

Операционные системы

Тема 2. Процессы и потоки. Планирование и синхронизация



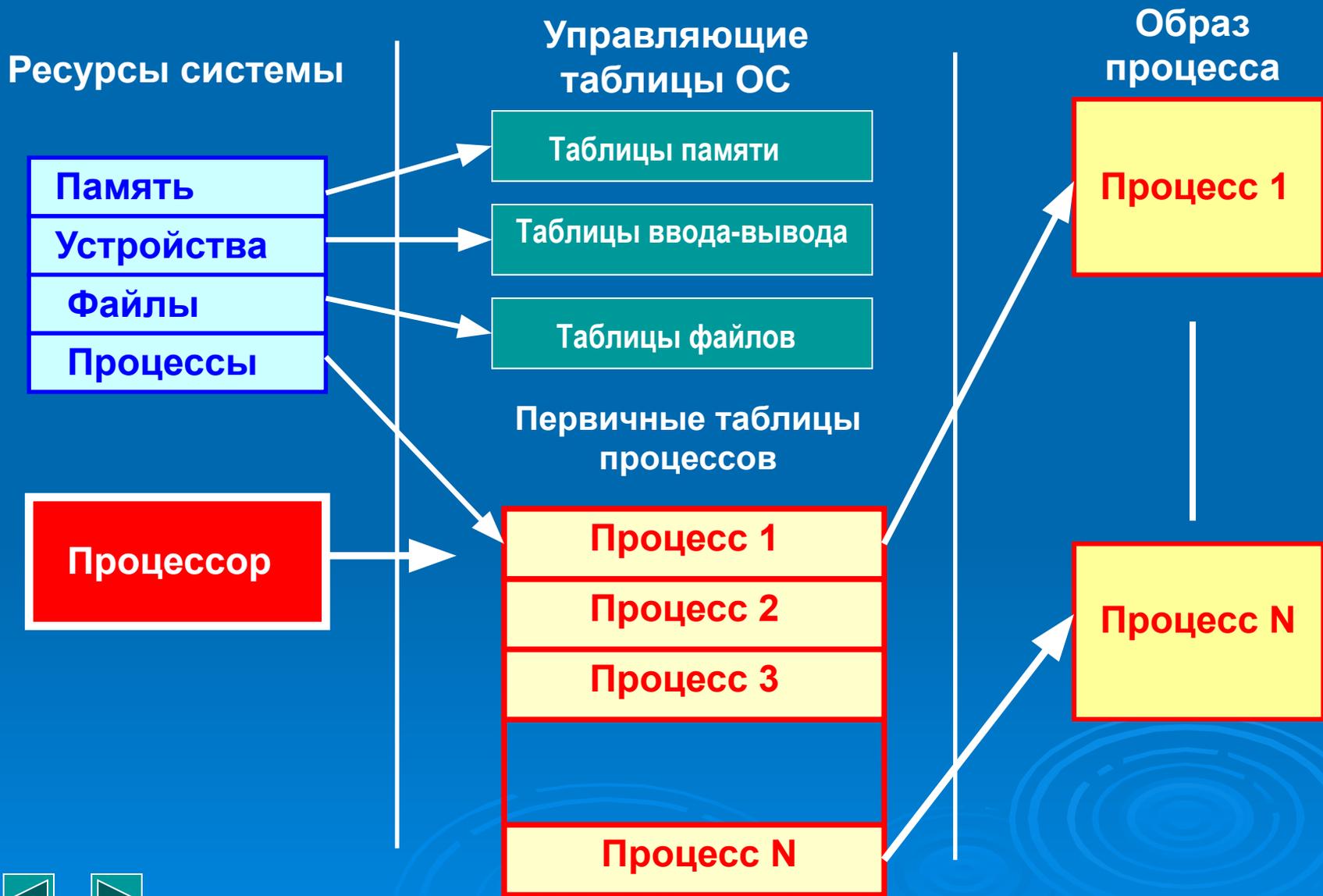
Тема 2. Процессы и потоки.

Планирование и синхронизация

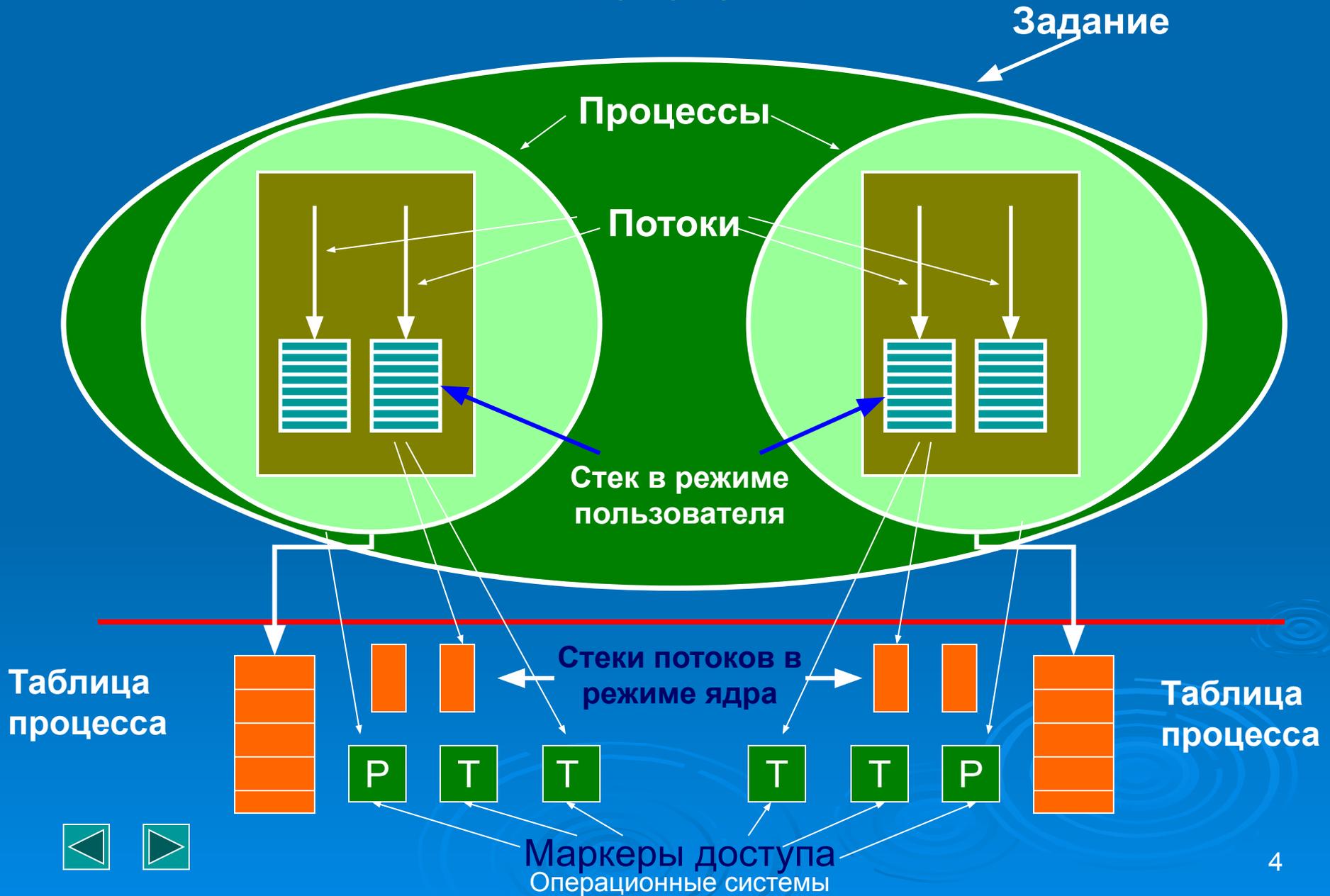
- 2.1. Концепция процессов и потоков. Задания, процессы, потоки (нити), волокна
- 2.2. Мультипрограммирование. Формы многопрограммной работы
- 2.3. Управление процессами и потоками
- 2.4. Создание процессов и потоков. Модели процессов и потоков
- 2.5. Планирование процессов и потоков
- 2.6. Взаимодействие и синхронизация процессов и потоков
- 2.7. Аппаратно-программные средства поддержки мультипрограммирования

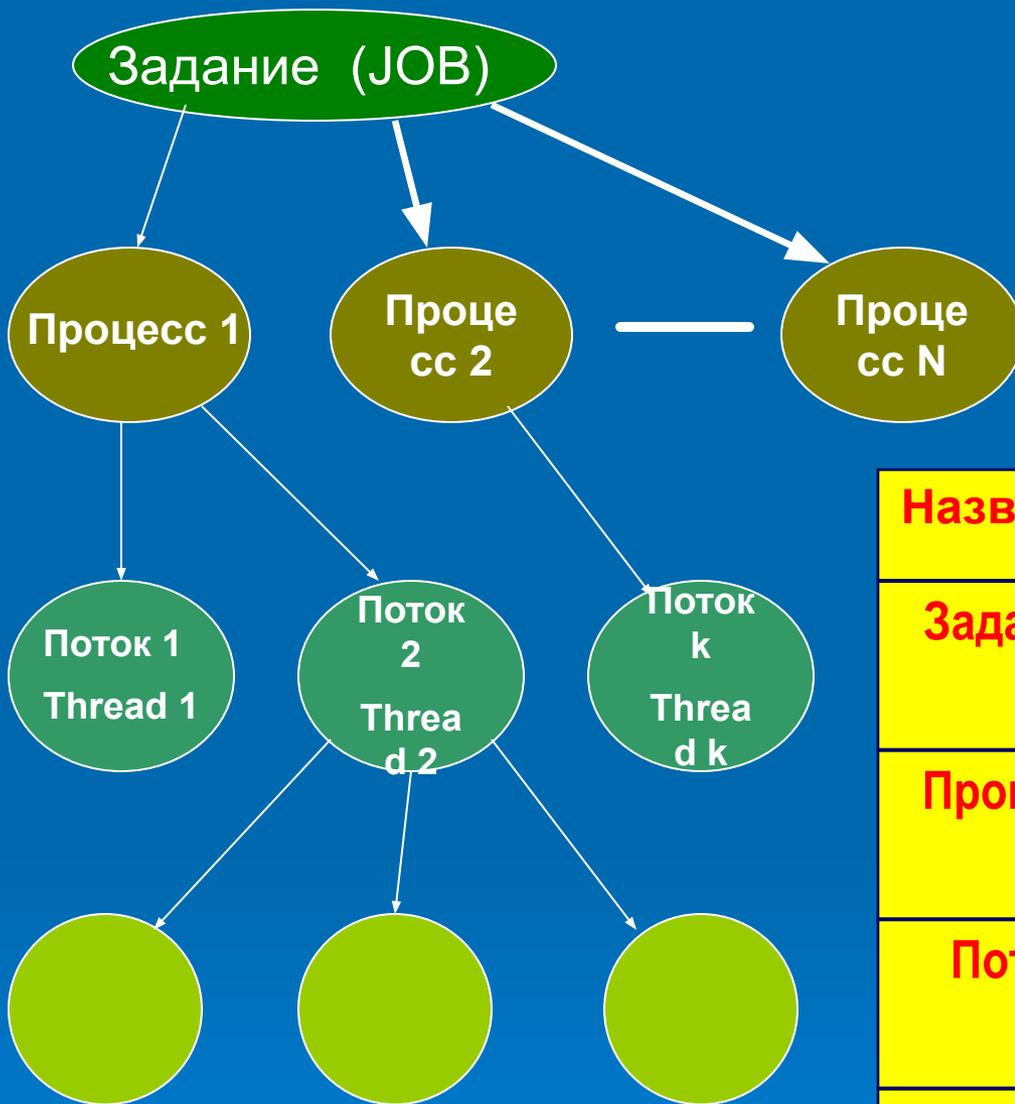


2.1. Концепция процессов и потоков. Задания, процессы, потоки (нити), волокна



Взаимосвязь между заданиями, процессами и потоками





Волокна (Fibers)

