурс. Если возобновленный процесс вновь попытается получить данный ресурс, он переводится в режим ожидания освобождения этого ресурса.**
3. **Восстановление путем уничтожения процессов.**

Недопущение тупиков путем безопасного распределения ресурсов. Подобные алгоритмы базируются на концепции безопасных состояний. Например, Дейкстрой был разработан алгоритм планирования, позволяющий избегать взаимоблокировок (алгоритм банкира).

Синхронизирующие объекты ОС

Для синхронизации потоков, принадлежащих разным процессам, ОС должна предоставлять потокам системные объекты синхронизации.

К таким объектам относятся *события* (event), *мьютексы* (mutex – mutual exclusion – взаимное исключение), *системные семафоры* и др.

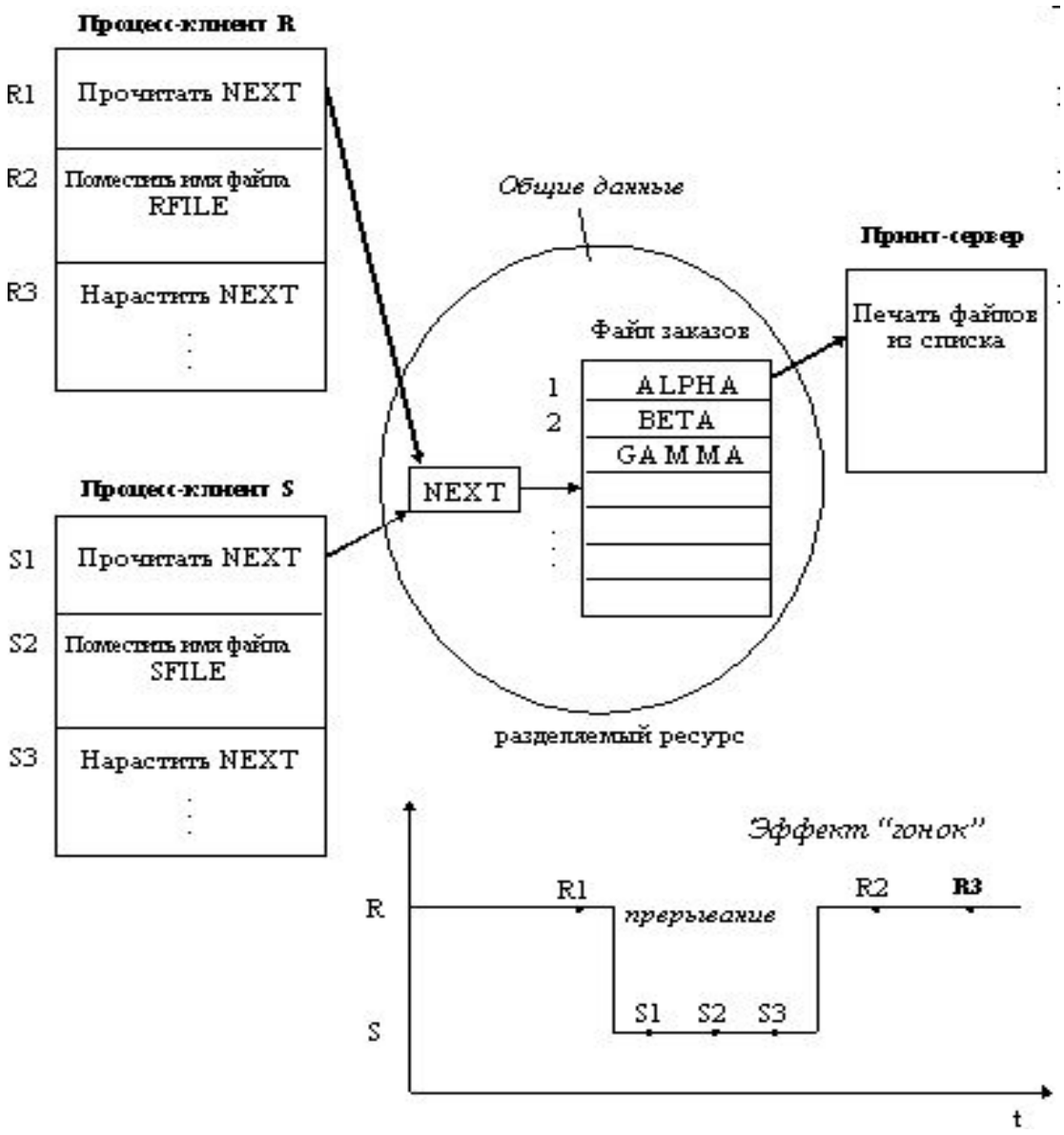
Объект-событие используется для того, чтобы оповестить потоки о том, что некоторые действия завершены.

