Операционные системы семейства **Windows**

Литература

- **Руссинович М., Соломон Д.**

Внутреннее устройство Microsoft Windows. 6-е изд. — СПб.: Питер, 2013. — 800 с.

История развития

1. Появление фирмы MICROSOFT и интерпретатора языка BASIC (1981 г.) для микропроцессора Intel8088.
2. Первый ПК IBM PC и MS DOS 1.0 (1981 г.), PC AT и MS DOS 3.0 (1984 г.).
3. 3. Проект Lisa (графический интерфейс GUI Xerox, ПК Apple, С. Джобс, 1983 г.).
4. Оболочка MS DOS – Windows 1.0 (1985 г.).
5. Версия Windows 2.0 для PC AT (1987 г.).

Операционная система Windows 2000

6. Предшественником ОС Windows является одноименная операционная оболочка, появившаяся как надстройка над ОС DOS.

Версия Windows 3.0 для ПК с Intel 386 (1990 г.).
7. Версии Windows 3.1 и 3.11 для ПК с Intel 386, 486 (1992 - 1994 гг.).

Наиболее популярной оболочкой стала Windows 3.11 for Workgroup, где были реализованы многозадачность, графический интерфейс, поддержка одноранговой сети.

8. Windows 95 с большинством особенностей монолитной ОС на основе MS DOS 7.0, содержащая в значительной части 16-разрядный код (1995 Г.).

9. Windows 98 со значительным наследием MS DOS, содержащая частично 16-разрядный код, и ориентацией на работу в Интернет (1998 г.).

10. Windows ME, в основе повторяющая Windows 98, но с

Полноценная операционная система MS Windows появилась в 1995 г., как однопользовательская операционная система, поддерживающая вытесняющую многозадачность, работу в сети, использование длинных имен и ряд других новых и удобных функций.

8. Windows 95 с большинством особенностей монолитной ОС на основе MS DOS 7.0, содержащая в значительной части 16-разрядный код (1995 Г.).
9. Windows 98 со значительным наследием MS DOS, содержащая частично 16-разрядный код, и ориентацией на работу в Интернет (1998 г.).
10. Windows ME, в основе повторяющая Windows 98, но с возможностью восстановления настроек ПК при неверной установке параметров (2000 г.).

Другая линейка ОС корпорации Microsoft была связана с развитием операционной системы OS/2. Сетевая оболочка LAN Manager послужила основой для создания ОС Windows NT.

В Windows NT реализован ряд важных решений: возможность организации двухуровневой сети, использование данной ОС для организации файлового сервера, сервера приложений, поддержка различных сетевых протоколов и сервисов, поддержка более надежной файловой системы NTFS.

11. Полностью 32-разрядная операционная систем Windows NT 3.1 (1993 г.).

12. Windows XP (2001г.) — Windows NT 5.1
13. Windows XP 64-bit Edition (2006г.) — Windows NT 5.2
14. Windows Server 2003 (2003г.) — Windows NT 5.2
15. Windows Vista (2006г.) — Windows NT 6.0

Обновлена подсистема управления памятью и вводом-выводом

16. Windows Home Server (2007г.) — Windows NT 5.2
17. Windows Server 2008 (2008г.) — Windows NT 6.0
18. Windows Small Business Server (2008г.) — Windows NT 6.0
19. Windows 7 — Windows NT 6.1 (2009г.)

В Windows 7 была улучшена совместимость со старыми приложениями, некоторые из которых было невозможно запустить на Windows Vista

Максимальный размер оперативной памяти (для 64-битных версий) 16 Гб-192 Гб

20. Windows Server 2008 R2 — Windows NT 6.1 (2009г.)
21. Windows Home Server 2011 — Windows NT 6.1 (2011г.)
22. Windows 8 — Windows NT 6.2 (2012г.)

Архитектура IA-32 (32-bit) или x86-64 (64-bit)

20. Windows Server 2012 — Windows NT 6.2 (2012г.)

Windows 10 (29 июля 2015)

- Требования к персональному компьютеру:
- Процессор: 1 ГГц или более или система на кристалле.
- Оперативная память: 1 ГБ для 32-разрядной системы или 2 ГБ для 64-разрядной системы.
- Место на диске: 16 ГБ для 32-разрядной или 20 ГБ для 64-разрядной системы.
- Видеокарта с поддержкой Microsoft DirectX 9 и с драйвером WDDM.
- Дисплей с разрешением не менее 800 x 600 пикселей.
- Учётная запись пользователя Microsoft и подключение к Интернету[33] (при отсутствии Интернет-подключения будет предложено создать локальную учётную запись).Интернет-подключения будет предложено создать локальную учётную запись).

Семейство ОС для карманных компьютеров

- Это семейство операционных систем реального времени было специально разработано для мобильных устройств. Поддерживаются процессоры ARM, MIPS, SuperH и x86. В отличие от остальных операционных систем Windows, операционные системы этого семейства продаются только в составе готовых устройств, таких как смартфоны, карманные компьютеры, GPS-навигаторы, MP3-проигрыватели и другие.
- В настоящее время под термином «Windows CE» понимают только ядро операционной системы.

1. Windows CE (1996 г), была «урезанной» версией настольной операционной системы MS Windows 95

2. Windows Mobile мобильная операционная система, разработанная Microsoft для собственных аппаратных платформ Pocket PC (коммуникатор) и Smartphone. В настоящее время переживает постепенный отказ от поддержки и разработки
3. Windows Phone 7 (2010г.) Операционная система является преемником Windows Mobile, хотя и несовместима с ней, с полностью новым интерфейсом и с интеграцией сервисов Microsoft. Microsoft начала процесс создания единой экосистемы на базе ОС Windows в 2012 году. На Build 2014 Microsoft представила концепцию «универсальных» приложений Windows, при создании которых использован единый код и интерфейс для всех версий Windows.

Windows Embedded

- Windows Embedded — это семейство операционных систем реального времени, было специально разработано для применения в различных встраиваемых системах. Ядро системы имеет общее с семейством ОС Windows CE и поддерживает процессоры ARM, MIPS, SuperH и x86.
- Windows Embedded включает дополнительные функции по встраиванию, среди которых фильтр защиты от записи, загрузка с флеш-памяти, CD-ROM, сети, использование собственной оболочки системы и т. п.
- В отличие от операционных систем Windows, операционные системы этого семейства продаются только в составе готовых устройств, таких как: банкоматы, медицинские приборы, навигационное оборудование, «тонкие» клиенты, медиапроигрыватели, цифровые рамки (альбомы), кассовые терминалы, платёжные терминалы, роботы, игровые автоматы, музыкальные автоматы и другие.

Особенности Windows 2000

- ОС Windows 2000 поддерживает службу каталогов Active Directory и на ее основе службу безопасности Public Key Infrastructure (PKI) и протокол Kerberos, терминальные службы, службы IIS.
- Система поддерживает до 4 Гб оперативной памяти и многопроцессорную симметричную обработку (SMP) – Windows 2000 Prof (до 2 процессоров), Windows 2000 Server SE (до 4 процессоров), Windows 2000 Sever AE (до 8 процессоров).

- Windows 2000 рассчитана на рабочие станции и серверы;
- Отказоустойчива;
- Защищенная ОС;
- Содержит богатый набор утилит для администрирования локального компьютера и сети;
- Ядро ОС написано на C и C++, что обеспечивает переносимость ОС;
- Поддержка Unicode, что обеспечивает поддержку различных языков;
- Высокоэффективная подсистему управления памятью;
- Поддержка структурной обработки исключений (SEH), что облегчает восстановление после сбоев;
- Поддержка динамически подключаемых библиотек (DLL);
- Поддержка многопоточной и многопроцессорной обработки;
- Поддержка файловых систем NTFS, FAT, FAT32.

Особенности Windows 7

Максимальные аппаратные требования для Windows 7

Edition	Processor architecture		Физических процессоров	Ядер	
	x86 (32-bit)	x64 (64-bit)		x86 (32-bit)	x64 (64-bit)
Ultimate	4 GB	192 GB	2 CPU sockets	32 ядра	256 ядер
Enterprise					
Professional					
Home Premium		16 GB	1 CPU socket		
Home Basic		8 GB			
Starter	2 GB	N/A			N/A

Структура операционной системы Windows

ОС Windows можно разделить на 2 части:

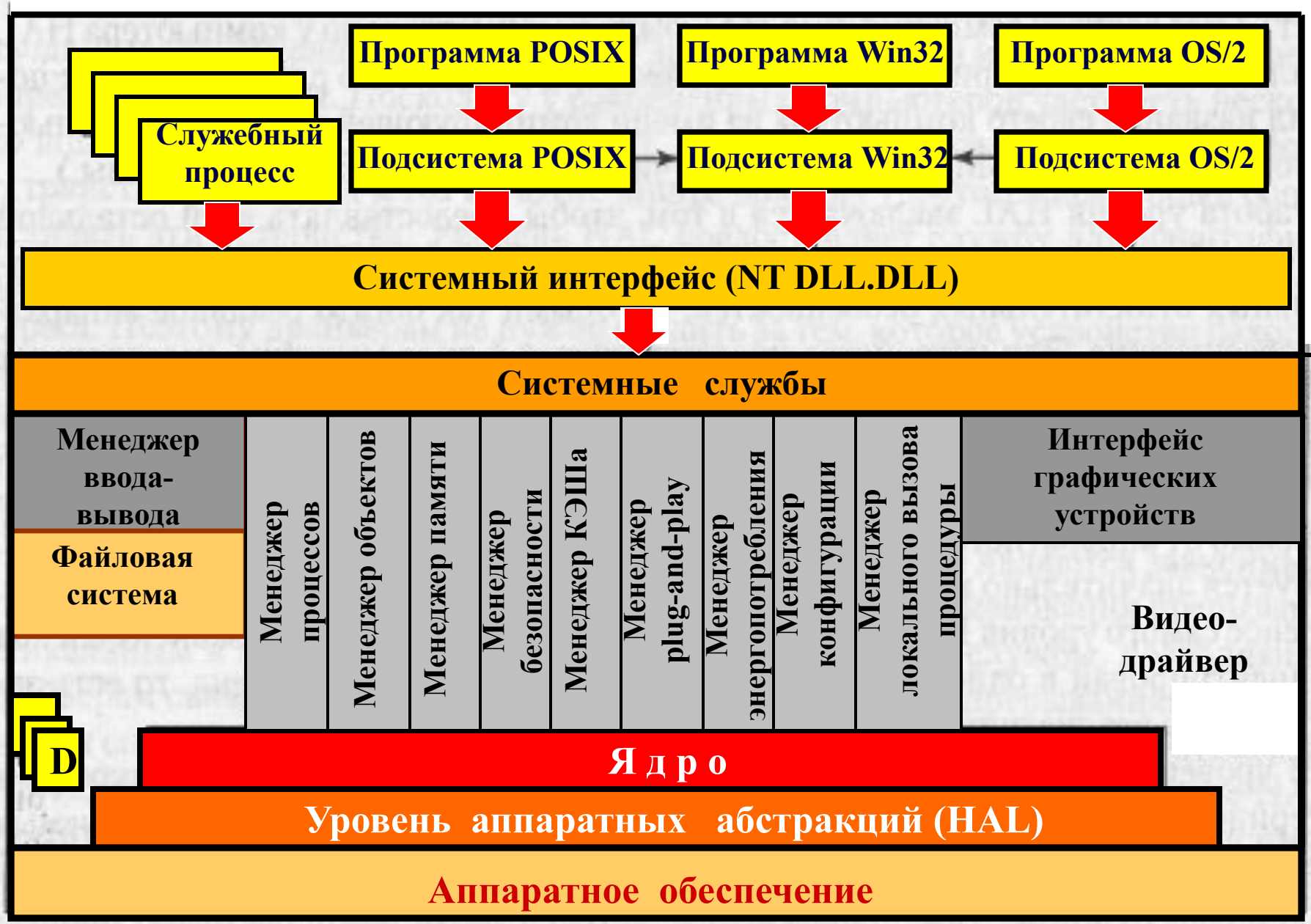
1. Основная часть ОС, работающая в режиме ядра (управление процессами, памятью, файловой системой , устройствами и т. д.).

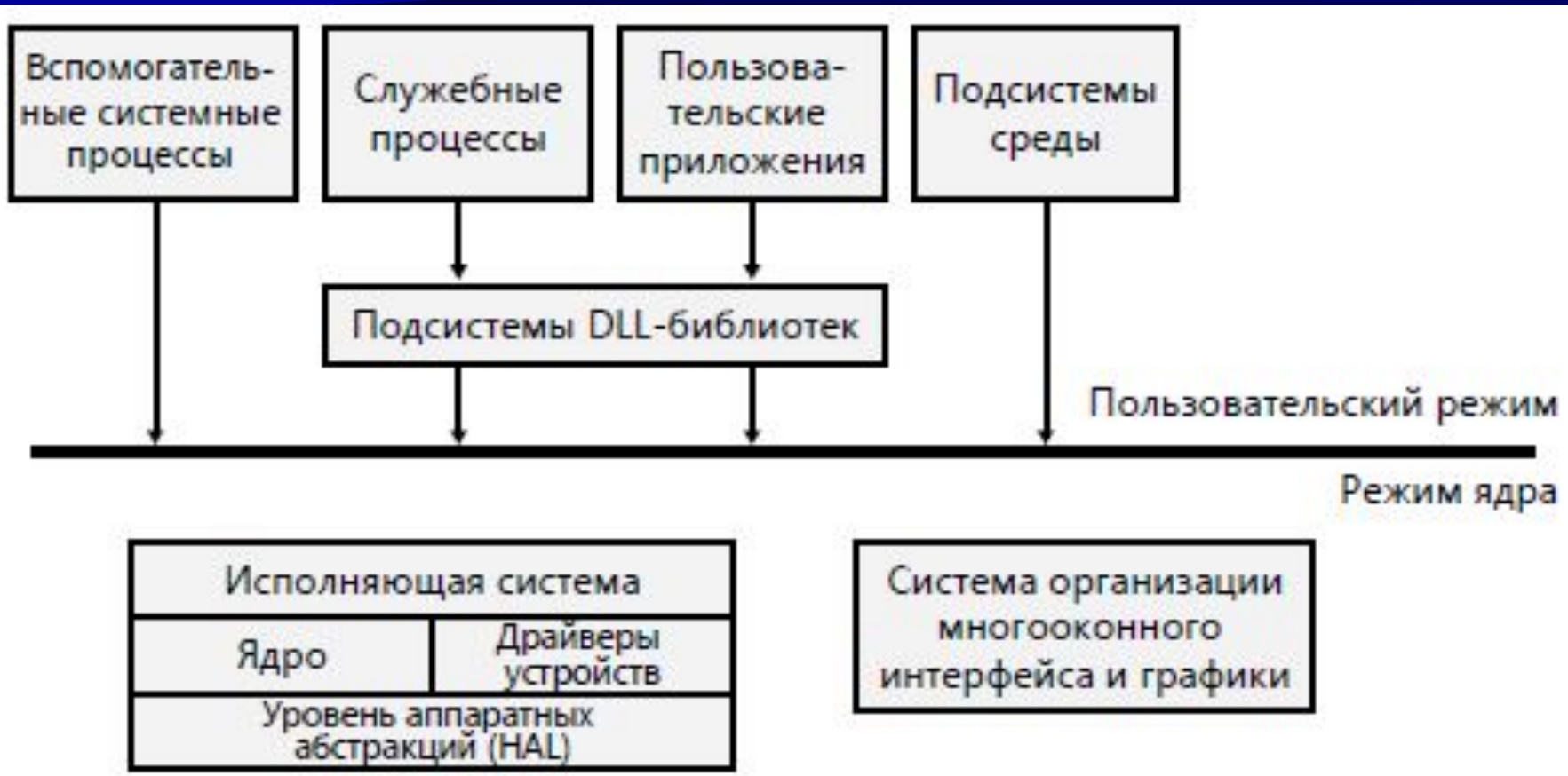
2. Подсистемы окружения (среды), работающие в режиме пользователя (процессы, помогающие пользователям выполнять определенные системные функции).

Основная часть разделена на несколько уровней, каждый из которых пользуется службами лежащего ниже уровня. Основными уровнями являются:

- **системные службы (сервисные процессы, являющиеся системными демонами);**
- **исполняющая система (супервизор или диспетчер);**
- **драйверы устройств;**
- **ядро операционной системы;**
- **уровень аппаратных абстракций (HAL).**

Два нижних уровня написаны на языке С и ассемблере и являются частично машинно-зависимыми. Верхние уровни написаны исключительно на языке С и почти полностью машинно-независимы. Драйверы написаны на С и в некоторых случаях на С++.

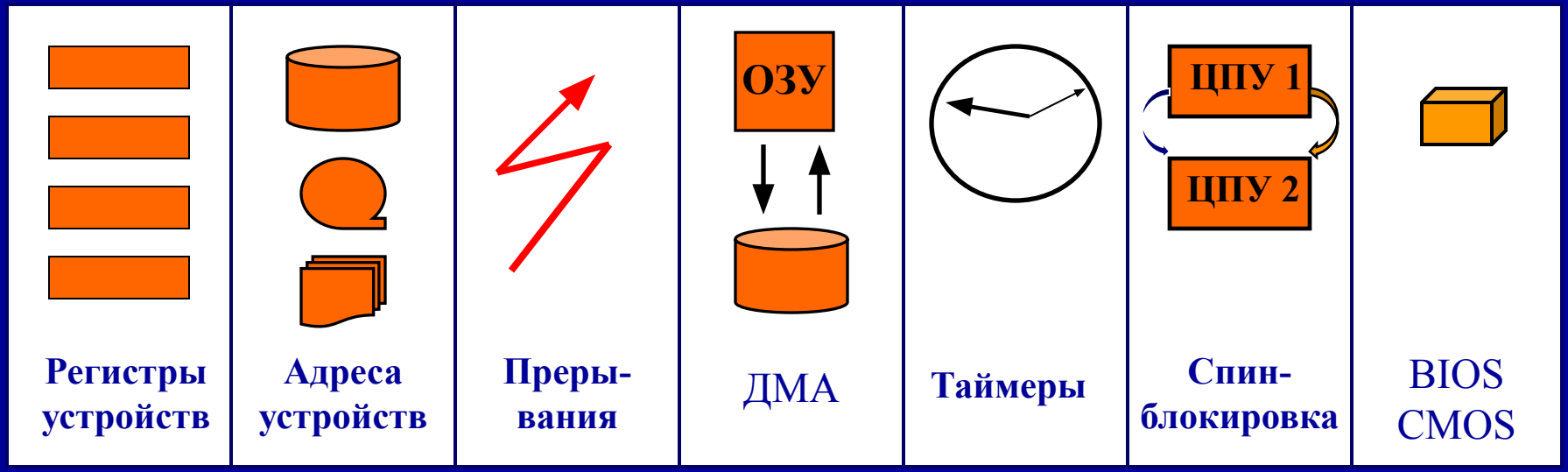

Программа POSIX
Программа Win32
Программа OS/2
Служебный процесс
Подсистема POSIX
Подсистема Win32
Подсистема OS/2
Системный интерфейс (NT DLL.DLL)
Режим пользователя
Системные службы
Менеджер ввода-вывода
Файловая система
Менеджер процессов
Менеджер объектов
Менеджер памяти
Менеджер безопасности
Менеджер КЭШа
Менеджер plug-and-play
Менеджер энергопотребления
Менеджер конфигурации
Менеджер локального вызова процедуры
Интерфейс графических устройств
Видео-драйвер
D
Я д р о
Уровень аппаратных абстракций (HAL)
Аппаратное обеспечение
Режим ядра


Вспомогатель-
ные системные
процессы
Служебные
процессы
Пользова-
тельские
приложения
Подсистемы
среды
Подсистемы DLL-библиотек
Пользовательский режим
Режим ядра
Исполняющая система
Ядро
Драйверы
устройств
Уровень аппаратных
абстракций (HAL)
Система организации
многооконного
интерфейса и графики

Уровень аппаратных абстракций (Hardware Abstraction Layer – HAL)

Работа уровня HAL заключается в том, чтобы предоставить всей остальной системе абстрактные аппаратные устройства, свободные от индивидуальных особенностей аппаратуры. Эти устройства представляются в виде машинно-независимых служб (процедурных вызовов и макросов), которые могут использоваться остальной ОС и драйверами.

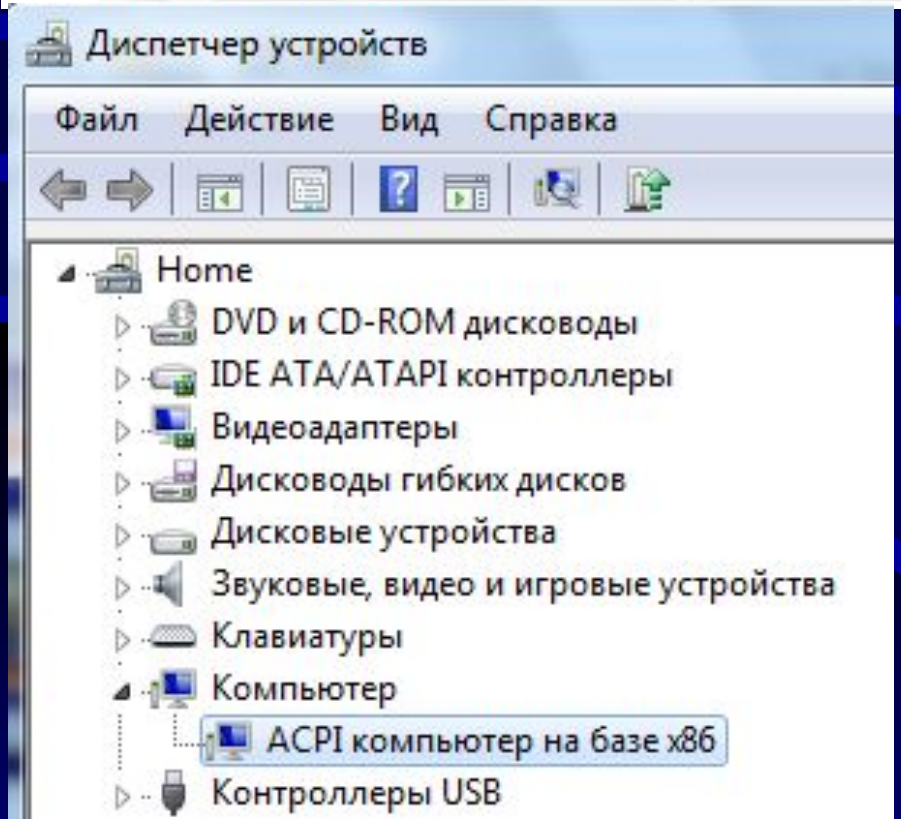
Некоторые функции уровня HAL



В уровень HAL включены те службы, которые зависят от набора микросхем материнской платы и меняются от машины к машине в разумных пределах:

- набор низкоуровневых функций (около 100), обеспечивающий стандартный интерфейс взаимодействия с аппаратнозависимыми элементами для функций , вызываемых компонентами ядра , драйверов и исполнительной системы , позволяющий абстрагироваться от того , на какой конкретно элементной базе (чипе контроллера прерывания, контроллера ПДП) реализовано выполнение доступа к шине, таймеру и т .д .
- Хотя в операционную систему включено несколько HAL-модулей , у Windows есть возможность определить во время загрузки, какой HAL-модуль нужно использоваться.

Имя файла HAL	Поддерживаемые системы
Hal.dll	Стандартные персональные компьютеры (ПК)
Halacpi.dll	ПК с ACPI (Advanced Configuration and Power Interface)
Halapic.dll	ПК с APIC (Advanced Programmable Interrupt Controller)
Halaacpi.dll	ПК с APIC и ACPI
Halmps.dll	Многопроцессорные ПК
Halmacpi.dll	Многопроцессорные ПК с ACPI
Halborg.dll	Рабочие станции Silicon Graphics (только для Windows 2000; больше не продаются)
Halsp.dll	Compaq SystemPro (только для Windows XP)



- **Halacpi.dll -** Персональные компьютеры с усовершенствованным интерфейсом управления конфигурированием и энергопотреблением — Advanced configuration and Power Interface (ACPI).
- **Halmacpi.dll -** Персональные компьютеры с усовершенствованным программируемым контроллером прерываний — Advanced Programmable Interrupt Controller (APIC).
- На x64-машинах имеется только один HAL-образ по имени Hal.dll. Это обусловлено наличием у всех x64-машин материнских плат одинаковой конфигурации, поскольку процессы требуют поддержки ACPI и APIC.