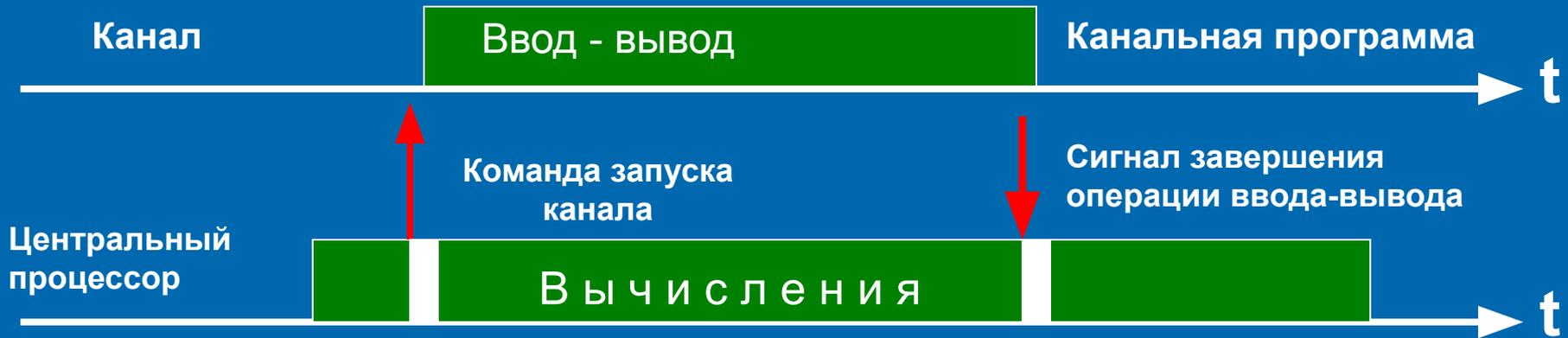
Объекты

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

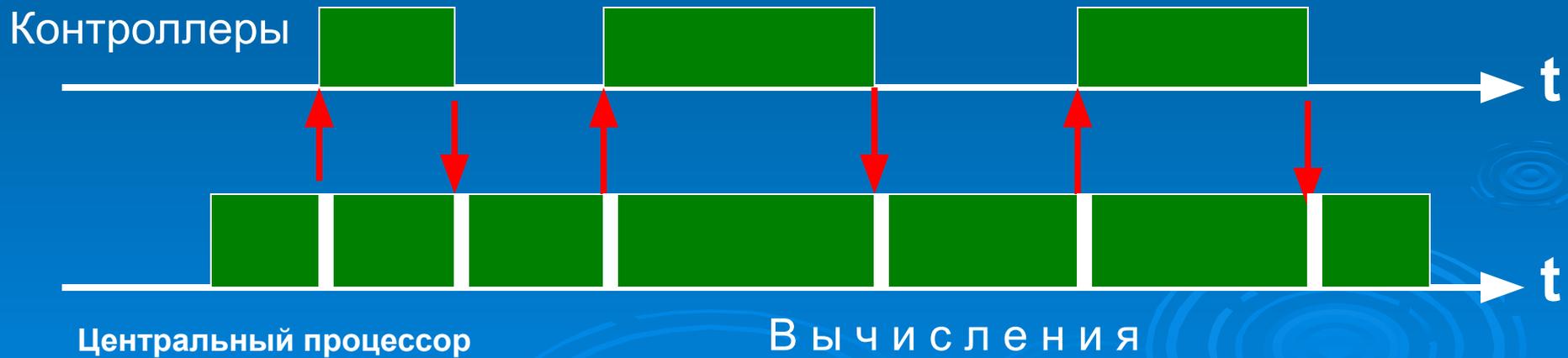


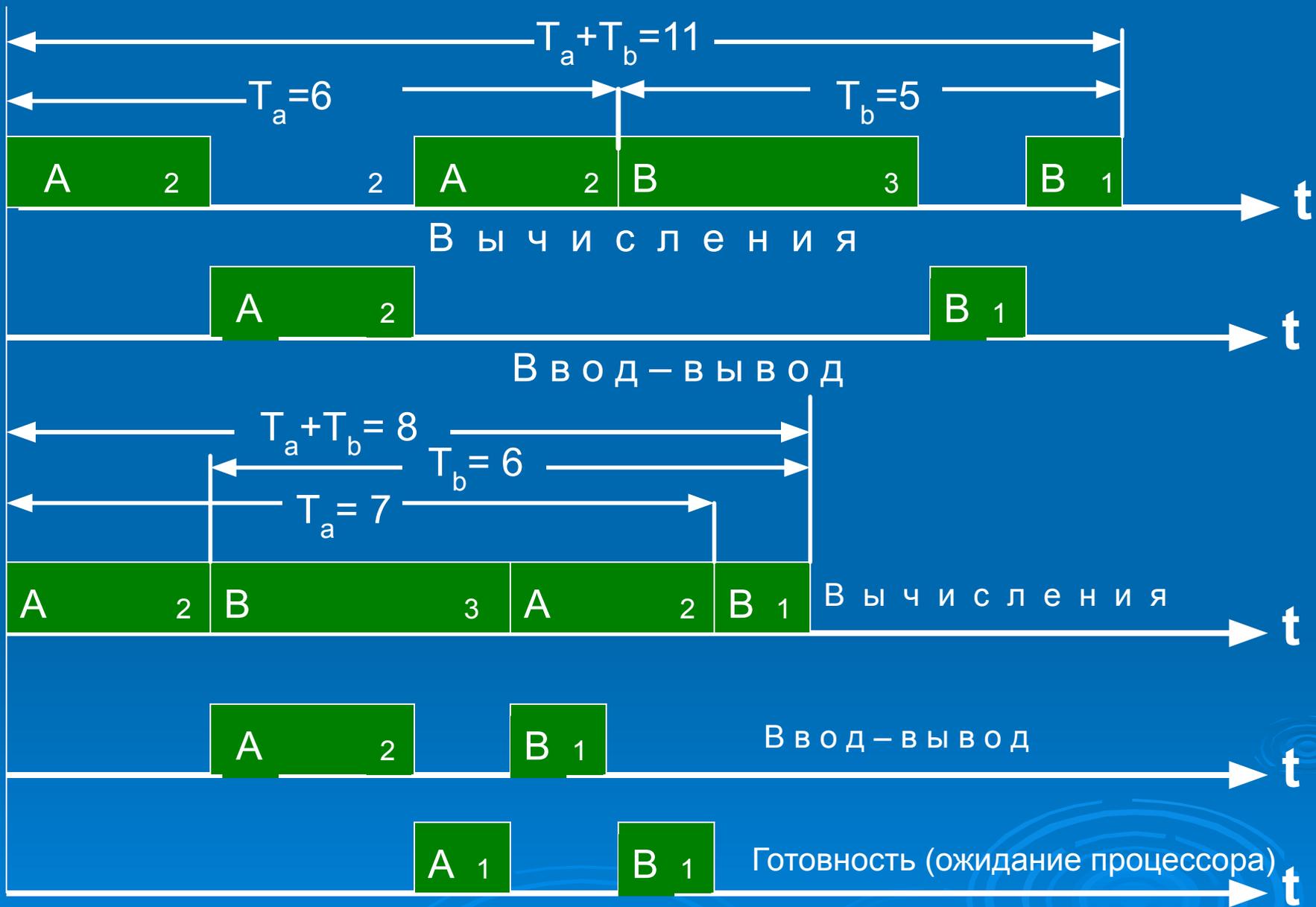
2.2. Мультипрограммирование. Формы многопрограммной работы

2.2.1. Мультипрограммирование в системах пакетной обработки



Операции ввода - вывода





2.2.2. Мультипрограммирование в системах разделения времени



2.2.3. Мультипрограммирование в системах реального времени

1. Управление техническими объектами, технологическими процессами, системами обслуживания и т. п.
2. Фиксированный набор заранее разработанных задач.
3. Жесткие ограничения на время обслуживания.
4. Режим типа запрос – ответ.

2.2.4. Мультипроцессорная обработка

1. Операционные системы: Windows NT/2000/2003, Sun Solaris 2/x, Santa Cruz Operations Open Server 3.x, OS/2 и др.
2. Симметричная архитектура и асимметричная архитектура.



2.3.2. Роль процессов, потоков и волокон в мультипрограммировании

1. Отдельный процесс не может быть выполнен быстрее, чем в однопрограммном режиме.
2. Сложно создать программу, реализующую параллелизм в рамках одного процесса.
3. Стандартные средства современных ОС не позволяют создать для одного приложения несколько процессов для параллельных работ.
4. Многопоточная обработка позволяет распараллелить вычисления в рамках одного процесса.
5. Многопоточная (multithreading) обработка эффективна в многопроцессорных вычислительных системах.
6. Использование аппарата волокон (Windows 2000) повышает эффективность мультипрограммирования за счет сокращения переключения процессов, но увеличивает трудоемкость разработки приложений.



2.3. Управление процессами и потоками

2.3.1. Основные функции управления процессами и потоками

1. Создание процессов и потоков.
2. Обеспечение процессов и потоков необходимыми ресурсами.
3. Изоляция процессов.
4. Планирование выполнения процессов и потоков.
5. Диспетчеризация потоков.
6. Синхронизация процессов и потоков.
7. Завершение и уничтожение процессов и потоков.

События, приводящие к созданию процессов:

1. Инициализация (загрузка) ОС.
2. Запрос процесса на создание дочернего процесса.
3. Запрос пользователя на создание процесса (например, при входе в систему в интерактивном режиме).
4. Инициирование пакетного задания.
5. Создание операционной системой процесса какой-либо службы.



2.4. Создание процессов и потоков. Модели процессов и потоков

2.4.1. Процессы и их модели

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





- закрытое *виртуальное адресное пространство* — диапазон адресов виртуальной памяти, которым может пользоваться процесс;
- исполняемую программу — начальный код и данные, проецируемые на виртуальное адресное пространство процесса;
- список открытых дескрипторов (*handles*) различных системных ресурсов — семафоров, коммуникационных портов, файлов и других объектов, доступных всем потокам в данном процессе;
- контекст защиты (*security context*), называемый *маркером доступа* (*access token*) и идентифицирующий пользователя, группы безопасности и привилегии, сопоставленные с процессом;
- уникальный идентификатор процесса (во внутрисистемной терминологии называемый идентификатором клиента);
- минимум один поток.



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



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 вершин стеков, хранящие параметры и адреса вызова процедур и системных служб.

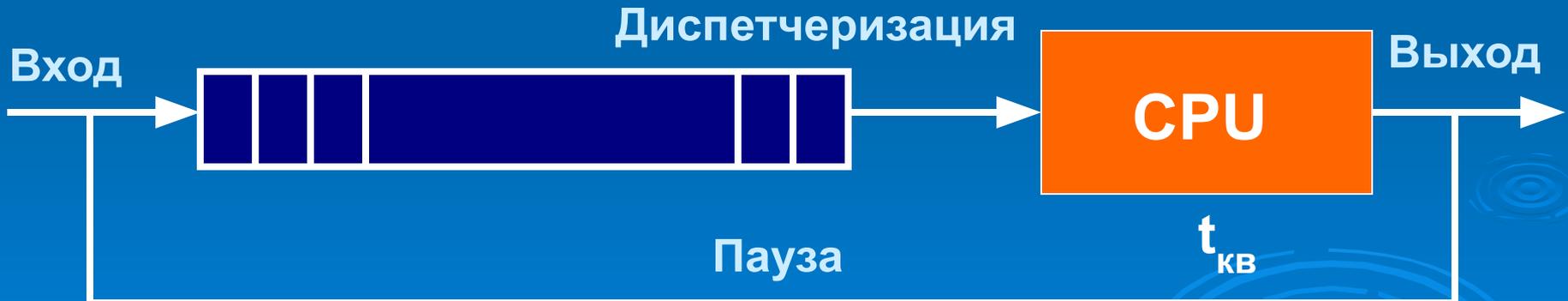
Значительная часть этой информации фиксируется в виде слова состояния программы PSW (program status word – EFLAGS в процессоре Pentium).