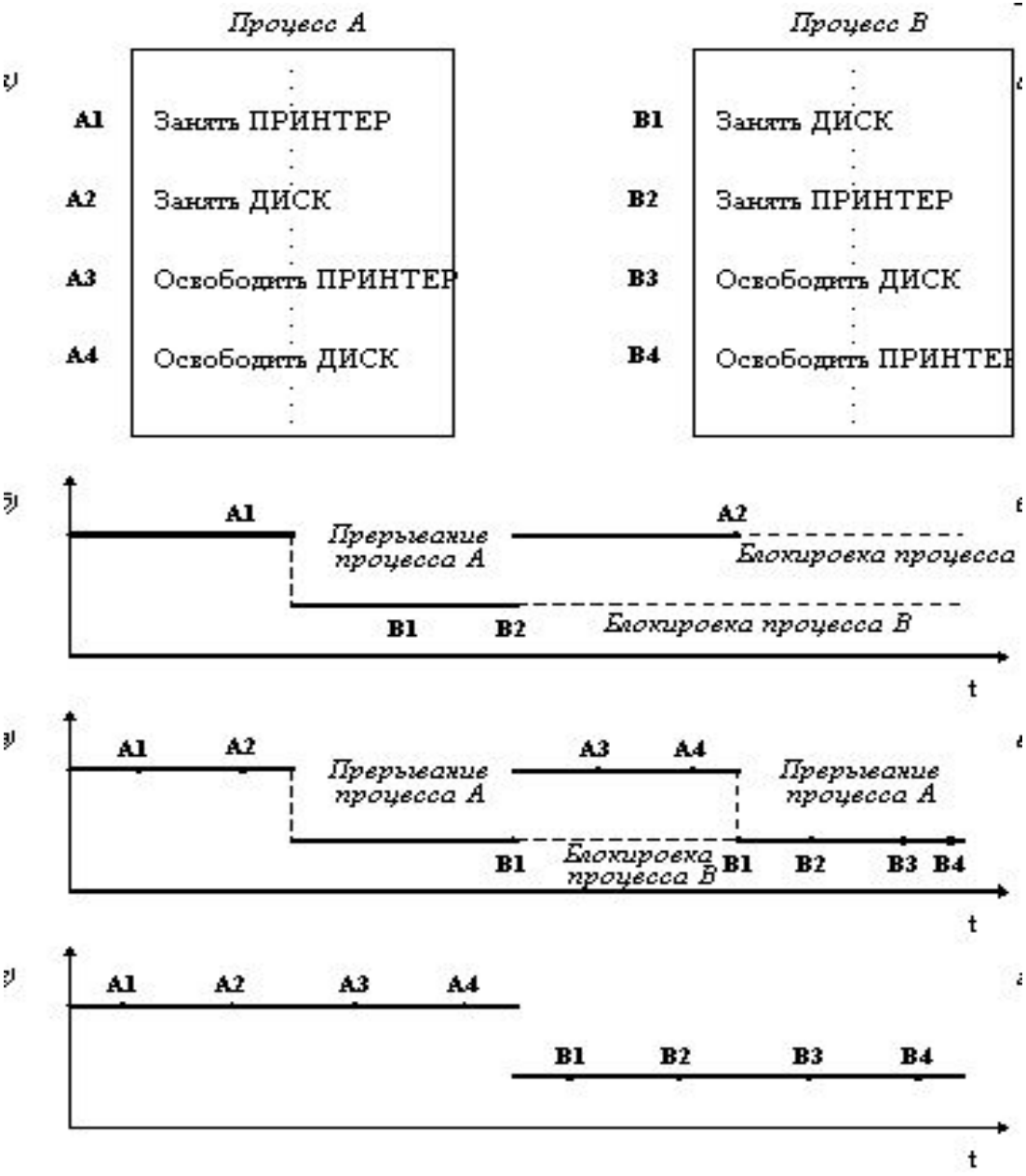
Мьютекс (простейший двоичный семафор) используется для управления доступом к данным.

Семафоры используются для оповещения свершения последовательности событий.

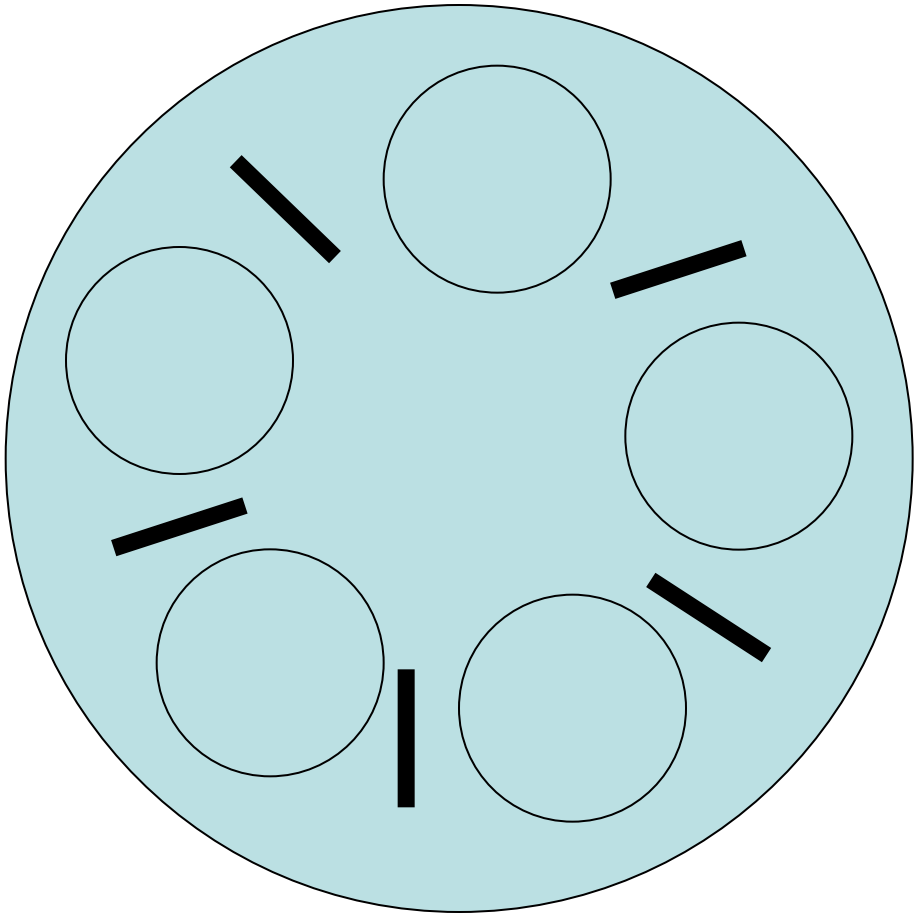
Для синхронизации используются также “обычные ” объекты ОС: файлы, процессы, потоки

Все объекты синхронизации могут находиться в сигнальном и несигнальном (свободном) состоянии. Поток с помощью системного вызова WAIT(X) может синхронизировать свое выполнение с объектом синхронизации X. С помощью системного вызова SET(X) поток может перевести объект X в сигнальное состояние. Кроме того, в ОС определен набор сигналов для логической связи меду процессами, а также процессами и пользователями (терминалами).


