Операционная система Windows 2000

История создания

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. 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.).
6. 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.).
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.).

11. Полностью 32-разрядная операционная систем Windows NT 3.1 (1993 г.).
12. Значительно усовершенствованная Windows NT 4.0 (1996 г.).
13. Windows NT 5.0 - Windows 2000

История ОС Windows

- Предшественником ОС Windows является одноименная операционная оболочка, появившаяся как надстройка над ОС DOS.
- Наиболее популярной оболочкой стала Windows 3.11 for Workgroup, где были реализованы многозадачность, графический интерфейс, поддержка одноранговой сети.
- Полноценная операционная система MS Windows появилась в 1995 г., как однопользовательская 32-разрядная операционная система, поддерживающая вытесняющую многозадачность, работу в сети, использование длинных имен и ряд других новых и удобных функций.
- Развитием линии явились операционные системы Windows’98, Windows ME.

История ОС Windows

- 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.

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).

Особенности Windows 2000

- 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.

Структура системы Windows 2000

ОС Windows можно разделить на 2 части:

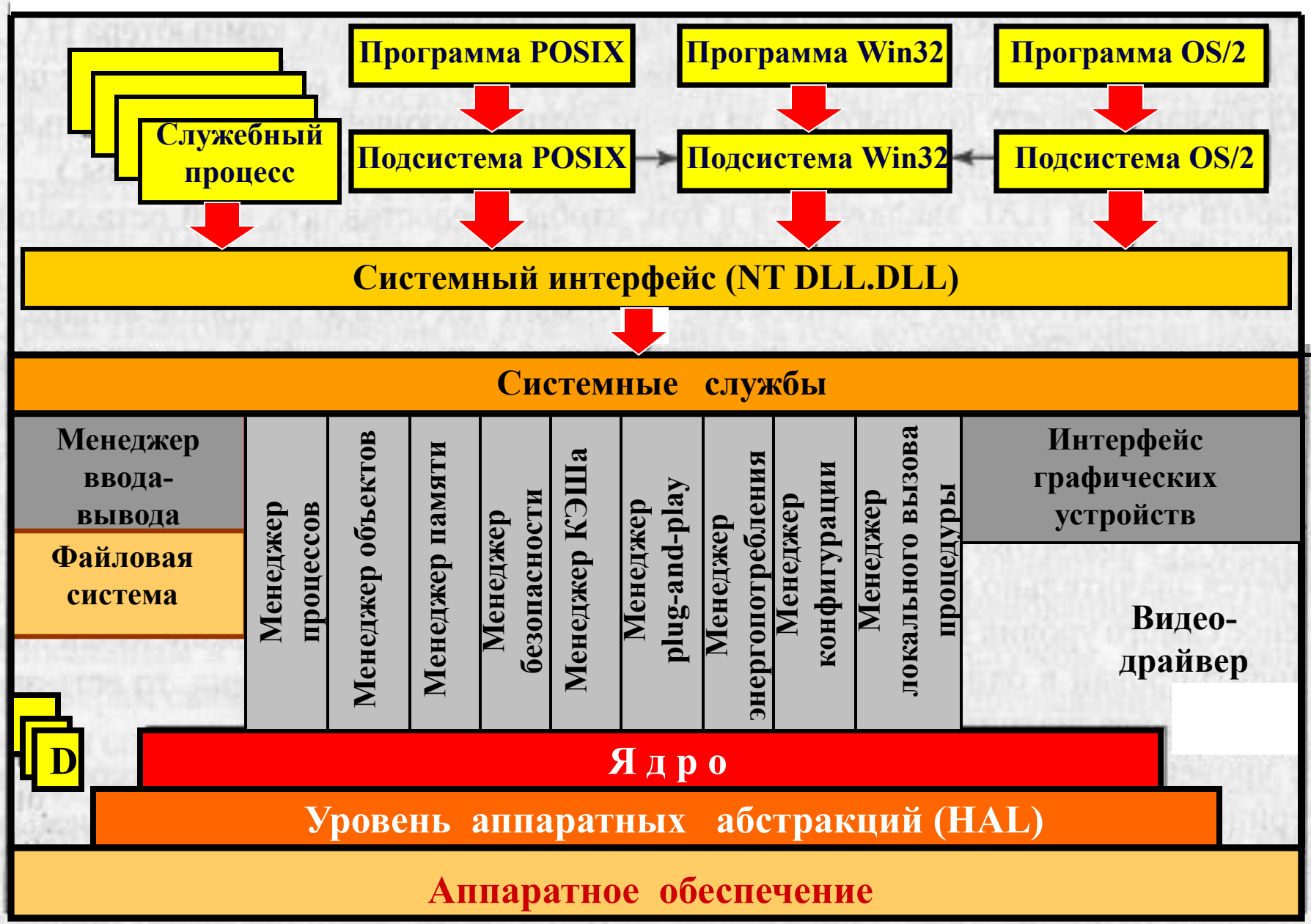
1. Основная часть ОС, работающая в режиме ядра (управление процессами, памятью, файловой системой , устройствами и т. д.).

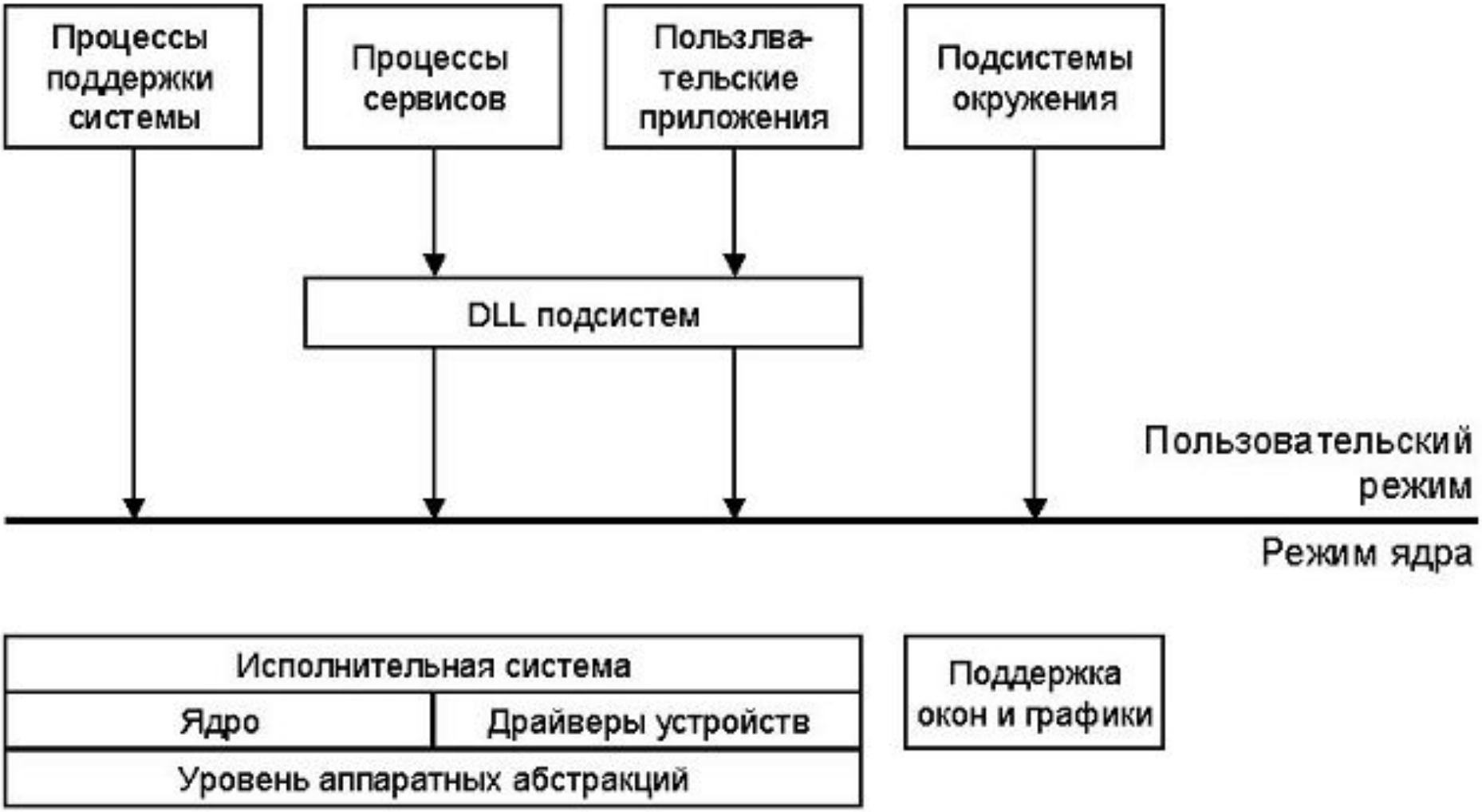
2. Подсистемы окружения (среды), работающие в режиме пользователя (процессы, помогающие пользователям выполнять определенные системные функции).

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

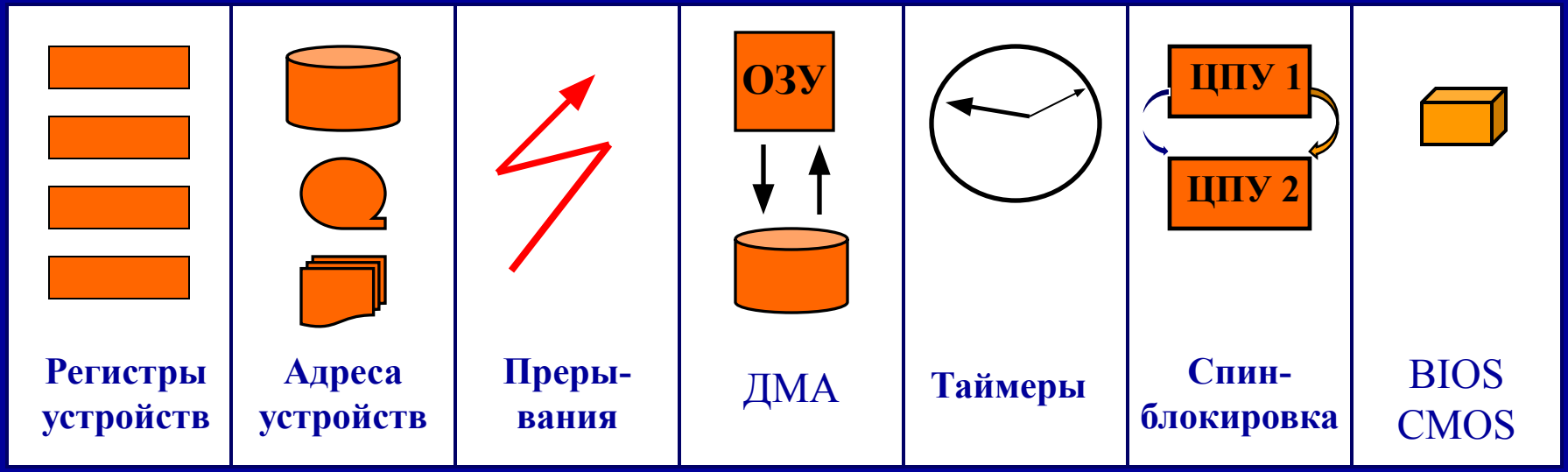

Процессы поддержки системы
Процессы сервисов
Пользлва-тельские приложения
Подсистемы окружения
DLL подсистем
Пользовательский режим
Режим ядра
Исполнительная система
Ядро
Драйверы устройств
Уровень аппаратных абстракций
Поддержка окон и графики

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.