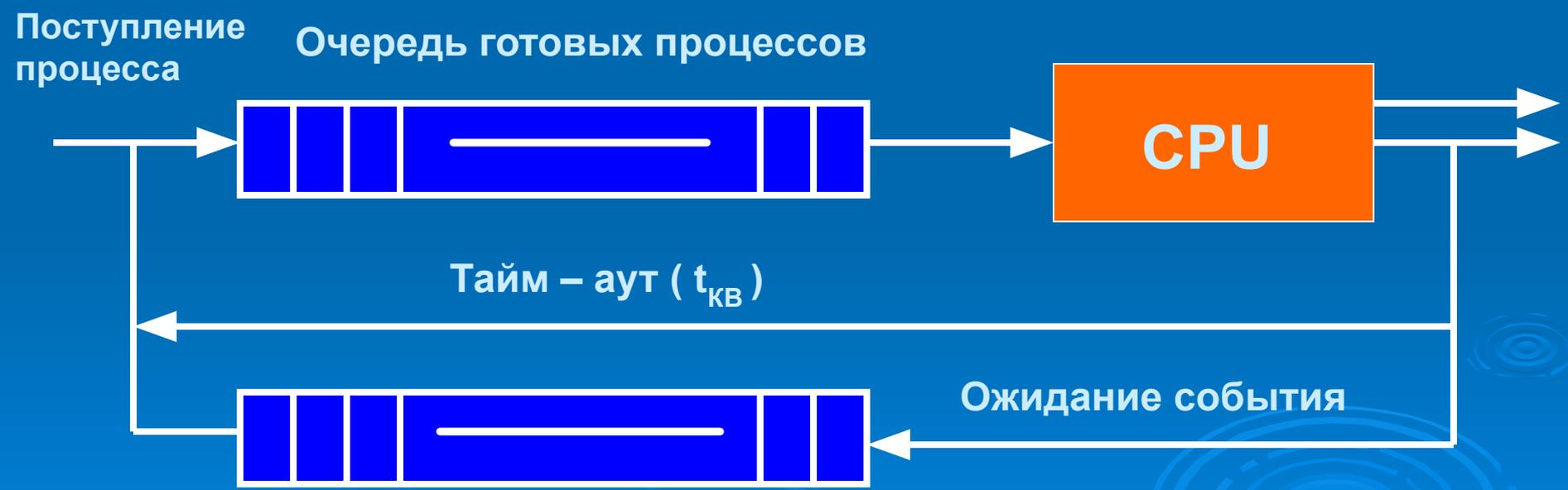
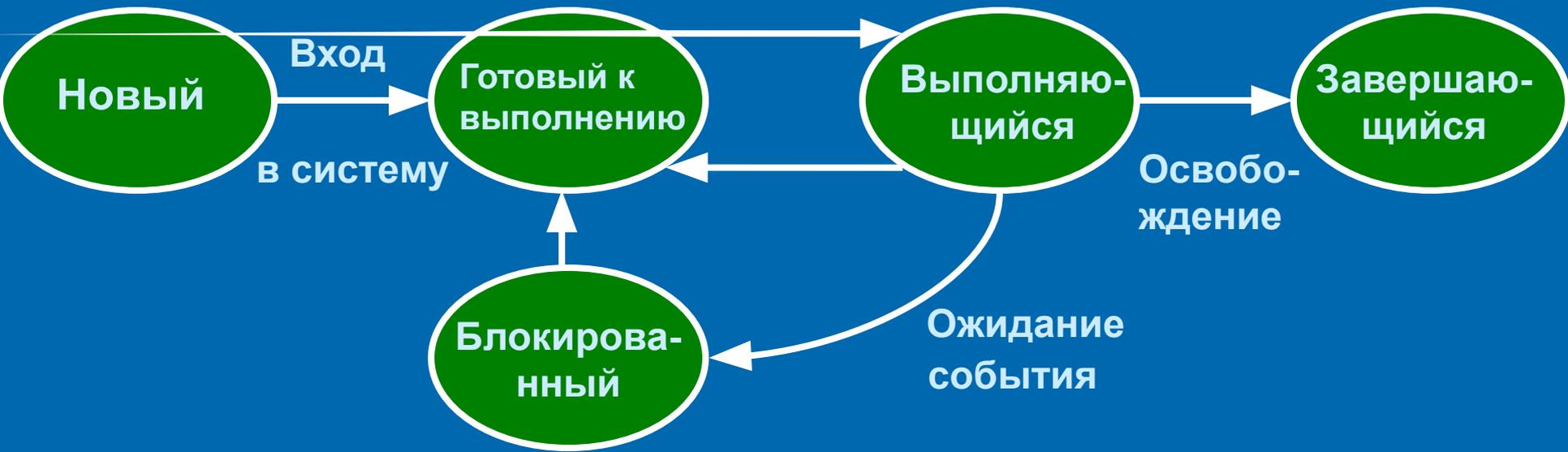


Простейшая модель процесса



Граф состояний и переходов





2.4.2. 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).



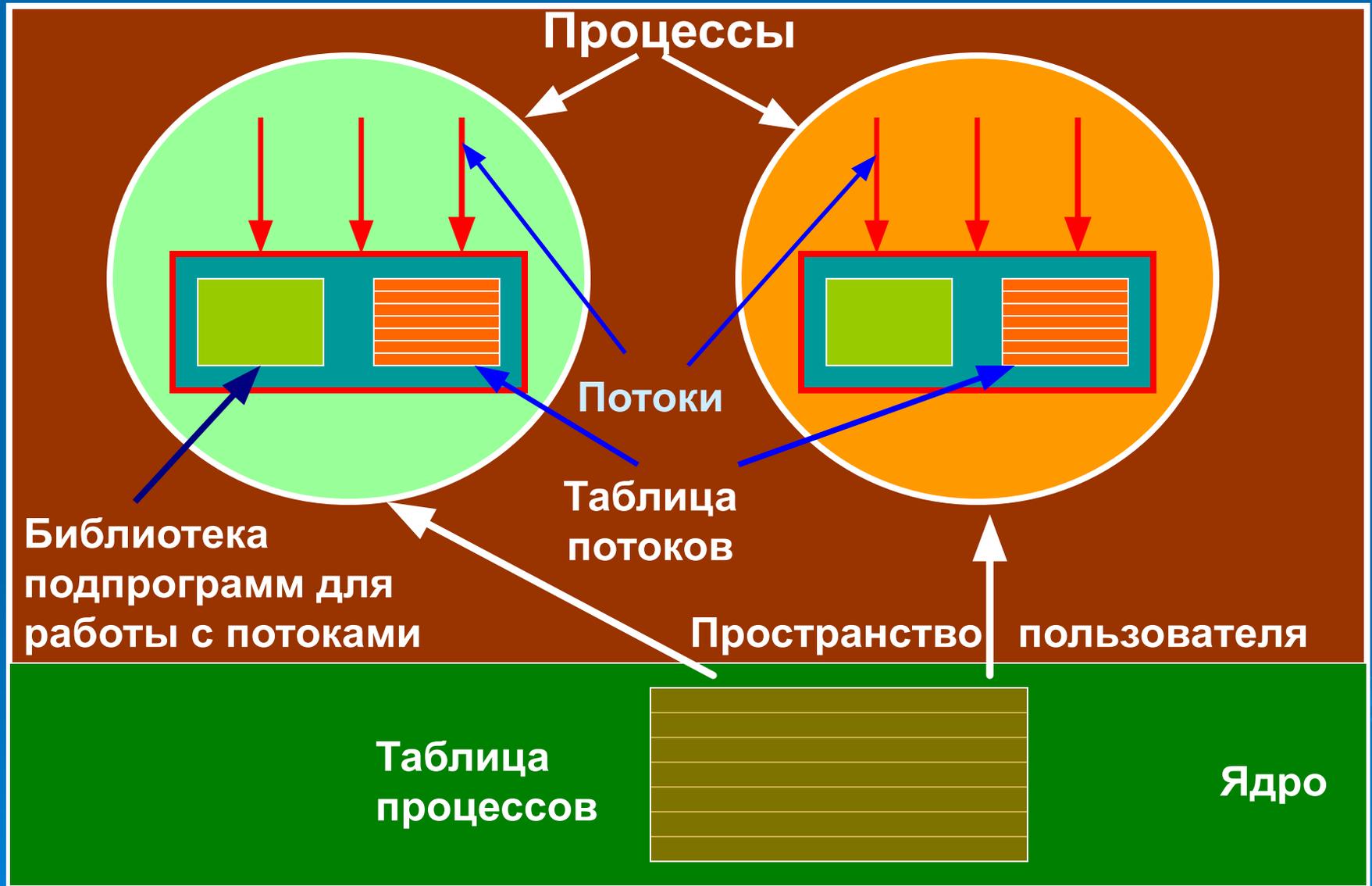
Поток (thread) — некая сущность внутри процесса, получающая процессорное время для выполнения. Без потока программа процесса не может выполняться. Поток включает следующие наиболее важные элементы:

- содержимое набора регистров процессора, отражающих состояние процессора;
- два стека, один из которых используется потоком при выполнении в режиме ядра, а другой — в пользовательском режиме;
- закрытую область памяти, называемую локальной памятью потока (thread-local storage, TLS) и используемую подсистемами, библиотеками исполняющих систем (run-time libraries) и DLL;
- уникальный идентификатор потока (во внутрисистемной терминологии также называемый идентификатором клиента: идентификаторы процессов и потоков генерируются из одного пространства имен и никогда не перекрываются);
- иногда потоки обладают своим контекстом защиты, который обычно используется многопоточными серверными приложениями, подменяющими контекст защиты обслуживаемых клиентов.

Переменные регистры, стеки и локальные области памяти называются *контекстом потока*. Поскольку эта информация различна на каждой аппаратной платформе, на которой может работать Windows, соответствующая



Поток на уровне пользователя (в пользовательском пространстве)



Поток на уровне пользователя

ДОСТОИНСТВА:

- можно реализовать в ОС, не поддерживающей потоки без каких-либо изменений в ОС;
- высокая производительность, поскольку процессу не нужно переключаться в режим ядра и обратно;
- ядро о потоках ничего не знает и управляет однопоточными процессами;
- имеется возможность использования любых алгоритмов планирования потоков с учетом их специфики;
- управление потоками возлагается на программу пользователя.



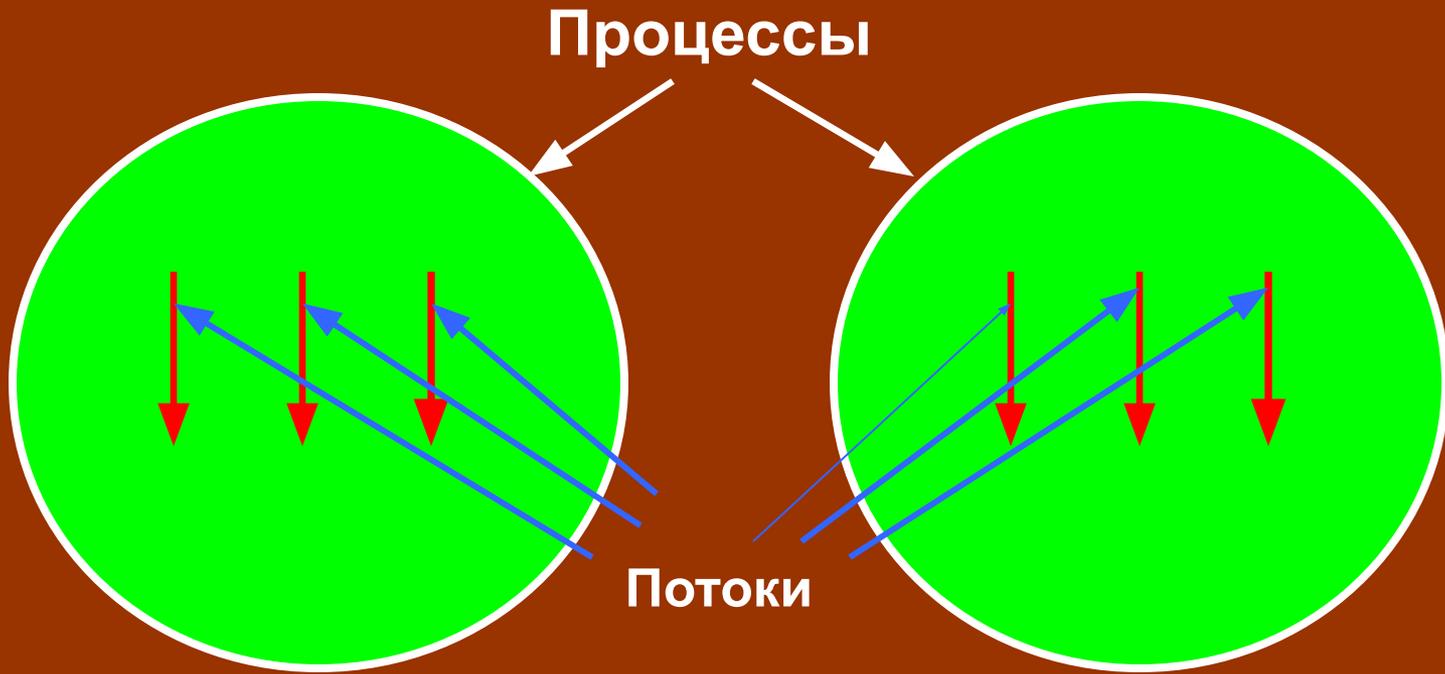
Поток на уровне пользователя

НЕДОСТАТКИ:

- ❑ системный вызов блокирует не только работающий поток, но и все потоки того процесса, к которому он относится;
- ❑ приложение не может работать в многопроцессорном режиме, так как ядро закрепляет за каждым процессом только один процессор;
- ❑ при запуске одного потока ни один другой поток в рамках одного процесса не будет запущен пока первый добровольно не отдаст процессор;
- ❑ внутри одного потока нет прерываний по таймеру, в результате чего невозможно создать планировщик по таймеру для поочередного выполнения потоков.