Уровень ядра

Назначение ядра – сделать остальную часть ОС независимой от аппаратуры. Для этого ядро на основе низкоуровневых служб HAL формирует абстракции более высоких уровней. Например, у уровня HAL есть вызовы для связывания процедур обработки прерываний с прерываниями и установки их приоритетов. Больше ничего в этом отношении HAL не делает. Ядро же предоставляет полный механизм для переключения контекста и планирования потоков.

Ядро также предоставляет низкоуровневую поддержку двум классам объектов ядра – управляющим объектам и объектам диспетчеризации. Эти объекты используются системой и приложениями для управления ресурсами компьютерной системы: процессами, потоками , файлами и т. д.

Каждый объект ядра – это блок памяти, выделенный ядром, доступный только ему и представляющий собой структуру данных, в которой содержится информация об объекте.

К управляющим объектам ядра относятся: объекты заданий, процессов, потоков, прерываний, DPC (Deferred Procedure Call – отложенный вызов процедуры), APC (Asynchronous Procedure Call – асинхронный вызов процедуры)

К объектам диспетчеризации ядра относятся объекты, изменение состояния которых могут ждать потоки. Это –семафоры, мьютексы, события, таймеры, очереди, файлы, порты, маркеры доступа и др.

Исполняющая система

Написана на языке С, не зависит от архитектуры машины и относительно просто может быть перенесена на новые машины. Исполняющая система состоит из 10 компонентов, между которыми нет жестких границ. Компоненты одного уровня могут вызывать друг друга. Большинство компонентов представляют собой процедуры, которые выполняются потоками системы в режиме ядра.

Менеджер объектов управляет всеми объектами, создаваемыми или известными операционной системе (процессами, потоками, семафорами, файлами и т. д.). Он управляет пространством имен, в которое помещается созданный объект, чтобы к нему можно было обратиться по имени. Остальные компоненты ОС пользуются объектами во время своей работы.

Менеджер ввода-вывода предоставляет остальной части ОС независимый от устройств ввод-вывод, вызывая для выполнения физического ввода-вывода соответствующий драйвер.

Менеджер процессов управляет процессами и потоками, включая их создание и завершение. Он основывается на объектах **потоков** и процессов ядра и добавляет к ним дополнительные функции. Это ключевой элемент многозадачности.

Менеджер памяти реализует архитектуру виртуальной памяти со страничной подкачкой по требованию ОС. Он управляет преобразованием виртуальных страниц в физические и реализует правила защиты, ограничивающие доступ каждому процессу только теми страницами, которые принадлежат его адресному пространству.

Менеджер безопасности приводит в исполнение сложный механизм безопасности Windows, удовлетворяющий требованиям класса C2 Оранжевой книги Министерства обороны США.

Менеджер кэша хранит в памяти блоки диска, которые использовались последнее время, чтобы ускорить доступ к ним и обслуживает все файловые системы. Взаимодействует с менеджером виртуальной памяти, чтобы обеспечить требуемую непротиворечивость.

Менеджер plug-and-play управляет установкой новых устройств, контролирует их динамическое подключение и осуществляет при необходимости загрузку драйверов.

Менеджер энергопотребления управляет потреблением энергии, следит за состоянием батарей, взаимодействует с ИБП.

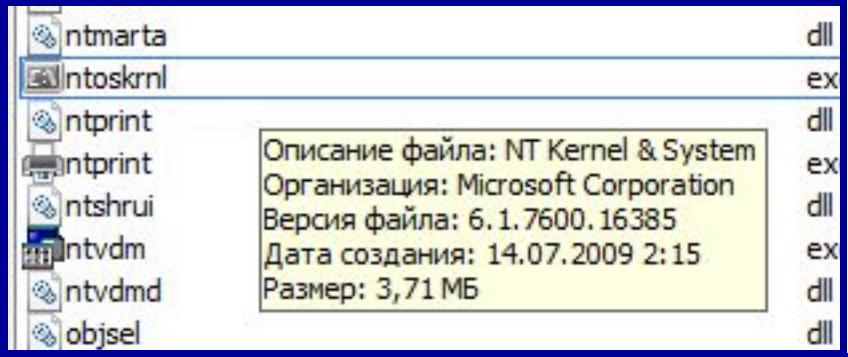
Менеджер конфигурации отвечает за сохранение реестра.

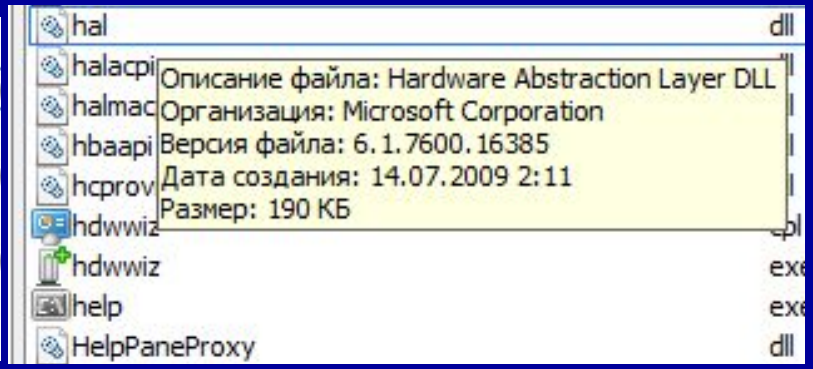
Менеджер вызова локальной процедуры обеспечивает высокоэффективное взаимодействие между процессами и их подсистемами. Поскольку этот путь нужен для выполнения некоторых системных вызовов, эффективность оказывается критичной, поэтому здесь не используются стандартные механизмы межпроцессного взаимодействия.

Интерфейс графических устройств GDI (Graphic Device Interface) управляет графическими изображениями для монитора и принтеров, предоставляя системные вызовы для пользовательских программ. (Интерфейс Win32 и модуль GDI превосходят по объему всю остальную исполняющую систему.)

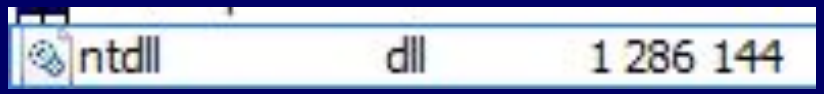
Системные службы предоставляют интерфейс к исполняющей системе. Они принимают системные вызовы Windows и вызывают необходимые части исполняющей системы для их выполнения.

Драйверы устройств устанавливаются в систему, добавляются в реестр и затем динамически загружаются при каждой загрузке системы. Драйверы не являются частью файла **ntoskrnl.exe.**





Основная часть ОС, состоящая из ядра и исполняющей системы, хранится в файле **ntoskrnl.exe .** Уровень **HAL**, представляющий собой библиотеку общего доступа, находится в файле **hal.dll.** Интерфейс Win32 и графических устройств хранятся в файле **win32.sys**. Системный интерфейс для связи пользовательского режима с режимом ядра – файл **NTDLL.DLL**



Системные процессы и службы предоставляют интерфейс к исполняющей системе. Они принимают системные вызовы Windows и вызывают необходимые части исполняющей системы для их выполнения.

В каждой системе Windows выполняются перечисленные ниже процессы. (Два из них, Idle и System, не являются процессами в строгом смысле этого слова, поскольку они не выполняют какой-либо код пользовательского режима.)

- Процесс Idle (включает по одному потоку на процессор для учета времени простоя процессора).
- Процесс System (содержит большинство системных потоков режима ядра).
- Диспетчер сеансов (Smss.exe).
- Подсистема Windows (Csrss.exe).
- Процесс входа в систему (Winlogon.exe).
- Диспетчер управления сервисами (Services.exe) и создаваемые им дочерние процессы сервисов (например, универсальный процесс для хостинга сервисов, Svchost.exe).
- Серверный процесс локальной аутентификации (Lsass.exe).

Подсистемы окружения

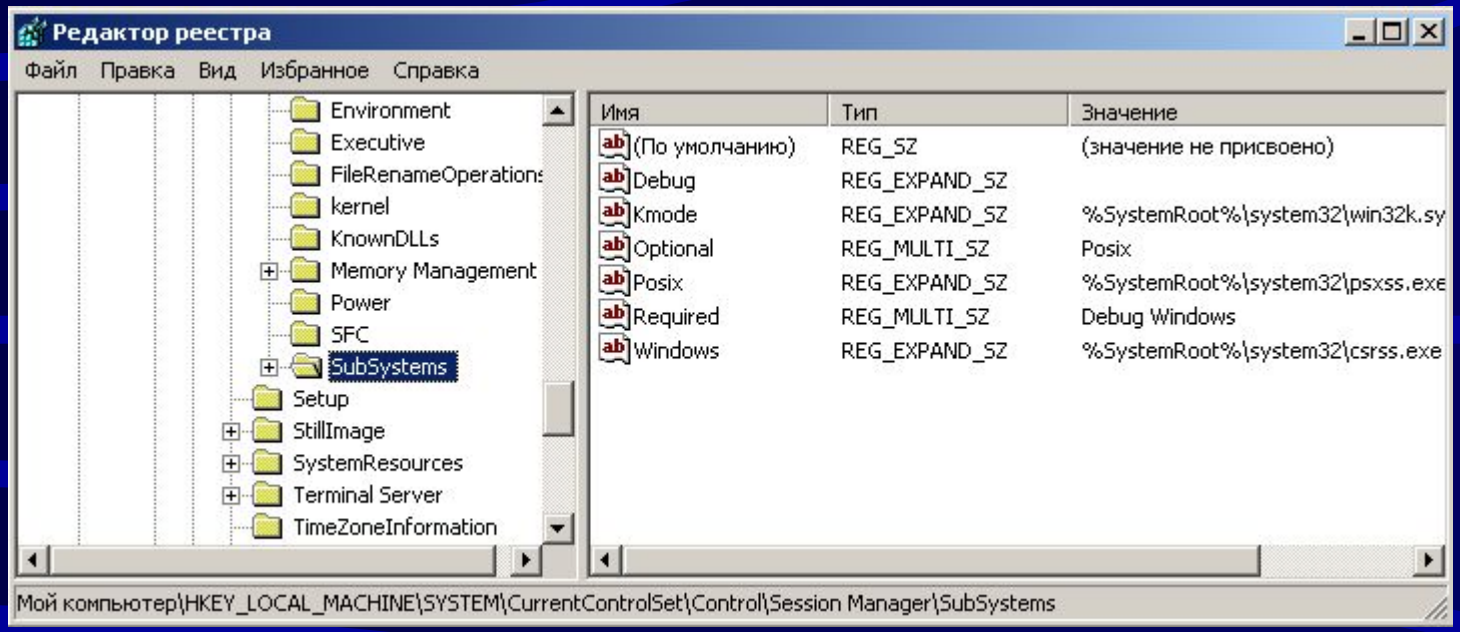
В Windows существует три типа компонентов, работающих в режиме пользователя: динамические библиотеки DLL, подсистемы окружения и служебные процессы. Эти компоненты работают вместе, предоставляя пользовательским процессам три различных документированных интерфейса прикладного программирования API (Win32, POSIX, OS/2).

У каждого интерфейса есть список библиотечных вызовов, которые используются программистами. Работа библиотек DLL и подсистем окружения заключается в том, чтобы реализовать функциональные возможности опубликованного интерфейса, скрывая истинный интерфейс системных вызовов от прикладных программ.

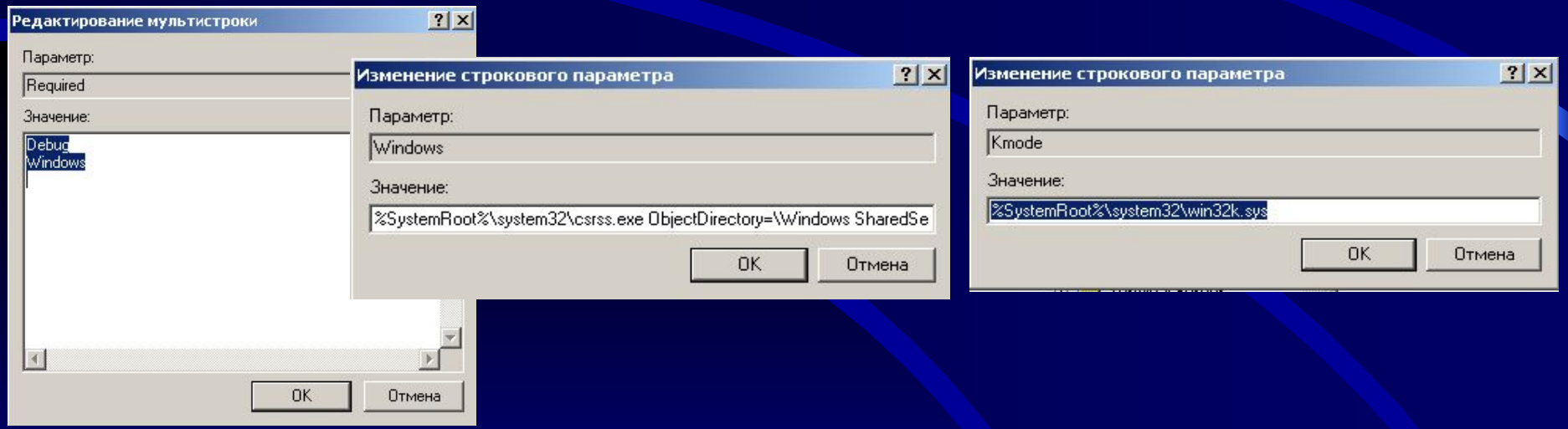
Программы, пользующиеся интерфейсом Win32, содержит, как правило, большое количество обращений к функциям Win32 API. Поэтому, если работает одновременно несколько программ, то в памяти будут находиться многочисленные копии одних и тех же библиотечных процедур. Чтобы избежать подобной проблемы Windows поддерживает динамически присоединяемые библиотеки DLL. При этом при запуске нескольких приложений, использующих одну и ту же DLL, в памяти будет находиться только одна копия текста DLL.

В каталоге \winnt\system32 есть более 2000 отдельных файлов DLL

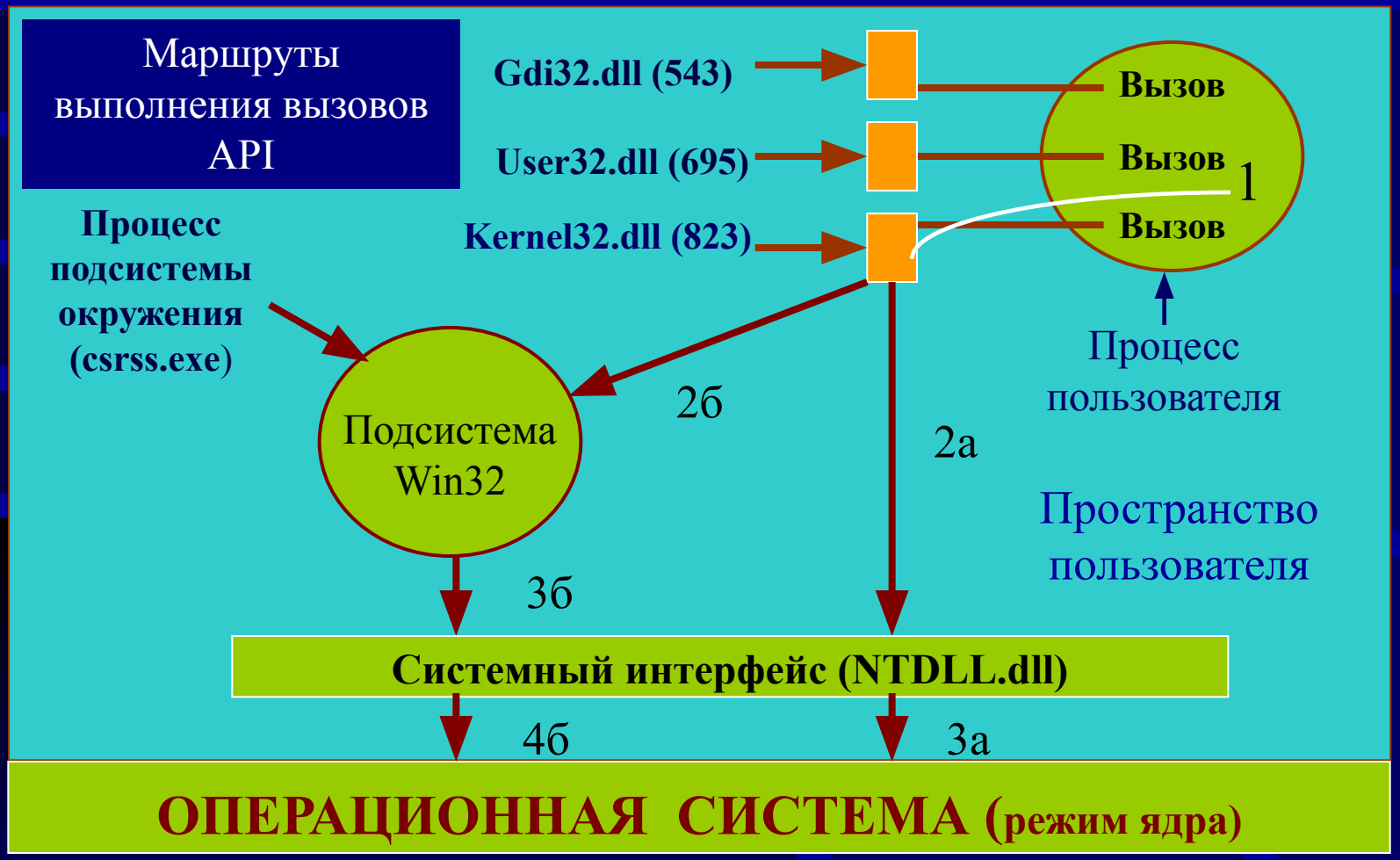
- Изначально Windows NT поставляется тремя подсистемами среды: Windows, POSIX и OS/2. Но подсистемы POSIX и OS/2 последний раз поставлялись с Windows 2000. Выпуски клиентской версии Windows Ultimate и Enterprise, а также все серверные версии включают поддержку для усовершенствованной подсистемы POSIX, которая называется подсистемой для приложений на основе Unix (Unix-based Applications, SUA).

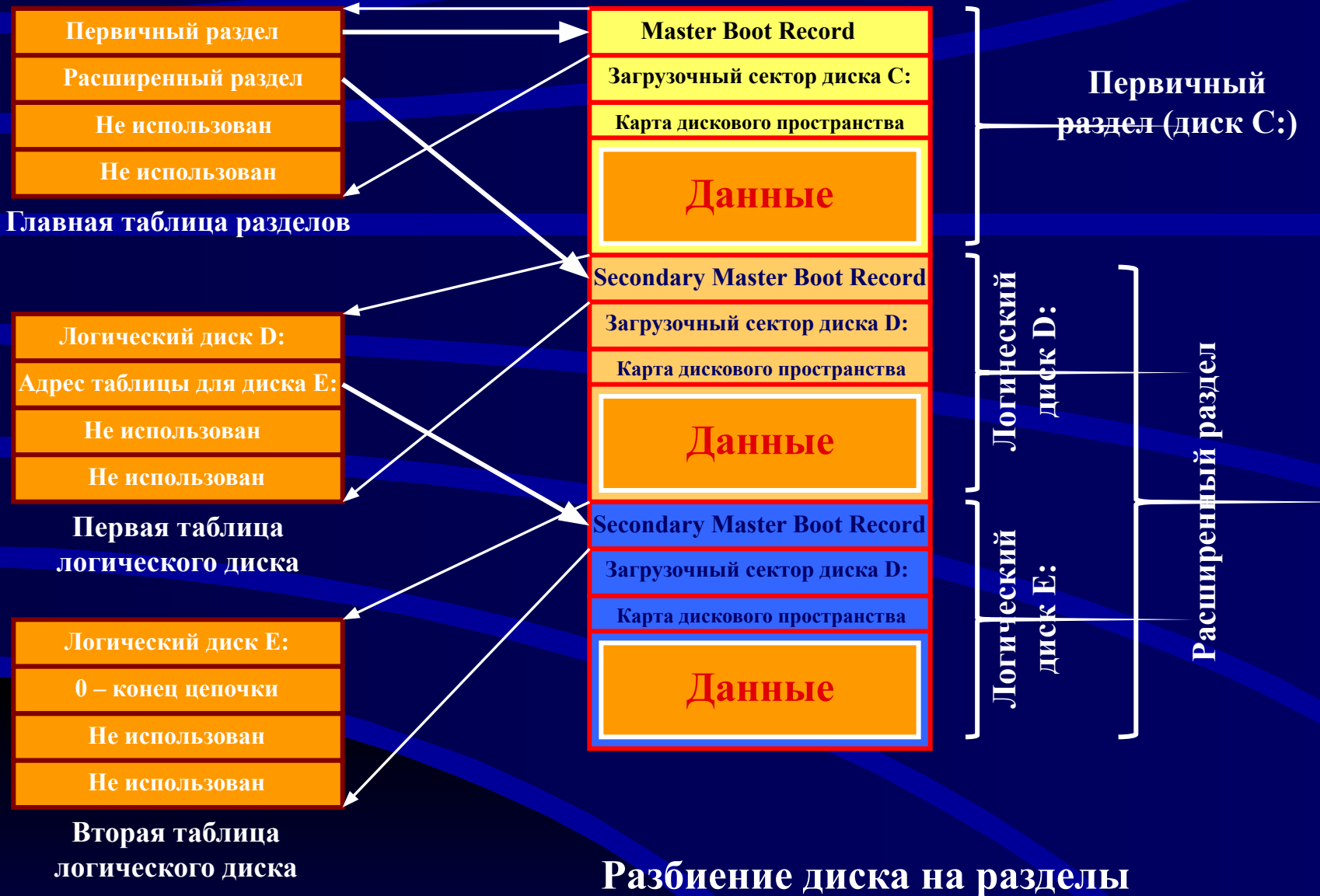


Подсистема Windows работает всегда (обеспечивает клавиатуру, мышь, экран), остальные подсистемы запускаются опционально (Optional). Параметр Required определяет список подсистем, загружаемых при запуске ОС. В параметре Windows указывается спецификация файла Csrss. Параметр Kmode содержит имя файла подсистемы Windows, работающего в режиме ядра.



Пользовательские приложения вызывают системные сервисы через DLL подсистем.
1. Функция реализована в пользовательском режиме внутри DLL подсистемы.
2. DLL – библиотека обращается к другой DLL (NTDLL.dll), которая обращается к ядру ОС (2а-3а).
3. Для выполнения функции происходит обращение к подсистеме Win32, которая выполняет всю работу самостоятельно или обращается к системному вызову (2б-3б-4б).



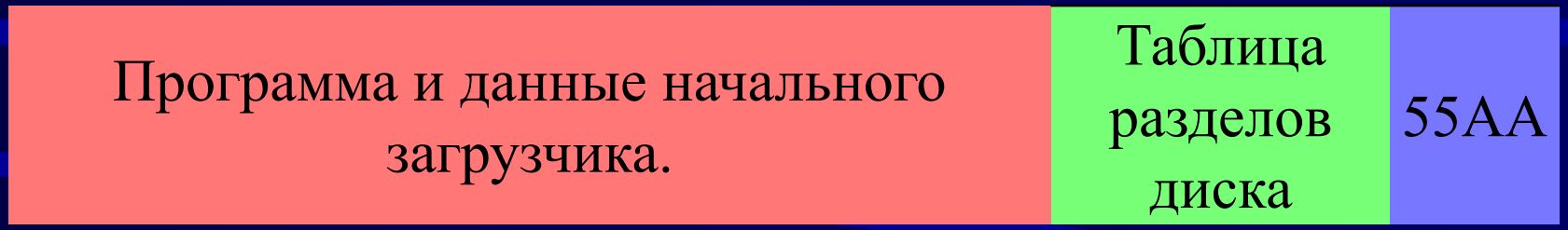


Разбиение диска на разделы

Механизм загрузки операционной системы Windows 7

Процесс загрузки любой операционной системы начинается всегда одинаково - после проверки оборудования, управление получает подпрограмма BIOS, (Basic Input/Output System), считывающая с устройства загрузки первый сектор, являющийся главной загрузочной записью **MBR** (**M**aster **B**oot **R**ecord). Стандартно MBR располагается в первом секторе загрузочного диска и занимает 512 байт (стандартная длина сектора). Структура MBR включает в себя 2 основных элемента - программный код первичного загрузчика и таблицу разделов. Обязательным признаком наличия записи MBR является специальный код (сигнатура) в двух последних байтах - **55AA**.