Процесс-клиент R
R1
Прочитать NEXT
R2
Поместить имя файла RFILE
R3
Нарастить NEXT
Общие данные
Принт-сервер
Печать файлов из списка
Файл заказов
1
2
ALPHA
BETA
GAMMA
NEXT
Процесс-клиент S
S1
Прочитать NEXT
S2
Поместить имя файла SFILE
S3
Нарастить NEXT
разделяемый ресурс
Эффект "гонок"
R
S
R1
прерывание
R2
R3
S1
S2
S3
t


Процесс A
A1 Занять ПРИНТЕР
A2 Занять ДИСК
A3 Освободить ПРИНТЕР
A4 Освободить ДИСК
Процесс B
B1 Занять ДИСК
B2 Занять ПРИНТЕР
B3 Освободить ДИСК
B4 Освободить ПРИНТЕР
A1
A2
Прерывание процесса A
Блокировка процесса
B1
B2
Блокировка процесса B
t
A1
A2
Прерывание процесса A
A3
A4
Прерывание процесса A
B1
Блокировка процесса B
B1
B2
B3
B4
t
A1
A2
A3
A4
B1
B2
B3
B4
t

Проблема обедающих философов.
Полезна для моделирования процессов, соревнующихся за монопольный доступ к ограниченному количеству ресурсов, например к устройствам ввода/вывода.