Поток на уровне ядра



Пространство пользователя

Ядро

Таблица
процессов

Таблица
потоков



Поток на уровне ядра

ДОСТОИНСТВА:

- возможно планирование работы нескольких потоков одного и того же процесса на нескольких процессорах;
- реализуется мультипрограммирование в рамках всех процессов (в том числе одного);
- при блокировании одного из потоков процесса ядро может выбрать другой поток этого же (или другого процесса);
- процедуры ядра могут быть многопоточными.

НЕДОСТАТКИ:

Необходимость двукратного переключения режима пользователь – ядро, ядро – пользователь для передачи управления от одного потока к другому в рамках одного и того же процесса.



2.5. Планирование заданий, процессов и потоков

1. Виды планирования

Вид планирования	Выполняемые функции
Долгосрочное	Решение о добавлении задания (процесса) в пул выполняемых в системе
Среднесрочное	Решение о добавлении процесса к числу процессов полностью или частично размещенных в основной памяти
Краткосрочное	Решение о том, какой из доступных процессов (потоков) будет выполняться процессором
Планирование ввода-вывода	Решение о том, какой из запросов процессов (потоков) на операцию ввода-вывода будет выполняться свободным устройством ввода-вывода



Схема планирования с учетом очередей заданий (процессов)



Типичный граф состояния потока



Алгоритмы планирования потоков

1. Невытесняющие (non-preemptive)

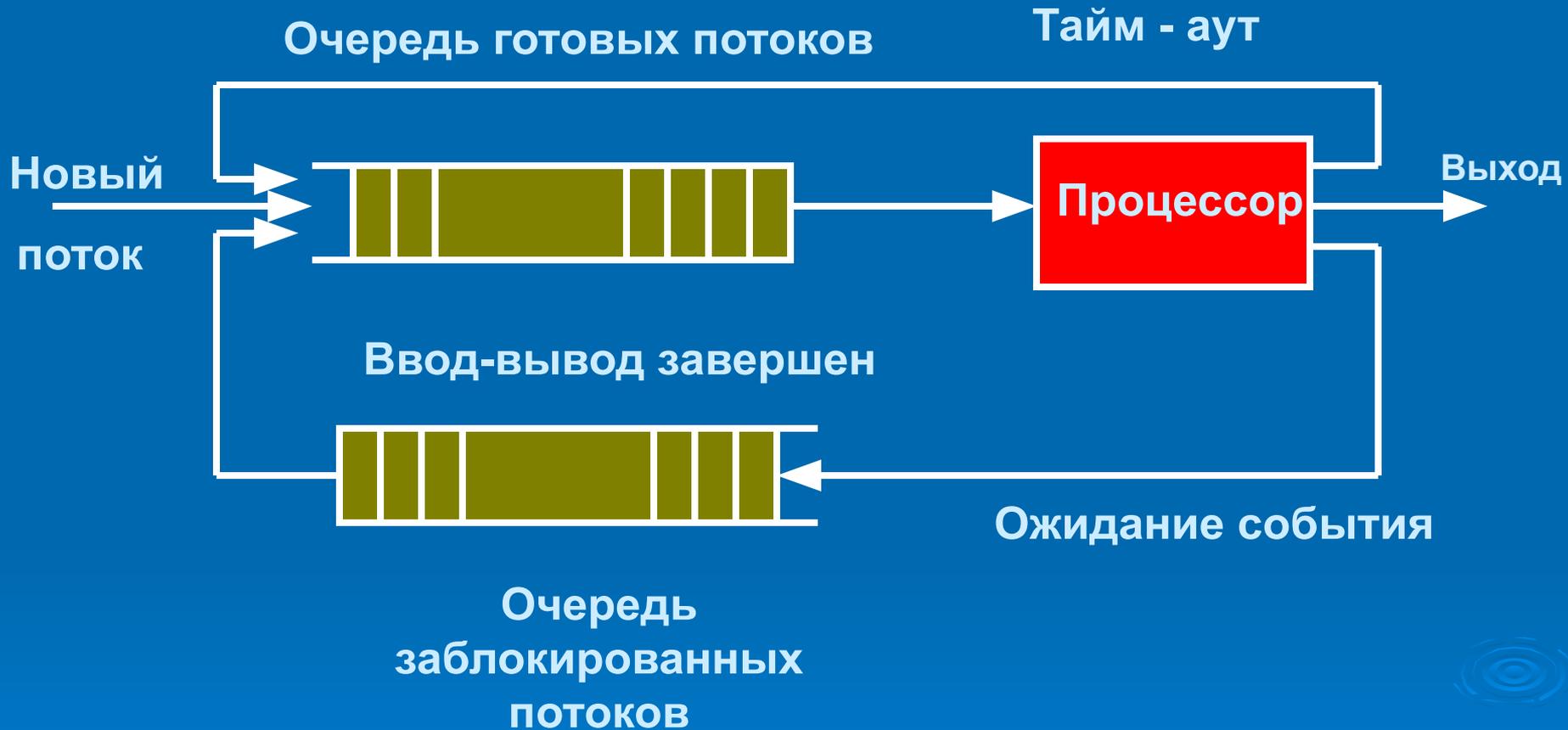
- ❑ планирование распределяется между ОС и прикладными программами;
- ❑ необходимость частых передач управлений ОС, в противном случае возможна монополизация процессора приложением;
- ❑ зависания приложений могут привести к краху системы

2. Вытесняющие (preemptive)

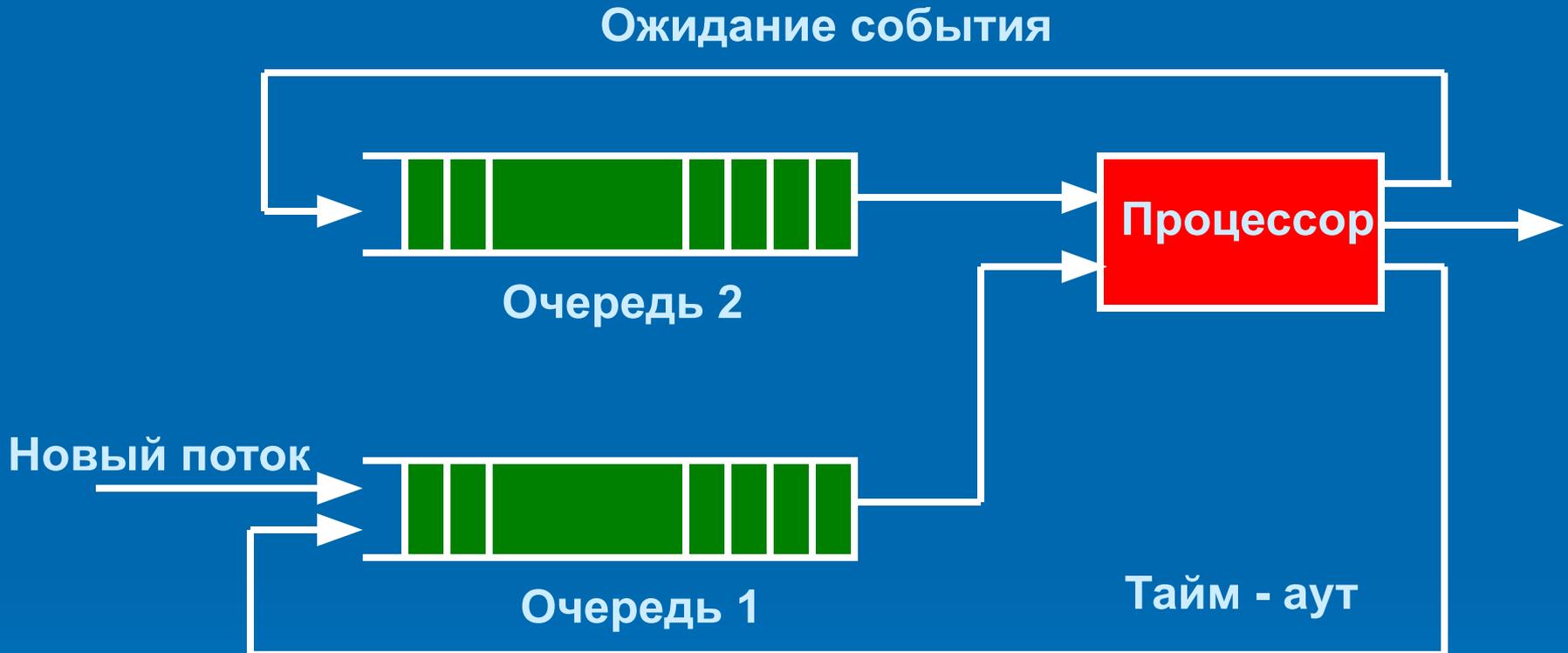
- ❑ функции планирования сосредоточены в ОС;
- ❑ планирование на основе квантования процессорного времени;
- ❑ планирование на основе приоритетов потоков: статических, динамических, абсолютных, относительных, смешанных;



Простейший алгоритм планирования, реализующий состояния потока по кадру 27



Алгоритм планирования, реализующий предпочтения потокам с интенсивным вводом-выводом



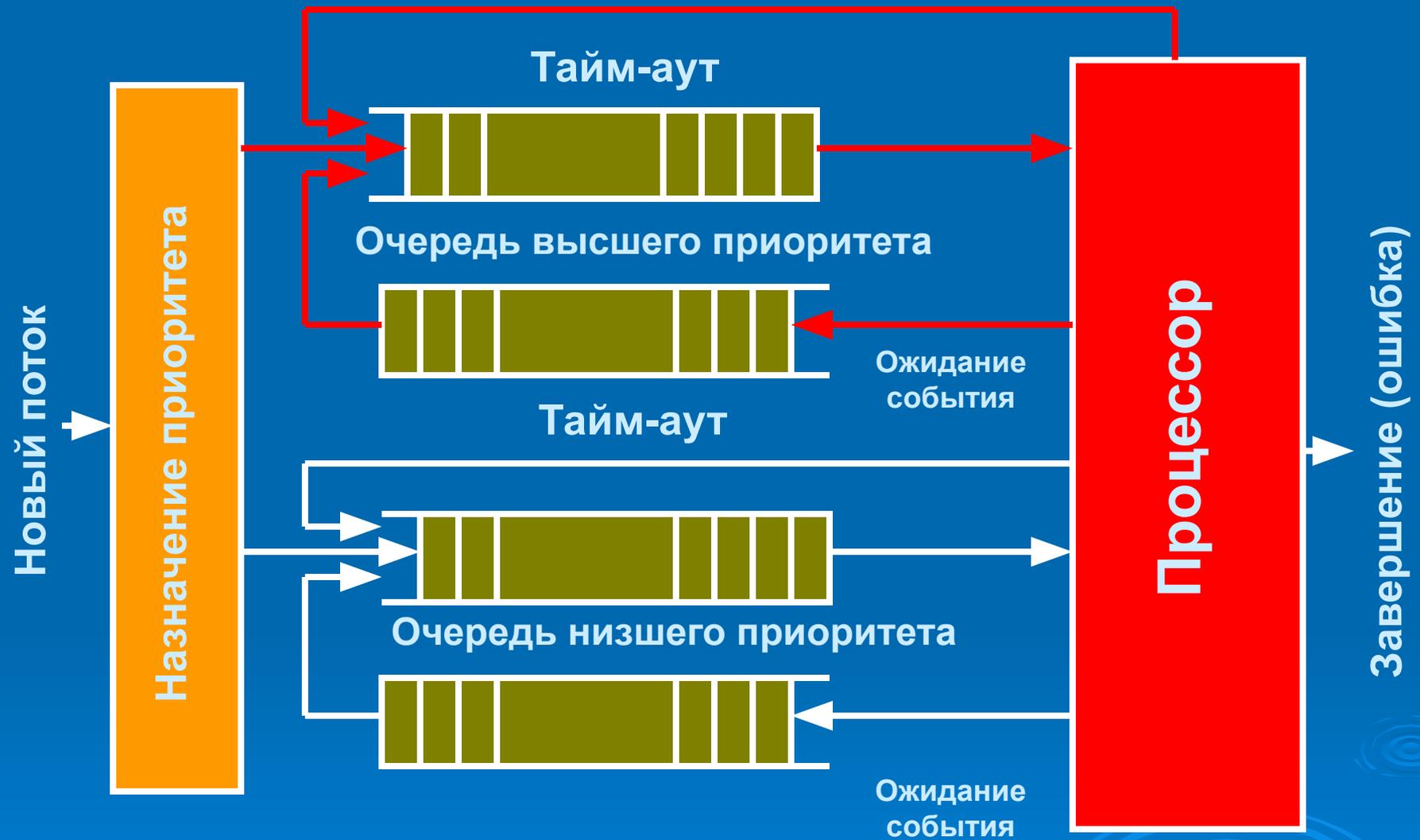
1. Переключение контекстов потоков связано с потерями процессорного времени.
2. С увеличением времени кванта ухудшается обслуживание пользователей.
3. В алгоритмах, основанных на квантовании, ОС не имеет никаких сведений о характеристиках решаемых задач.