Структура главной загрузочной записи MBR:
- программный код и данные начального загрузчика. (446 байт.)
- таблица разделов диска (4 поля по 16 байт - 64 байта)
- сигнатура 55AA (2 байта)



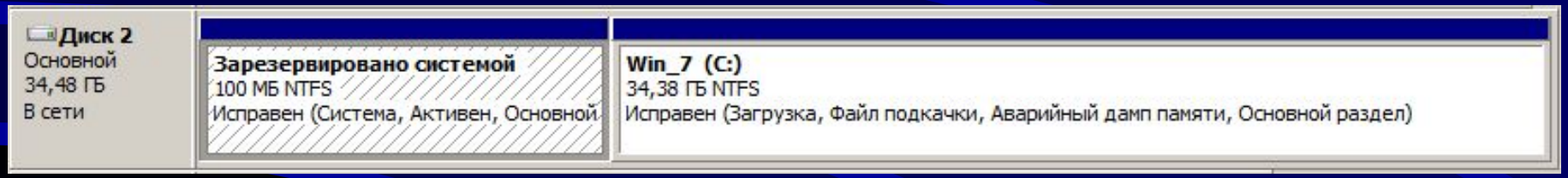
После считывания в оперативную память, программный код начального загрузчика выполняет поиск активного раздела (Active), с которого может выполняться загрузка конкретной операционной системы. Такой раздел имеет свою загрузочную запись, называемую загрузочной записью раздела **PBR** (**P**artition **B**oot **R**ecord) .

Этап OSLoader

В случае с загрузкой Windows 7 (и последующих ОС семейства Windows) программный код загрузчика раздела выполняет считывание в оперативную память и передачу управления специальной программе - диспетчеру загрузки **BOOTMGR**
Диспетчер загрузки **bootmgr** – файл в корневом каталоге активного раздела. Основное его предназначение - обеспечение дальнейшей процедуры загрузки в соответствии с существующей **конфигурацией**, хранящейся в специальном **хранилище** - хранилище данных конфигурации (**BCD - B**oot **C**onfiguratin **D**ata), представляющем собой файл с именем **BCD**, находящийся в каталоге **BOOT** активного раздела.
Диспетчер загрузки может выполнить не только загрузку ядра установленной на данном компьютере Windows, но и другие, имеющиеся в конфигурации варианты - загрузку Windows NT/2000/XP, операционных систем семейства Linux, загрузку ОС из образов (файлов **wim**) , виртуальных дисков (**файлов VHD**)

При стандартной установке операционной системы Windows 7 на новый жесткий диск, в качестве активного раздела используется, автоматически создаваемый при инсталляции в первой части диска, раздел небольшого размера (около 100Мб). Данному разделу не присваивается буква, и в проводнике он не отображается. Это сделано с целью защиты загрузчика от небезопасных для него действий пользователя

При просмотре в Диспетчере логических дисков, активный раздел отображается под названием "Зарезервировано системой" :



Диспетчер **bootmgr** считывает из хранилища конфигурации данные, необходимые для загрузки ядра системы. Активируются модуль Winload.exe, системные службы, компоненты ядра Hal.dll и Ntoskrnl.exe и ряд других компонентов – данный этап сопровождается анимированным логотипом.

Этап OSLoader можно представить в виде цепочки из последовательно выполняемых этапов:

Программа из записи MBR	Программа из записи PBR	BOOTMGR+запись конфигурации Win7	Загрузчик ядра WINLOAD.EXE

Этап MainPathBoot

Фаза PreSMSS

Во время этой фазы полностью инициализируется ядро Windows 7, запускается диспетчер оборудования plug and play, инициализируются ранее запущенные драйвера BOOT_START и драйвера оборудования.

Фаза SMSSInit

Фаза начинается с момента передачи управления диспетчеру сеансов — SMSS.exe. В это время инициализируются остальные кусты реестра, загружаются драйвера с параметром запуска «авто». В конце фазы управление переходит к файлу Winlogon.exe — программе пользовательского входа в Windows. Визуально об окончании SMSSInit свидетельствует появление на экране приглашения входа.

.

Фаза WinLogonInit

Эта фаза начинается со старта Winlogon.exe (экрана приветствия) и заканчивается загрузкой рабочего стола — началом работы оболочки Windows — файла Explorer.exe. Во время ее хода система считывает и выполняет сценарии групповых политик и запускает службы (системные и сторонние).

Фаза ExplorerInit

Начинается стартом оболочки и заканчивается запуском процесса диспетчера окон рабочего стола. Во время хода этой фазы на экране появляются значки рабочего стола.
Одновременно с этим происходит дальнейший запуск служб, старт автозагружаемых приложений, кэширование данных и т. п.

Этап PostBoot

Начинается появлением рабочего стола и заканчивается после загрузки всего, что прописано в автозапуск. В этот период начинает работать основная часть приложений, запускаемых вместе с Windows. После окончания этапа система переходит в бездействие.

Загрузка Windows

Самотестирование при включении (Power-On Self-Test. POST)

Системная BIOS ищет загрузочный диск (в порядке, заданном пользователем)

Считывание главной загрузочной записи MBR загружаемого диска

Анализ таблицы разделов и определение раздела, в котором находится загружаемая ОС

Передача управления загрузочному сектору 0 раздела, в котором находится ОС

Программа загрузочного сектора находит в корневом каталоге раздела файл ntldr

Программа ntldr считывает файл Boot.ini, в котором хранятся списки файлов hal.dll, ntoskernl.exe (в Boot.ini указано количество ЦПУ, ОП, F часов реального времени). Загружает и выполняет программу ntdetect.com.

Загружаются файлы hal.dll, ntoskernl.exe и bootvid.dll. Программа ntldr считывает реестр, чтобы найти драйверы, необходимые для завершения загрузки (для микросхем мат. платы, клавиатуры, мыши и т. д.).

Загрузчик считывает драйверы и передает управление программе ntoskernl.exe


Общие процедуры инициализации и инициализация компонентов исполняющей системы. Загрузка и инициализация драйверов устройств и сервисов, создание файлов подкачки
Создание сеансового менеджера smss.exe
Запуск подсистемы окружения Win32 (csrss.exe), считывание реестра и выполнение указанных команд, создание файлов подкачки и открытие нужных DLL
Создание демона регистрации winlogon.exe
Демон регистрации
Менеджер аутентификации
Создание менеджера аутентификации lsass.exe
Запуск родительского процесса всех служебных процессов services.exe. По информации, хранящейся в реестре определяет, какие демоны в пространстве пользователя надо запустить (сервер принтера, файловый сервер, обработчик входящей электронной почты, факсов и т. д.).
Выбор из реестра профиля пользователя и запуск требуемой оболочки.

Объекты Windows

- В Windows любой ресурс системы, который одновременно может быть использован более чем одним процессом, включая файлы, совместно используемую память и физические устройства, реализован в виде объекта и управляется рядом функций.

Менеджер объектов выполняет следующие функции:

- Выделяет память для объекта.
- Присоединяет к объекту так называемый дескриптор безопасности, который определяет, кому разрешено использовать объект, и что они могут с ним делать.
- Создает и манипулирует структурой каталога объектов, в котором хранятся имена объектов.
- Создает описатель объекта и возвращает его вызывающему процессу.

- Каждый объект состоит из двух частей - заголовка объекта и тела объекта.
- Менеджер объектов работает с заголовком объекта, а другие компоненты executive работают с телами объектов тех типов, которые они сами создают.
- Заголовок объекта используется менеджером без учета типа объекта. В заголовке объекта любого типа содержится имя, каталог, дескриптор безопасности, квоты на использование ресурсов, счетчик открытых описателей, база данных открытых описателей, признак постоянный/временный, режим пользователя/ядра, указатель на тип объекта.

- Объект-процесс, как и другие объекты, содержит заголовок, который создает и инициализирует менеджер объектов.
- Кроме заголовка объекта, каждый объект имеет тело объекта, формат и содержание которого уникально определяется типом этого объекта; у всех объектов одного и того же типа одинаковый формат тела. При создании объекта исполнительная часть может оперировать данными в телах всех объектов этого типа.

В число атрибутов тела объекта-процесса входят:

1. Идентификатор процесса - уникальное значение, которое идентифицирует процесс в рамках операционной системы.
2. Токен доступа - исполняемый объект, содержащий информацию о безопасности.
3. Базовый приоритет - основа для исполнительного приоритета потоков процесса.
4. Процессорная совместимость - набор процессоров, на которых могут выполняться потоки процесса.
5. Предельные значения квот - максимальное количество страничной и нестраничной системной памяти, дискового пространства, предназначенного для выгрузки страниц, процессорного времени - которые могут быть использованы процессами пользователя.
6. Время исполнения - общее количество времени, в течение которого выполняются все потоки процесса.

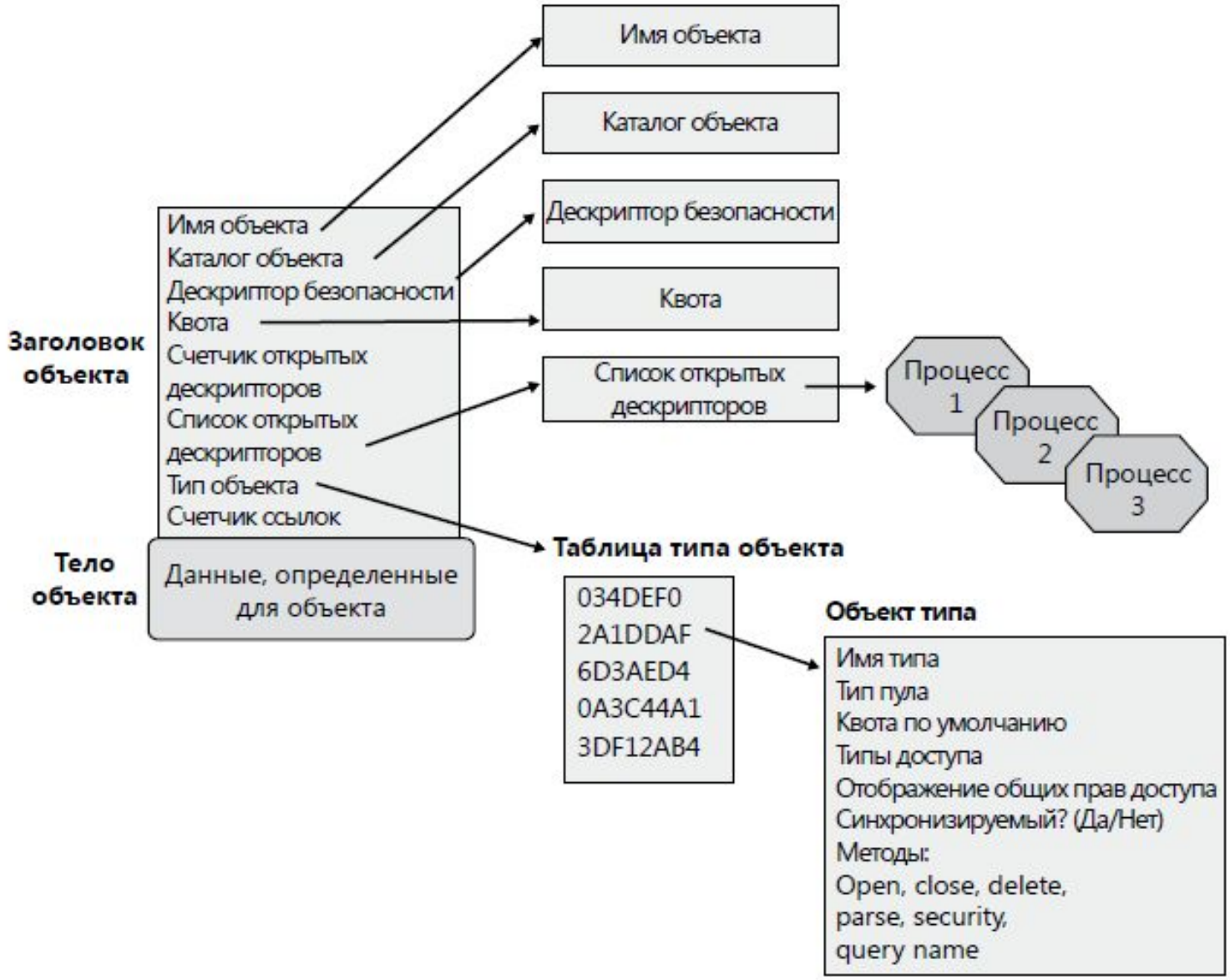
Объект- поток имеет следующие атрибуты тела:

1. Идентификатор клиента - уникальное значение, которое идентифицирует поток при его обращении к серверу.
2. Контекст потока- информация, которая необходима ОС для того, чтобы продолжить выполнение прерванного потока. Контекст содержит текущее состояние регистров, стеков и индивидуальной области памяти, которая используется подсистемами и библиотеками.
3. Текущий приоритет - значение приоритета потока в данный момент.
4. Базовый приоритет - основа текущего приоритета потока.
5. Процессорная совместимость - перечень типов процессоров, на которых может выполняться поток.
6. Время выполнения потока - суммарное время выполнения в пользовательском режиме и в режиме ядра, накопленное за период существования.
7. Состояние предупреждения - флаг, который показывает, что поток должен выполнять вызов асинхронной процедуры.
8. Счетчик приостановок - текущее количество приостановок выполнения.

Объекты исполняющей системы, видимые функциям Windows API

Тип объекта	Что он представляет
Process (Процесс)	Виртуальное адресное пространство и управляющую информацию, необходимую для выполнения набора объектов типа «поток»
Thread (Поток)	Исполняемая категория внутри процесса
Job (Задание)	Коллекция процессов, управляемых как единое целое в рамках задания
Section (Раздел)	Область разделяемой памяти (известная в Windows как проекция файла)
File (Файл)	Экземпляр открытого файла или устройства ввода-вывода
Token (Маркер)	Профиль безопасности (идентификатор безопасности, права пользователя и т.д.) процесса или потока
Event (Событие)	Объект, имеющий постоянное состояние (о котором поступил или не поступил сигнал), который может использоваться для синхронизации или уведомления
Semaphore (Семафор)	Счетчик, ограничивающий доступ к ресурсу путем разрешения доступа к этому ресурсу, защищенному семафором, вполне определенному максимальному количеству потоков
Mutex (Мьютекс)	Механизм синхронизации, используемый для последовательного доступа к ресурсу
Timer (Таймер)	Механизм уведомления потока об истечении конкретного периода времени
IoCompletion (Завершение ввода-вывода)	Метод для потоков по постановке в очередь и извлечении из нее уведомлений о завершении операций ввода-вывода (известный в Windows API как порт завершения ввода-вывода)
Key (Раздел реестра)	Механизм ссылки на данные реестра. Хотя разделы появляются в пространстве имен диспетчера объектов, они управляются диспетчером конфигурации точно так же, как файловые объекты управляются драйверами файловой системы. С объектом раздела (key) связано от нуля до нескольких значений раздела, эти значения содержат данные о разделе

Структура объекта



Общие службы объекта

- Менеджер объектов предоставляет небольшой набор служб, позволяющий работать с атрибутами, сохраненными в любом заголовке объекта, и применимый к объектам любого типа.
- Хотя общие службы объектов поддерживаются для объектов всех типов, у каждого объекта есть свои службы создания (create), открытия (open) и запроса (query).
- Например, система ввода-вывода реализует службу создания файла для своих файловых объектов, а менеджер процессов реализует службу создания процесса для своих объектов-процессов.

Служба	Назначение
Close (Закрытия)	Закрывает дескриптор объекта
Duplicate (Дублирования)	Обеспечивает совместное использование объекта путем создания дубликата его дескриптора и предоставления его другому процессу
Make permanent/temporary (Превращения в постоянный или временный)	Изменяет сохранность объекта (рассматриваемую далее)
Query object (Запроса объекта)	Получает информацию о стандартных атрибутах объекта
Query security (Запроса безопасности)	Получает дескриптор безопасности объекта
Set security (Установки безопасности)	Изменяет степень защиты объекта
Wait for a single object (Ожидания одиночного объекта)	Синхронизирует выполнение потока с одним объектом
Signal an object and wait for another (Сообщения об объекте и ожидания другого объекта)	Сообщает об объекте (таком, как событие) и синхронизирует выполнение потока с другим объектом
Wait for multiple objects (Ожидания нескольких объектов)	Синхронизирует выполнение потока с несколькими объектами

Методы объекта

Когда компонент исполняющей системы создает новый объект типа, он с помощью менеджера объектов может зарегистрировать один или несколько методов. После этого менеджер объектов вызывает методы в определенные моменты жизни объектов данного типа (при создании , его удалении или изменении). Методы, поддерживаемые менеджером объектов

Метод	Когда он вызывается
Open (Открыть)	При открытии дескриптора объекта
Close (Закрыть)	При закрытии дескриптора объекта
Delete (Удалить)	Перед удалением объекта диспетчером объектов
Query name (Запросить имя)	Когда поток запрашивает имя объекта, такого как ф которое существует в пространстве имен производн объекта
Parse (Провести разбор)	При поиске диспетчером объектов имени объекта, к существует в пространстве имен производного объе
Dump (Вывести дамп)	Не используется
Okay to close (Подтвер-дить закрытие)	Когда диспетчер объектов получает указание закры дескриптор
Security (Определить степень безопасности)	Когда процесс читает состояние защиты или измен защиту такого объекта, как файл, то есть имеет дел сведениями, существующими в пространстве имен водного объекта

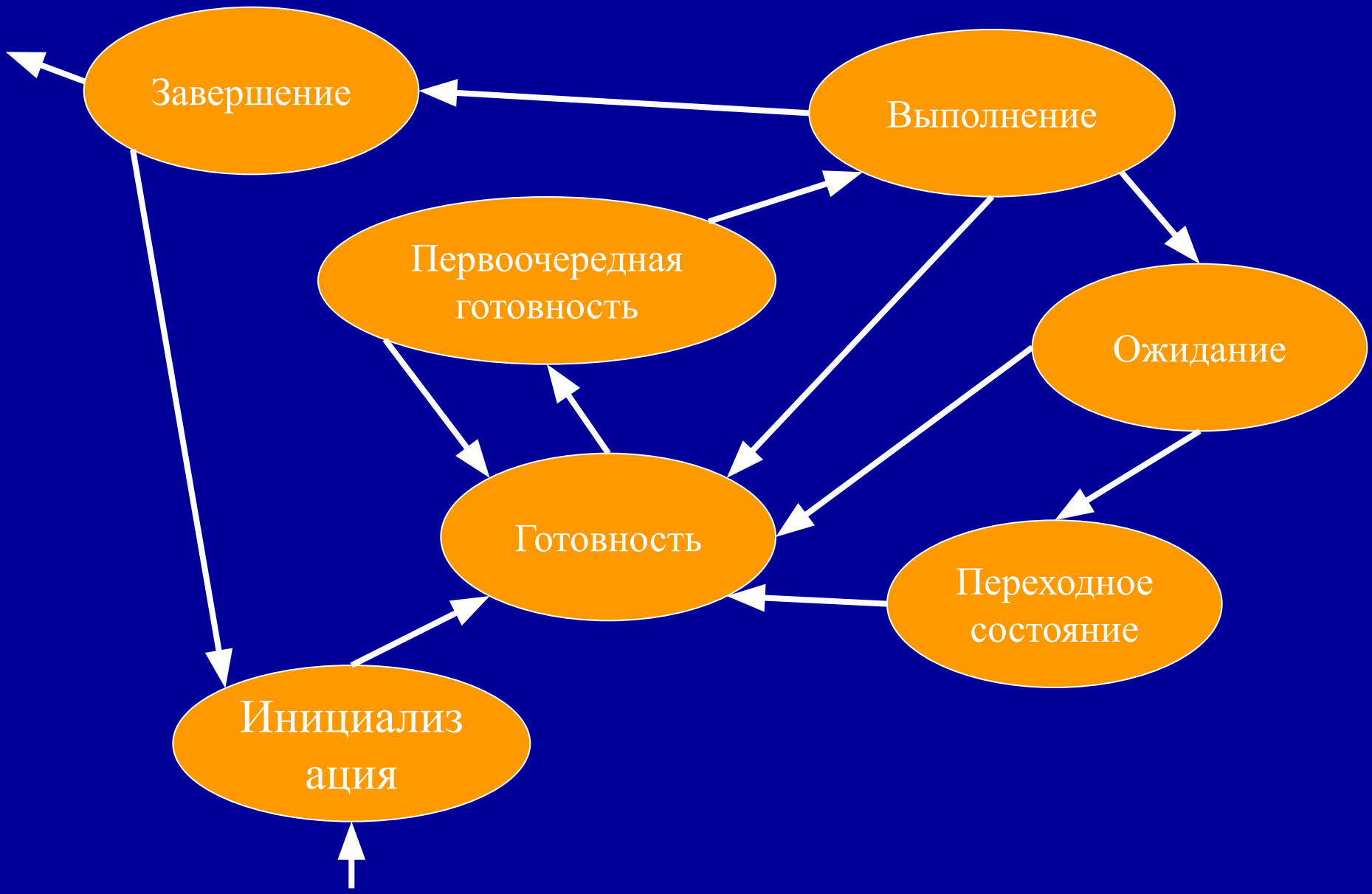
Безопасность объекта

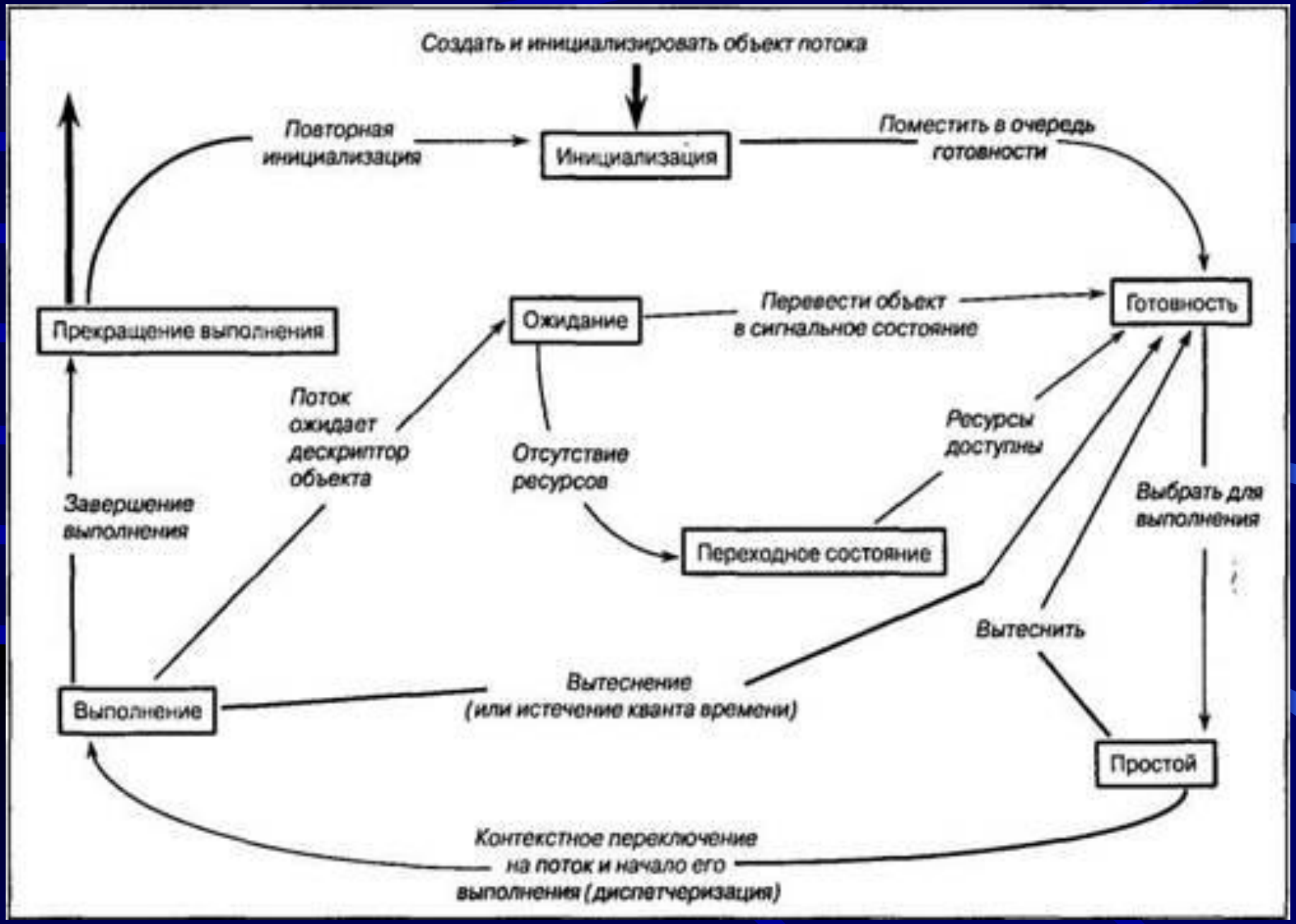
- Когда процесс открывает дескриптор объекта, менеджер объектов вызывает *монитор безопасности* (security reference monitor), ту часть системы безопасности, которая работает в режиме ядра, передавая ему от процесса набор желаемых прав доступа. Монитор безопасности ссылок проверяет, разрешает ли дескриптор безопасности объекта тот вид доступа, который запрашивается процессом. Если да, монитор возвращает набор *выделенных прав доступа*, которым разрешено пользоваться процессу, а менеджер объектов сохраняет права доступа в создаваемом им дескрипторе объекта.