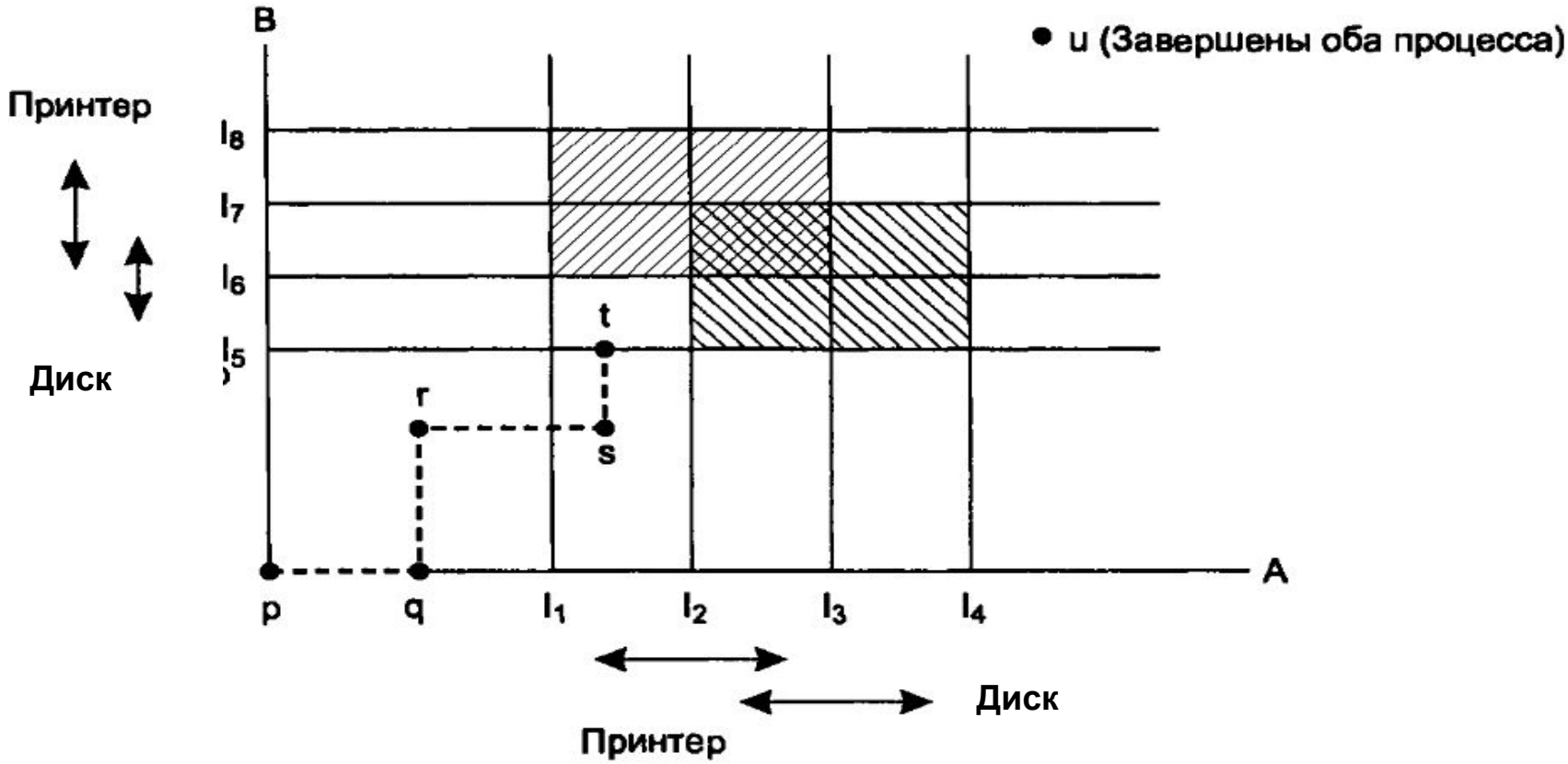


Решение задачи обедающих философов

```
#define N       5           /* Количество философов */
#define LEFT (i+N-1)%N      /* Номер левого соседа философа с номером i */
#define RIGHT (i+1)%N /* Номер правого соседа философа с номером i */
#define THINKING 0      /* Философ размышляет */
#define HUNGRY 1        /* Философ пытается получить вилки */
#define EATING 2        /* Философ ест */
typedef int semaphore;      /* Семафоры - особый вид целочисленных
            переменных */
int state[N];           /* Массив для отслеживания состояния каждого
     философа */
semaphore mutex =1;   /* Взаимоисключение для критических областей */
semaphore s[N];    /* Каждому философу по семафору */
void philosopher(int i)   /* i - номер философа, от 0 до N-1 */
{
while (TRUE) {      /* Повторять до бесконечности */
think();          /* Философ размышляет */
take_forks(i);  /* Получает две вилки или блокируется */
eat();            /* Философ ест спагетти */
put_forks(i);   /* Кладет на стол обе вилки */
}
}
```

```
void take_forks(int i)     /* i - номер философа, от 0 до N-1*/
{
down(&mutex);              /* Вход в критическую область */
state[i] = HUNGRY;         /* Фиксация наличия голодного философа */
test(i);                   /* Попытка получить две вилки */
up(&mutex);                /* Выход из критической области */
down(&s[i]);               /* Блокировка, если вилок не досталось */
}
void put_forks(i)          /* i - номер философа, от 0 до N-1*/
{
down(&mutex);              /* Вход в критическую область */
state[i] = THINKING;       /* Философ перестал есть */
test(LEFT);                /* Проверить, может ли есть сосед слева */
test(RIGHT);               /* Проверить, может ли есть сосед справа */
up(&mutex);                /* Выход из критической области */
}
void test(i)               /* i - номер философа, от 0 до N-1*/
{
if (state[i] == HUNGRY && state[LEFT] != EATING && state[RIGHT] != EATING) {
State[i] = EATING;
up(&s[i]);
}
}
```

Траектории ресурсов



Две траектории ресурсов процессов

Алгоритм банкира

	Имеет	Max
Энди	0	6
Барбара	0	5
Марвин	0	4
Сьюзан	0	7

Свободно: 10

а

	Имеет	Max
Энди	1	6
Барбара	1	5
Марвин	2	4
Сьюзан	4	7

Свободно: 2

б

	Имеет	Max
Энди	1	6
Барбара	2	5
Марвин	2	4
Сьюзан	4	7

Свободно: 1

в

Три состояния распределения ресурсов: а — безопасное; *б* — безопасное *в* — небезопасное

Межпроцессное взаимодействие (англ. Inter-Process Communication, IPC) — набор способов обмена данными между множеством потоков в одном или более процессах. Процессы могут быть запущены на одном или более компьютерах, связанных между собой сетью. IPC-способы делятся на методы обмена сообщениями, синхронизации, разделяемой памяти и удаленных вызовов (RPC). Методы IPC зависят от пропускной способности и задержки взаимодействия между потоками и типа передаваемых данных.

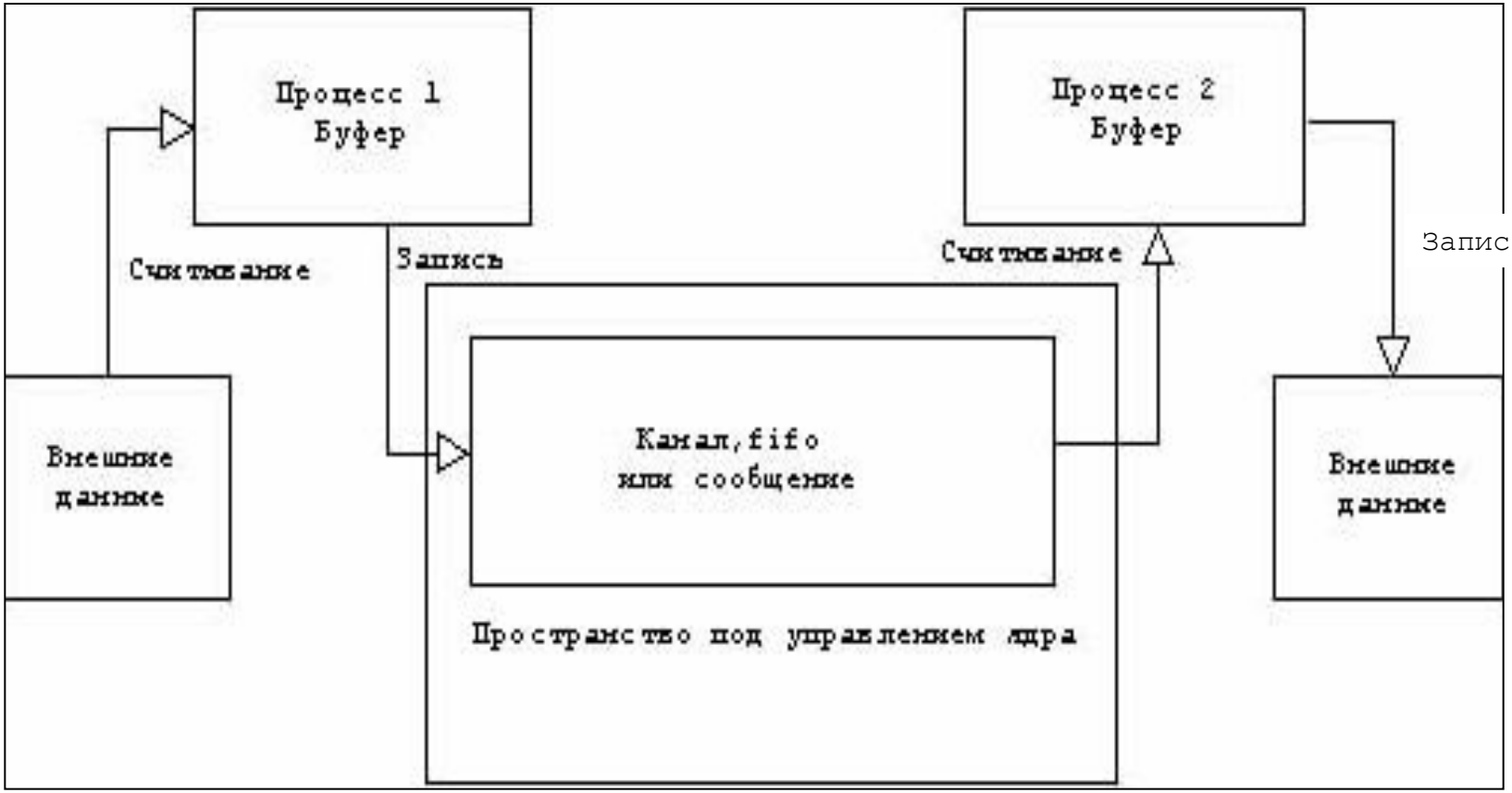
IPC также может упоминаться как межпотоковое взаимодействие (англ. inter-thread communication), межпоточное взаимодействие и межпрограммное взаимодействие (англ. inter-application communication).