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

Некоторые функции уровня HAL



- уровень аппаратных абстракций (Hardware Abstraction Layer, HAL) —
- набор низкоуровневых функций (около 92), 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 .;

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 2000, 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 2000 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 (1674 Кбайт).** Уровень **HAL**, представляющий собой библиотеку общего доступа, находится в файле **hal.dll (80 Кбайт).** Интерфейс Win32 и графических устройств хранятся в файле **win32.sys (1690 Кбайт)**. Системный интерфейс для связи пользовательского режима с режимом ядра – файл **NTDLL.DLL** содержит **484 Кбайт**.

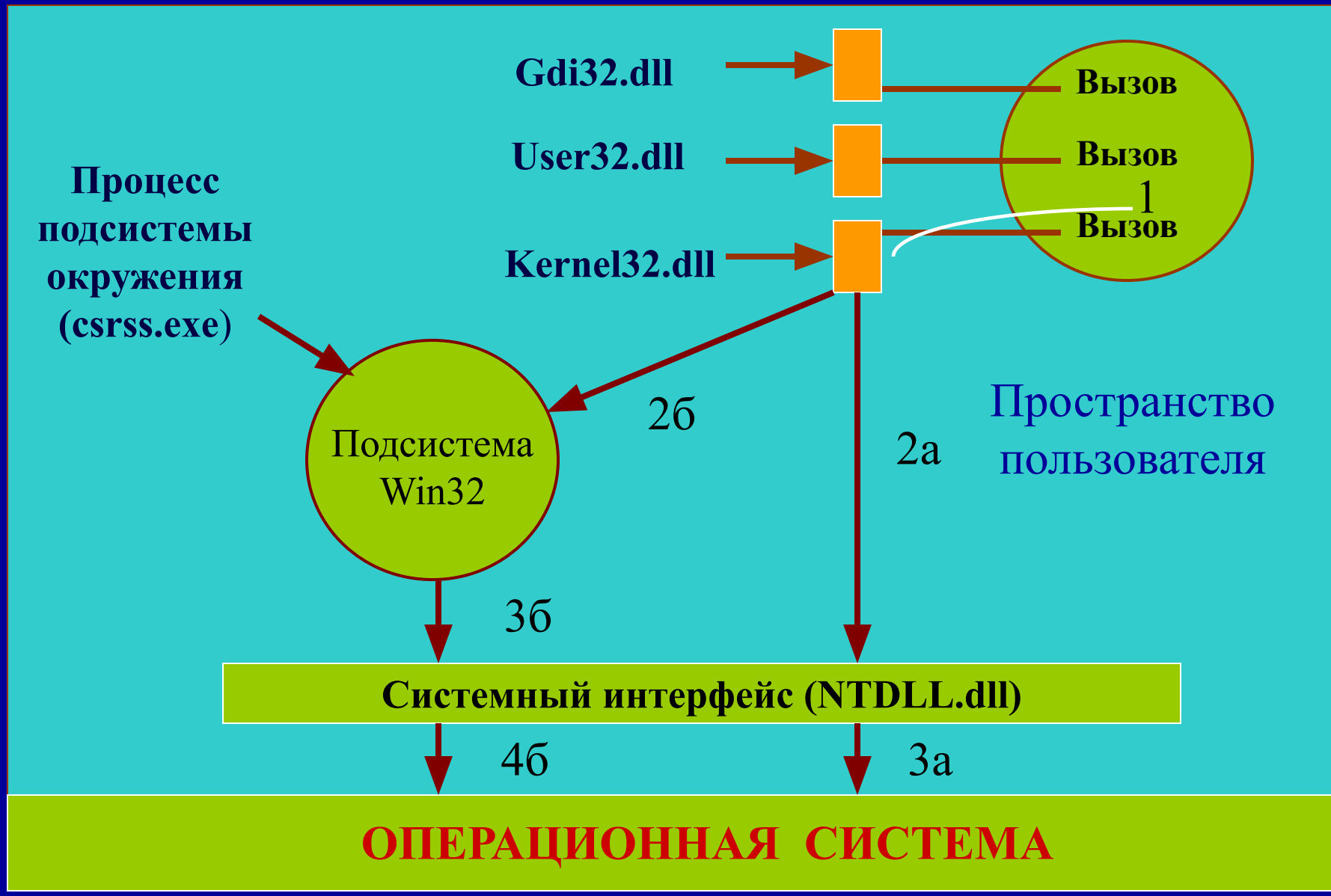
Подсистемы окружения

В Windows 2000 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 Wndows 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 800 отдельных файлов DLL (130 Мбайт при числе вызовов API более13000).


Gdi32.dll
User32.dll
Kernel32.dll
Вызов
Вызов
Вызов
1
Процесс подсистемы окружения (csrss.exe)
Подсистема Win32
2б
2а
Пространство пользователя
3б
Системный интерфейс (NTDLL.dll)
4б
3а
ОПЕРАЦИОННАЯ СИСТЕМА

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**

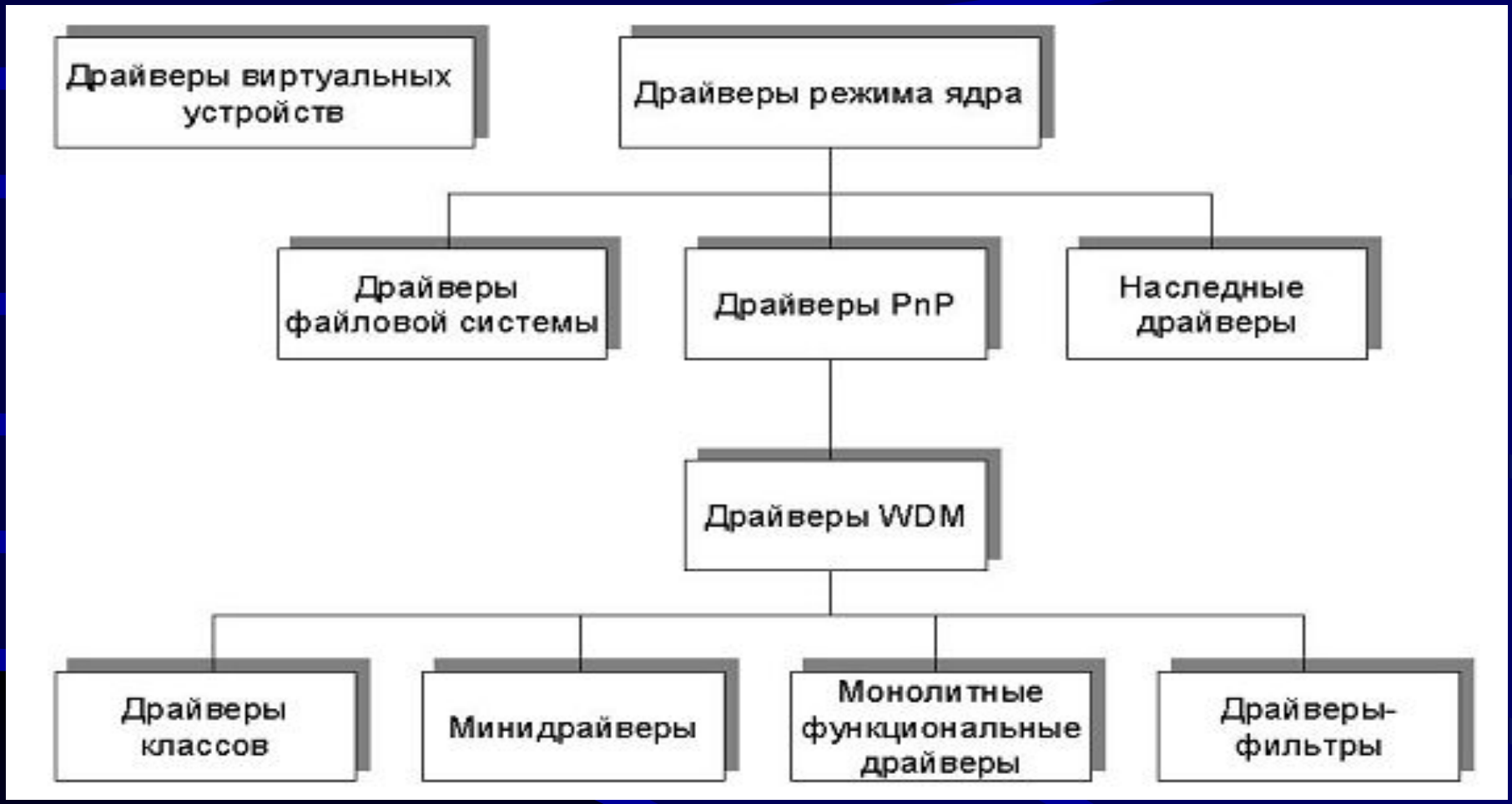
Запуск родительского процесса всех служебных процессов **services.exe.** По информации, хранящейся в реестре определяет, какие демоны в пространстве пользователя надо запустить (сервер принтера, файловый сервер, обработчик входящей электронной почты, факсов и т. д.)..

Выбор из реестра профиля пользователя и запуск требуемей оболочки.

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 Plag 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).