Граф состояния потока

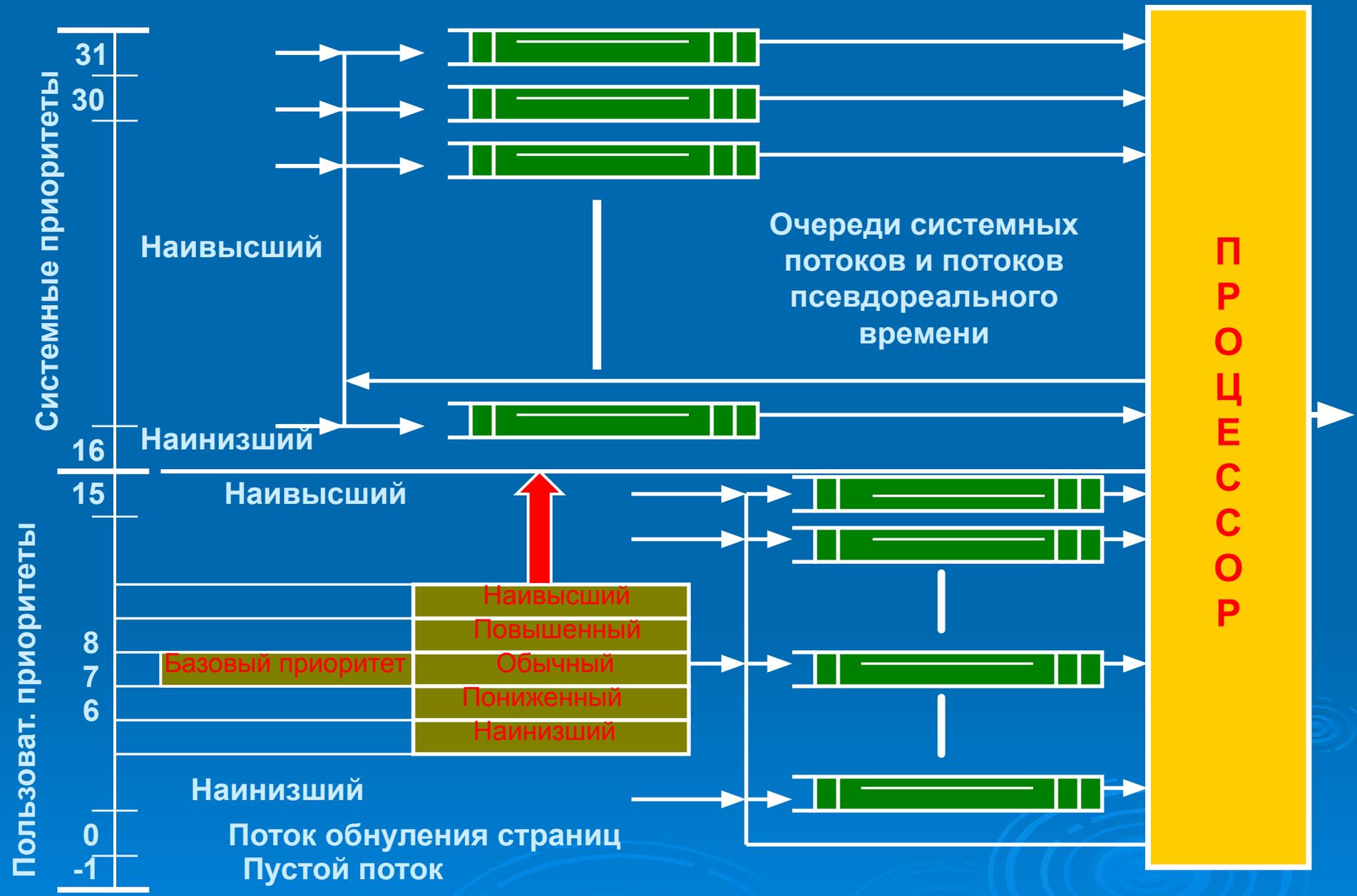


Алгоритмы приоритетного планирования



Приоритетное переключение с квантованием





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).



Работоспособные процессы (потoki)



Неработоспособные процессы (потoki)

2.6. Взаимодействие и синхронизация процессов и потоков

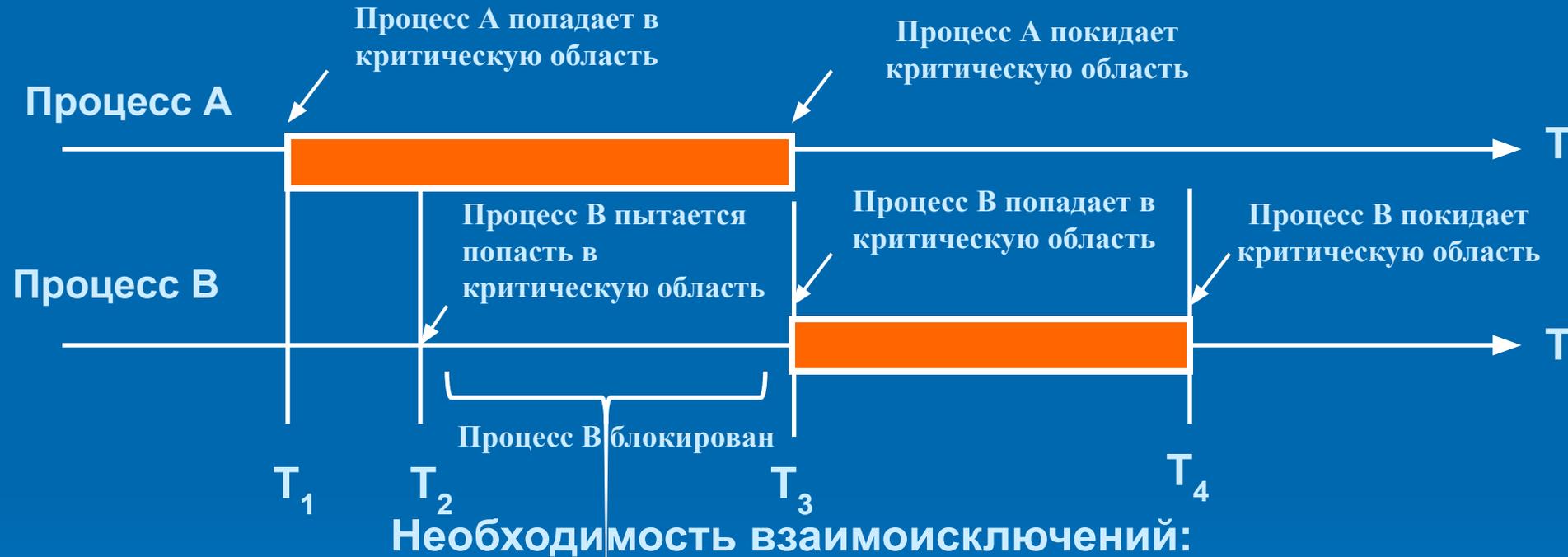
2.6.1. Проблемы взаимодействия и синхронизации

Степень осведомленности	Взаимосвязь	Влияние одного процесса на другой	Потенциальные проблемы
Процессы не осведомлены друг о друге	Конкуренция	<ul style="list-style-type: none"> ❑ Результат работы одного процесса не зависит от действий других. ❑ Возможно влияние одного процесса на время работы другого. 	<ul style="list-style-type: none"> ❑ Взаимоисключения ❑ Взаимоблокировки ❑ Голодание
Процессы косвенно осведомлены о наличии друг друга	Сотрудничество с использованием разделения	<ul style="list-style-type: none"> ❑ Результат работы одного процесса может зависеть от информации, полученной от других. ❑ Возможно влияние одного процесса на время работы другого. 	<ul style="list-style-type: none"> ❑ Взаимоисключения ❑ Взаимоблокировки ❑ Голодание ❑ Синхронизация
Процессы непосредственно осведомлены о наличии друг друга	Сотрудничество с использованием связи	<ul style="list-style-type: none"> ❑ Результат работы одного процесса зависит от информации, полученной от других процессов. ❑ Возможно влияние одного процесса на время работы другого. 	<ul style="list-style-type: none"> ❑ Взаимоблокировки (расходуемые ресурсы) ❑ Голодание



2.6.2. 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.



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



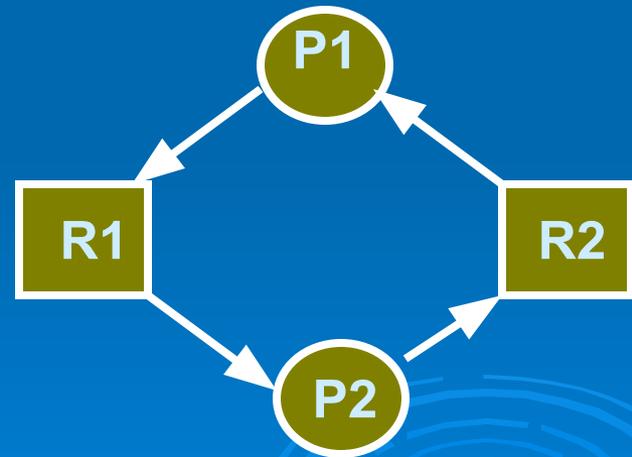
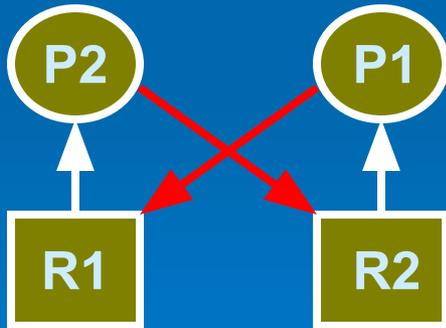
Процесс



Ресурс



Исходное
распределение
ресурсов



Тупиковая ситуация



Проблема “голодание”



Активный

Блокированные

Блокированные

Активный

Активный

Блокированные

Блокированные

Активный



2.6.3. 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**.

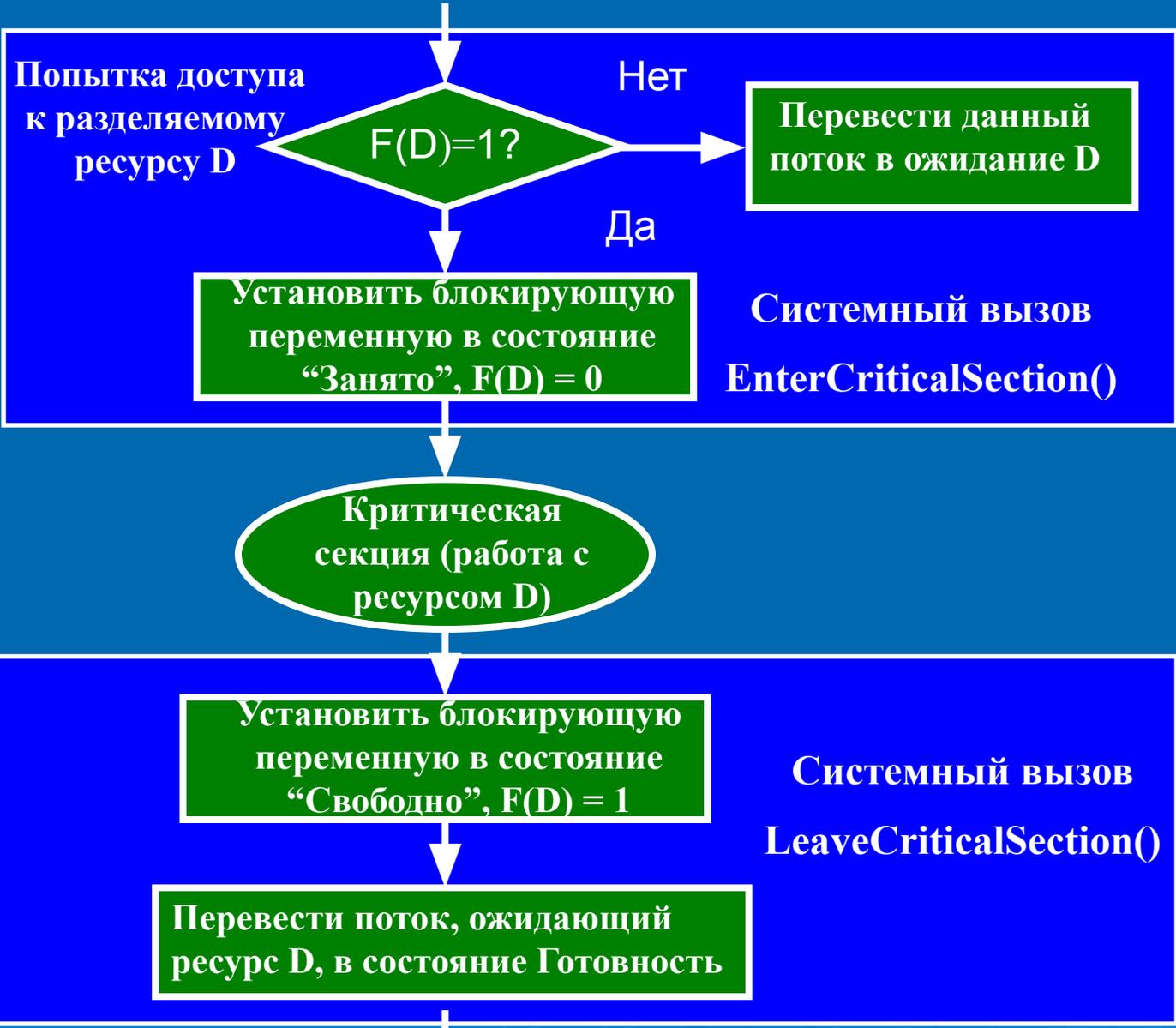


2.6.4. Методы взаимоисключений

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)**



3.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



4. 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 включает в себе потенциальную возможность перехода процесса, который ее выполняет, в состояние ожидания (если $S = 0$).