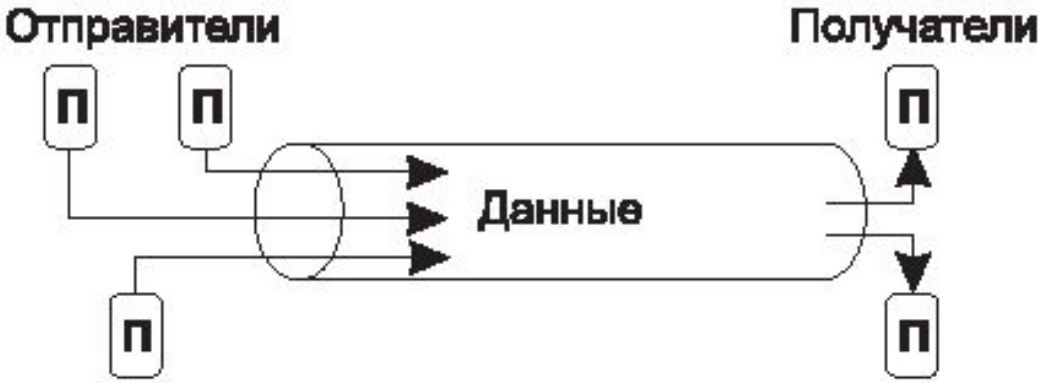
Виды механизмов межпроцессного взаимодействия (InterProcess Communication - IPC)

Механизм IPC	**Назначение**
Буфер обмена	Пересылка данных между программами выполняется по желанию и под контролем пользователя.
OLE 2.0	Имеются встроенные функции для пересылки данных через границы процессов. Излишне сложен для простого обмена данными.
Сообщения WM_COPYDATA	Способ пересылки блока данных из одной программы в другую. Используется в тех случаях, когда скорость передачи данных не является критической и не требуется синхронизировать передачу данных.
Объекты синхронизации	Объекты синхронизации тоже можно отнести к механизмам IPC. Конечно, объем передаваемых данных очень невелик: например, одному процессу нужно передать другому "я закончил работу" или "я начинаю работать с общей памятью".
Библиотеки динамической компоновки (DLL)	Библиотеки динамической компоновки также имеют способность обеспечивать обмен данными между процессами. Когда в рамках DLL объявляется переменная, ее можно сделать разделяемой (shared). Все процессы, обращающиеся к библиотеке, для таких переменных будут использовать одно и то же место в физической памяти.

Механизм IPC	**Назначение**
Сигналы	Применяются для уведомления о некотором событии, произошедшем в системе.
Анонимные каналы (Anonymous pipes)	Полезны для организации прямой связи между двумя процессами на одном ПК.
Именованные каналы (Named pipes)	Полезны для организации прямой связи между двумя процессами на одном ПК или в сети.
Почтовые ячейки (mailslots)	Полезны для организации связи одного процесса со многими на одном ПК или в сети.
Гнезда (sockets)	Полезны для организации пересылки данных как в Windows-программы, так и в прочие программы, функционирующие на одном ПК, в сети или в интрасети.
Очереди сообщений Microsoft Message Queue (MSMQ)	Данный протокол может работать с удаленными процессами и даже с процессами, которые на данный момент недоступны (например, не запущены). Доставка сообщения по адресу гарантируется.
Файлы отображаемой памяти	Обеспечивают одновременный доступ к объектам файла- отображения из нескольких процессов.

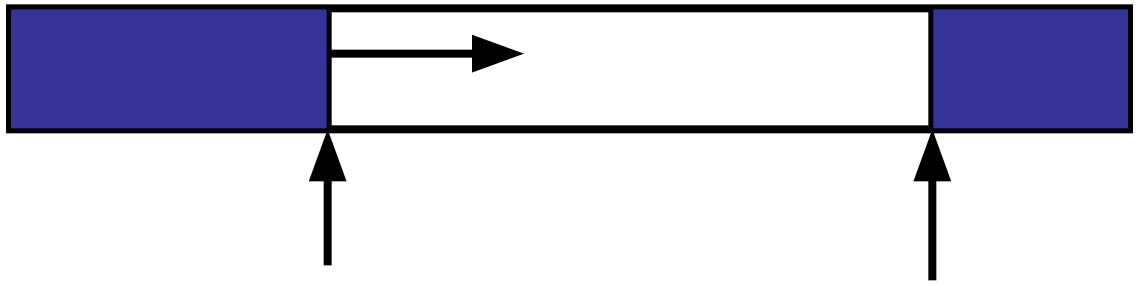
Межпроцессный обмен на локальном компьютере