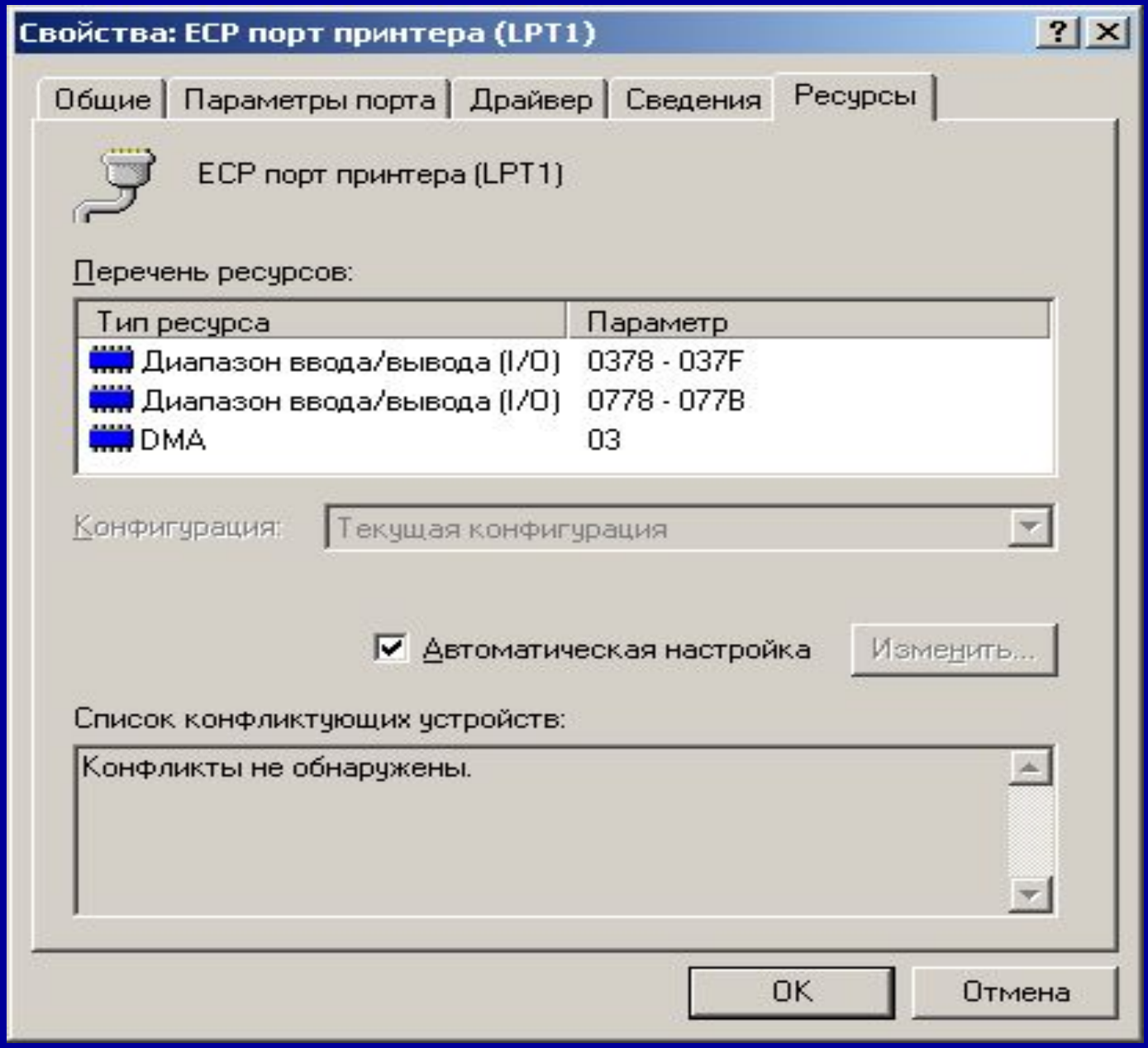
Одно- и многоуровневые драйверы

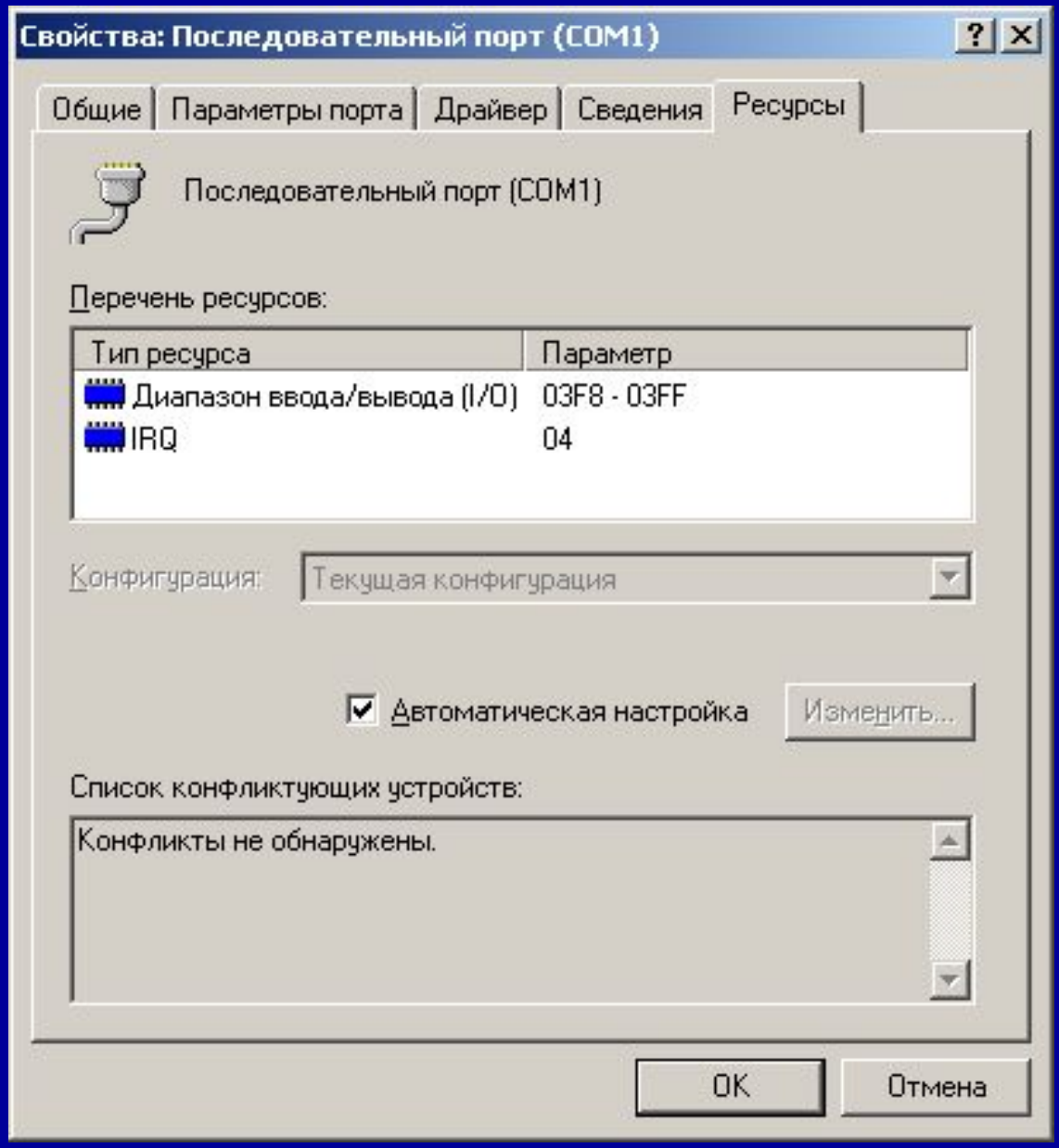
- Большинство драйверов управляющих физическими устройствами являются многоуровневыми (layered drivers). Обработка запроса ввода-вывода разделяется между несколькими драйверами. Каждый выполняет свою часть работы. Например, запрос на чтение файла передается драйверу файловой системы, который, выполнив 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).
- одноуровневые (monolithic drivers)

- 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, 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.

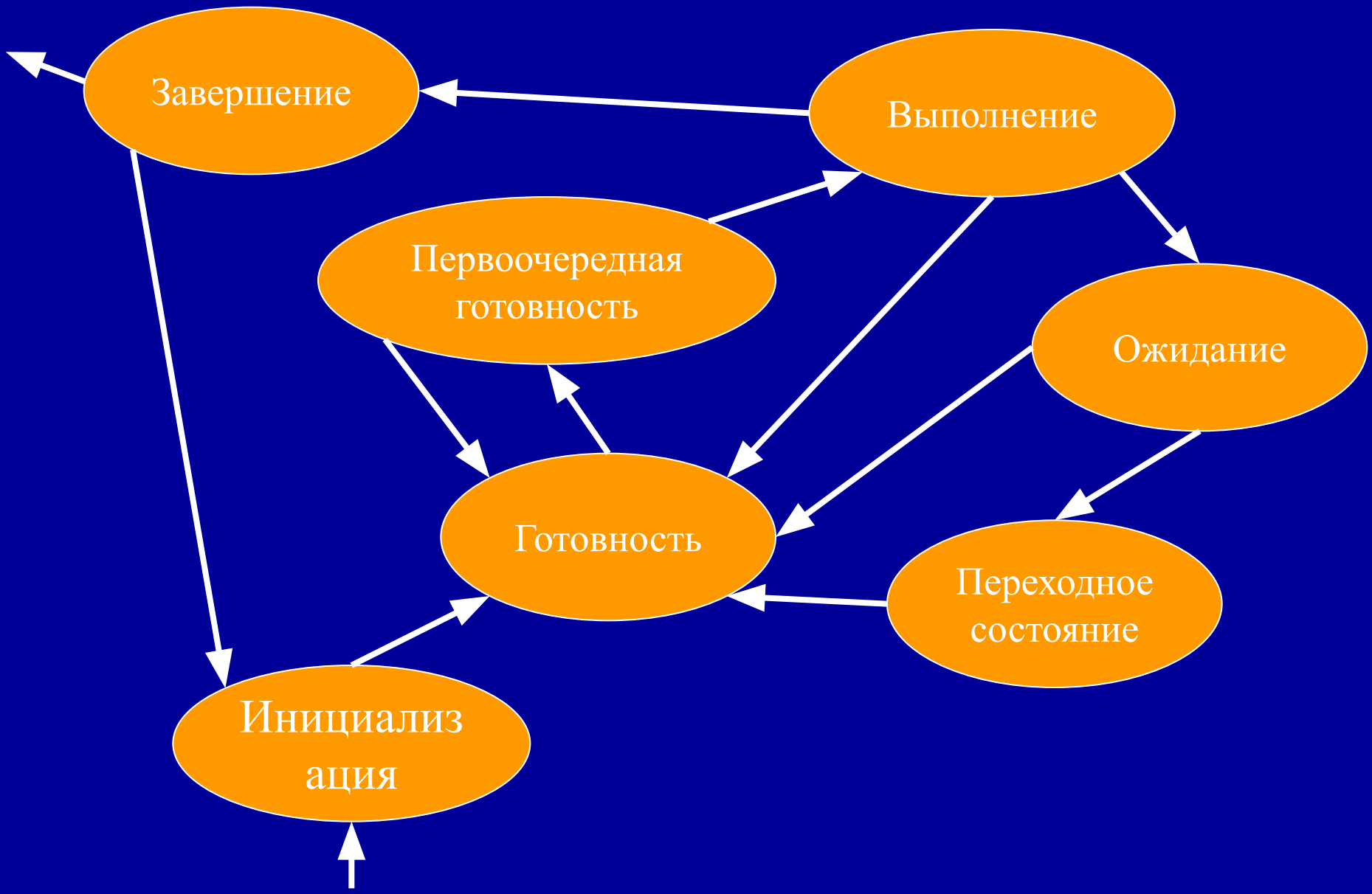
При установке устройства менеджер ввода-вывода назначает уникальный набор системных ресурсов.
Это могут быть:

- 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

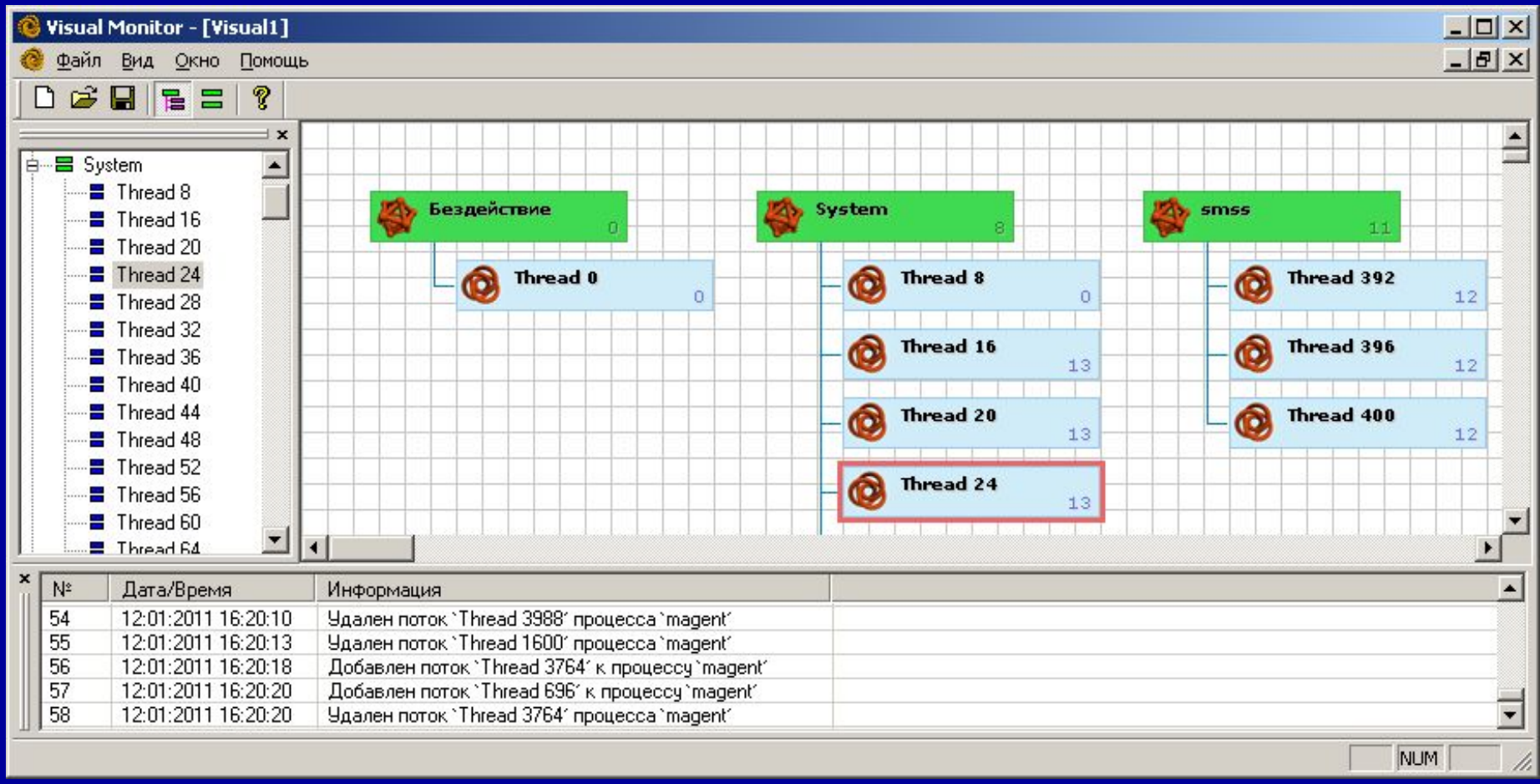

Свойства: Последовательный порт (COM1)
Общие
Параметры порта
Драйвер
Сведения
Ресурсы
Последовательный порт (COM1)
Перечень ресурсов:
Тип ресурса
Параметр
Диапазон ввода/вывода (I/O)
03F8 - 03FF
IRQ
04
Конфигурация:
Текущая конфигурация
Автоматическая настройка
Изменить...
Список конфликтующих устройств: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

Состояния потока



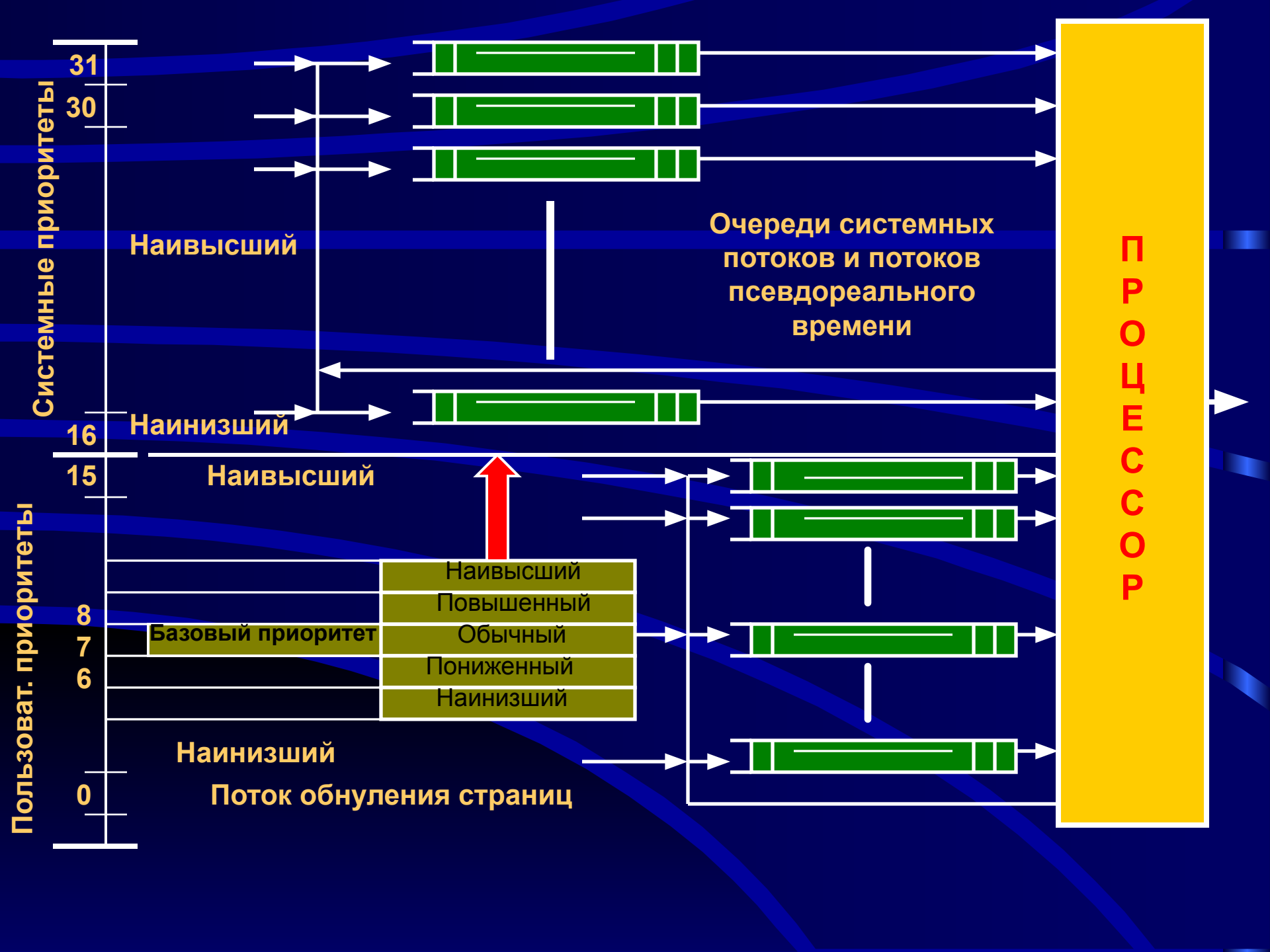
Состояния потока

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

Планирование 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.


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

- Динамические приоритеты имеют значения в диапазоне 1-15. ОС может динамически 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.

- 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 (механизм вытесняющей многозадачности не задействован). Такой поток должен сам освободить процессор.
- Однако в любом случае при появлении потока с большим или равным значением приоритета задействуется механизм вытесняющей многозадачности.

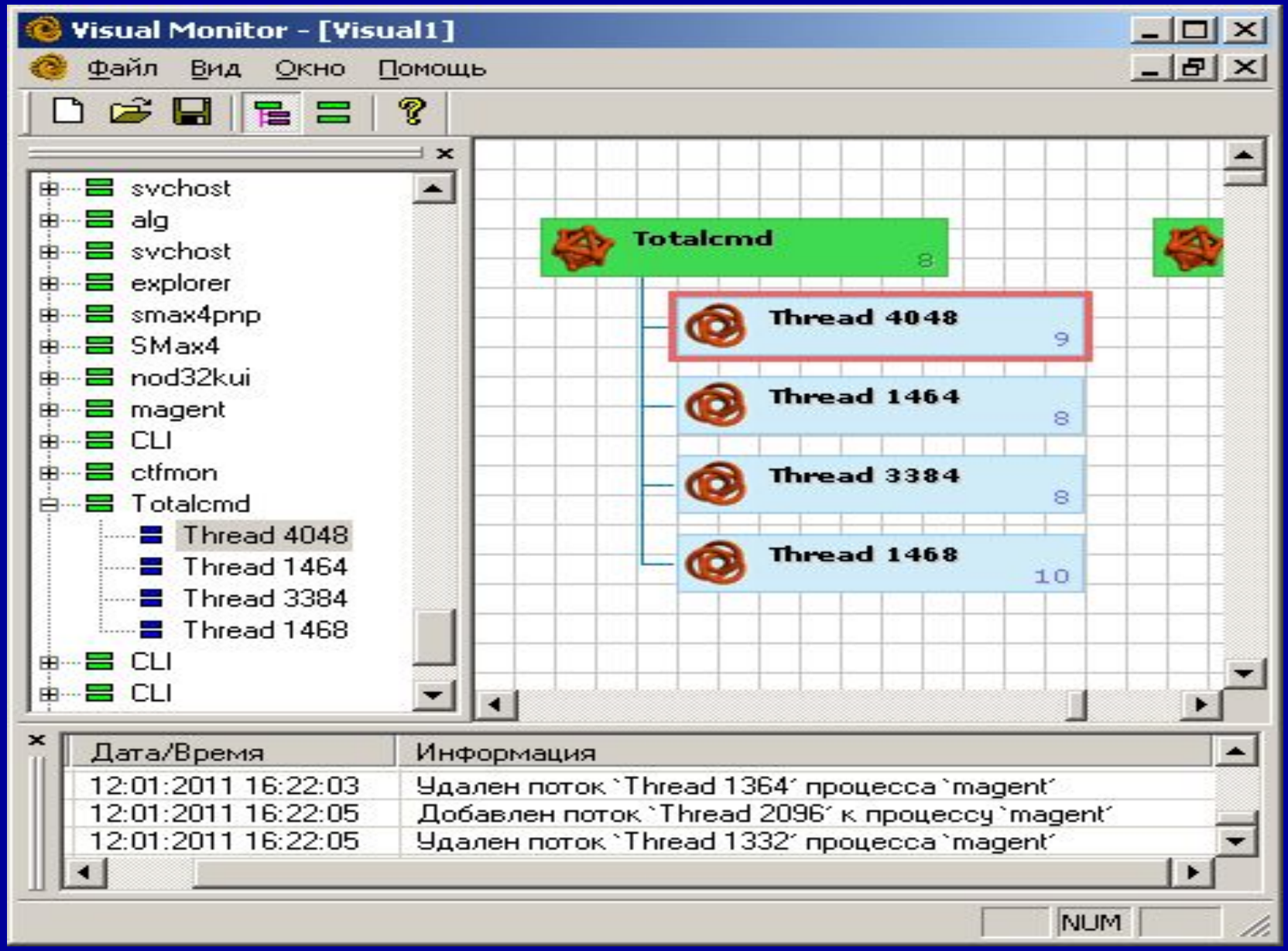
- Ядро Windows всегда запускает тот из потоков, готовых к выполнению, который обладает наивысшим приоритетом. Поток не является готовым к выполнению, если он находится в состоянии ожидания, приостановлен или блокирован по той или иной причине.
- Процесс может изменить или установить свой собственный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