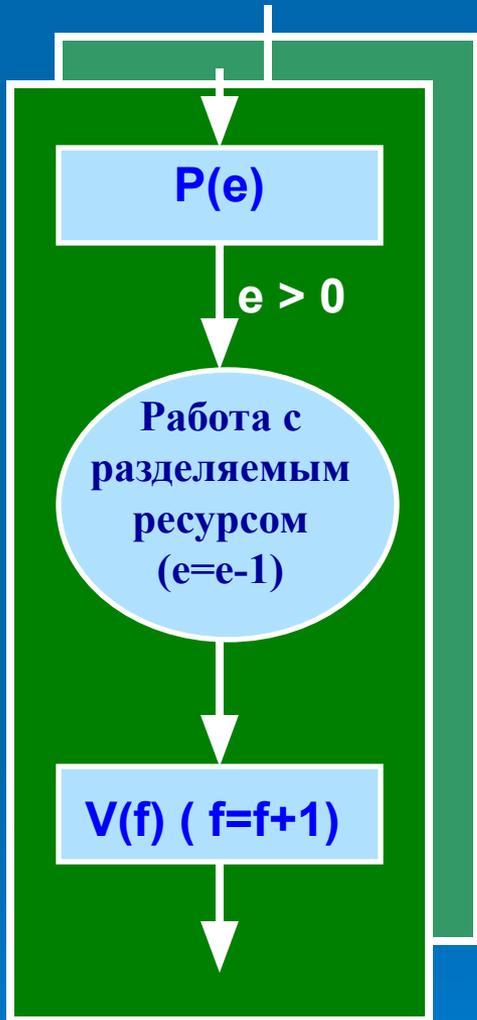
Операция V может при некоторых обстоятельствах активизировать процесс, приостановленный операцией P .

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

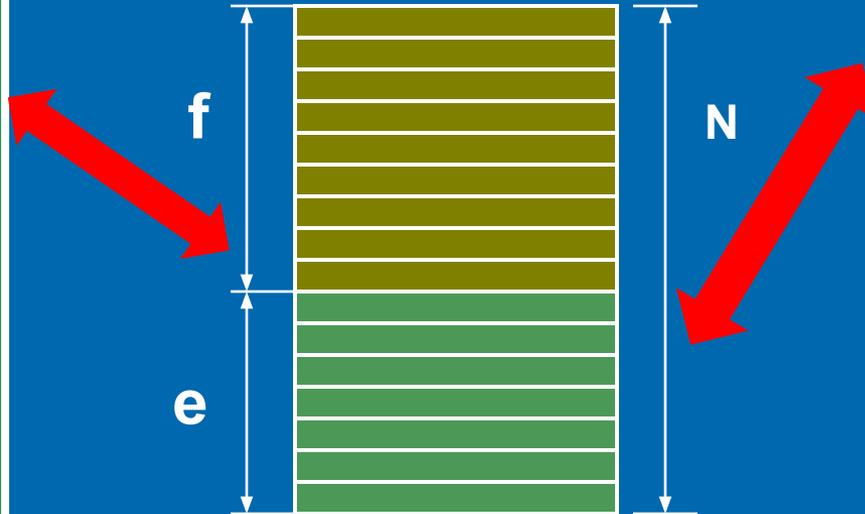


Начальные значения семафоров

$$e = N; f = 0$$



Потоки-писатели



Буферный пул

e – пустые буферы,
 f – занятые буферы



Потоки-читатели



2.6.5. 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:

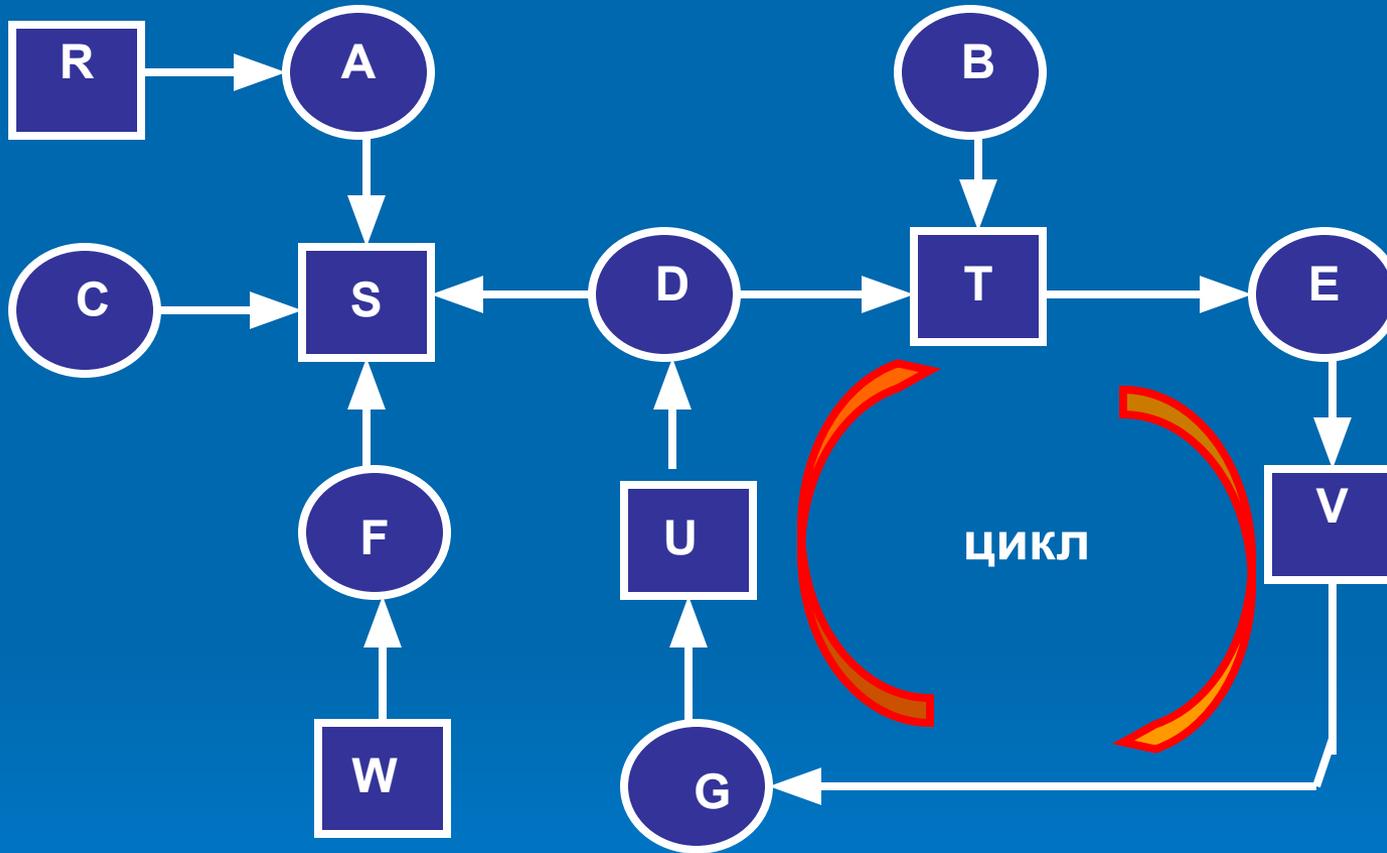
- ❑ Процесс A занимает ресурс R и хочет получить ресурс S.
- ❑ Процесс B ничего не использует, но хочет получить ресурс T.
- ❑ Процесс C ничего не использует, но хочет получить ресурс S.
- ❑ Процесс D занимает ресурс U и хочет получить ресурсы S и T.
- ❑ Процесс E занимает ресурс T и хочет получить ресурс 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.

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

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

$A = \{A_1, A_2, \dots, A_m\}$ – вектор свободных ресурсов; $A_j \leq E_j, j = \overline{1, m}$;

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

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

Существующие ресурсы

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

$$\begin{array}{cccc} c_{11} & c_{12} & \dots & c_{1m} \\ c_{21} & c_{22} & \dots & c_{2m} \\ \cdot & \cdot & \dots & \cdot \\ c_{n1} & c_{n2} & \dots & c_{nm} \end{array}$$

Доступные ресурсы

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

$$\begin{array}{cccc} r_{11} & r_{12} & \dots & r_{1m} \\ r_{21} & r_{22} & \dots & r_{2m} \\ \cdot & \cdot & \dots & \cdot \\ r_{n1} & r_{n2} & \dots & r_{nm} \end{array}$$

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



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.

Алгоритм

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



2.6.6. 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 между процессами, а также процессами и пользователями (терминалами).



2.7. Аппаратно-программные средства поддержки мультипрограммирования

2.7.1. Системы прерываний

Классы прерываний: внешние, внутренние, программные

1. Внешние прерывания – результат действий пользователя, сигналы от периферийных устройств компьютера и управляемых объектов.
2. Внутренние прерывания – результат появления аварийных ситуаций при выполнении инструкции программы.
3. Программные прерывания – результат выполнения запланированных в программе особых инструкций (системный вызов).

Принципы построения систем прерываний:

- аппаратная поддержка (контроллер прерываний, контроллер DMA, контроллеры внешних устройств, шины подключения внешних устройств, средства микропроцессора);
- векторный, опрашиваемый и комбинированный способы прерываний;
- приоритетный механизм обслуживания (с абсолютными и относительными приоритетами);
- маскирование прерываний;
- диспетчер прерываний и процедуры обслуживания прерываний.