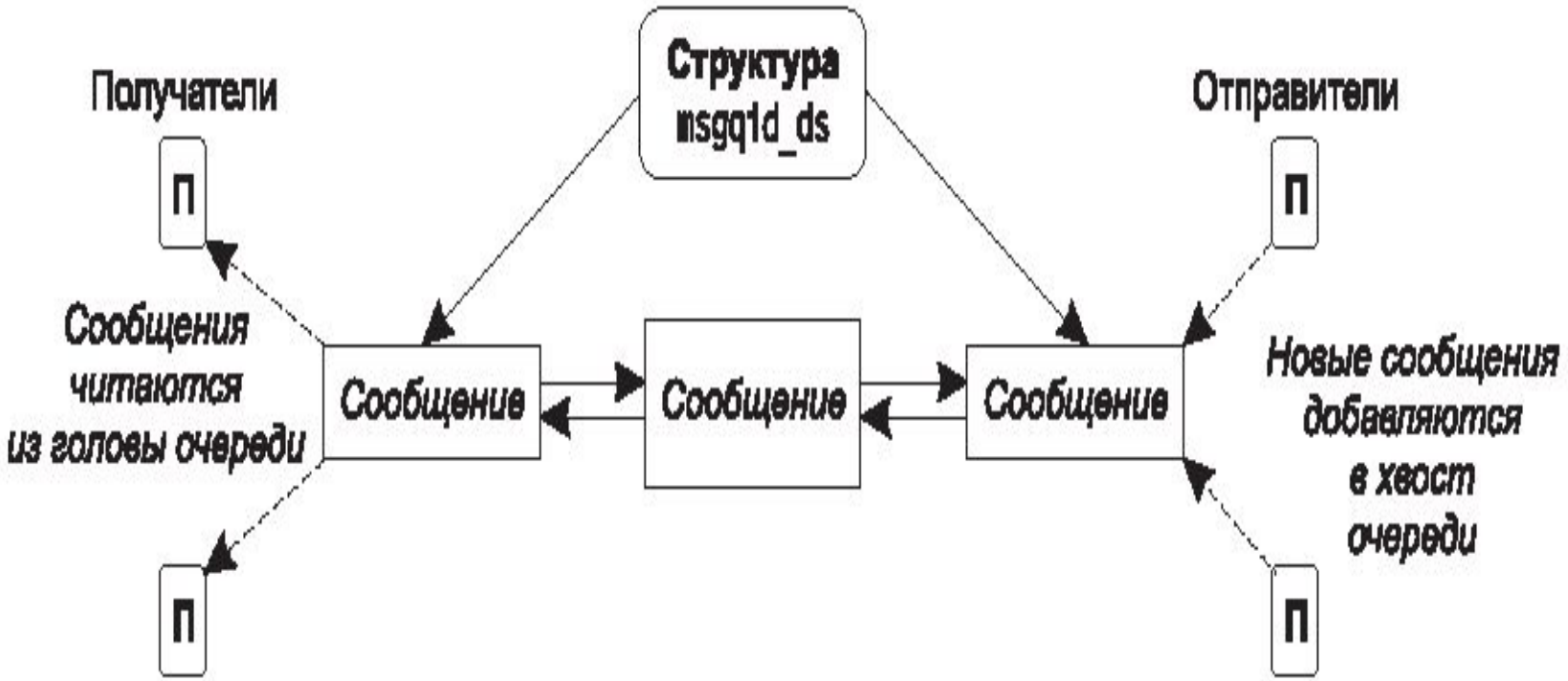




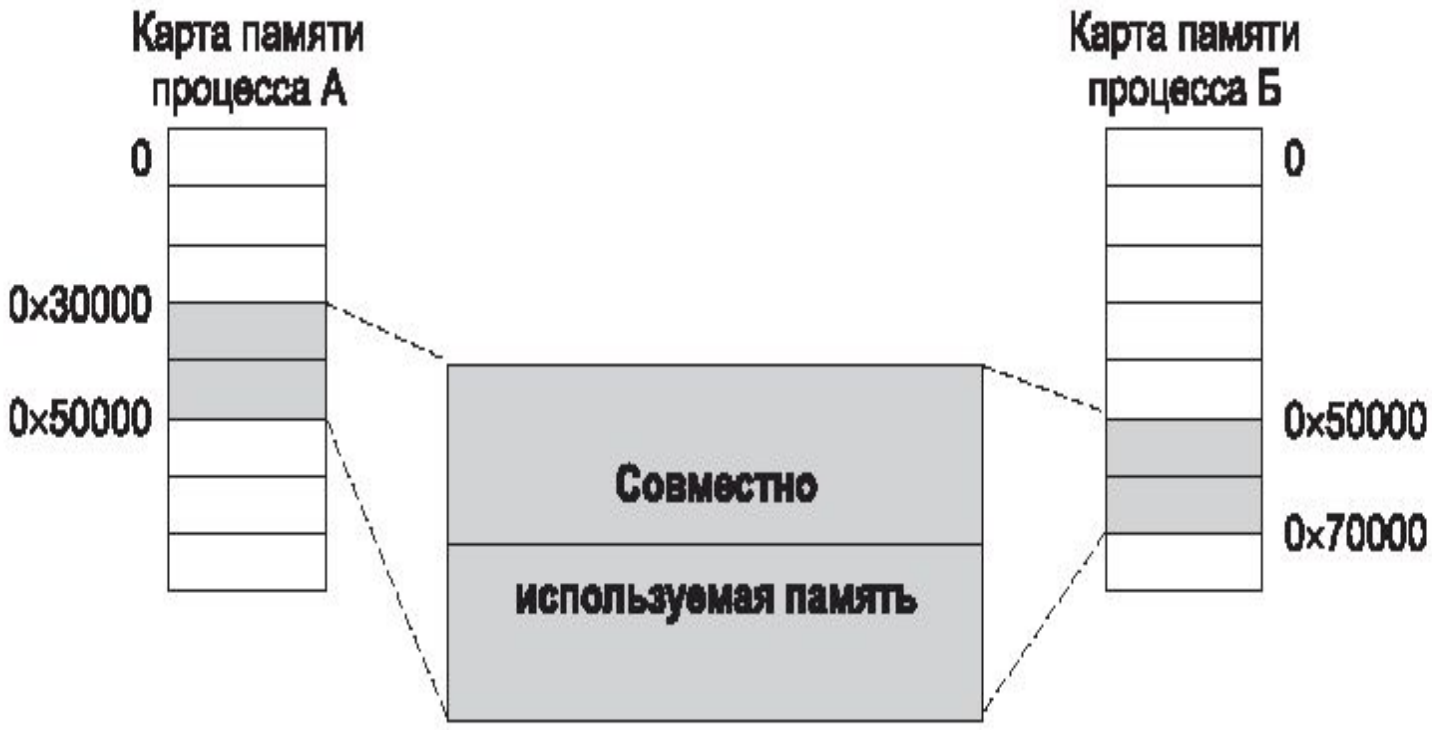
Движение потока данных по программному каналу