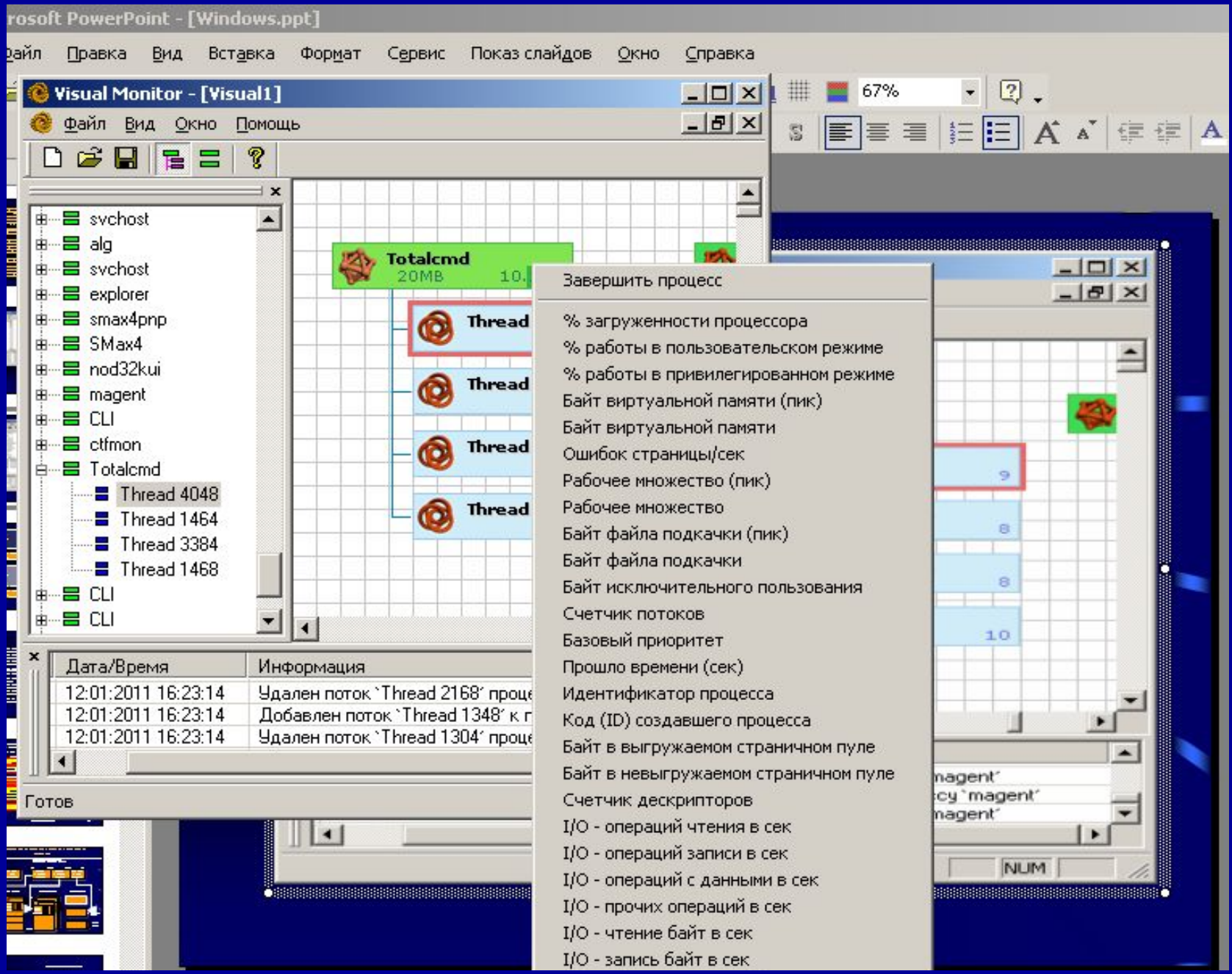
BOOL SetPriorityClass(HANDLE hProcess, DWORD dwPriority)

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


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

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.

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 (неделимым).

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.

Какой объем ОЗУ поддерживают различные версии и выпуски Windows

Version (only in X64 editions **)** **Limit on X64**

- Windows Server 2012 Datacenter 4 TB
- Windows Server 2012 Standard 4 TB
- Windows Server 2012 Essentials 64 GB
- Windows Server 2012 Foundation 32 GB
- Windows Storage Server 2012 Workgroup 32 GB
- Windows Storage Server 2012 Standard 4 TB

Version	Limit on X86	Limit on X64
Windows 8 Enterprise	4 GB	512 GB
Windows 8 Professional	4 GB	512 GB
Windows 8	4 GB	128 GB

Диспетчер памяти

Максимальный объем физической памяти, поддерживаемый Windows, варьируется от 2 до 1024 Гб в зависимости от версии и редакции Windows. Так как виртуальное адресное пространство может быть больше или меньше объема физической памяти в компьютере, диспетчер управления памятью решает две главные задачи.

- Трансляция, или проецирование (mapping), виртуального адресного пространства процесса на физическую память. Это позволяет ссылаться на корректные адреса физической памяти, когда потоки, выполняемые в контексте процесса, читают и записывают в его виртуальном адресном пространстве. Физически резидентное подмножество виртуального адресного пространства процесса называется рабочим набором (working set).
- Подкачка части содержимого памяти на диск, когда потоки или системный код пытаются задействовать больший объем физической памяти, чем тот, который имеется в наличии, и загрузка страниц обратно в физическую память по мере необходимости.

Диспетчер памяти является частью исполнительной системы Windows, содержится в файле Ntoskrnl.exe и включает следующие **компоненты**.

1. Набор сервисов исполнительной системы для выделения, освобождения и управления виртуальной памятью; большинство этих сервисов доступно через Windows API или интерфейсы драйверов устройств режима ядра.
2. Обработчики ловушек трансляции недействительных адресов (transla-tion-not-vaiid) и нарушений доступа для разрешения аппаратно обнаруживаемых исключений, связанных с управлением памятью, а также загрузки в физическую память необходимых процессу страниц.

- Страницы в адресном пространстве процесса могут быть:
- **свободными (free)**,
- **зарезервированными (reserved)**
- **переданными (committed)**.
- Приложения могут резервировать (reserve) адресное пространство и передавать память (commit) зарезервированным страницам по мере необходимости. Резервировать страницы и передавать им память можно и одним вызовом. Эти сервисы предоставляются через Windows-функции VirtualAlloc и VirtualAllocEx.