Обработчики ловушек



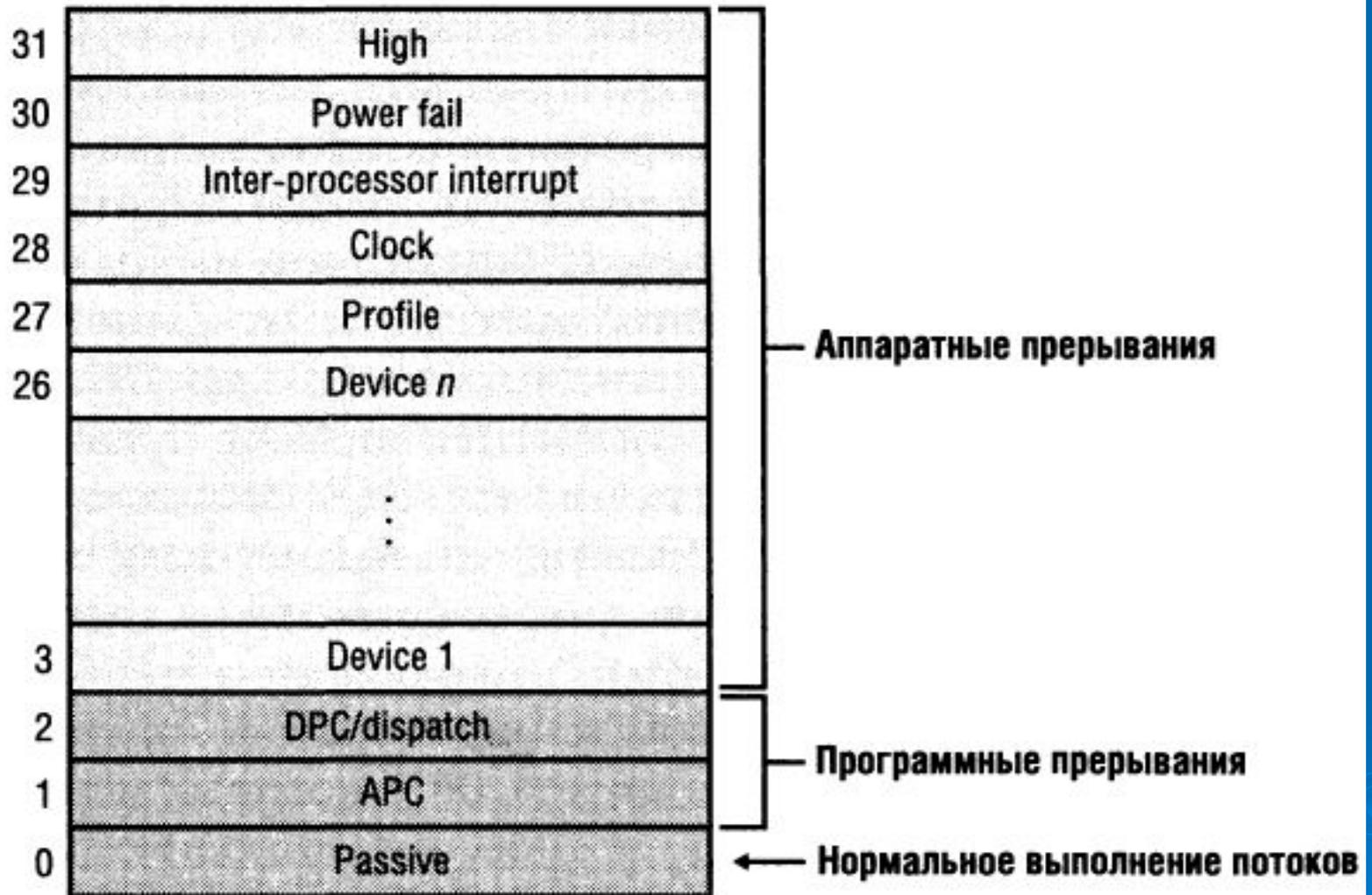


Рис. 3-3. Уровни запросов прерываний (IRQL) в x86-системах



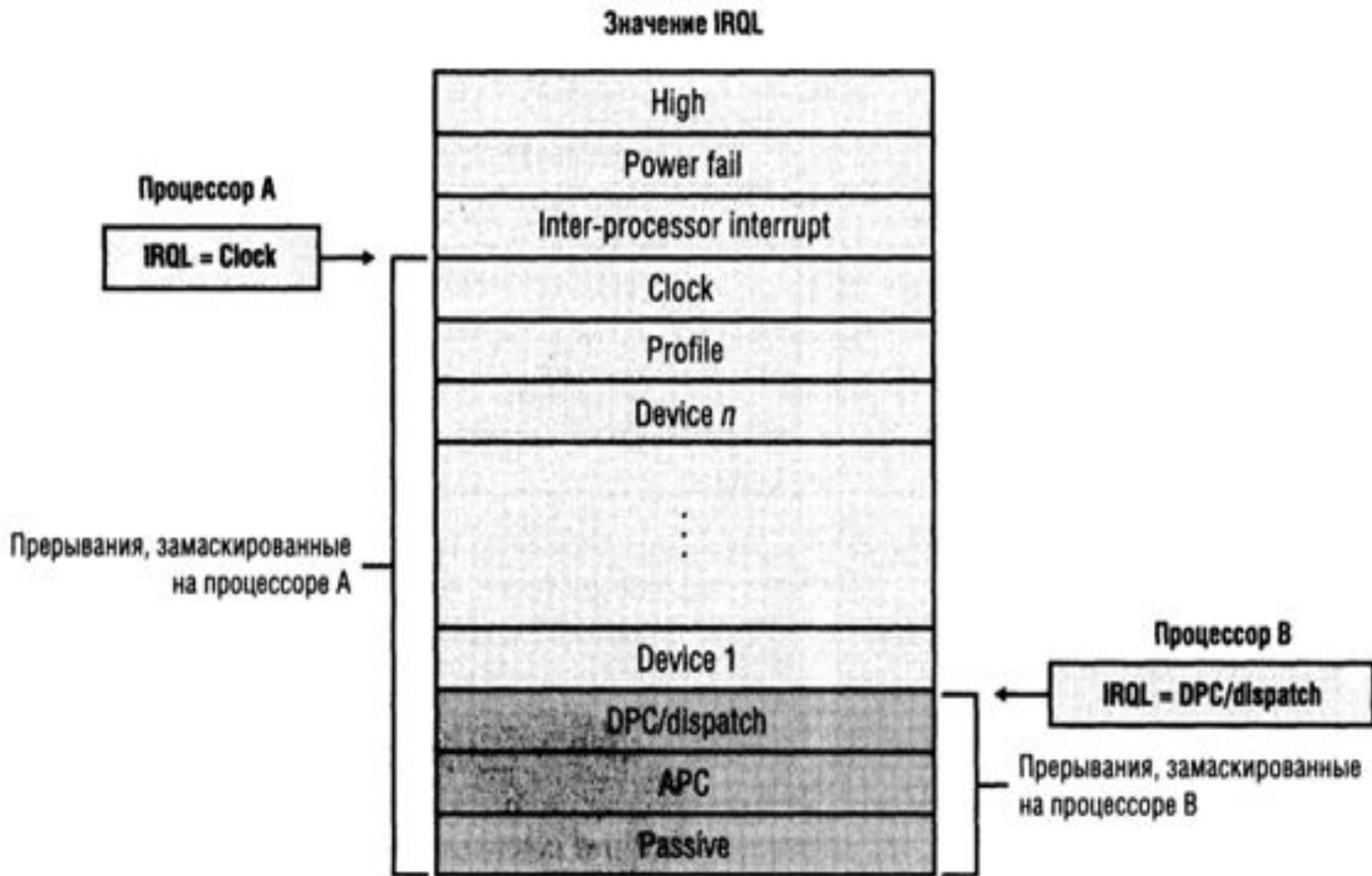


Рис. 3-5. Маскирование прерываний



- Уровень «high» (высокий) используется ядром, только если оно останавливает систему в функции *KeBugCheckEx* и маскирует все прерывания.
- Уровень «power fail» (отказ электропитания) был заложен еще в самый первый проект Microsoft Windows NT. Он определяет поведение системы при отказе электропитания, но никогда не применялся.
- Уровень «interprocessor interrupt» (межпроцессорное прерывание) используется для того, чтобы запрашивать от другого процессора выполнение какой-либо операции, например, при постановке в очередь прерывания DISPATCH_LEVEL для планирования конкретного потока к выполнению, при обновлении кэша TLB, завершении работы или крахе системы.
- Уровень «clock» (часы) используется для системных часов, с помощью которых ядро отслеживает время суток, измеряет и распределяет процессорное время между потоками.
- Уровень «profile» (профиль) используется системным таймером реального времени, если активизирован механизм профилирования ядра (kernel profiling), т. е. измерения его производительности. Когда он активен, обработчик ловушки профилирования регистрирует адрес команды, выполнявшейся на момент прерывания. Со временем создается таблица адресов, которую можно извлечь и проанализировать с помощью соответствующих утилит. Вы можете скачать утилиту Kernrate (<http://www.microsoft.com/whdc/system/sysperf/krview.mspx>), позволяющую просматривать статистику, полученную при использовании механизма профилирования ядра. Подробнее об этой утилите см. описание эксперимента с Kernrate.
- Уровень «device» (устройство) применяется для задания приоритетов прерываний от устройств (о принципах увязки аппаратных прерываний с IRQL см. предыдущий раздел).
- Прерывания уровней «DPC/dispatch» и «APC» являются программными; они генерируются ядром и драйверами устройств (о DPC и APC будет рассказано позже).
- Самый низкий уровень IRQL, «passive» (пассивный), на самом деле вообще не является уровнем прерывания. При этом значении IRQL потоки выполняются обычным образом и могут возникать любые прерывания.



Программные прерывания

Хотя большинство прерываний генерируется аппаратно, ядро Windows тоже может генерировать прерывания — только они являются программными. Этот вид прерываний служит для решения многих задач, в том числе:

- инициации диспетчеризации потоков;
- обработки прерываний, не критичных по времени;
- обработки событий таймеров;
- асинхронного выполнения какой-либо процедуры в контексте конкретного потока;
- поддержки асинхронного ввода-вывода.

Эти задачи подробно рассматриваются ниже.

Прерывания DPC или диспетчеризации Когда дальнейшее выполнение потока невозможно, например из-за его завершения или перехода в ждущее состояние, ядро напрямую обращается к диспетчеру, чтобы вызвать немедленное переключение контекста. Однако иногда ядро обнаруживает, что перераспределение процессорного времени (rescheduling) должно произойти при выполнении глубоко вложенных уровней кода. В этой ситуации ядро запрашивает диспетчеризацию, но саму операцию откладывает до выполнения текущих действий. Такую задержку удобно организовать с помощью программного прерывания DPC (deferred procedure call).



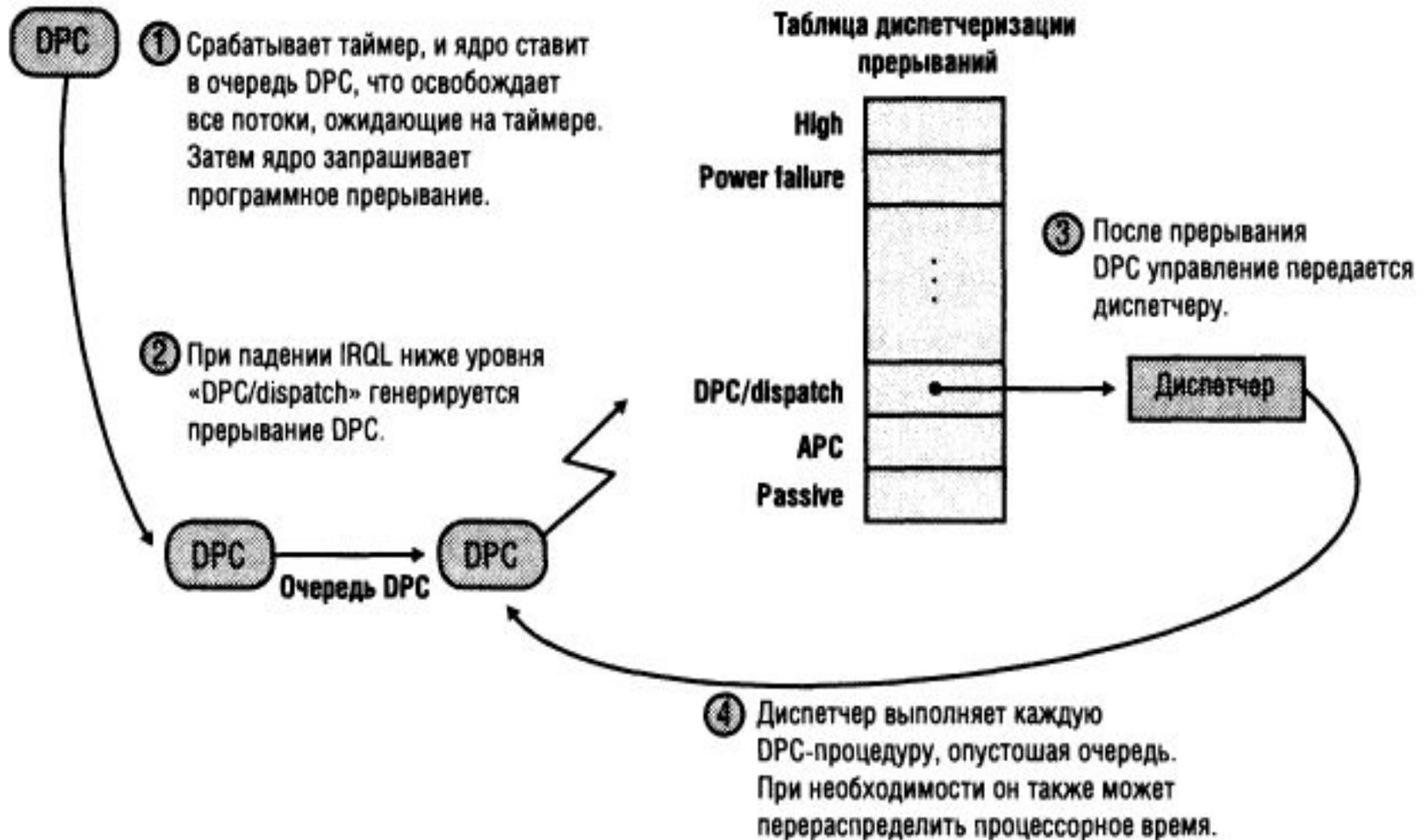


Рис. 3-7. Обработка DPC



Таблица 3-2. Исключения в системах типа x86 и соответствующие им номера прерываний