Организации очереди в массиве



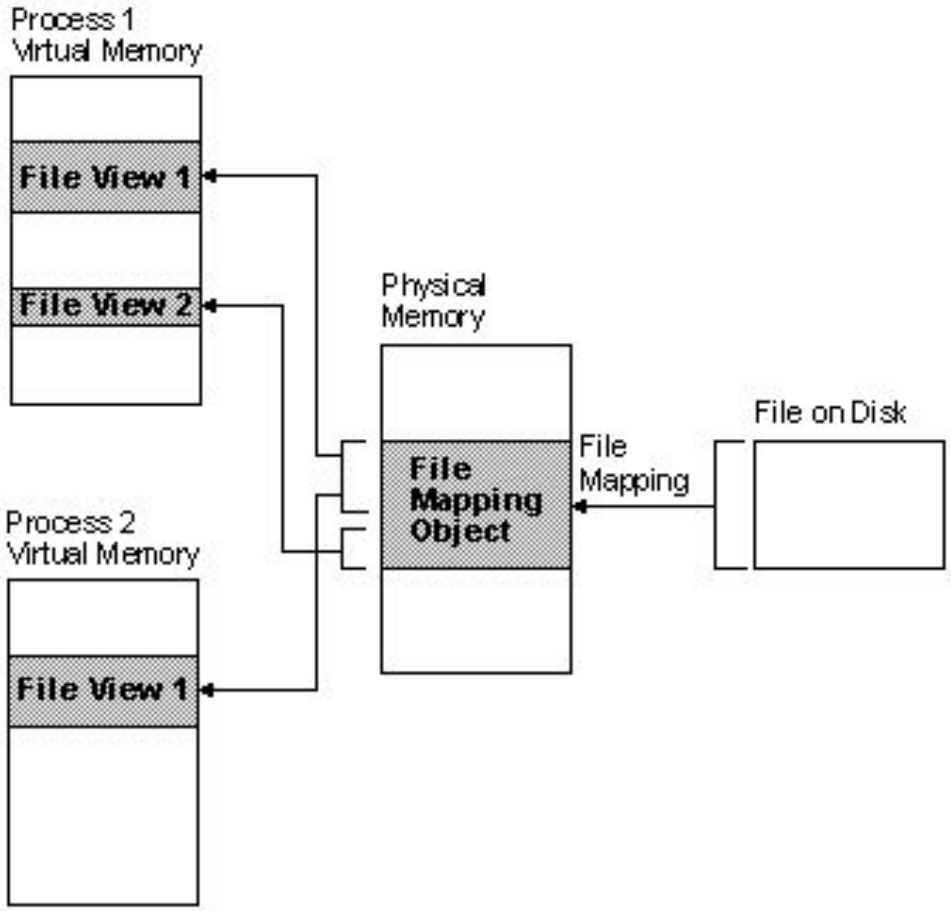


Применение очередей сообщений

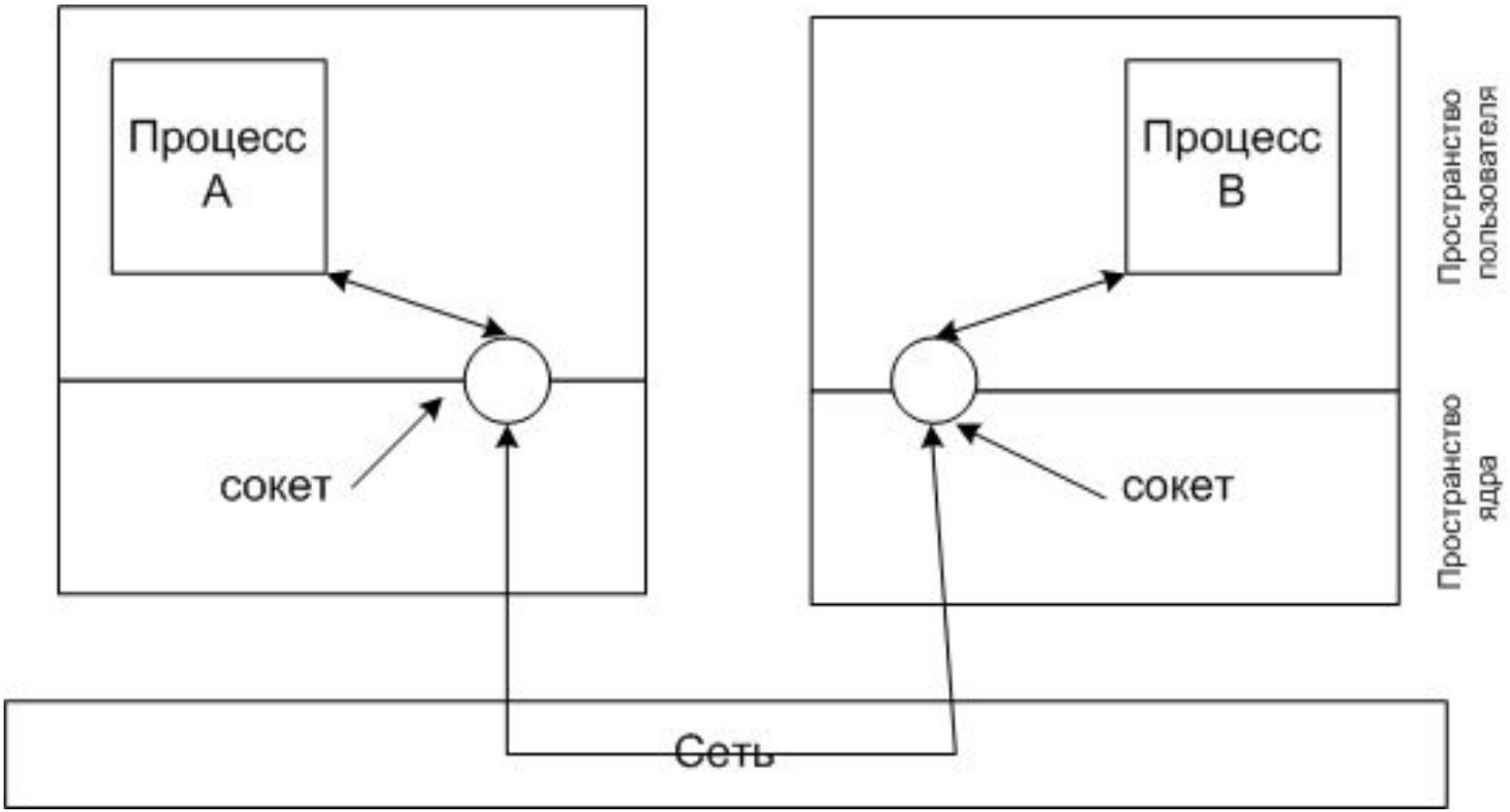


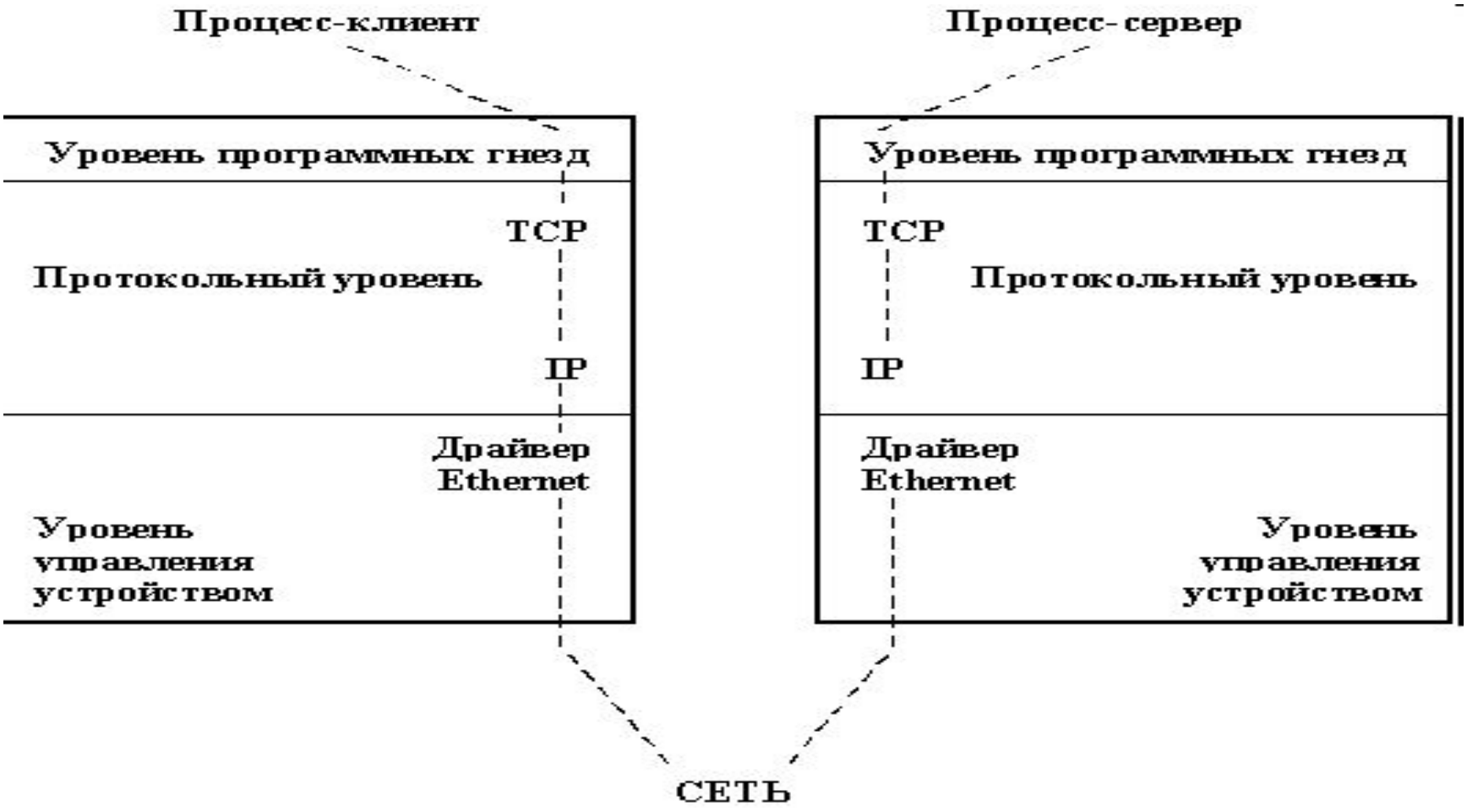
Присвоение разделяемой области памяти

Отображаемые файлы




Заголовок сообщения
Тип
Размер
Локальный порт
Порт назначения
Идентификатор сообщения
Дескриптор типа
Имя
Размер
Номер
Флаги
Данные
Дескриптор типа
Имя
Размер
Номер
Флаги
Данные


Процесс
A
Процесс
B
сокет
сокет
Сеть
Пространство
пользователя
Пространство
ядра


Процесс-клиент
Процесс-сервер
Уровень программных гнезд
TCP
Протокольный уровень
IP
Драйвер Ethernet
Уровень управления устройством
Уровень программных гнезд
TCP
Протокольный уровень
IP
Драйвер Ethernet
Уровень управления устройством
СЕТЬ