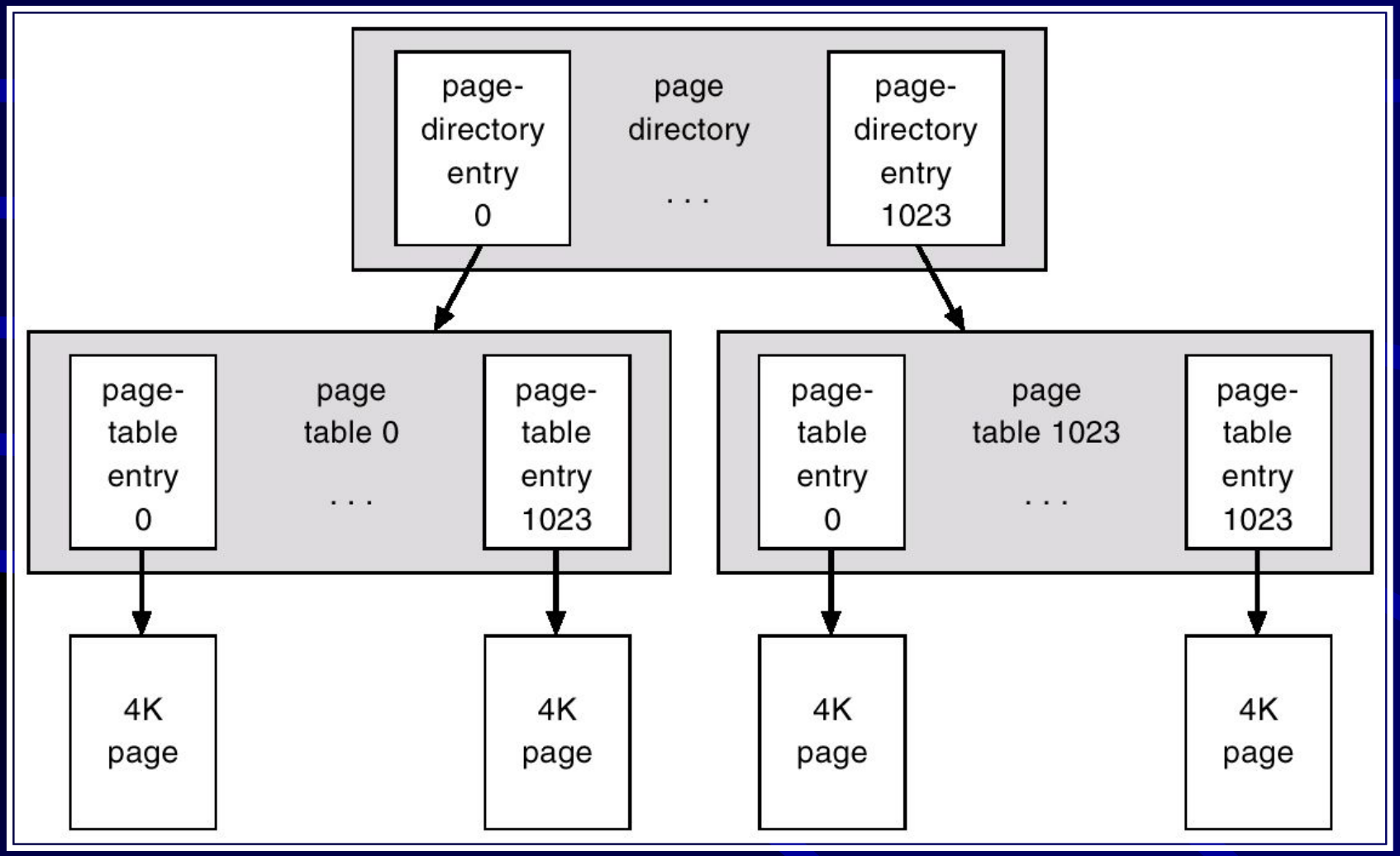
Менеджер виртуальной памяти

- При проектировании менеджера виртуальной памяти предполагалось, что процессор поддерживает для отображения виртуальных адресов в физические механизм страничной организации, прозрачный кэш для многопроцессорных систем.
- VM – менеджер в Windows использует страничную организацию с размером страницы 4 KB (4096) .
- Используется двухуровневая схема выделения памяти.
  - На первом шаге резервируется часть адресного пространства процесса.
  - На втором шаге выделяется пространство в файле подкачки (paging file).

Менеджер виртуальной памяти

- **Трансляция виртуальных адресов в Windows 2000 использует несколько структур данных.**
  - **Каждый процесс имеет каталог страниц (*page directory*), содержащий 1024 *элемента* размером по 4 байта.**
  - **Каждый элемент каталога ссылается на *таблицу страниц*, которая содержит 1024 *элемента* (page table entries - PTEs) размером по 4 байта.**
  - **Каждый PTE ссылается на фрейм страницы (4 KB) в физической памяти.**
- **Ссылка на элемент всегда занимает 10 битов (0..1023).**
- **Это свойство используется при трансляции виртуальных адресов в физические.**
- **Станица может находиться в следующих состояниях: valid, zeroed, free standby, modified, bad.**

Распределение виртуальной памяти



Невыгружаемый и выгружаемый пулы

Ядро и драйверы устройств используют невыгружаемый пул для хранения данных, к которым можно обратиться в случае, когда система не может обработать страничные ошибки.

Ядро входит в такой режим в случаях, когда оно выполняет процедуры обработки прерываний (**ISR**) и отложенные вызовы процедур (**DPC**) - функции, связанные с аппаратными прерываниями.

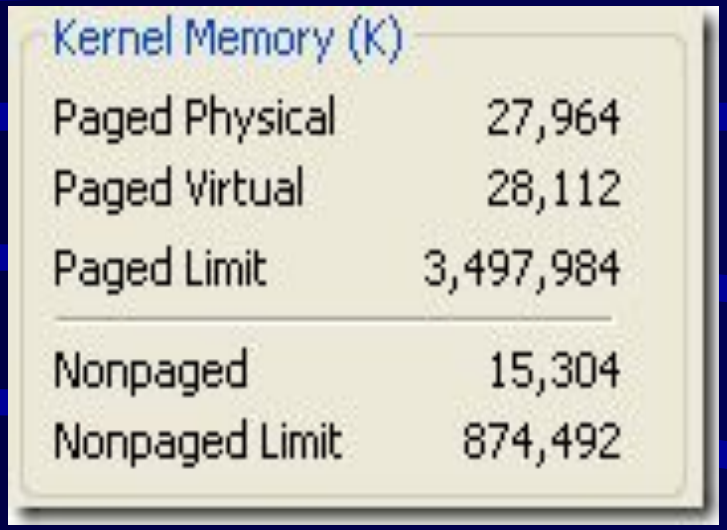
Выгружаемый пул, получил свое название потому, что **Windows** может записать данные, которые он хранит, в файл подкачки, позволяя тем самым использовать физическую память, которая при этом освобождается, в других целях.

Выгружаемый пул

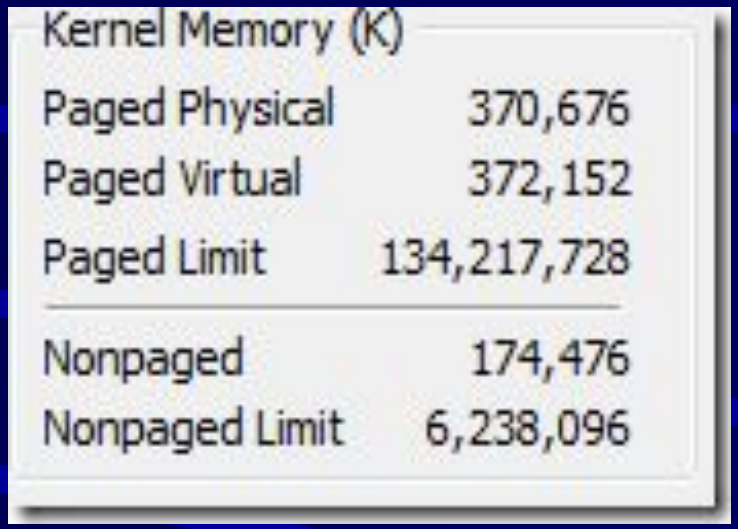
Выгружаемый пул, получил свое название потому, что **Windows** может записать данные, которые он хранит, в файл подкачки, позволяя тем самым использовать физическую память, которая при этом освобождается, в других целях.

Выгружаемый пул, получил свое название потому, что **Windows** может записать данные, которые он хранит, в файл подкачки, позволяя тем самым использовать физическую память, которая при этом освобождается, в других целях.

Ограничения на размер невыгружаемого пула

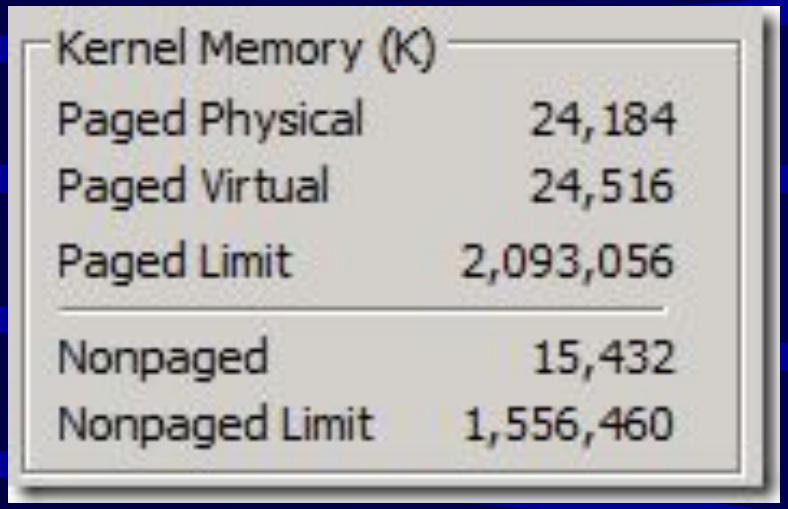


64-х битная Windows XP
с 2 Гб ОЗУ



64-битная Windows 7
с 8 Гб памяти

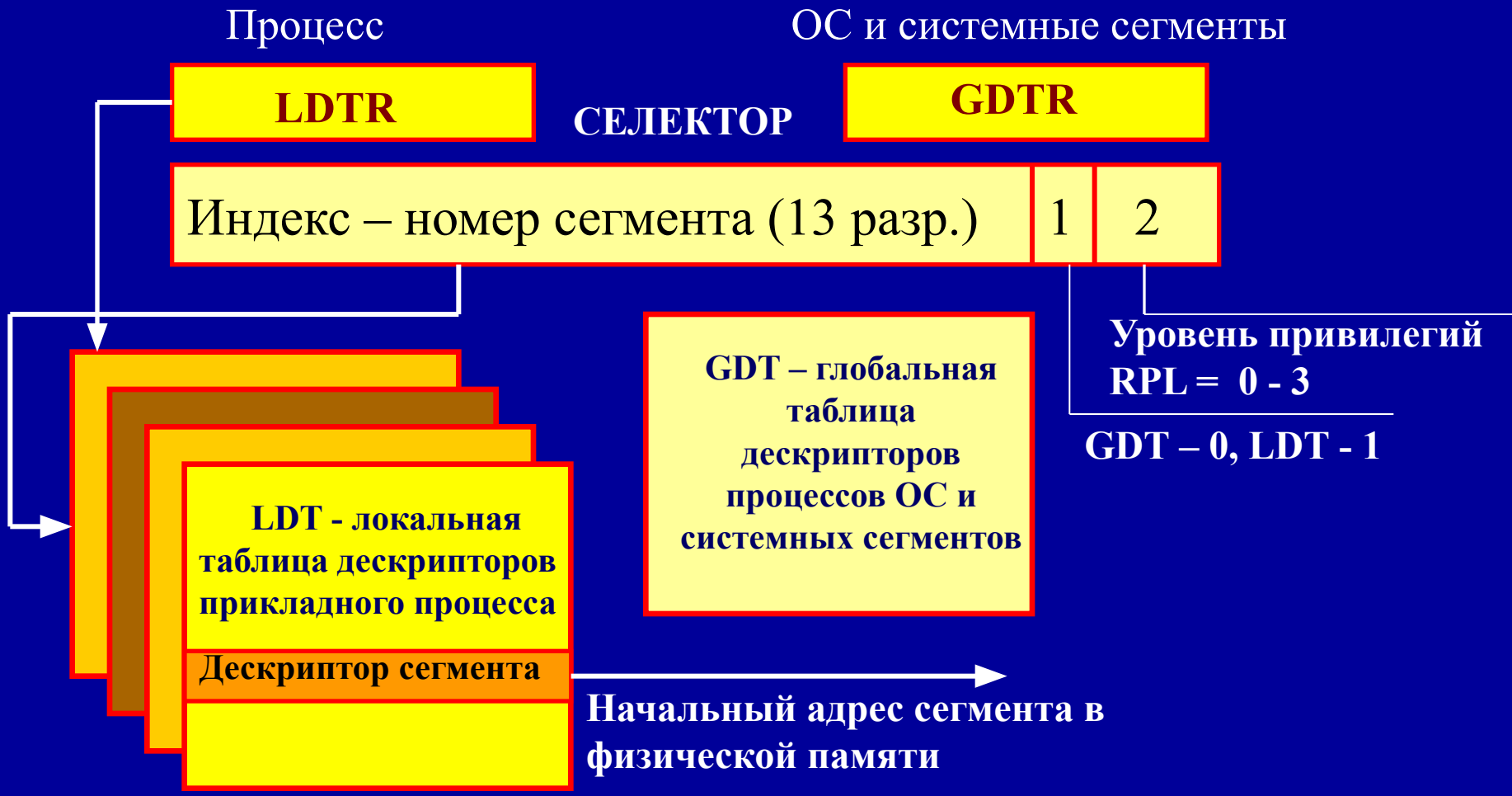
Ограничения на размер невыгружаемого пула