Процессы и потоки

упрощенная схема структур данных процесса и потока

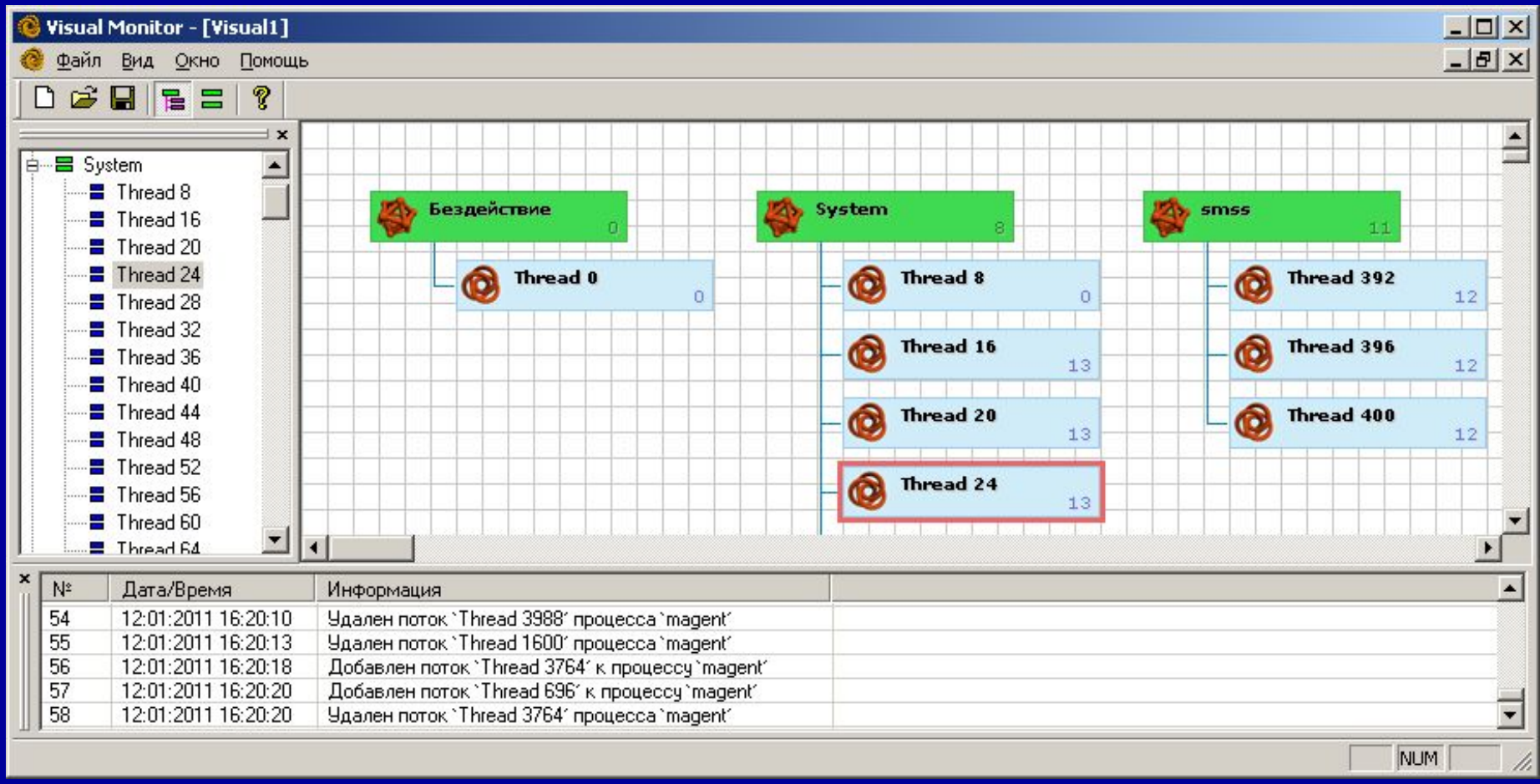
Состояния потока

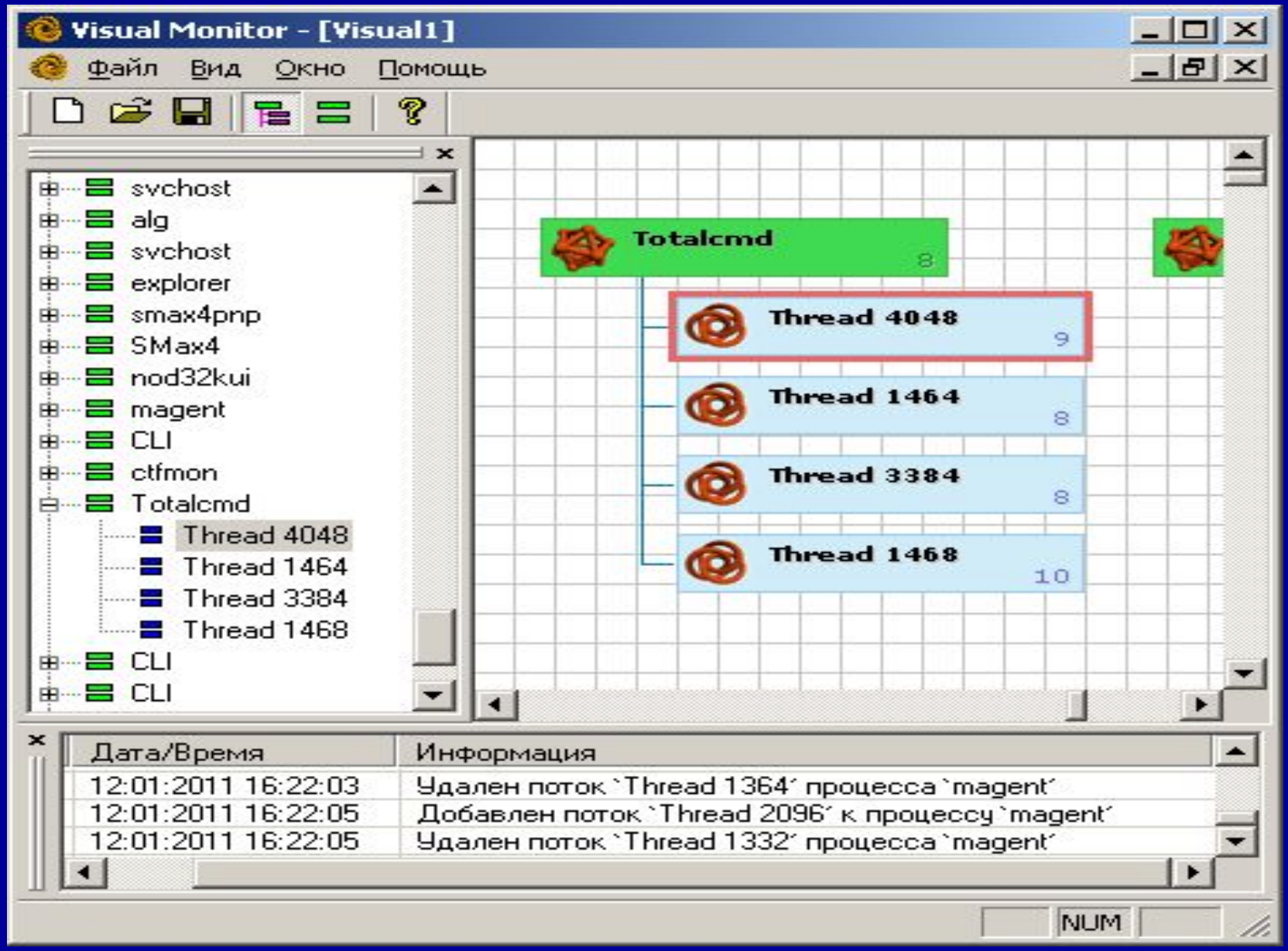


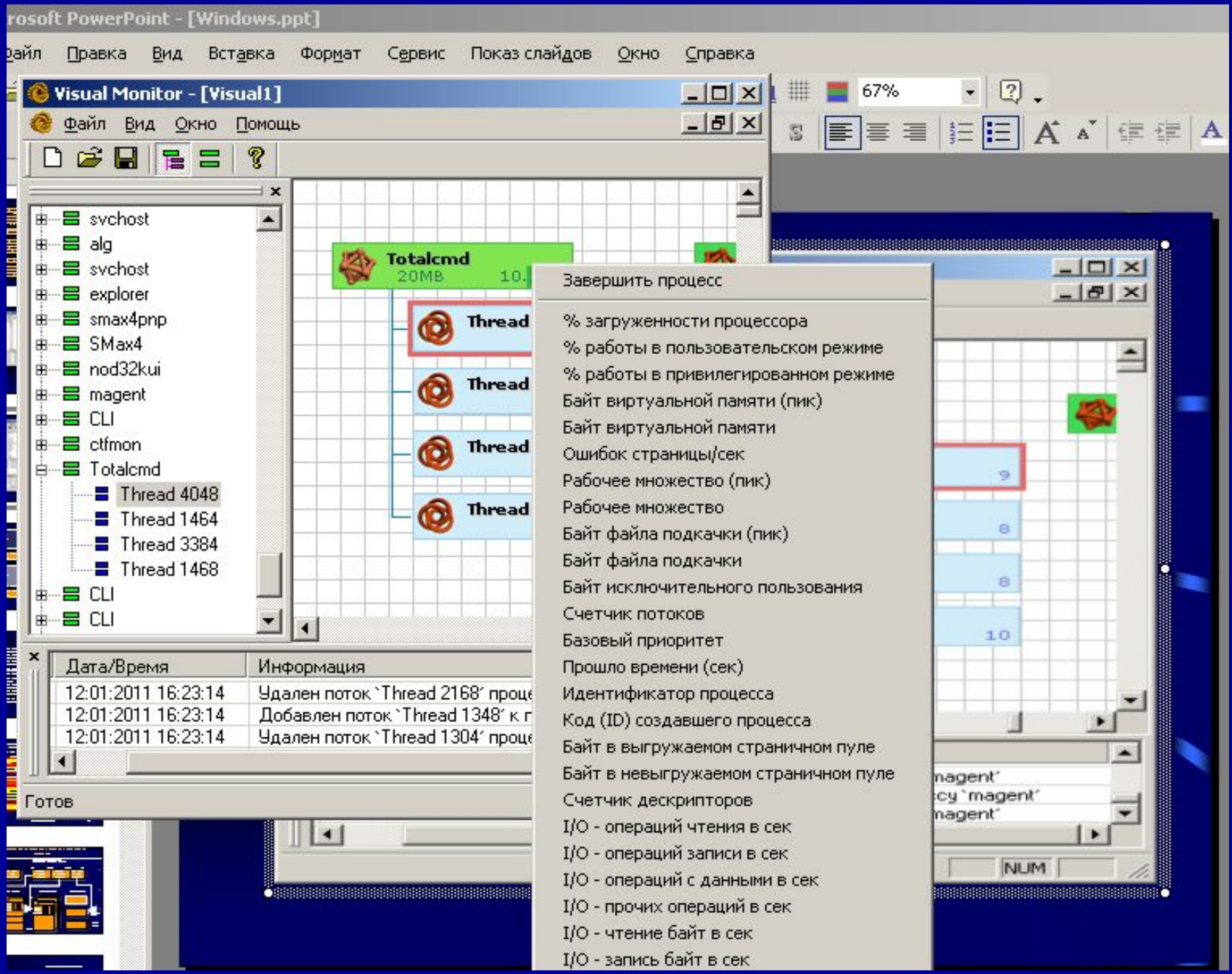

Создать и инициализировать объект потока
Повторная инициализация
Инициализация
Поместить в очередь готовности
Прекращение выполнения
Ожидание
Перевести объект в сигнальное состояние
Готовность
Поток ожидает дескриптор объекта
Отсутствие ресурсов
Ресурсы доступны
Завершение выполнения
Переходное состояние
Выбрать для выполнения
Вытеснить
Выполнение
Вытеснение (или истечение кванта времени)
Простой
Контекстное переключение на поток и начало его выполнения (диспетчеризация)

Состояния потока

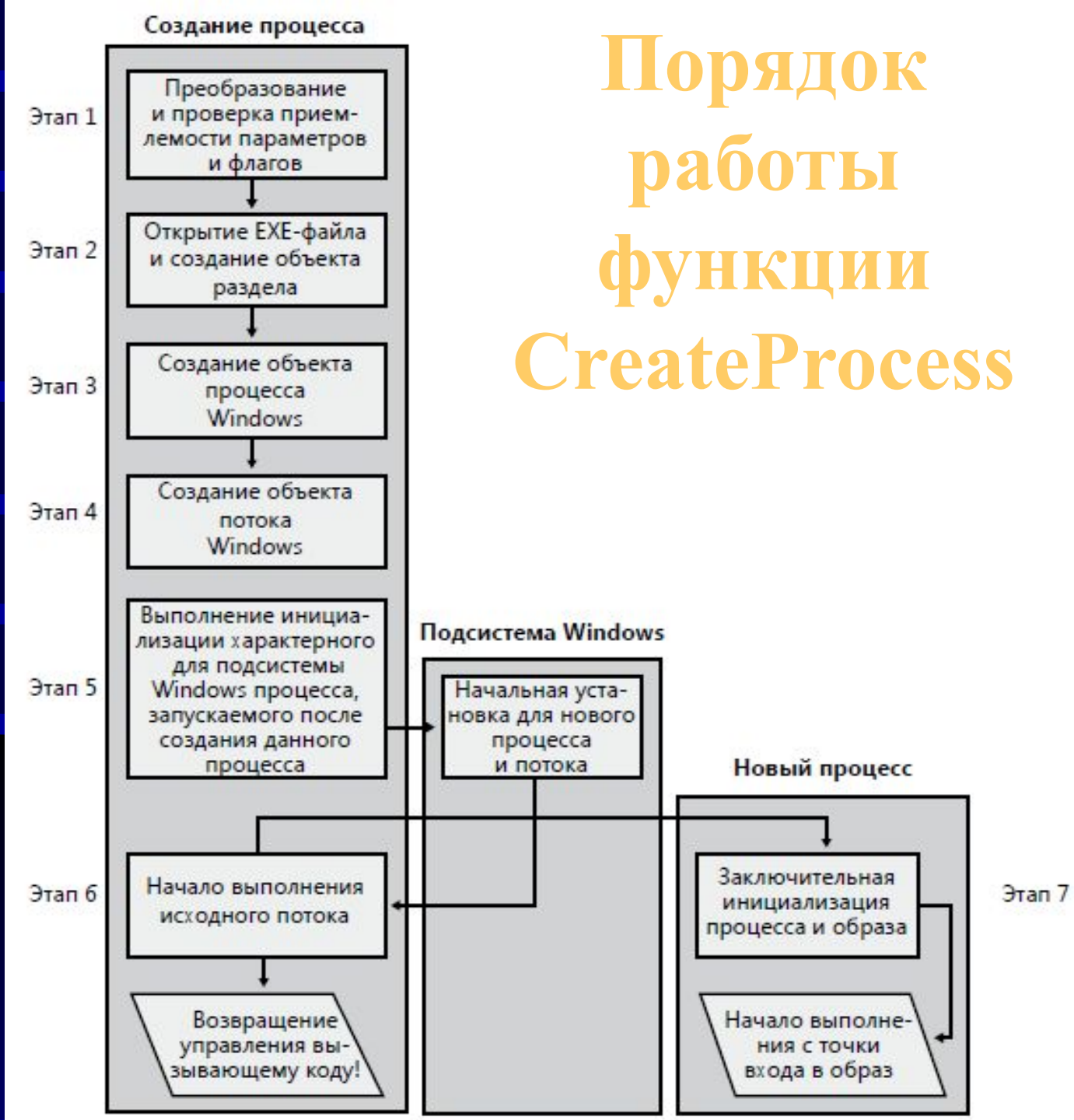
1. ***Готовность(Ready)*. У потока есть все, но не хватает только процессора (пул потоков).**
2. ***Первоочередная готовность* (*standby*). Для каждого процессора системы выбирается один поток, который будет выполняться следующим (самый первый поток в очереди). Когда условия позволяют, происходит переключение на контекст этого потока.**
3. ***Выполнение (Running)*. Поток выполняется процессором и покинет это состояние либо, если он завершится, либо, если появился более приоритетный поток или закончился квант времени, либо, если он ожидает какое-либо событие.**
4. ***Ожидание (Waiting)*. Поток может оказаться в этом состоянии либо по своей инициативе, если он ожидает некоторый объект для того, чтобы синхронизировать свое выполнение, либо операционная система (например, подсистема ввода-вывода) может ожидать чего-то в интересах потока, либо подсистема окружения может заставлять поток приостановить себя.**
5. ***Переходное состояние (Transition)*. Поток входит в переходное состояние, если он готов к выполнению, но страница, содержащая стек потока, выгружена из оперативной памяти на диск в файл подкачки.**
6. ***Завершение(Terminate)*. Когда выполнение всех команд потока закончилось, он находится в состоянии завершения до тех пор, пока его не удалит менеджер объектов. Если в исполнительной части имеется указатель на этот же поток, то он может быть инициализирован и использован снова.**


Visual Monitor - [Visual1]
Файл Вид Окно Помощь
System
Thread 8
Thread 16
Thread 20
Thread 24
Thread 28
Thread 32
Thread 36
Thread 40
Thread 44
Thread 48
Thread 52
Thread 56
Thread 60
Thread 64
Бездействие 0
Thread 0 0
System 8
Thread 8 0
Thread 16 13
Thread 20 13
Thread 24 13
smss 11
Thread 392 12
Thread 396 12
Thread 400 12
№ Дата/Время Информация
54 12:01:2011 16:20:10 Удален поток `Thread 3988' процесса `magent'
55 12:01:2011 16:20:13 Удален поток `Thread 1600' процесса `magent'
56 12:01:2011 16:20:18 Добавлен поток `Thread 3764' к процессу `magent'
57 12:01:2011 16:20:20 Добавлен поток `Thread 696' к процессу `magent'
58 12:01:2011 16:20:20 Удален поток `Thread 3764' процесса `magent'
NUM


Visual Monitor - [Visual1]
Файл
Вид
Окно
Помощь
svchost
alg
svchost
explorer
smax4pnp
SMax4
nod32kui
magent
CLI
ctfmon
Totalcmd
Thread 4048
Thread 1464
Thread 3384
Thread 1468
CLI
CLI
Totalcmd
8
Thread 4048
9
Thread 1464
8
Thread 3384
8
Thread 1468
10
Дата/Время
Информация
12:01:2011 16:22:03
Удален поток `Thread 1364' процесса `magent'
12:01:2011 16:22:05
Добавлен поток `Thread 2096' к процессу `magent'
12:01:2011 16:22:05
Удален поток `Thread 1332' процесса `magent'
NUM


rosoft PowerPoint - [Windows.ppt]
Файл Правка Вид Вставка Формат Сервис Показ слайдов Окно Справка
Visual Monitor - [Visual1]
Файл Вид Окно Помощь
svchost
alg
svchost
explorer
smax4pnp
SMax4
nod32kui
magent
CLI
ctfmon
Totalcmd
Thread 4048
Thread 1464
Thread 3384
Thread 1468
CLI
CLI
Totalcmd
20MB
Thread
Thread
Thread
Thread
Дата/Время
Информация
12:01:2011 16:23:14
12:01:2011 16:23:14
12:01:2011 16:23:14
Готов
Завершить процесс
% загруженности процессора
% работы в пользовательском режиме
% работы в привилегированном режиме
Байт виртуальной памяти (пик)
Байт виртуальной памяти
Ошибок страницы/сек
Рабочее множество (пик)
Рабочее множество
Байт файла подкачки (пик)
Байт файла подкачки
Байт исключительного пользования
Счетчик потоков
Базовый приоритет
Прошло времени (сек)
Идентификатор процесса
Код (ID) создавшего процесса
Байт в выгружаемом страничном пуле
Байт в невыгружаемом страничном пуле
Счетчик дескрипторов
I/O - операций чтения в сек
I/O - операций записи в сек
I/O - операций с данными в сек
I/O - прочих операций в сек
I/O - чтение байт в сек
I/O - запись байт в сек
67%
NUM

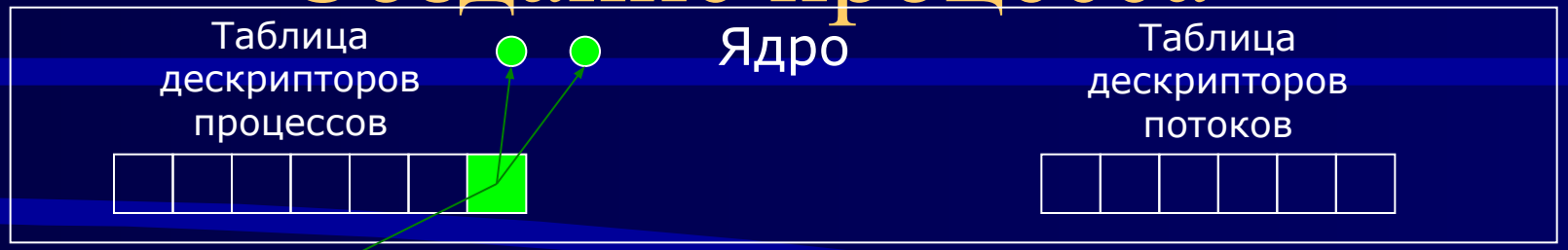
Порядок работы функции CreateProcess

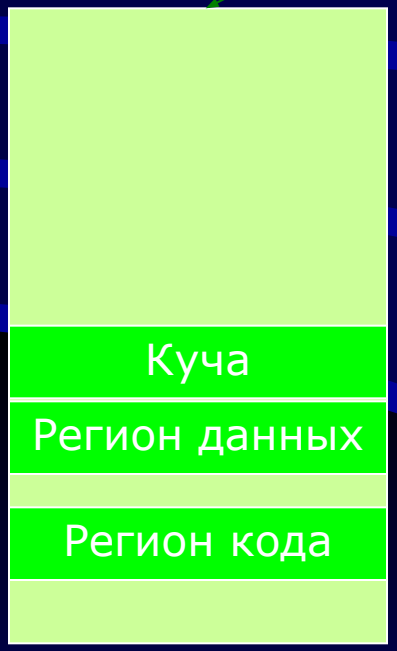


Создание процесса

- Создать дескриптор процесса и поместить его в таблицу процессов
- Проинициализировать значения полей общего назначения дескриптора процесса
- Создать виртуальное адресное пространство (ВАП) процесса и сформировать его структуру
- Заполнить необходимыми данными ВАП процесса (разместить в нем код, данные и т.д.)
- Выделить процессу ресурсы, которые он может использовать сразу после создания
- Оповестить подсистемы, принимающие участие в управлении процессами, о создании нового процесса
- Создать первичный поток процесса