Номер прерывания	Исключение
0	Divide Error (ошибка деления)
1	DEBUG TRAP (ловушка отладки)
2	NMI/NPX Error (ошибка NMI/NPX)
3	Breakpoint (точка прерывания)
4	Overflow (переполнение)
5	BOUND/Print Screen
6	Invalid Opcode (неправильный код операции)
7	NPX Not Available (NPX недоступен)
8	Double Exception (двойное исключение)
9	NPX Segment Overrun (выход за пределы сегмента NPX)
A	Invalid Task State Segment (TSS) (неправильный TSS)
B	Segment Not Present (сегмент отсутствует)
C	Stack Fault (ошибка стека)
D	General Protection (ошибка общей защиты)
E	Page Fault (ошибка страницы)
F	Зарезервировано Intel
10	Floating Point (ошибка в операции с плавающей точкой)
11	Alignment Check (ошибка контроля выравнивания)



Последовательность действий при обработке прерываний

1. **Первичное аппаратное распознавание типа прерывание. Если прерывания запрещены, продолжается текущая программа. В противном случае вызывается диспетчер прерываний и в зависимости от поступившей в процессор информации (вектор прерывания, приоритет и др.) производится вызов процедуры обработки прерывания.**
2. **Сохраняется некоторая часть контекста прерванного потока, которая позволит возобновить его исполнение после обработки прерывания (обычно слово состояния процессора – регистр EFLAGS в Pentium, регистры общего назначения). Может быть сохранен и полный контекст, если ОС обслуживает прерывание со сменой процесса.**
3. **В счетчик команд загружается адрес процедуры обработки прерывания и устанавливается новое PSW, которое определяет привилегированный режим работы процессора при обработке прерывания.**
4. **Маскированием прерываний временно запрещаются прерывания, чтобы не образовалась очередь вложенных друг в друга потоков одной и той же процедуры.**
5. **После обработки прерывания ядром операционной системы, прерванный контекст восстанавливается (частично аппаратно – PSW, содержимое счетчика команд, частично программно – извлечение данных из стека), снимается обработка прерываний данного типа и работа потока возобновляется с прерванного места.**



Обработка прерываний в UNIX

1. Аппаратное обеспечение сохраняет в стеке счетчик команд и другую информацию.
2. Аппаратное обеспечение загружает новое значение счетчика команд из вектора прерываний.
3. Процедура на ассемблере сохраняет регистры.
4. Процедура на ассемблере устанавливает новый стек.
5. Запускается программа обработки прерываний на C. (Она обычно считывает и буферизирует входные данные).
6. Планировщик выбирает следующий процесс.
7. Программа на c передает управление процедуре на ассемблере.
8. Процедура на ассемблере запускает новый процесс (возможно прерванный программой обработки прерывания).



2.7.2. Системные вызовы

Системный вызов позволяет приложению обратиться к ОС с просьбой выполнить то или иное действие, оформленное как процедура кодового сегмента ОС.

Реализация системных вызовов должна удовлетворять следующим требованиям:

- обеспечивать переключение в привилегированный режим;
- обладать высокой скоростью вызова процедур ОС;
- обеспечивать по возможности единообразное обращение к системным вызовам для всех аппаратных платформ, на которых работает ОС;
- допускать простое расширение системных вызовов;
- обеспечивать контроль со стороны ОС за корректным использованием системных вызовов.

Возможные схемы обслуживания системных вызовов:

1. Децентрализованная – за каждым системным вызовом закреплен свой вектор прерываний. Достоинство – высокая скорость обработки системных вызовов, недостаток – разрастание таблицы векторов прерываний.
2. Централизованная – с помощью диспетчера системных вызовов.

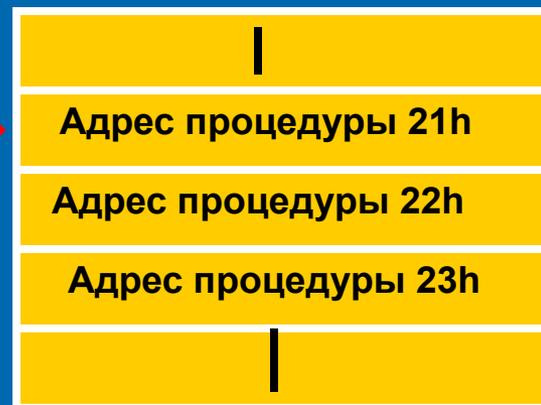
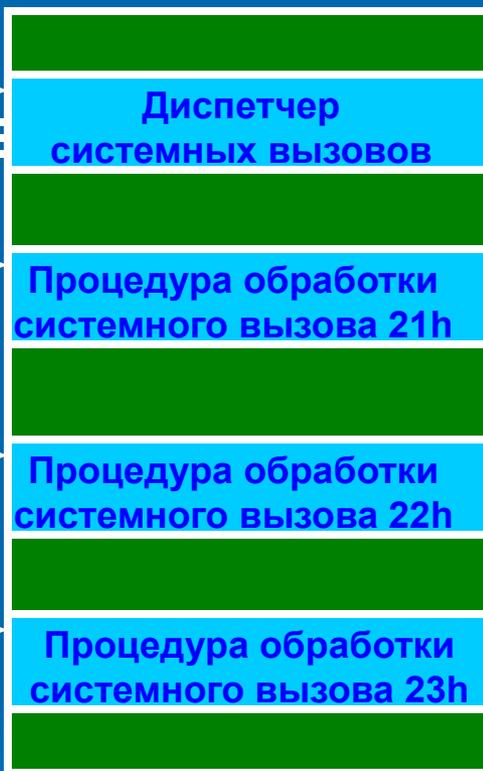


Таблица прерываний системы



Вектор
80h
Linux
RQ=21h

Виртуальное адресное пространство



INT 2Eh
Pentium

Централизованная схема обработки системных вызовов



Диспетчеризация системных сервисов

Как показано на рис. 3-1, обработчики ловушек ядра обслуживают прерывания, исключения и вызовы системных сервисов. Из предыдущих разделов вы знаете, как проводится обработка прерываний и исключений. Здесь будет рассказано о вызовах системных сервисов. Диспетчеризация системных сервисов начинается с выполнения инструкции, закрепленной за такой диспетчеризацией. Эта инструкция зависит от процессора, на котором работает Windows.

Диспетчеризация 32-разрядных системных сервисов

На процессорах x86 до Pentium II использовалась инструкция *int 0x2e* (десятичное значение 46). В результате выполнения этой инструкции срабатывает ловушка, и Windows заносит в запись IDT под номером 46 указатель на

диспетчер системных сервисов (см. таблицу 3-1). Эта ловушка заставляет выполняемый поток переключиться в режим ядра и войти в диспетчер системных сервисов. Номер запрошенного системного сервиса указывается числовым аргументом, переданным в регистр процессора EAX. Содержимое регистра EBX указывает на список параметров, передаваемый системному сервису вызывающей программой.

На x86-процессорах Pentium II и выше Windows использует инструкцию *sysenter*, которую Intel специально определил для быстрой диспетчеризации системных сервисов. Для поддержки этой инструкции Windows сохраняет на этапе загрузки адрес процедуры ядра — диспетчера системных сервисов в регистре, сопоставленном с данной инструкцией. Выполнение инструкции приводит к переключению в режим ядра и запуску диспетчера системных сервисов. Номер системного сервиса передается в регистре процессора EAX, а регистр EDI указывает на список аргументов, предоставленных вызвавшим кодом. Для возврата в пользовательский режим диспетчер системных сервисов обычно выполняет инструкцию *sysexit*. (В некоторых случаях, например, когда в процессоре включен флаг *single-step*, диспетчер системных сервисов использует вместо *sysexit* инструкцию *iretd*.)



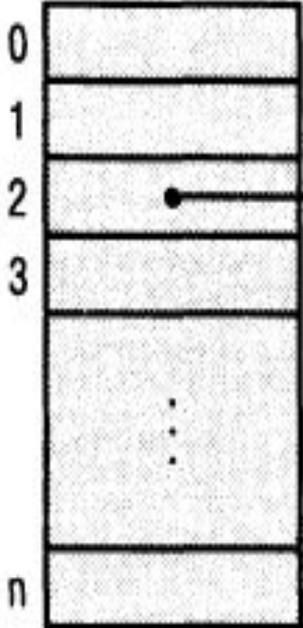
Пользовательский режим

Режим ядра

Вызов системного
сервиса



Таблица диспетчеризации
системных сервисов



Системный сервис 2



Инструкции для диспетчеризации сервисов исполнительной системы Windows содержатся в системной библиотеке Ntdll.dll. DLL-модули подсистем окружения вызывают функции из Ntdll.dll для реализации своих документированных функций. Исключением являются функции USER и GDI — здесь инструкции для диспетчеризации системных сервисов реализованы непосредственно в User32.dll и Gdi32.dll, а не в Ntdll.dll. Эти два случая иллюстрирует рис. 3-16.

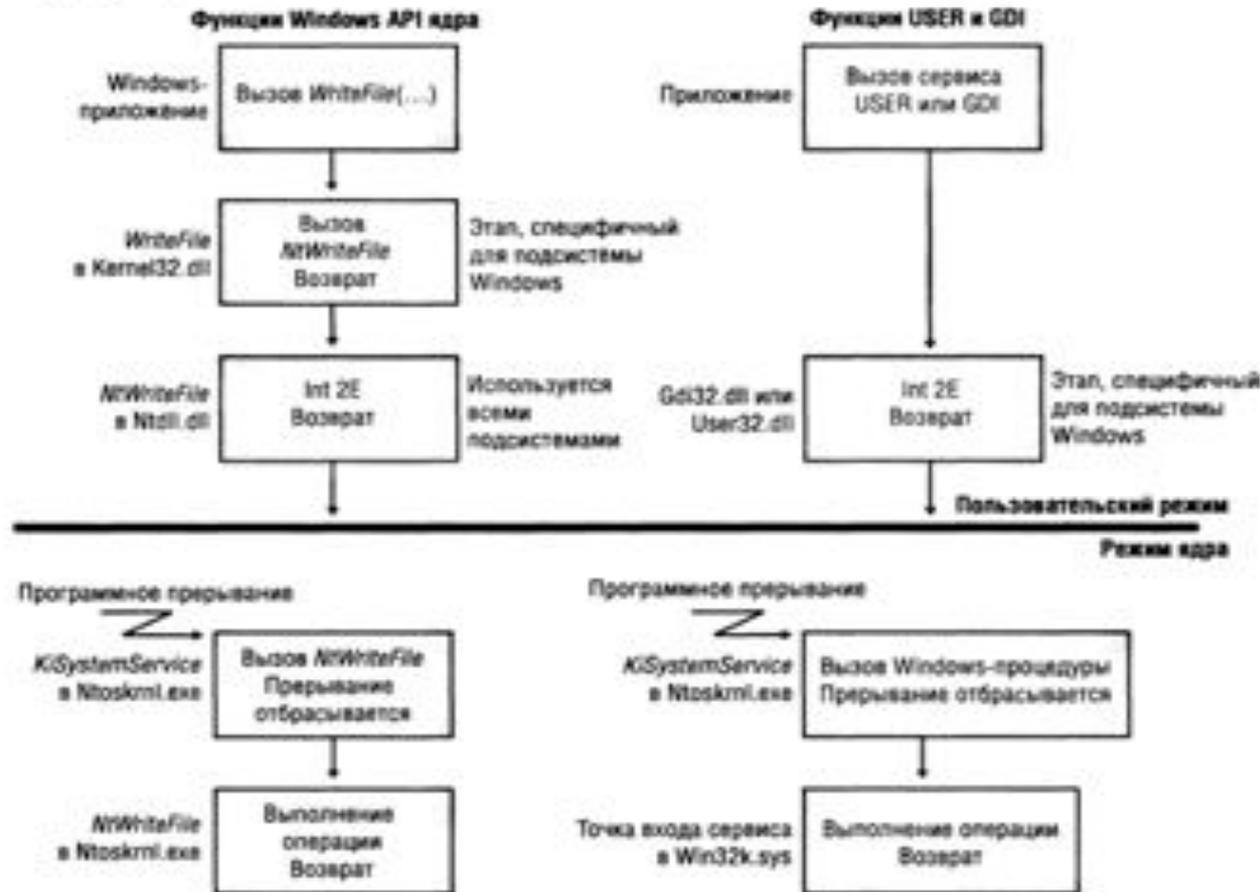


Рис. 3-16. Диспетчеризация системных сервисов