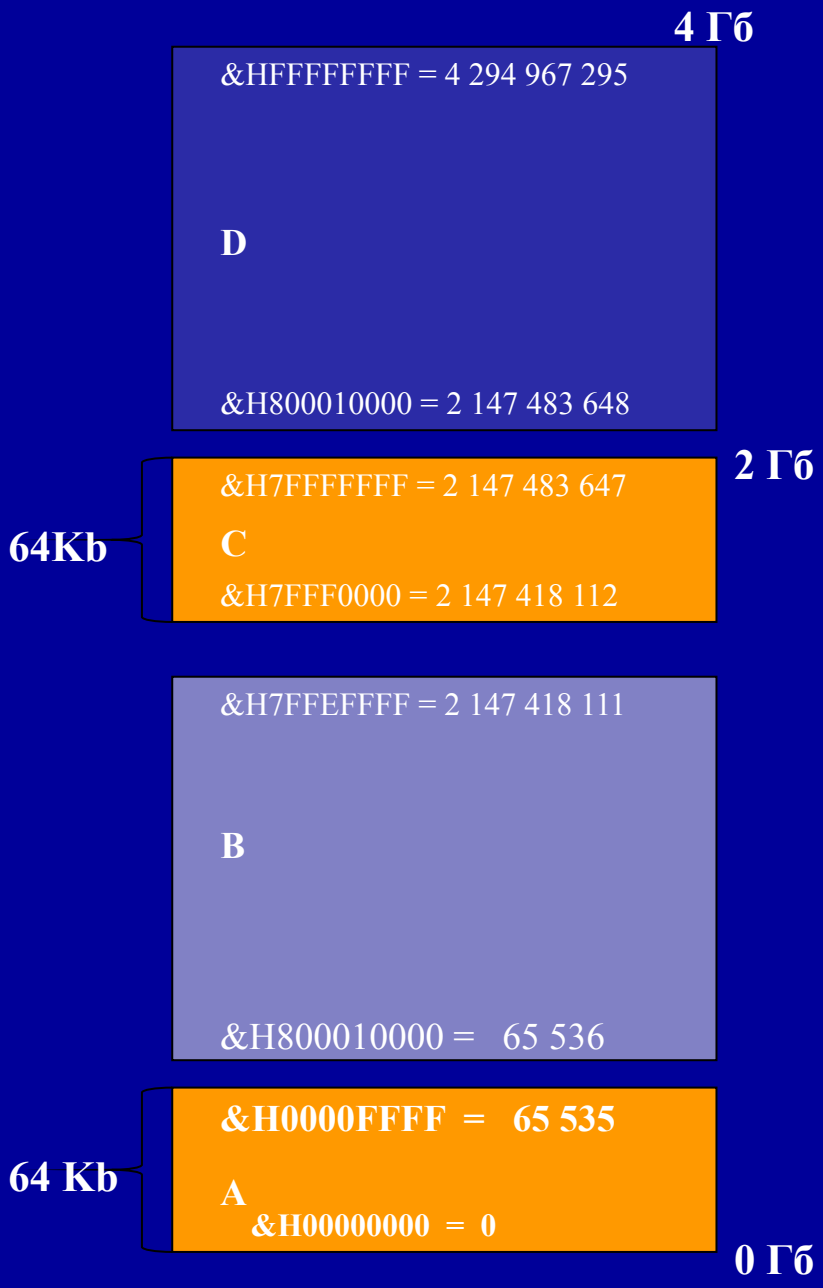


32-х битный
Windows Server 2008
с 2 Гб ОЗУ

Для **32-х битной Windows XP** данный предел вычисляется исходя из того, сколько адресного пространства выделено другим ресурсам, в особенности таблице **PTE**, с максимальным значением в **491Мб**.

Виртуальная память Windows обеспечивает каждому процессу: 1. 4 Гбайт виртуального адресного пространства (2 Гбайт – ОС, 2 Гбайт – пользовательская программа). 2. 16 К независимых сегментов (8к локальных и 8К глобальных).





Зарезервировано Windows для исполнительной системы Windows, ядра и драйверов устройств.
Недоступно в пользовательском режиме.
(2 Гб)

Используется для некорректно инициализированных указателей.
Недоступно в пользовательском режиме. (64 Кб)

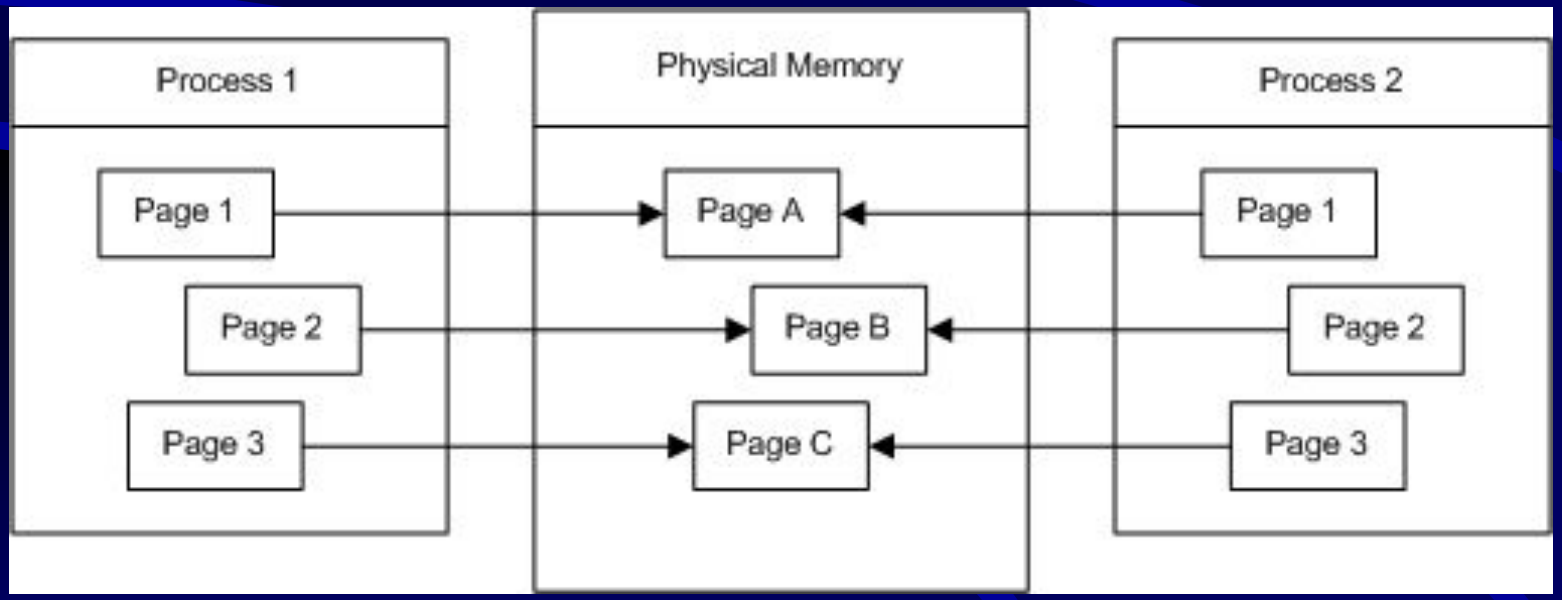
Адресное пространство процессов содержит прикладные модули EXE и DLL, Win32 DLL (kernel32.dll, user.dll и т.д.), файлы, отображаемые в память.
Доступно в пользовательском режиме.
(2 Гб – 128 Кб)

Защита памяти

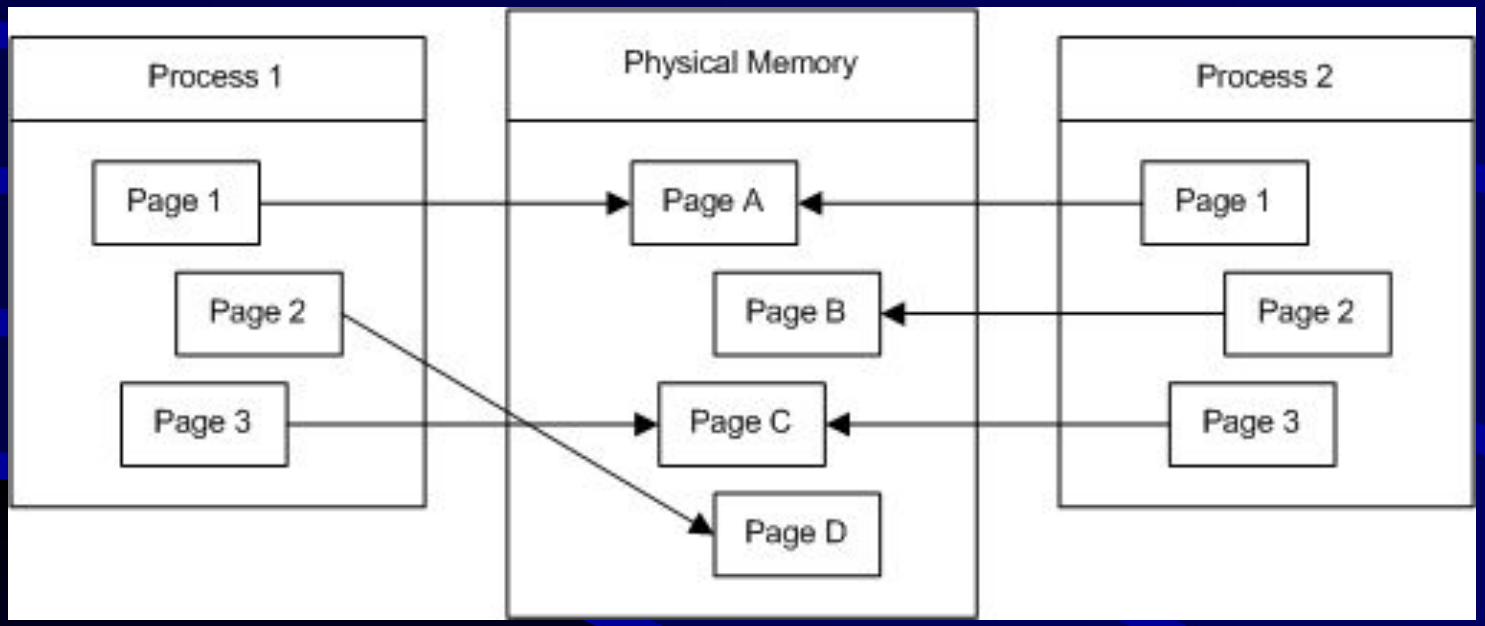
- PAGE_READONLY присваивает доступ «только для чтения» выделенной виртуальной памяти;
- PAGE_READWRITE назначает доступ «чтение-запись» выделенной виртуальной памяти;
- PAGE_WRITECOPY устанавливает доступ «запись копированием» (copy-onwrite) выделенной виртуальной памяти.
- PAGE_EXECUTE разрешает доступ «выполнение» выделенной виртуальной памяти. Тем не менее, любая попытка чтения - записи этой памяти приведет к нарушению доступа;
- PAGE_EXECUTE_READ назначает доступ «выполнение» и «чтение»;
- PAGE_EXECUTE_READWRITE разрешает доступ «выполнение», «чтение» и «запись»;
- PAGE_EXECUTE_WRITECOPY присваивает доступ «выполнение», «чтение» и «запись копированием»;
- PAGE_NOACCESS запрещает все виды доступа к выделенной виртуальной памяти.