Создание процесса





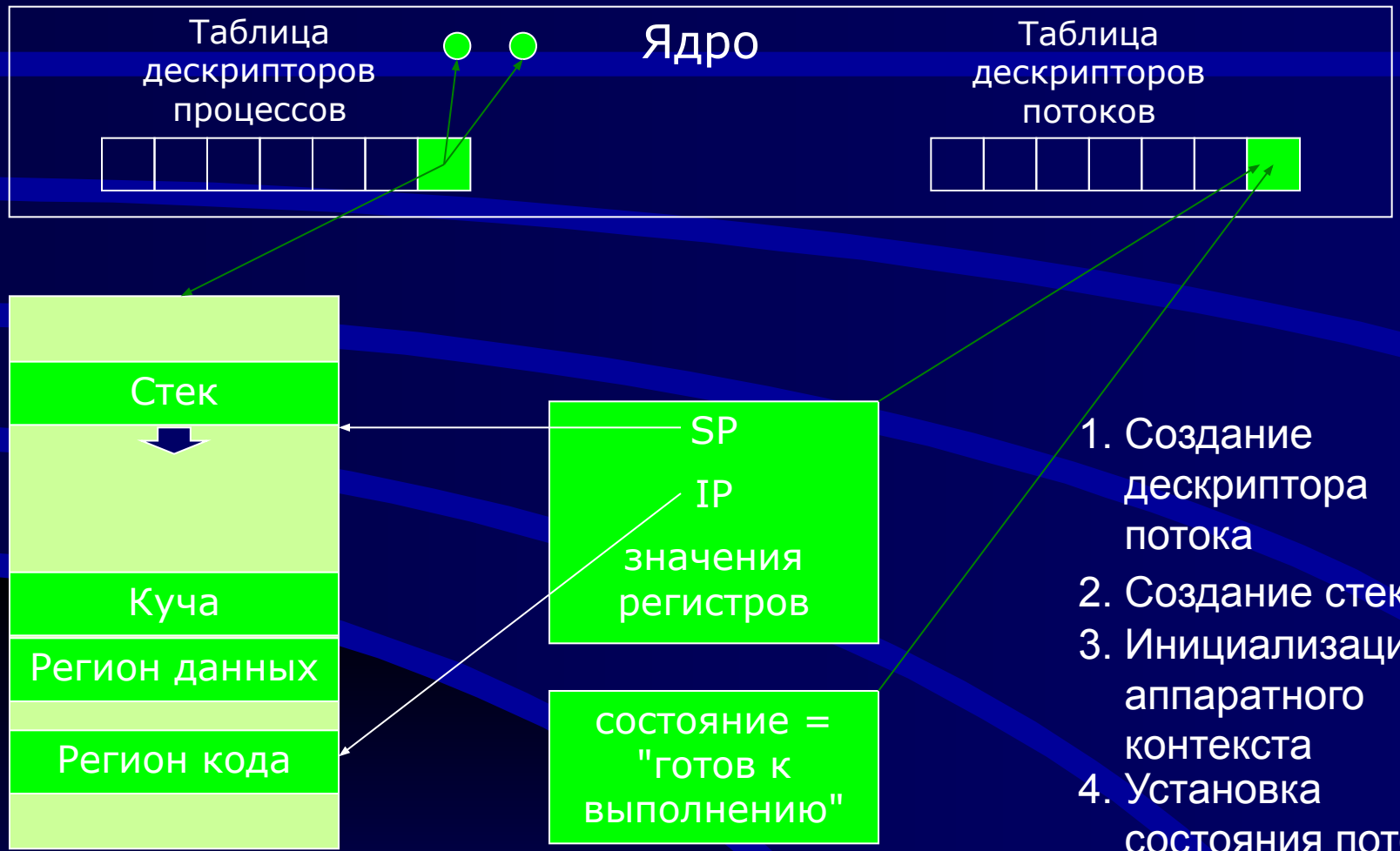
1. Создание дескриптора процесса
2. Создание ВАП процесса
3. Формирование структуры ВАП процесса и его заполнение
4. Выделение ресурсов по умолчанию

Создание первичного потока

Создание потока

- Создать дескриптор потока и поместить его в таблицу потоков
- Проинициализировать значения полей общего назначения дескриптора потока
- Создать области данных, необходимые для функционирования потока в данной аппаратной архитектуре
- Инициализировать поле дескриптора «аппаратный контекст выполнения потока»
- Оповестить подсистемы, принимающие участие в управлении потоками, о создании нового потока
- Перевести поток в состояние «готов к выполнению»

Создание потока



1. Создание дескриптора потока
2. Создание стека
3. Инициализация аппаратного контекста
4. Установка состояния потока

Завершение потока

- Сохранить статистические данные потока и код возврата в его дескрипторе
- Перевести все ресурсы, принадлежащие потоку, в непротиворечивое и стабильное состояние
- Освободить все ресурсы, принадлежавшие потоку или использовавшиеся потоком
- Оповестить подсистемы, принимающие участие в управлении потоками, о завершении потока
- Установить состояние потока в значение «завершен»
- Если данный поток является последним активным потоком в процессе – завершить процесс

После выполнения всех действий остается дескриптор потока, содержащий его код возврата и статистические данные; момент уничтожения дескриптора зависит от реализации

Завершение процесса

- Завершить выполнение всех потоков процесса
- Сохранить статистические данные процесса и код возврата в его дескрипторе
- Перевести все ресурсы, принадлежащие процессу, в непротиворечивое и стабильное состояние
- Освободить все ресурсы, принадлежавшие процессу или использовавшиеся процессом
- Освободить ВАП и уничтожить его
- Оповестить подсистемы, принимающие участие в управлении процессами, о завершении процесса
- Установить состояние процесса в значение «завершен»

После выполнения всех действий остается дескриптор процесса, содержащий его код возврата и статистические данные; момент уничтожения дескриптора зависит от реализации

Создание/завершение процесса – Win32

- ```
  BOOL CreateProcess(
   LPCTSTR lpszImageName,
   LPCTSTR lpszCommandLine,
   LPSECURITY_ATTRIBUTES lpsaProcess,
   LPSECURITY_ATTRIBUTES lpsaThread,
   BOOL fInheritHandles,
   DWORD fdwCreate,
   LPVOID lpvEnvironment,
   LPTSTR lpszCurDir,
   LPSTARTUPINFO lpsiStartInfo,
   LPPROCESS_INFORMATION lppiProcInfo);
  ```
- ```
  VOID ExitProcess(UINT fuExitCode);
  ```
- ```
  BOOL TerminateProcess(
   HANDLE hProcess,
   UINT uExitCode );
  ```
  ```

```
STARTUPINFO si;
PROCESS_INFORMATION pi;
ZeroMemory( &si, sizeof(si) );
si.cb = sizeof(si);
ZeroMemory( &pi, sizeof(pi) );

CreateProcess( NULL,        // No module name (use command line).
    TEXT("MyChildProcess"),// Command line.
    NULL,             // Process handle not inheritable.
    NULL,             // Thread handle not inheritable.
    FALSE,            // Set handle inheritance to FALSE.
    0,                // No creation flags.
    NULL,             // Use parent's environment block.
    NULL,             // Use parent's starting directory.
    &si,              // Pointer to STARTUPINFO structure.
    &pi );            // Pointer to PROCESS_INFORMATION structure.

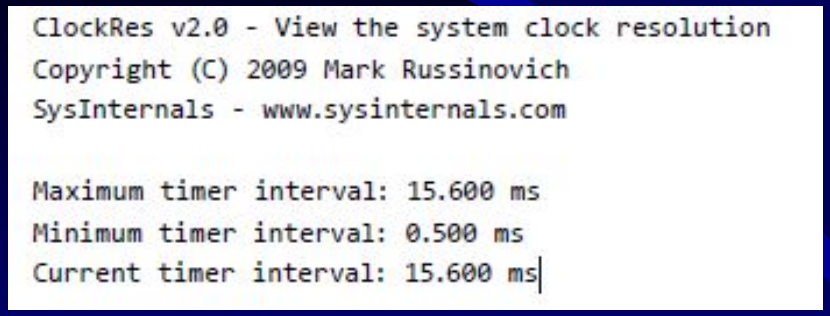
// Wait until child process exits.
WaitForSingleObject( pi.hProcess, INFINITE );
// Close process and thread handles.
CloseHandle( pi.hProcess );
CloseHandle( pi.hThread );
```

Создание/завершение потока – Win32

- ```
  HANDLE CreateThread(
   LPSECURITY_ATTRIBUTES lpThreadAttributes,
   SIZE_T dwStackSize,
   LPTHREAD_START_ROUTINE lpStartAddress,
   LPVOID lpParameter,
   DWORD dwCreationFlags,
   LPDWORD lpThreadId);
  ```
- ```
  VOID ExitThread(DWORD dwExitCode);
  ```
- ```
  BOOL TerminateThread(
   HANDLE hThread,
   DWORD dwExitCode);
  ```
  ```

```
int GlobalVar = 0;

DWORD WINAPI ThreadProc( LPVOID arg ){
    *(int*)arg = *(int*)arg + 1;
    ExitThread(0);
}

int main(void){
  int i;
  HANDLE Threads[10];

  for( i = 0; i < 10; i ++ ){
    Threads[i] = CreateThread(NULL, 0, ThreadProc,
                              (LPVOID)&GlobalVar, 0, NULL);
  }

  for( i = 0; i < 10; i ++ ){
    WaitForSingleObject(Threads[i],INFINITE);
  }
  return 0;
}
```

Планирование выполнения процессов и потоков

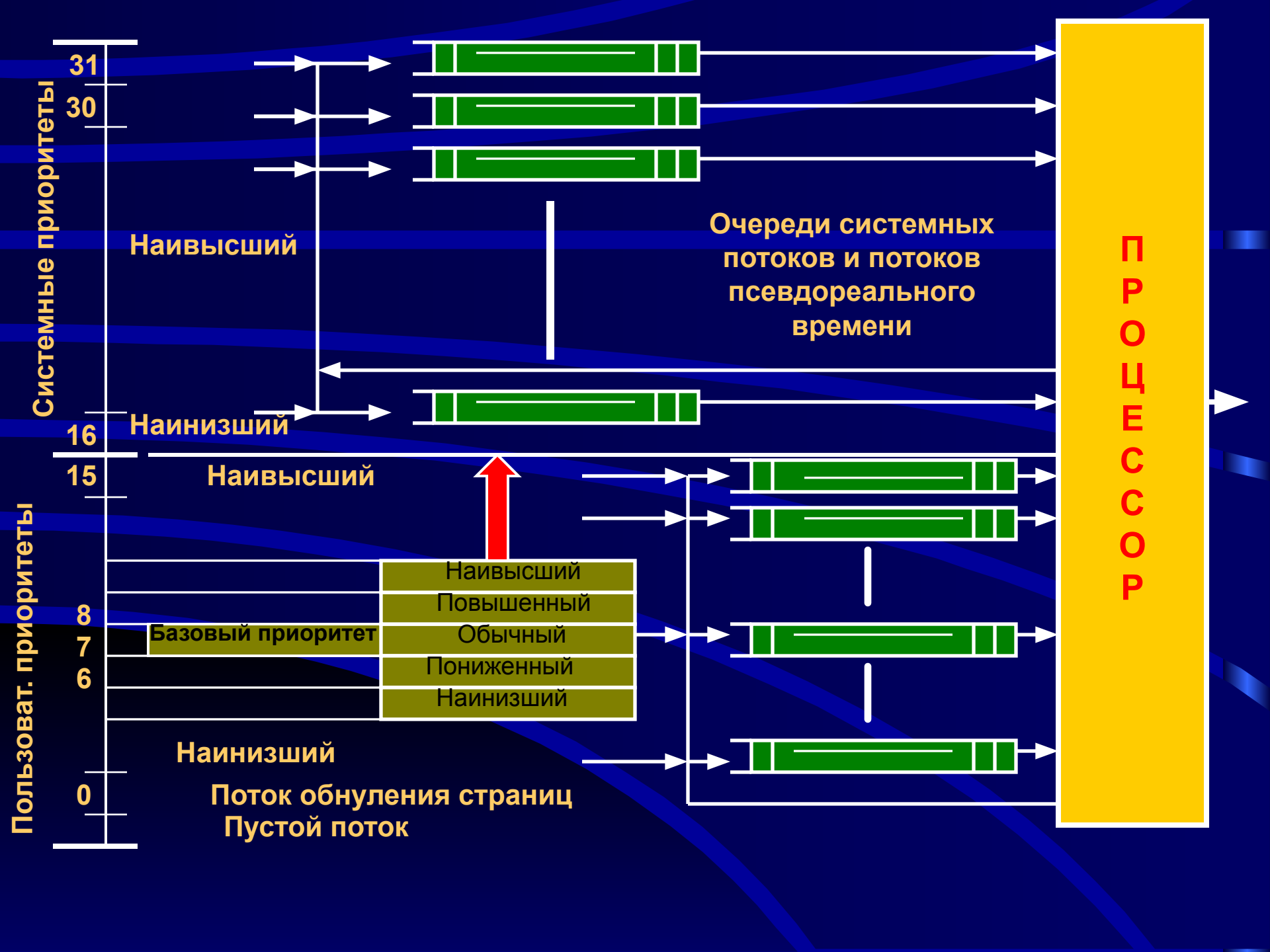
- Операционная система распределяет процессорное время между потоками, выделяя каждому из них определенную долю времени (квант). Процессорное время выделяется по очереди каждому потоку, но при этом учитывается также приоритет потока. Когда прекращается выполнение первичного потока процесса, уничтожается и сам процесс.

- На клиентских версиях Windows потоки по умолчанию выполняются в течение двух интервалов таймера (clock intervals), а на серверных системах по умолчанию поток выполняется в течение 12 интервалов таймера (чтобы минимизировать контекстные переключения).
- Продолжительность интервала таймера варьируется в зависимости от аппаратной платформы и зависит от HAL.
- Например, интервал таймера на большинстве однопроцессорных систем x86 составляет около 10 миллисекунд, на большинстве мультипроцессорных систем x86 и x64 - порядка 15 миллисекунд. Этот интервал таймера хранится в переменной ядра *KeMaximumIncrement* в виде сотен наносекунд.

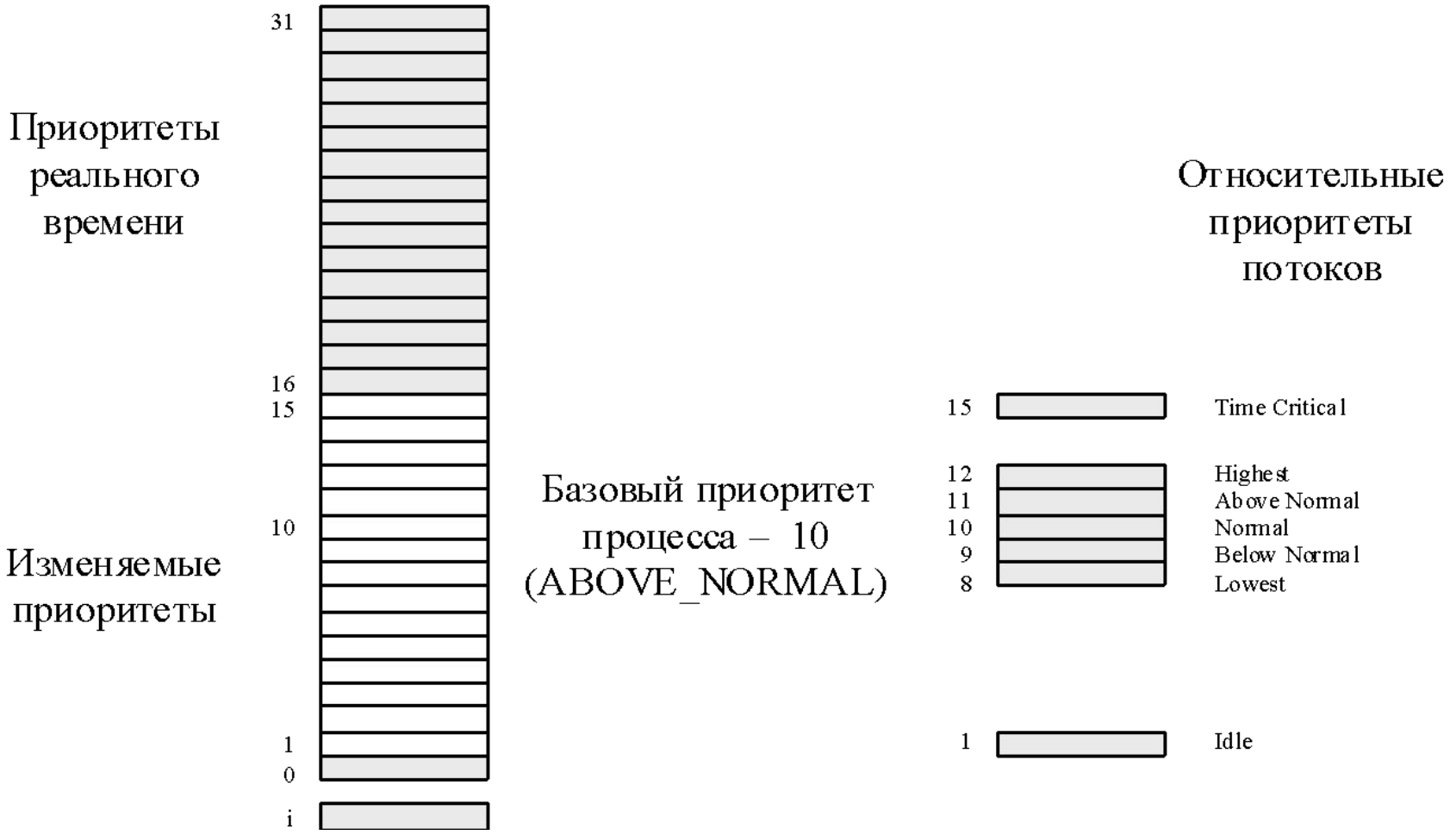


- Когда система запускается, делается вычисление, результатом которого является количество тактовых циклов, которому равен каждый квант времени (значение сохраняется в переменной ядра KiCyclesPerClockQuantum). Это вычисление делается путем умножения тактовой частоты процессора в герцах (числа тактовых импульсов центрального процессора в секунду) на количество секунд, затрачиваемое на запуск одного такта системных часов (на основе KeMaximumIncrement).
- Настройки кванта хранит параметр реестра HKLM\SYSTEM\CurrentControlSet\Control\PriorityControl\Win32PrioritySeparation

	Короткий индекс кванта			Длинный индекс кванта		
Переменное значение	6	12	18	12	24	36
Фиксированное значение	18	18	18	36	36	36


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

Планирование в Windows



- **Динамические** приоритеты имеют значения в диапазоне 1-15. ОС может изменять приоритет потока в этом диапазоне.
- **Приоритеты реального времени** имеют значения в диапазоне 16-31. ОС не может изменять значение приоритета потока, находящееся в этом диапазоне.
- Низший приоритет планирования со значением 0 зарезервирован для потока обнуления страниц (Zero Page Thread), который выполняется в случае, когда больше нечего исполнять. Этот поток является компонентом диспетчера памяти, и его работа состоит в обнулении страниц из списка свободных страниц.

- Имеется два важных отличия между динамическими приоритетами и приоритетами реального времени.
- Поток с *приоритетом реального времени* может сохранять контроль над процессором до тех пор, пока не появится поток с большим или равным значением приоритета. Таким образом, пока выполняется поток реального времени, потоки с меньшим значением приоритета никогда не получат шанса исполниться.
- Однако при появлении потока с большим или равным значением приоритета задействуется механизм вытесняющей многозадачности.

- Ядро Windows всегда запускает тот из потоков, готовых к выполнению, который обладает наивысшим приоритетом. Поток не является готовым к выполнению, если он находится в состоянии ожидания, приостановлен или блокирован по той или иной причине.
- Процесс может изменить или установить свой собственный приоритет или приоритет другого процесса, если это разрешено атрибутами защиты.

BOOL SetPriorityClass(HANDLE hProcess, DWORD dwPriority)

DWORD GetPriorityClass(HANDLE hProcess)

- Потоки получают приоритеты на базе классов приоритета **своих процессов**.

- Приоритеты **потоков** устанавливаются относительно базового приоритета процесса, и во время создания потока его приоритет устанавливается равным приоритету процесса. Приоритеты потоков могут принимать значения в интервале ±2 относительно базового приоритета процесса. Результирующим пяти значениям приоритета присвоены следующие символические имена:
- THREAD_PRIORITY_LOWEST
- THREAD_PRIORITY_BELOW_NORMAL
- THREAD_PRIORITY_NORMAL
- THREAD_PRIORITY_HIGHEST

Первоначально функцией CreateProcess устанавливаются четыре класса приоритета, каждый из которых имеет *базовый приоритет* (base priority):

- IDLE_PRIORITY_CLASS, базовый приоритет 4.
- NORMAL_PRIORITY_CLASS, базовый приоритет 9 - 7.

Нормальный базовый приоритет равен 9, если фокус ввода с клавиатуры находится в окне; в противном случае этот приоритет равен 7.

- HIGH_PRIORITY_CLASS, базовый приоритет 13.
- REALTIME_PRIORITY_CLASS, базовый приоритет 24.

Классом REALTIME_PRIORITY_CLASS следует пользоваться с осторожностью, чтобы не допустить вытеснения других процессов.

Для установки и выборки относительного приоритета потока следует использовать эти значения. Обратите внимание на использование целых чисел со знаком вместо чисел типа DWORD.

BOOL SetThreadPriority(HANDLE hThread, int nPriority)

int GetThreadPriority(HANDLE hThread)

Существуют два дополнительных значения приоритета потоков. Они являются абсолютными, а не относительными, и используются только в специальных случаях.

- THREAD_PRIORITY_IDLE имеет значение 1 (или 16 — для процессов, выполняющихся в режиме реального времени).
- THREAD_PRIORITY_TIME_CRITICAL имеет значение 15 (или 31 — для процессов, выполняющихся в режиме реального времени).

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).

Устройство	Повышение приоритета
Жесткий диск, привод компакт-дисков, параллельный порт, видеоустройство	1
Сеть, почтовый слот, именованный канал, последовательный порт	2
Клавиатура, мышь	6
Звуковое устройство	8

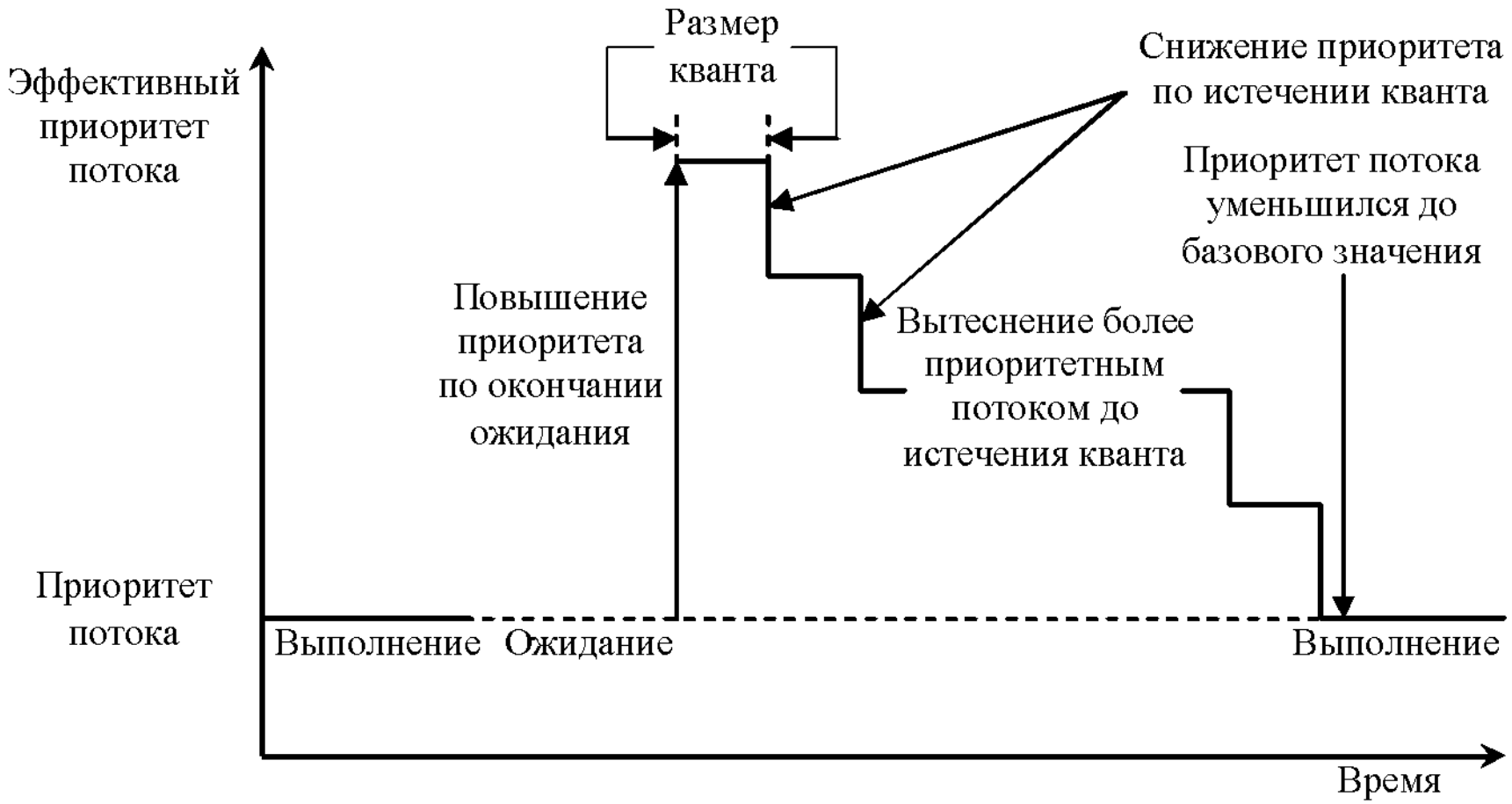
Алгоритмы приоритетного планирования

- Для предотвращения бесконечной работы высокоприоритетных потоков планировщик может уменьшать с каждым тактом таймера текущий приоритет выполняющегося потока
- Для предоставления низкоприоритетным потокам процессорного времени планировщик может проверять время, в течение которого поток находился в состоянии готовности к исполнению
- Возможно адаптивное и динамическое изменение приоритетов
  - адаптивное изменение – коррекция эффективных приоритетов на основании набора правил
  - динамическое изменение – регулярный пересчет эффективных приоритетов на основании значений некоторой совокупности параметров

‰Планировщик Windows периодически настраивает текущий приоритет потоков, используя внутренний механизм повышения приоритета, используя ряд сценариев:

- Повышение вследствие событий диспетчера (сокращение задержек).
- ‰‰Повышение вследствие завершения ввода-вывода (сокращение задержек).
- ‰‰Повышение вследствие ввода из пользовательского интерфейса (сокращение задержек и времени отклика).
- ‰‰Повышение вследствие слишком продолжительного ожидания ресурса исполняющей системы (предотвращение зависания).
- ‰‰Повышение в случае, когда готовый к запуску поток не был запущен в течение определенного времени (предотвращение зависания и смены приоритетов).

- Интересной аномалией, относящейся к процессу простоя **Idle** , является то, что Windows сообщает, что у процесса простоя **Idle** уровень приоритета равен 0. Но в действительности у приоритетов потоков простоя нет значения, потому что такие потоки выбираются для диспетчеризации, только если нет никаких других потоков, готовых к выполнению. Их приоритет никогда не сравнивается с приоритетами других потоков, не используется для помещения потока простоя в очередь готовых потоков, потоки простоя никогда не стоят ни в каких очередях готовых потоков. (Только один поток в каждой системе Windows действительно выполняется с приоритетом 0 — это поток обнуления страниц (zero page thread).)


Размер кванта
Снижение приоритета по истечении кванта
Эффективный приоритет потока
Приоритет потока уменьшился до базового значения
Повышение приоритета по окончании ожидания
Вытеснение более приоритетным потоком до истечения кванта
Приоритет потока
Выполнение
Ожидание
Выполнение
Время

В клиентские версии Windows была внедрена служба MMCSS (MultiMediaClass Scheduler Service). Ее целью было гарантировать проигрывание мультимедийного контента приложений, зарегистрированных с этой службой, без каких-либо сбоев. MMCSS работает со следующими задачами:

- ‰‰аудио;
- ‰‰захват;
- ‰‰распределение;
- ‰‰игры;
- ‰‰проигрывание;
- ‰‰аудио профессионального качества;
- ‰‰задачи администратора многооконного режима.

Каждая из этих задач включает информацию о категории планирования — Scheduling Category, которое является первичным фактором, определяющим приоритет потоков, зарегистрированных с MMCSS.

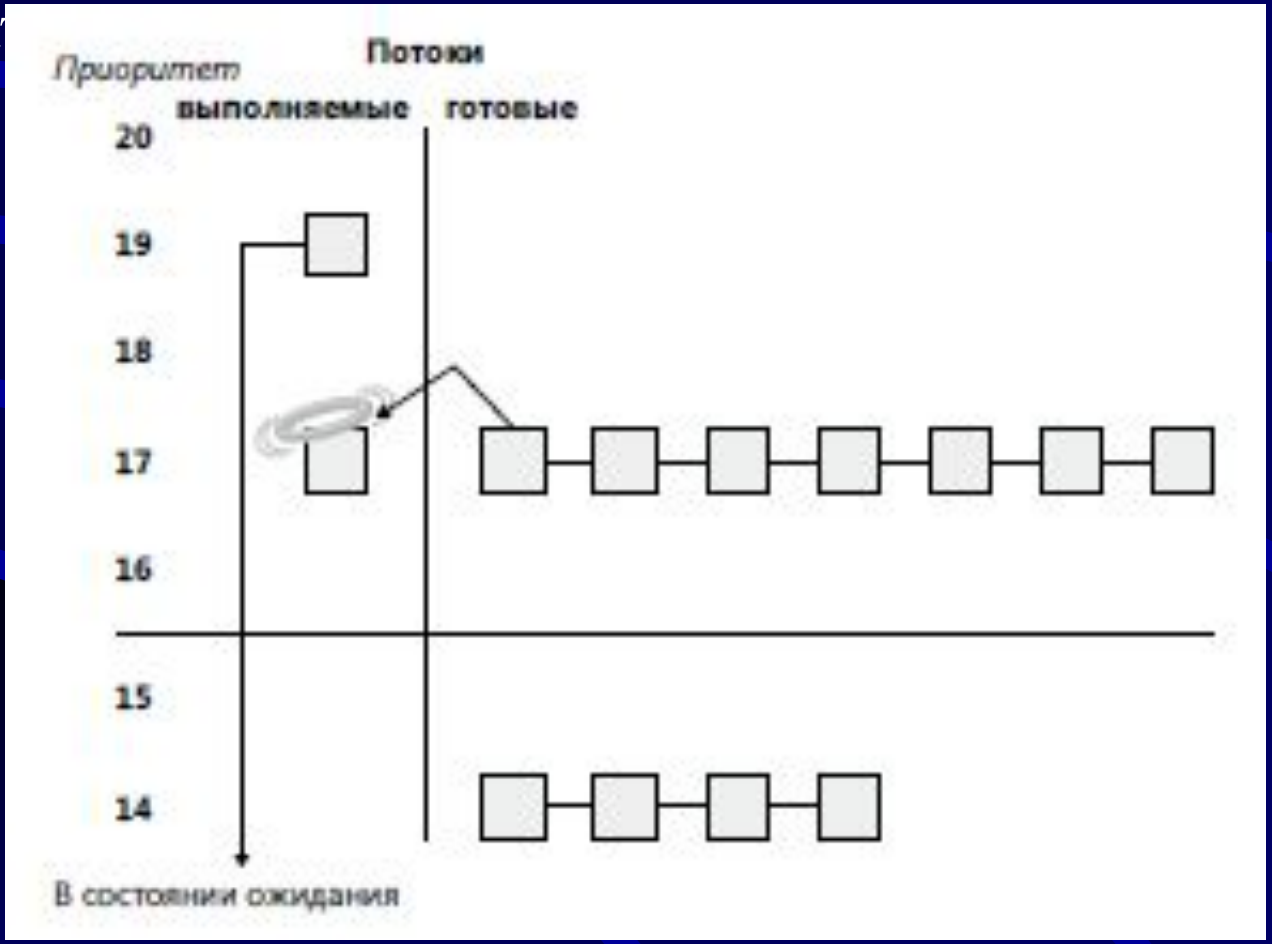
Категория	Приоритет	Описание
High (Высокая)	23–26	Потоки профессионального аудио (Pro Audio), запущенные с приоритетом выше, чем у других потоков на системе, за исключением критических системных потоков
Medium (Средняя)	16–22	Потоки, являющиеся частью приложений первого плана, например Windows Media Player
Low (Низкая)	8–45	Все остальные потоки, не являющиеся частью предыдущих категорий
Exhausted (Исчерпавших потоков)	1–7	Потоки, исчерпавшие свою долю времени центрального процессора, выполнение которых продолжится, только если не будут готовы к выполнению другие потоки с более высоким уровнем приоритета

По умолчанию мультимедийные потоки получают 80 % доступного времени центрального процессора, а другие потоки получают 20 % этого. Сама служба MMCSS выполняется с приоритетом 27, поскольку ей нужно вытеснять любые Pro Audio-потоки с целью снижения их приоритета до категории Exhausted.

Моменты смены потоков

1. **Самостоятельное переключение**

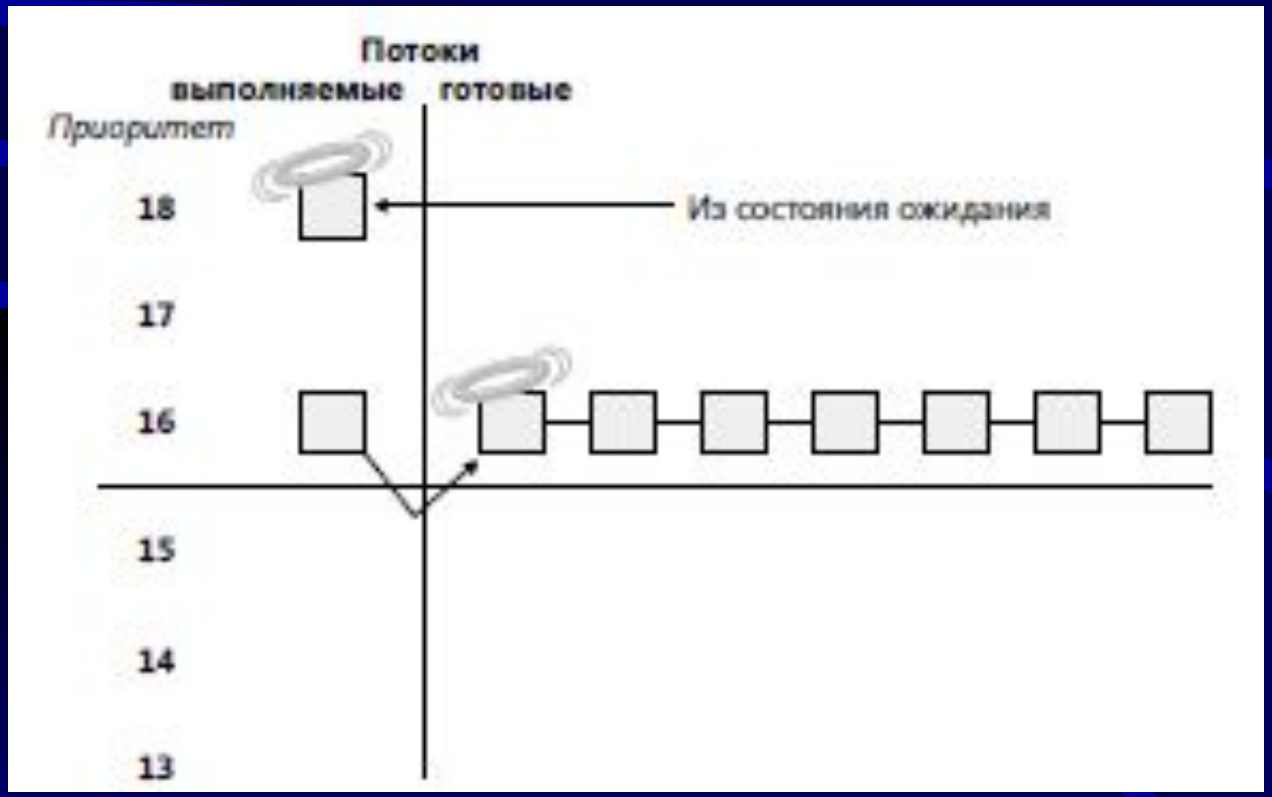
Поток может добровольно отказаться от использования процессора из-за входа в состояние ожидания какого-нибудь объект



2. Планирование с вытеснением потоков

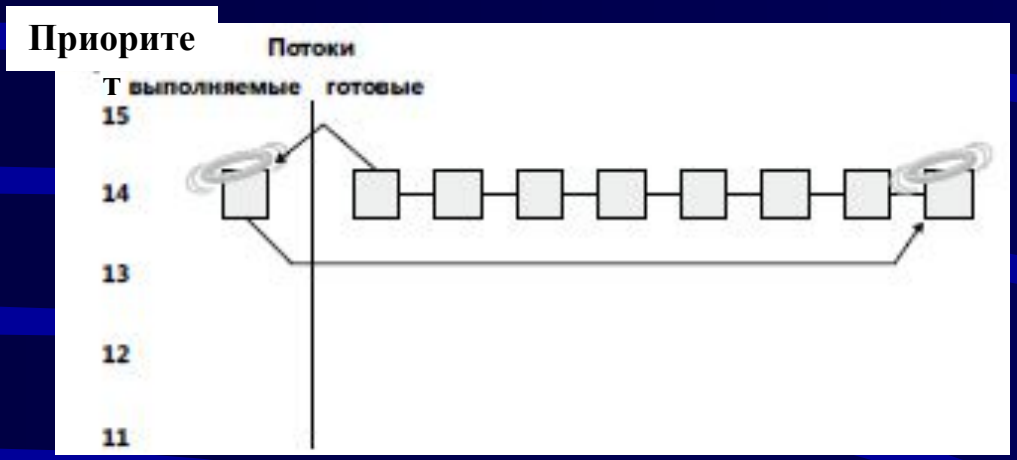
Поток с более низким приоритетом вытесняется, когда становится готовым к выполнению поток с более высоким приоритетом. Такая ситуация может сложиться по двум причинам:

- ‰‰Поток с более высоким приоритетом завершил ожидание. (Произошло ожидаемое им событие.)
- ‰‰Произошло повышение или снижение приоритета потока.



3. Истечение кванта времени

Windows использует для обслуживания квантовых целей точный счетчик тактовых циклов центрального процессора, это обеспечивает справедливое распределение процессорного времени.



- ‰Потоки А и Б стали готовы к выполнению в середине интервала.
- ‰‰Поток А начал выполняться, но его выполнение на какое-то время было прервано. Тактовые циклы центрального процессора, затраченные на обработку прерывания, потоку не возмещаются.
- ‰‰Обработка прерывания завершается, и поток А снова запускается на выполнение, но быстро достигает следующего интервала таймера. Планировщик смотрит на количество тактовых циклов центрального процессора, выделенных потоку, и сравнивает его с ожидаемым количеством тактовых циклов центрального процессора, которые должны были быть выделены до конца кванта времени. ‰‰ Поскольку предшествующее количество намного меньше того, каким оно должно быть, планировщик предполагает, что поток А начал выполняться в середине интервала таймера и, кроме того, его выполнение могло быть прервано. ‰‰
- Поток А получает повышение кванта своего времени на еще один интервал таймера, и квантовая цель пересчитывается. Теперь у потока А есть шанс выполняться в течение полного интервала таймера. ‰‰
- В следующий интервал таймера поток А выберет свой квант времени, и теперь шанс на выполнение получит поток Б.

Объект - задание

Объект - задание представляет собой объект ядра, который позволяет контролировать один или несколько процессов, сведенных в группу. У него может быть имя, механизм защиты и механизм общего использования. Основной функцией объекта-задания является предоставление возможности управления группой процессов как единым целым и работы с этим объединением.

- Процесс может быть членом только одного объекта задания.
- По умолчанию связь процесса с объектом задания не может быть разорвана, и все созданные им процессы и их потомки также связаны с тем же объектом задания.
- Объект задания записывает основную учетную информацию для всех процессов, связанных с заданием, и для всех процессов, которые были связаны с заданием, но работа которых уже была завершена.

Синхронизация процессов и потоков

- В Windows реализована вытесняющая многозадачность - это значит, что в любой момент система может прервать выполнение одного потока и передать управление другому.
- Поэтому необходим механизм, позволяющий потокам согласовывать свою работу с общими ресурсами. Этот механизм получил название механизма синхронизации потоков (thread synchronization).

Объектов синхронизации существует несколько, самые важные из них:

- взаимоисключение (mutex),
- критическая секция (critical section),
- событие (event)
- семафор (semaphore).
- Также в качестве объектов синхронизации могут использоваться сами процессы и потоки (когда один поток ждет завершения другого потока или процесса); а также файлы, коммуникационные устройства, консольный ввод и уведомления об изменении.
- Любой объект синхронизации может находиться в так называемом **сигнальном состоянии**.
- Потоки могут проверять текущее состояние объекта и/или ждать изменения этого состояния и таким образом согласовывать свои действия. При этом гарантируется, что когда поток работает с объектами синхронизации (создает их, изменяет состояние) система не прервет его выполнения, пока он не завершит это действие.
- Таким образом, все конечные операции с объектами синхронизации являются атомарными (неделимыми).

Работа с объектами синхронизации

- Чтобы создать тот или иной объект синхронизации, производится вызов специальной функции WinAPI типа Create... (напр. CreateMutex). Этот вызов возвращает дескриптор объекта (HANDLE), который может использоваться всеми потоками, принадлежащими данному процессу. Есть возможность получить доступ к объекту синхронизации из другого процесса - либо унаследовав дескриптор этого объекта, либо, что предпочтительнее, воспользовавшись вызовом функции открытия объекта (Open...). После этого вызова процесс получит дескриптор, который в дальнейшем можно использовать для работы с объектом.
- Объекту, если только он не предназначен для использования внутри одного процесса, обязательно присваивается имя. Имена всех объектов должны быть различны (даже если они разного типа). Нельзя, например, создать событие и семафор с одним и тем же именем.

- По имеющемуся дескриптору объекта можно определить его текущее состояние. Это делается с помощью «ожидающих функций». Чаще всего используется функция WaitForSingleObject.
- Эта функция имеет два параметра, первый из которых - дескриптор объекта, второй - время ожидания в мсек. Функция возвращает:
- WAIT_OBJECT_0, если объект находится в сигнальном состоянии,
- WAIT_TIMEOUT - если истекло время ожидания,
- WAIT_ABANDONED, если объект-взаимоисключение не был освобожден до того, как владеющий им поток завершился.
- Если время ожидания указано равным нулю, функция возвращает результат немедленно, в противном случае она ждет в течение указанного промежутка времени.
- В случае, если состояние объекта станет сигнальным до истечения этого времени, функция вернет WAIT_OBJECT_0, в противном случае функция вернет WAIT_TIMEOUT.

- Если в качестве времени указана символическая константа INFINITE, то функция будет ждать неограниченно долго, пока состояние объекта не станет сигнальным.
- Если необходимо узнавать о состоянии сразу нескольких объектов, следует воспользоваться функцией WaitForMultipleObjects.
- Чтобы закончить работу с объектом и освободить дескриптор вызывается функция CloseHandle.
- Очень важен тот факт, что обращение к ожидающей функции блокирует текущий поток, т. е. пока поток находится в состоянии ожидания, ему не выделяется процессорного времени.

Конструкция	Назначение
Sleep	Блокировка на указанное время
Join	Ожидание окончания другого потока

Конструкция	Назначение	Доступна из других процессов?	Скорость
lock	Гарантирует, что только один поток может получить доступ к ресурсу или секции кода.	нет	быстро
Mutex	Гарантирует, что только один поток может получить доступ к ресурсу или секции кода. Может использоваться для предотвращения запуска нескольких экземпляров приложения.	да	средне
Semaphore	Гарантирует, что не более заданного числа потоков может получить доступ к ресурсу или секции кода.	да	средне

Конструкция	Назначение	Доступна из других процессов?	Скорость
EventWaitHandle	Позволяет потоку ожидать сигнала от другого потока.	да	средне
Wait and Pulse*	Позволяет потоку ожидать, пока не выполнится заданное условие блокировки.	нет	средне

Семафоры

- Семафоры – примитивы синхронизации более высокого уровня абстракции, чем признаки блокировки; предложены Дийкстрой (Dijkstra) в 1968 г. в качестве компонента операционной системы THE
- Семафор - это целочисленная неотрицательная переменная, для которой определены 2 операции: **P** и **V**
  - **P**(sem) (wait/down) ожидает выполнения условия sem > 0, затем уменьшает sem на 1 и возвращает управление
  - **V**(sem) (signal/up) увеличивает sem на 1
- Операции **P** и **V** выполняются **атомарно**

- С каждым семафором ассоциирована очередь потоков
  - при вызове потоком **P**(sem)
    - если семафор "свободен" (>0), его значение уменьшается на 1, и выполнение потока продолжается
    - если семафор "занят" (<=0), поток переводится в состояние ожидания и помещается в очередь, соответствующую данному семафору
    - запускается какой-либо другой готовый к выполнению поток
  - при вызове потоком **V**(sem)
    - если очередь потоков, ассоциированная с данным семафором, не пуста – один из них разблокируется и помещается в очередь готовых к выполнению
    - поток, вызвавший V (sem), продолжает свое выполнение
    - в противном случае (нет потоков, ожидающих освобождения семафора), значение семафора увеличивается
- Таким образом, все вызовы изменяют состояние семафора, то есть семафор сохраняет некоторую информацию о прошедших вызовах

Виды семафоров

- **Двоичный семафор**
  - S может принимать значения 0 и 1, инициализируется значением 1
  - обеспечивает эксклюзивный доступ к ресурсу (например, при работе в критической секции)
  - одновременно может выполняться только один поток
- **Использование**

```
Semaphore S = 1;
while( true ){
  P(S);
  Использование ресурса
  V(S);
}
```

Виды семафоров

- **Счетный семафор**
  - S инициализируется значением N (число доступных единиц ресурса)
  - представляет ресурсы, состоящие из нескольких однородных элементов
  - позволяет потокам исполняться, пока есть неиспользуемые элементы
- Использование

```
Semaphore S1 = N, S2 = 1;
while( true ){
  P(S1);
  P(S2);
  Выбор свободного ресурса
  V(S2);
  Использование ресурса
  P(S2);
  Освобождение ресурса
  V(S2);
  V(S1);
}
```

Взаимоисключения (мьютексы)

- Объекты-взаимоисключения (мьютексы, mutex - от MUTual EXclusion) позволяют координировать взаимное исключение доступа к разделяемому ресурсу. Сигнальное состояние объекта (т.е. состояние "установлен") соответствует моменту времени, когда объект не принадлежит ни одному потоку и его можно "захватить". Состояние "сброшен" (не сигнальное) соответствует моменту, когда какой-либо поток уже владеет этим объектом.
- Для того, чтобы объявить взаимоисключение принадлежащим текущему потоку, надо вызвать одну из ожидающих функций. Поток, которому принадлежит объект, может его "захватывать" повторно сколько угодно раз (это не приведет к самоблокировке), но столько же раз он должен будет его освобождать с помощью функции ReleaseMutex.

События (event)

- Объекты-события используются для уведомления ожидающих потоков о наступлении какого-либо события. Различают два вида событий - с ручным и автоматическим сбросом. Ручной сброс осуществляется функцией ResetEvent. События с ручным сбросом используются для уведомления сразу нескольких потоков.
- При использовании события с автосбросом уведомление получит и продолжит свое выполнение только один ожидающий поток, остальные будут ожидать дальше.
- Функция CreateEvent создает объект-событие,
- SetEvent - устанавливает событие в сигнальное состояние, ResetEvent-сбрасывает событие.
- Функция PulseEvent устанавливает событие, а после возобновления ожидающих это событие потоков (всех при ручном сбросе и только одного при автоматическом), сбрасывает его. Если ожидающих потоков нет, PulseEvent просто сбрасывает событие.

Семафоры

- Объект-семафор - это фактически объект-взаимоисключение со счетчиком. Данный объект позволяет "захватить" себя определенному количеству потоков. После этого "захват" будет невозможен, пока один из ранее "захвативших" семафор потоков не освободит его.
- Семафоры применяются для ограничения количества потоков, одновременно работающих с ресурсом. Объекту при инициализации передается максимальное число потоков, после каждого "захвата" счетчик семафора уменьшается. Сигнальному состоянию соответствует значение счетчика больше нуля. Когда счетчик равен нулю, семафор считается не установленным (сброшенным).

- Семафоры обычно используются для учета ресурсов (текущее число ресурсов задается переменной S) и создаются при помощи функции CreateSemaphore, в число параметров которой входят начальное и максимальное значение переменной. Текущее значение не может быть больше максимального и отрицательным. Значение S, равное нулю, означает, что семафор занят.

Реализация семафоров Win32

- В следующем примере по очереди запускаются десять потоков, выполняющих вызов Sleep. Semaphore гарантирует, что не более трех потоков могут вызвать Sleep одновременно:

```
class SemaphoreTest
{
  static Semaphore s = new Semaphore(3, 3);  // Available=3; Capacity=3
  static void Main()
  {
    for (int i = 0; i < 10; i++)
      new Thread(Go).Start();
  }
  static void Go()
  {
    while (true)
    {
      s.WaitOne();
      // Только 3 потока могут находиться здесь одновременно
      Thread.Sleep(100);
      s.Release();
    }
  }
}
```

Защищенный доступ к переменным

- Существует ряд функций, позволяющих работать с глобальными переменными процесса из всех потоков не заботясь о синхронизации, т.к. эти функции сами за ней следят. Это функции InterlockedIncrement/InterlockedDecrement, InterlockedExchange, InterlockedExchangeAdd и InterlockedCompareExchange. Например, функция InterlockedIncrement увеличивает значение 32-битной переменной на единицу - удобно использовать для различных счетчиков.

Сравнительные характеристики объектов синхронизации Windows

	CRITICAL_ SECTION	Мьютекс	Семафор	Событие
Именованный защищаемый объект синхронизации	Нет	Да	Да	Да
Доступность из нескольких процессов	Нет	Да	Да	Да
Синхронизация	Вхождение	Ожидание	Ожидание	Ожидание
Освобождение	Выход	Мьютекс может быть освобожден или оставлен без контроля.	Освобождается любым потоком.	Функции SetEvent, PulseEvent.

Права владения	В каждый момент времени иметь права владельца может только один поток. Владеющий поток может осуществлять вхождение несколько раз, не блокируя свое выполнение.	В каждый момент времени иметь права владельца может только один поток. Владеющий поток может выполнять функцию ожидания несколько раз, не блокируя свое выполнение.	Понятие владения неприменимо. Доступ разрешен одновременно нескольким потокам, число которых ограничено максимальным значением счетчика.	Понятие владения неприменимо. Функции SetEvent и PulseEvent могут быть вызваны любым потоком.
Результат освобождения	Разрешается вхождение одного потока из числа ожидающих.	Вслед за последним освобождением права владения разрешается приобрести одному потоку из числа ожидающих.	Продолжать выполнение могут несколько потоков, число которых определяется текущим значением счетчика.	После вызова функций SetEvent или PulseEvent продолжать выполнение будет один или несколько ожидающих потоков.

Проблемы семафоров

- В Windows существуют важные ограничения, касающиеся реализации семафоров. Например, каким образом поток может потребовать, чтобы счетчик семафора уменьшился на 2? Для этого поток мог бы организовать ожидание два раза подряд, но эта операция не была бы атомарной, поскольку в промежутке между двумя вызовами функции ожидания данный поток может быть вытеснен. В результате этого может наступить тупик - взаимоблокировка (deadlock) потоков.

```
/* hsem – дескриптор семафора. Максимальное значение счетчика
     семафора равно 2. */
…
/* Уменьшить значение счетчика семафора на 2. */
WaitForSingleObject(hSem, INFINITE);
WaitForSingleObject(hSem, INFINITE);
…
/* Увеличить значение счетчика семафора на 2. */
ReleaseSemaphore(hSem, 2, &PrevCount);
```

Предположим, что максимальное и начальное значения счетчика устанавливаются равными 2 и что первый из двух потоков завершает первый цикл ожидания, а затем вытесняется. Далее второй поток может завершить первый цикл ожидания и уменьшить значение счетчика до 0. Оба потока окажутся блокированными на неопределенное время, поскольку ни один из них не сможет выполнить второй цикл ожидания.

- Один из возможных вариантов правильного решения заключается в том, чтобы защитить циклы ожидания при помощи мьютекса или объекта CRITICAL_SECTION, как показано в приведенном ниже фрагменте программного кода:

```
/* Уменьшаем значение счетчика семафора на 2. */
EnterCriticalSection(&csSem);
WaitForSingleObject(hSem, INFINITE);
WaitForSingleObject(hSem, INFINITE);
LeaveCriticalSection (&csSem);
…
ReleaseSemaphore(hSem, 2, &PrevCount);
```

- Но и эта реализация, в таком общем виде, страдает ограничениями. Предположим, например, что в счетчике семафора остается две единицы, и потоку A необходимы три единицы, а потоку B — только две. Если первой начнет выполняться поток A, то он выполнит два цикла ожидания и блокируется на третьем, продолжая владеть мьютексом. При этом поток B, которому были необходимы только две единицы, по-прежнему будет оставаться блокированным.

Влияние синхронизации на производительность

- Использование синхронизации в программах может и будет ухудшать их производительность, и в этом отношении следует быть особенно осмотрительным в случае SMP-систем.
- Симметричное мультипроцессирование (англ. Symmetric Multiprocessing, сокращённо SMP) — архитектура многопроцессорных компьютеров, в которой два или более одинаковых процессоров подключаются к общей памяти. Большинство многопроцессорных систем сегодня используют архитектуру SMP.

- Главный поток создает семафор с небольшим, например 4, максимальным значением параметра, представляющего максимально допустимое количество активных потоков, которое, например, можно принимать равным количеству процессоров, установленных в системе, для обеспечения приемлемой производительности. Начальное значение счетчика семафора также следует установить равным максимальному значению. Это число можно сделать параметром и подбирать его оптимальное значение экспериментальным путем. Другим возможным значением этого параметра может служить количество процессоров, которое может быть определено во время выполнения программы.

- Главный поток может регулировать, или, как говорят, "дросселировать" (throttle) выполнение рабочих потоков и динамически настраивать работу приложения, ожидая, пока не уменьшится значение счетчика, если он решает, что уровень состязательности слишком высок, или увеличивая значение счетчика с помощью функции ReleaseSemaphore, чтобы дать возможность выполняться большему количеству потоков.

- В приведенном ниже фрагменте кода представлен видоизмененный рабочий цикл, выполняющий две операции с семафором.

```
while (TRUE) { // Рабочий цикл
 WaitForSingleObject(hThrottleSem, INFINITE);
 WaitForSingleObject(hMutex, INFINITE);
 … Критический участок кода …
 ReleaseMutex(hMutex);
 ReleaseSemaphore(hThrottleSem, 1, NULL);
} // Конец рабочего цикла
```

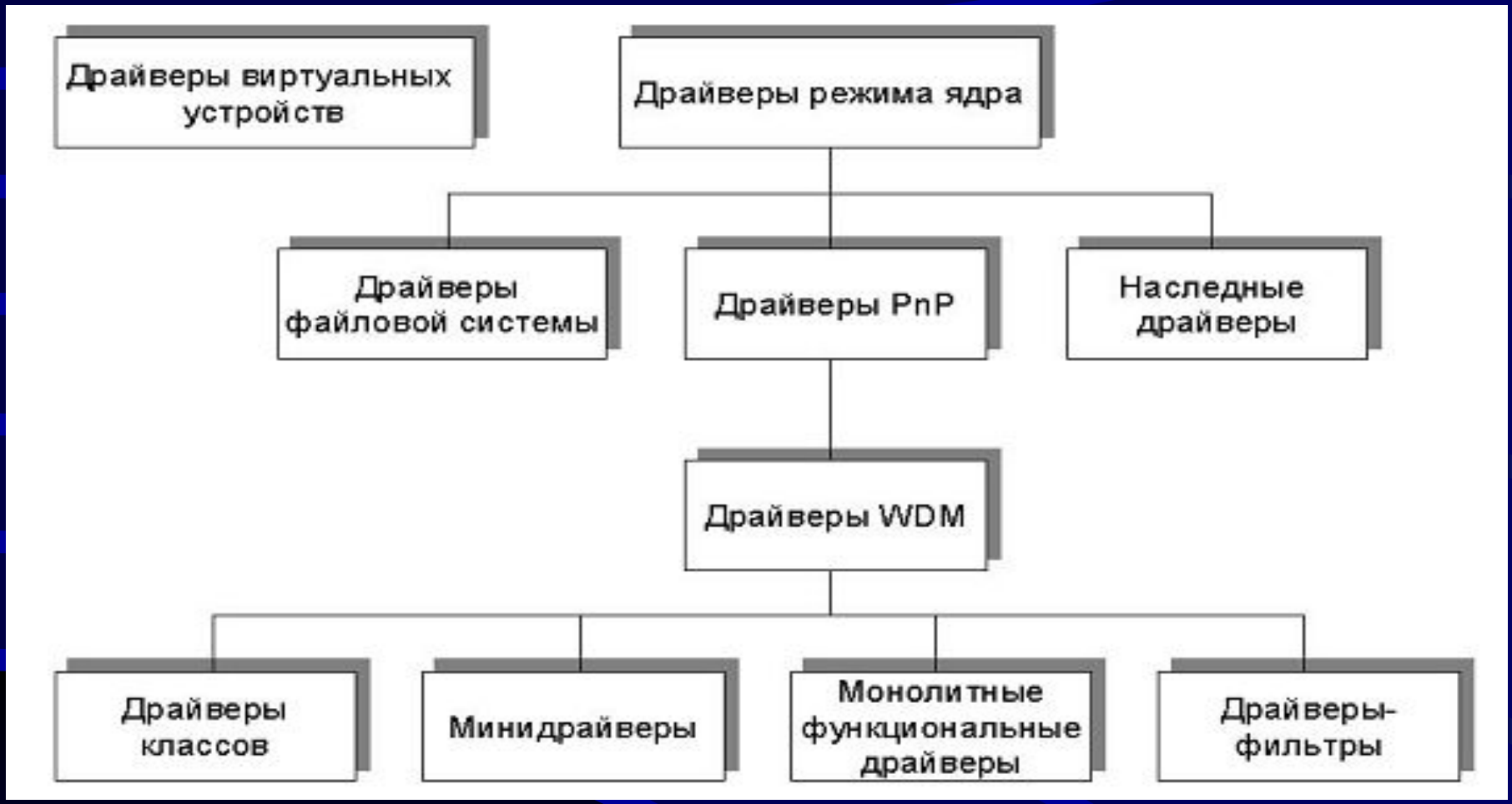
Повышение при ожидании ресурсов исполняющей системы

- Когда поток пытается получить ресурс исполняющей системы (ERESOURCE), который уже находится в исключительном владении другого потока, он должен войти в состояние ожидания до тех пор, пока другой поток не освободит ресурс. Для ограничения риска взаимных исключений исполняющая система выполняет это ожидание, не входя в бесконечное ожидание ресурса, а интервалами по пять секунд.
- Если по окончании этих пяти секунд ресурс все еще находится во владении, исполняющая система пытается предотвратить зависание центрального процессора путем завладения блокировкой диспетчера, повышения приоритета потока или потоков, владеющих ресурсом, до значения 14 (если исходный приоритет владельца был меньше, чем у ожидающего, и не был равен 14), перезапуска их квантов и выполнения еще одного ожидания.
- Поскольку ресурсы исполняющей системы могут быть либо общими, либо эксклюзивными, ядро сначала повышает приоритет эксклюзивного владельца, а затем проводит проверку общих владельцев и повышает приоритет всех этих владельцев. Когда ожидающий поток опять входит в состояние ожидания, появляется надежда, что планировщик так спланирует работу одного из потоков-владельцев, чтобы у него было достаточно времени для завершения своей

Windows 2000 поддерживает два базовых типа драйверов:

1. Драйверы пользовательского режима (User-Mode Drivers):
   - Драйверы виртуальных устройств (Virtual Device Drivers, VDD) - используются для поддержки программ MS-DOS;
   - Драйверы принтеров (Printer Drivers).
2. Драйверы режима ядра (Kernel-Mode Drivers):
   - Драйверы файловой системы (File System Drivers) - реализуют ввод-вывод на локальные и сетевые диски;
   - Унаследованные драйверы (Legacy Drivers) - написаны для предыдущих версий Windows NT;
   - Драйверы видеоадаптеров (Video Drivers) - реализуют графические операции;
   - Драйверы потоковых устройств (Streaming Drivers) - реализуют ввод-вывод видео и звука;
   - WDM-драйверы (Windows Driver Model, WDM) - поддерживают технологию Plug and Play и управления электропитанием.

Типы драйверов



Драйверы выполняются в режиме ядра в одном из трех контекстов:

- в контексте пользовательского потока инициировавшего запрос ввода-вывода;
- в контексте системного потока режима ядра (эти потоки принадлежат процессу System);
- как результат прерывания (а значит, не в контексте какого-либо процесса или потока, который был текущим на момент прерывания).

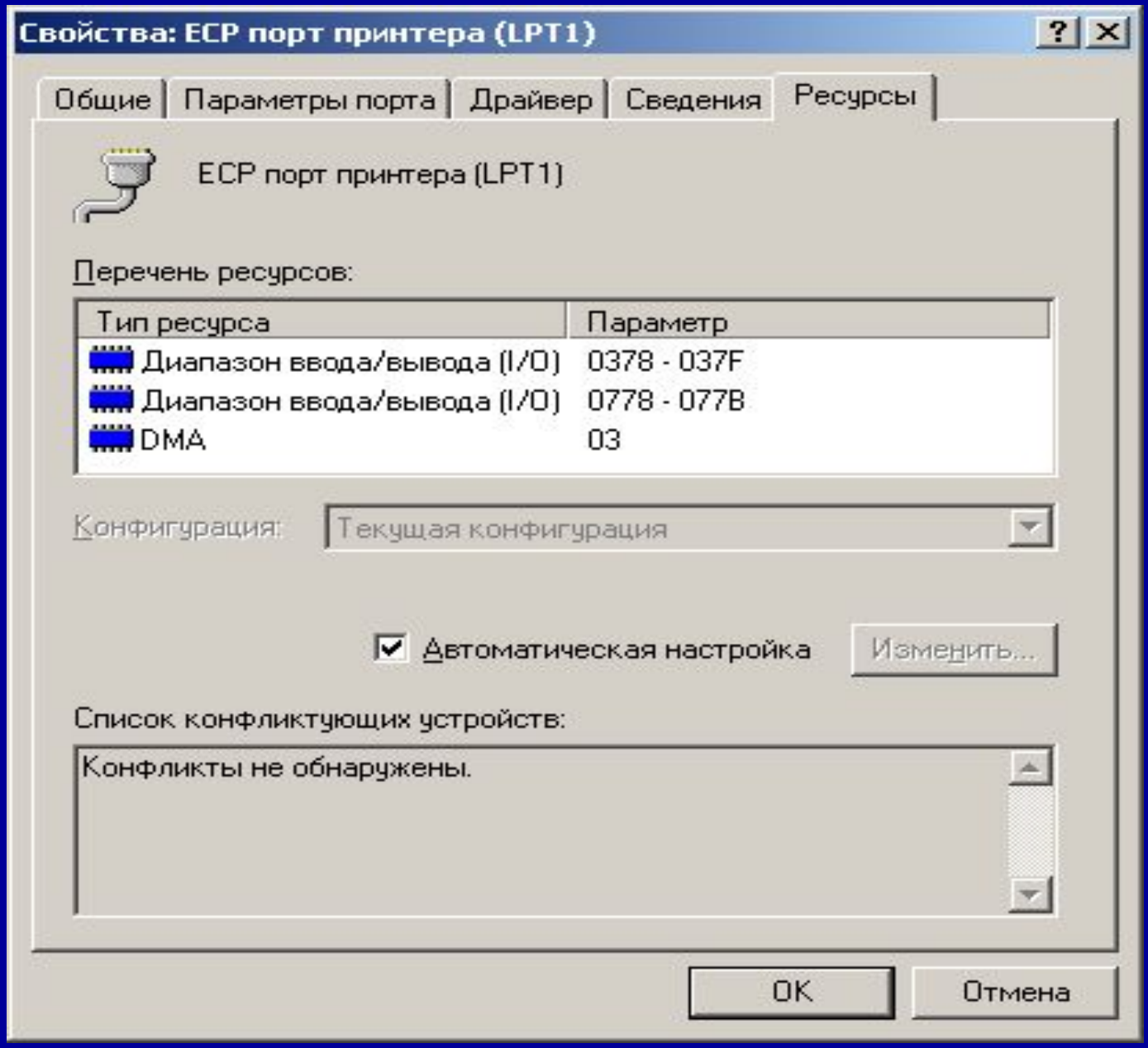
Одно- и многоуровневые драйверы

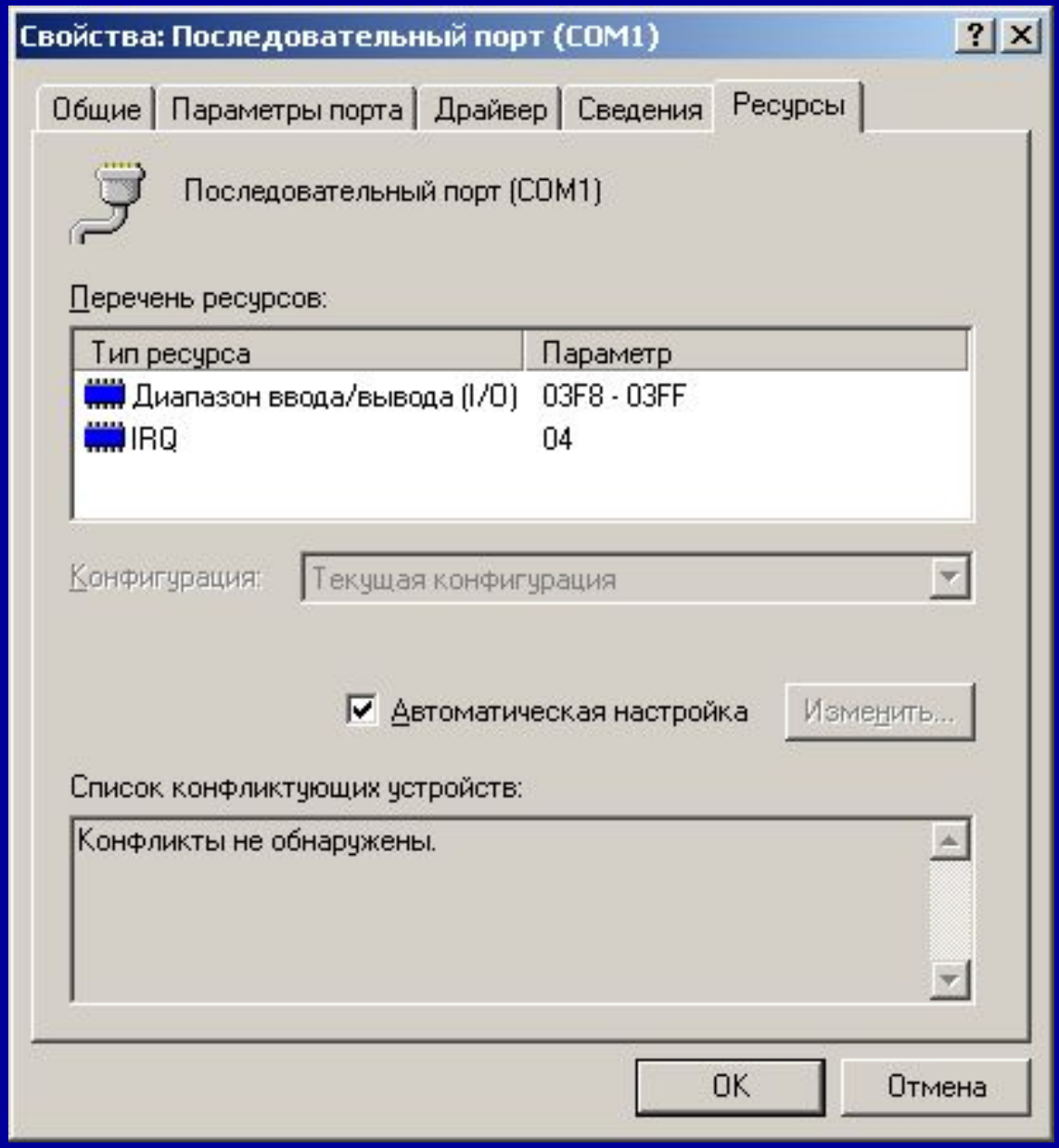
- **одноуровневые** (monolithic drivers)
- Но большинство драйверов, управляющих физическими устройствами являются **многоуровневыми** (layered drivers). Обработка запроса ввода-вывода разделяется между несколькими драйверами. Каждый выполняет свою часть работы. Например, запрос на чтение файла передается драйверу файловой системы, который, выполнив некоторые операции (например, разбиение запроса на несколько частей), передает его "ниже" - драйверу диска, а тот, в свою очередь, отправляет запрос драйверу шины. Кроме того между этими драйверами можно добавить любое количество драйверов-фильтров (например, шифрующих данные). Выполнив запрос нижестоящий драйвер (lower-level driver) передает его результаты "наверх" - вышестоящему (higher-level driver).

- По своей структуре драйвер устройства является файлом PE-формата (Portable Executable, PE). Таким же как обычные exe и dll. Драйверы можно рассматривать как DLL режима ядра, предназначенные для выполнения задач, не решаемых из пользовательского режима. Принципиальная разница (не считая уровня привилегий) в том, что нельзя напрямую обращаться к драйверу, ни к его коду, ни к его данным. Менеджер ввода-вывода (Input/Output Manager) обеспечивает среду для функционирования драйверов, а также предоставляет механизмы для их загрузки, выгрузки и управления ими.

При установке устройства менеджер ввода-вывода назначает ему уникальный набор системных ресурсов.
Это могут быть:

- Уровни запросов на прерывания (IRQ);
- Каналы прямого доступа к памяти DMA;
- Адреса портов ввода/вывода I/O;
- Диапазоны адресов памяти.

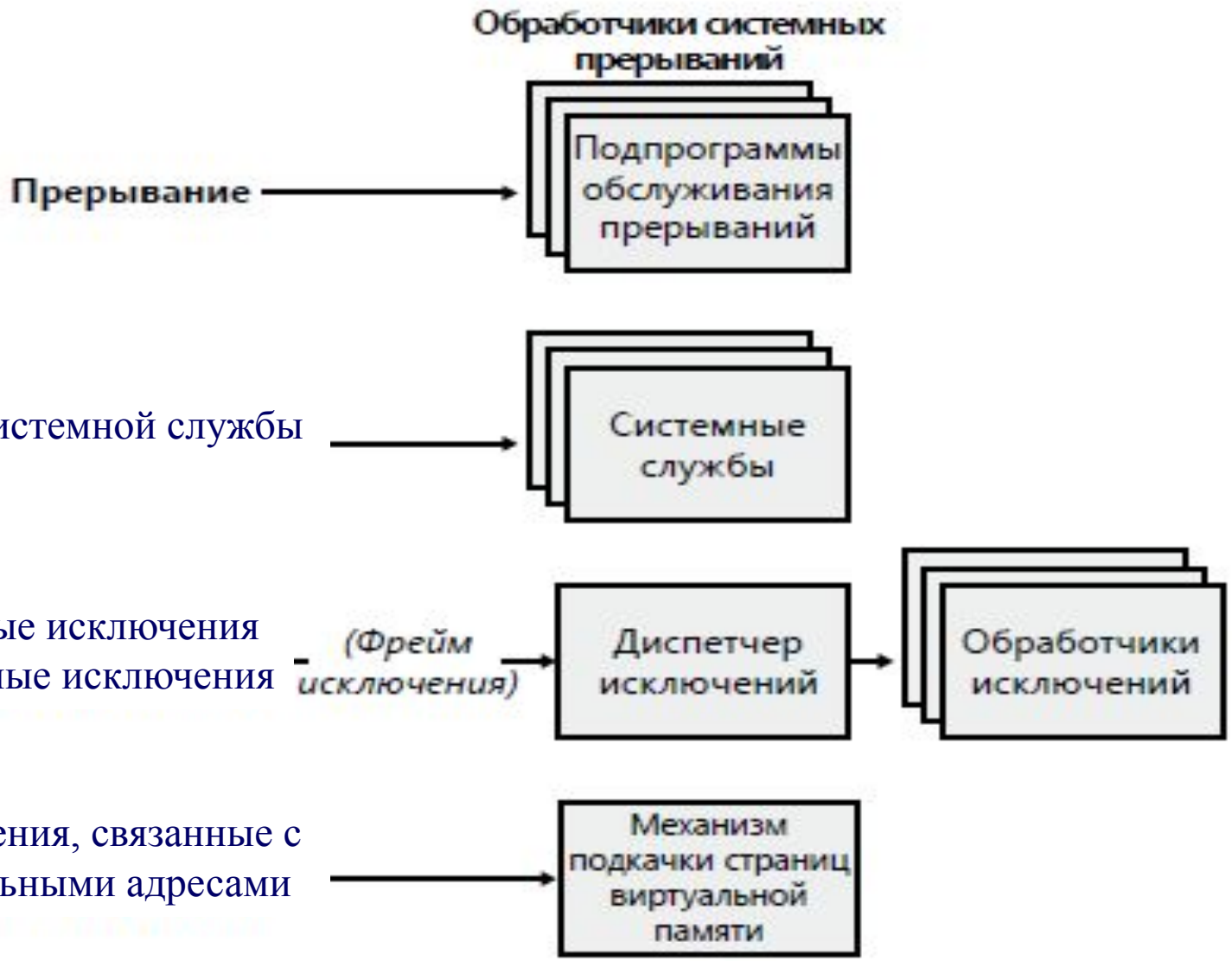

Свойства: ECP порт принтера (LPT1)
Общие
Параметры порта
Драйвер
Сведения
Ресурсы
ECP порт принтера (LPT1)
Перечень ресурсов:
Тип ресурса
Параметр
Диапазон ввода/вывода (I/O)
0378 - 037F
Диапазон ввода/вывода (I/O)
0778 - 077B
DMA
03
Конфигурация:
Текущая конфигурация
Автоматическая настройка
Изменить...
Список конфликтующих устройств:
Конфликты не обнаружены.
OK
Отмена

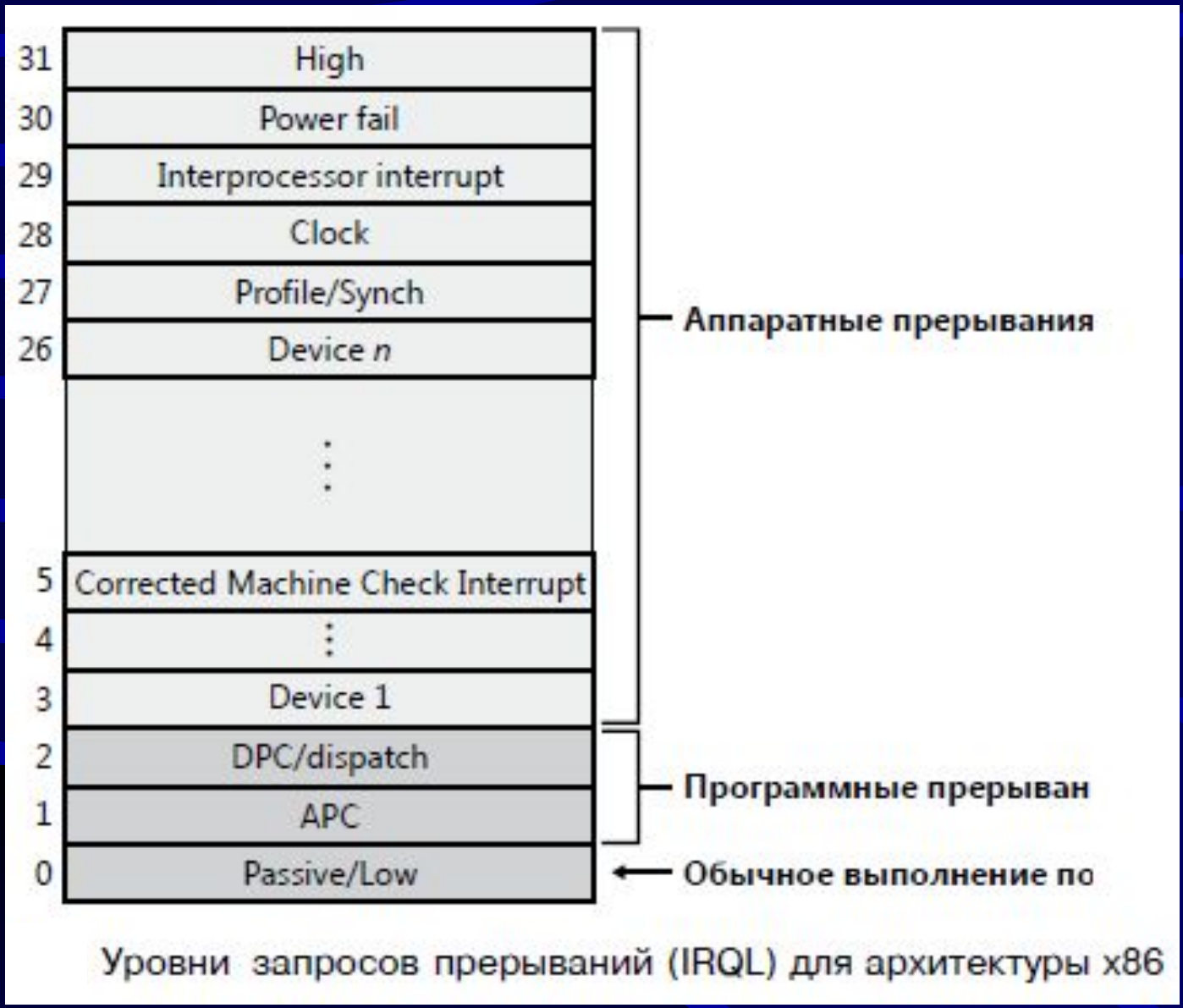

Свойства: Последовательный порт (COM1)
Общие
Параметры порта
Драйвер
Сведения
Ресурсы
Последовательный порт (COM1)
Перечень ресурсов:
Тип ресурса
Параметр
Диапазон ввода/вывода (I/O)
03F8 - 03FF
IRQ
04
Конфигурация:
Текущая конфигурация
Автоматическая настройка
Изменить...
Список конфликтующих устройств:
Конфликты не обнаружены.
OK
Отмена

Диспетчеризация прерываний

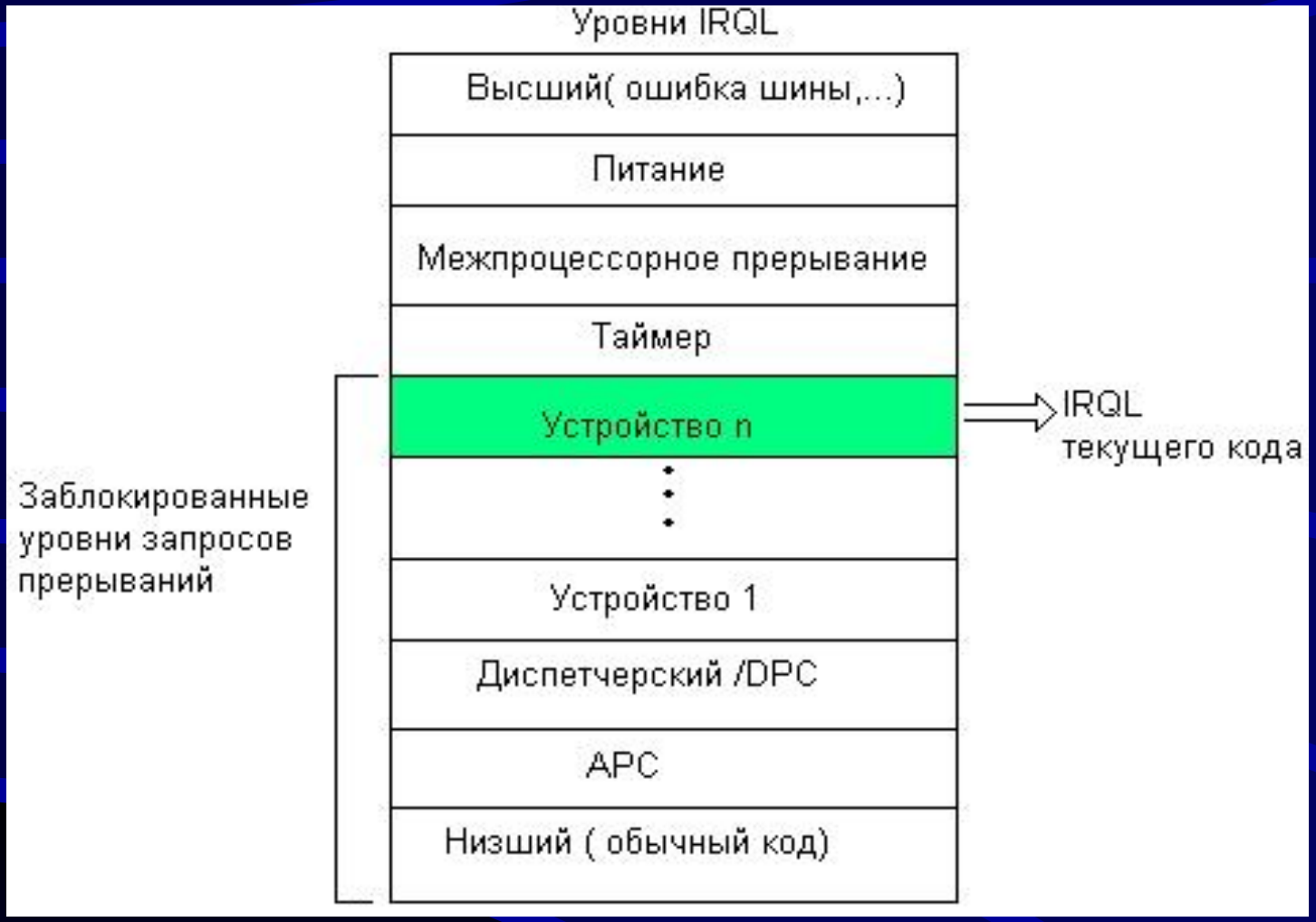
Диспетчер прерываний Windows NT (так называемый Trap Handler) работает с программной моделью прерываний, единой для всех аппаратных платформ, поддерживаемых Windows NT.

- Все источники прерываний (аппаратных и программных, а также некоторых важных для системы исключений, например исключения по ошибке шины) делятся на несколько классов, и каждому классу присваивается уровень запроса прерывания — Interrupt Request Level, IRQL. Этот уровень и представляет приоритет данного класса.
- ОС программным способом поддерживает внутреннюю переменную, называемую IRQL выполняемого процессором кода, которая по назначению соответствует уровню прерывания процессора. Если процессор, на котором работает ОС, поддерживает такую переменную, то она используется и IRQL выполняемого кода отображается на нее, в противном случае соответствующие функции процессора эмулируются программно.


Обработчики системных прерываний
Прерывание
Подпрограммы обслуживания прерываний
Вызов системной службы
Системные службы
Аппаратные исключения
Программные исключения
(Фрейм исключения)
Диспетчер исключений
Обработчики исключений
Исключения, связанные с виртуальными адресами
Механизм подкачки страниц виртуальной памяти



Уровни запросов прерываний (IRQL) для архитектуры x86

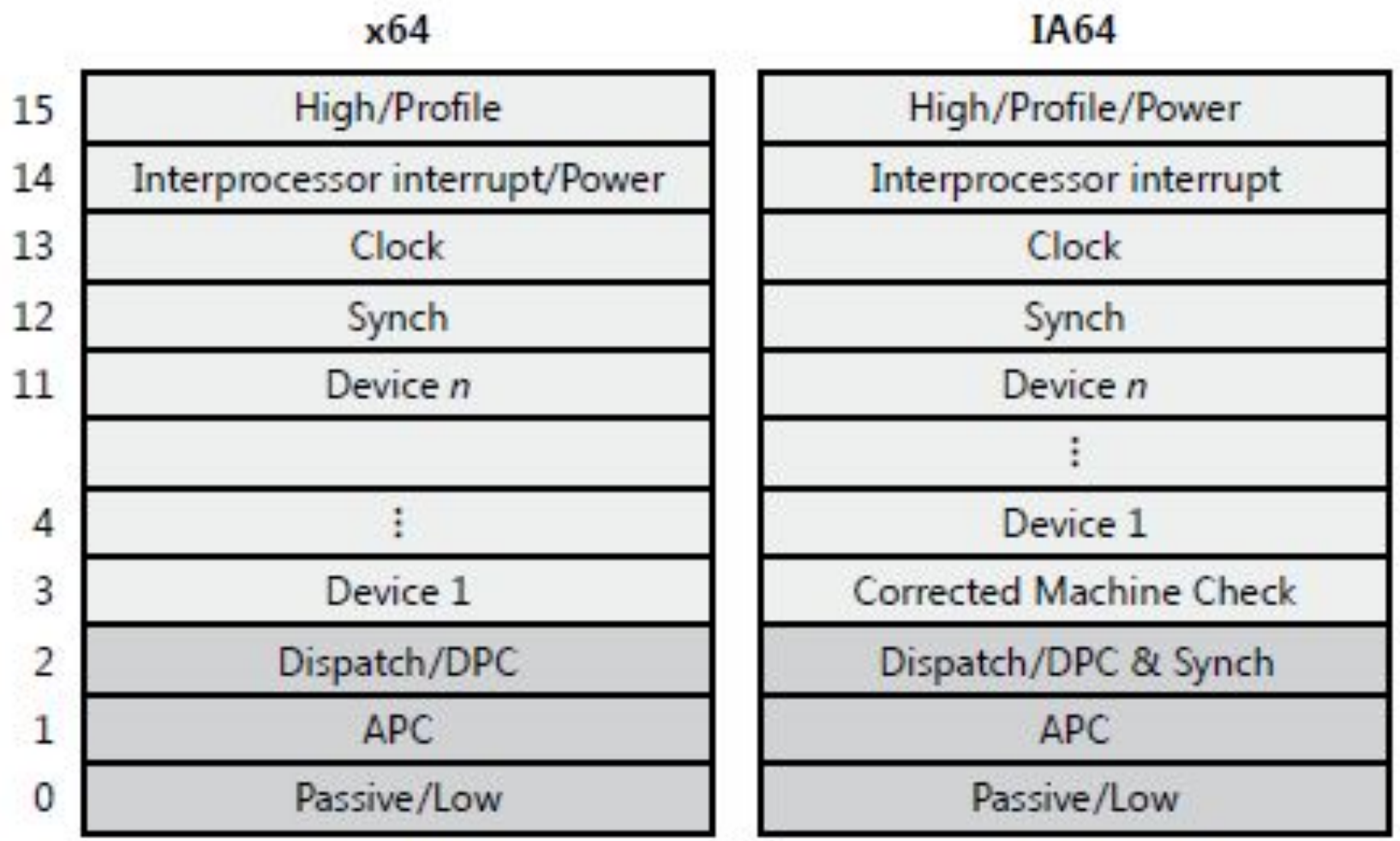


Второе название диспетчерского уровня, DPC, аббревиатура от Deffered Procedure Call (вызов отложенной процедуры)

Уровни запросов прерываний

- Всего существует 32 уровня, с 0 (passive), имеющего самый низкий приоритет, по 31 (high), имеющего соответственно самый высокий. Причем, прерывания с IRQL=0 (passive) по IRQL=2 (DPC\dispatch) являются программными, а прерывания с IRQL=3 (device 1) по IRQL=31 (high) являются аппаратными.
- Прерывание с уровнем IRQL=0, строго говоря, прерыванием не является, т.к. оно не может прервать работу никакого кода (ведь для этого этот код должен выполняться на еще более низком уровне прерывания, а такого уровня нет). На этом IRQL выполняются потоки пользовательского режима.

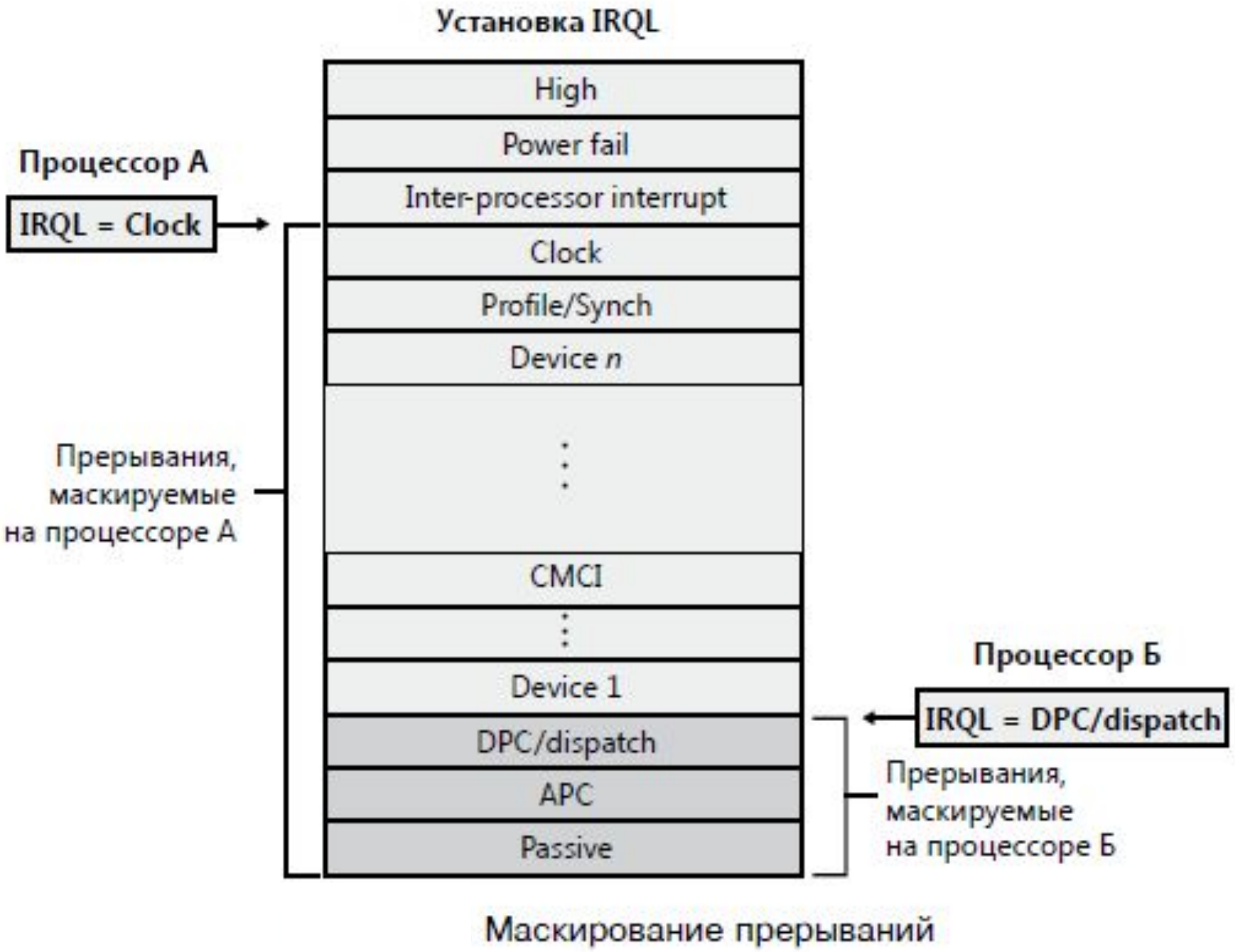
Не путайте уровни приоритета прерываний с уровнями приоритетов потоков



Уровни запросов прерываний (IRQL) для архитектур x64 и IA64

Хотя контроллеры прерываний устанавливают приоритетность прерываний, Windows устанавливает свою собственную схему приоритетности прерываний, известную как *уровни запросов прерываний* (IRQL). В ядре IRQL-уровни представлены в виде номеров от 0 до 31 на системах x86 и в виде номеров от 0 до 15 на системах x64 и IA64, где более высоким номерам соответствуют прерывания с более высоким приоритетом. Хотя ядро определяет для программных прерываний стандартный набор IRQL-уровней, номера аппаратных прерываний на IRQL-уровни отображает HAL.

Установка IRQL каждого процессора определяет, какие прерывания данный процессор может получать. Прерывания, поступающие от источника с IRQL, превышающим текущий уровень, прерывают работу процессора, а прерывания, поступающие от источников с IRQL-уровнями равными или ниже текущего уровня, *маскируются* до тех пор, пока выполняющийся поток не понизит IRQL.



Маскирование прерываний

Общая схема планирования обработки прерываний выглядит в Windows следующим образом.

1. При поступлении в процессор сигнала запроса на прерывание/исключение вызывается диспетчер прерываний, который запоминает информацию об источнике прерывания и анализирует его приоритет.
2. Если приоритет запроса ниже или равен IRQL прерванного кода, то обслуживание этого запроса откладывается и данные о запросе помещаются в соответствующую очередь запросов, после чего происходит быстрый возврат к прерванному обработчику прерываний.
3. По завершении обработки высокоприоритетного прерывания управление возвращается диспетчеру прерываний, который просматривает очереди отложенных прерываний и выбирает из них наиболее приоритетное. При этом уровень IRQL снижается до уровня выбранного прерывания.

4. Если же запрос имеет более высокий приоритет, чем IRQL текущего кода, то текущий обработчик прерываний вытесняется и ставится в очередь, соответствующую его значению IRQL, а управление передается новому обработчику в соответствии с IRQL запроса.
5. Уровень IRQL процессора делается равным уровню IRQL принятого на выполнение запроса.

Таким образом, запрос на прерывание принимается диспетчером прерываний **всегда** независимо от текущего уровня IRQL выполняемого кода, но диспетчер прерываний не передает его на обработку соответствующей процедуре обработки прерываний, а помещает в программную очередь запросов, если в данный момент выполняется более приоритетная процедура обработки прерываний. Операционная система имеет полный контроль над ситуацией, не позволяя контроллерам устройств ввода-вывода принимать решения о ходе вычислительного процесса.

IRQ	Устройство	Приор.	Комментарии
0	Системный таймер	* 15	Системное прерывание. Генерируется 91 раз за 5 сек. В данном качестве применяется со времени первого PC.
1	Клавиатура	* 14	Системное прерывание, генерируемое контроллером клавиатуры.
2	Контроллер прерываний	* 13	Каскадировано (связано) с IRQ9. Могут возникнуть конфликты, когда одновременно на IRQ2 и IRQ9 должны работать различные устройства. Его использование системой сохраняется для совместимости.
3	COM 2	4	Используется вторым коммуникационным адаптером (UART2). Какое же устройство будет его генерировать? Это может быть второй последовательный порт COM2 (интегрирован на материнской плате), внутренний модем, настроенный на COM2 или COM4, или инфракрасный адаптер. Можно отключить UART2, но присвоить IRQ3 ничему не удастся. Делит одно и то же IRQ3 с COM4 (при наличии последнего). Возможен конфликт при одновременном использовании.
4	COM 1	3	Используется первым коммуникационным адаптером. Все практически идентично: генерируется первым последовательным портом COM1, модемом на COM1 или COM3 (за исключением инфрапорта). Делит одно и то же IRQ4 с COM3 (при наличии последнего). В системах с подключенной к COM1 мышью использовать COM3 не следует.
5	свободен	2	Прерывание изначально предназначалось для использования вторым параллельным портом LPT2. Практического применения такое решение не нашло, поэтому IRQ5 перешло в разряд свободных. В IBM XT на IRQ5 "висел" жесткий диск. Через некоторое время "Creative Labs", создавая звуковую карту "Sound Blaster Pro", нашла применение прерыванию. С тех пор IRQ5 стало излюбленным для большинства звуковых ISA-карт. Звуковые PCI-карты также иногда используют это прерывание для эмуляции "SB Pro". IRQ5 можно привязать к слоту PCI.
6	Контроллер FDD-дисковода	* 1	Прерывание используется контроллером флоппи-дисковода, начиная с первых ПК. Ныне его можно отключить, если перейти на совместимый с обычными дискетами накопитель LS-120 с интерфейсом EIDE. Однако прерывание все равно не может быть использовано: ISA-карты на работу с ним не рассчитаны, и к слоту PCI привязать его нельзя. Может быть использовано для привода на на магнитной ленте.

7	LPT 1	0	По умолчанию прерывание первого параллельного порта LPT1. При отключенном порте (если принтер отсутствует или рассчитан на USB) может использоваться различными устройствами: сетевыми, ISDN-картами. Это также "запасное" место для звуковых карт.
8	Часы реального времени (RTC)	* 12	Системное прерывание со времени первых IBM AT.
9	свободен	11	Каскадировано с IRQ2. В остальном может использоваться по усмотрению.
10	свободен	10	Может быть использовано по усмотрению. Устаревшие IDE-контроллеры на старых звуковых картах иногда используют это IRQ.
11	свободен	9	Может быть использовано по усмотрению, часто используется видеокартами. В современных ПК обычно резервируется для шины USB. При отключении последней в BIOS может быть задействовано иначе.
12	свободен или PS/2-мышь	8	Если используется мышь, это IRQ в большинстве случаев выдается лишь по разрешению BIOS. Почти всегда прерывание свободно.
13	Сопроцессор	* 7	Системное прерывание. Изначально применялось арифметическим сопроцессором, который в первых ПК являлся отдельной микросхемой. Ныне это прерывание зарезервировано для совместимости со старым ПО.
14	Первичный EIDE-контроллер	6	По разрешению BIOS может быть использовано в SCSI- системах.
15	Вторичный EIDE-контроллер	5	Также может использоваться SCSI-интерфейсом, хотя обычно на него "подгружают" дополнительные EIDE-диски. При отсутствии устройств может применяться в любых целях.

Примечания: 15 - наивысший приоритет,

* - эти системные компоненты жестко зафиксированы и их конфигурация не может быть изменена.