Copy-on-Write Protection

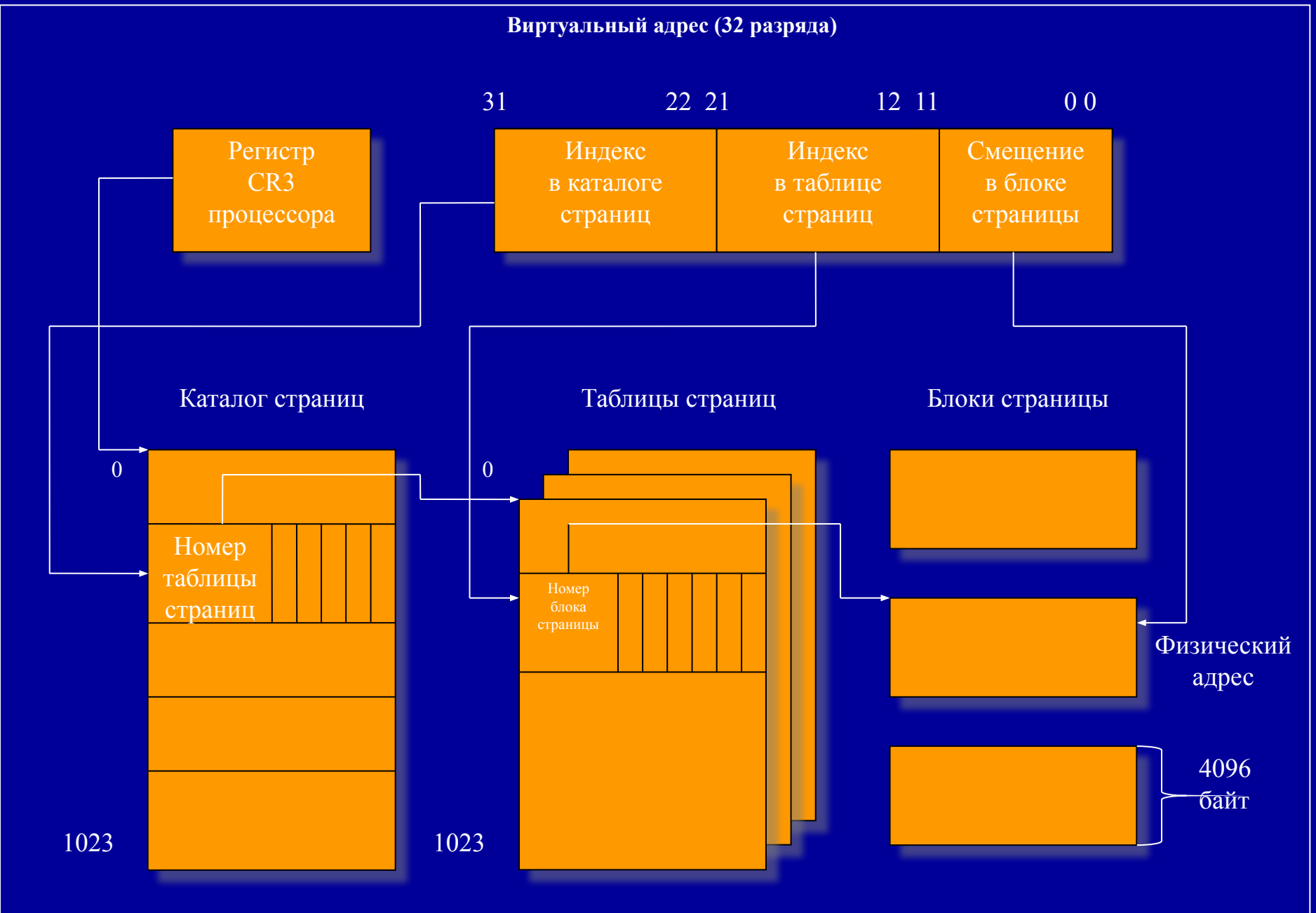
- Позволяет нескольким процессам организовать их карту виртуального адресного пространства (ВАП) таким образом, что они используют **одну** физическую страницу, пока один из процессов не изменяет страницу.
- Эта технология, называемая «*ленивые вычисления*», позволяет системе экономить физическую память и время, выполняя операцию, только когда это станет абсолютно необходимо.
  Например, два процесса загрузили страницы с одного DLL в свои ВАП. Эти страницы виртуальной памяти отображаются в той же физической странице памяти для обоих процессов. Пока ни один процесс не пишет на эти страницы, они могут отображаться на те же физические страницы.



- Если процесс1 записывает в одну из этих страниц, содержание физической страницы будет скопировано на другую физическую страницу и карта виртуальной памяти для процесса1 обновляется. Оба процесса теперь имеют свой собственный экземпляр страницы в физической памяти.

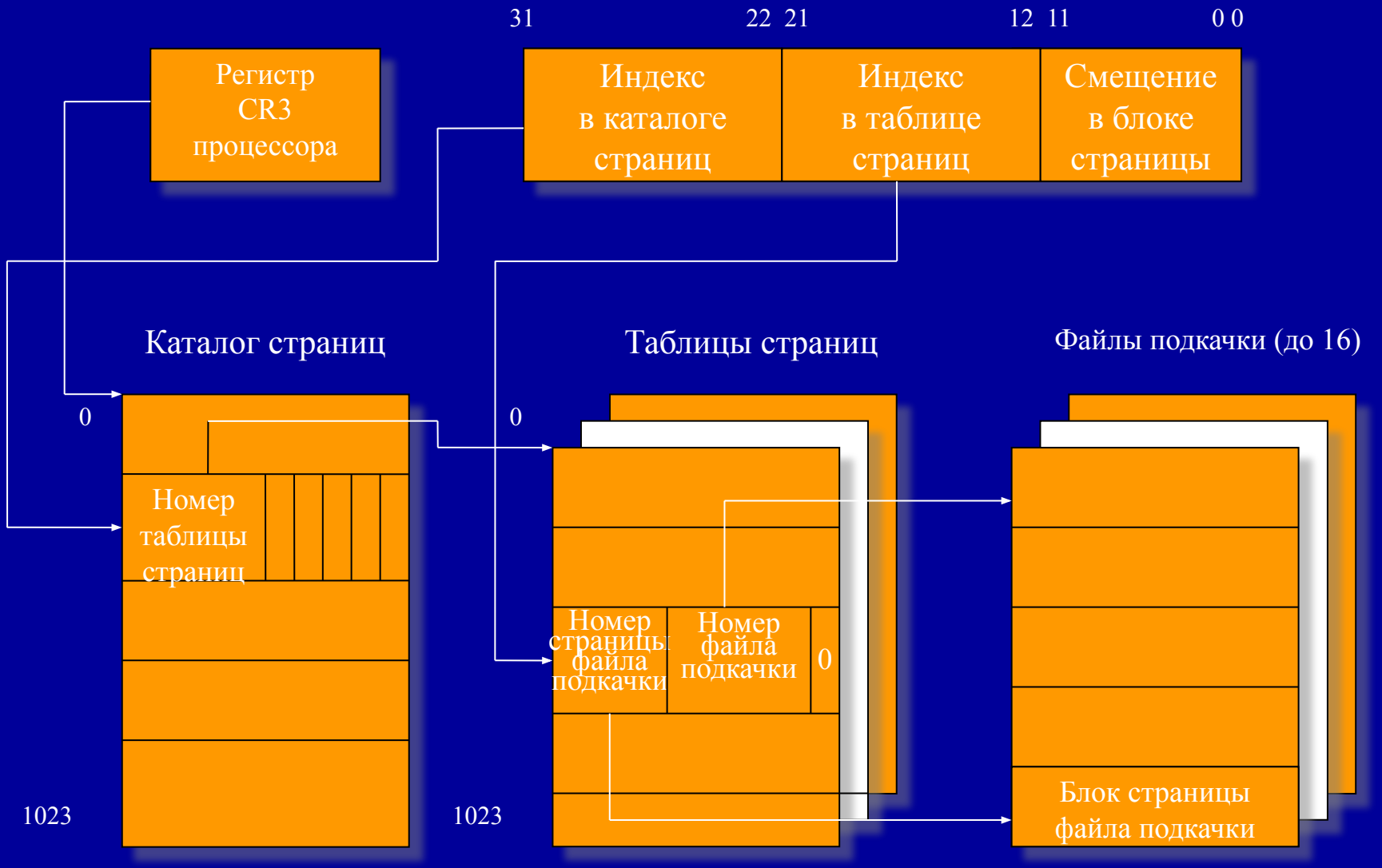


Преобразование виртуальных адресов в физические: попадание



Преобразование виртуальных адресов в физические: промах

Виртуальный адрес (32 разряда)




Бит записи (страница была записана)
Бит доступа (страница читалась)
Запрет кэширования
Запись в обход запрета
Разрешение доступа из пользовательского режима
Чтение/запись или только запись
Бит достоверности (страница отражается в физическую память)
Номер блока страницы в физической памяти
31
12
11
10
9
8
7
6
5
4
3
2
1
0

Проблемы ручного управления памятью

Традиционным для директивных языков способом управления памятью является ручной:

1. Для создания объекта в динамической памяти программист явно вызывает команду выделения памяти. Эта команда возвращает указатель на выделенную область памяти, который сохраняется и используется для доступа к ней.
2. До тех пор, пока созданный объект нужен для работы программы, программа обращается к нему через ранее сохранённый указатель.
3. Когда надобность в объекте проходит, программист явно вызывает команду освобождения памяти, передавая ей указатель на удаляемый объект.

В любом языке, допускающем создание объектов в динамической памяти, потенциально возможны две проблемы: висячие ссылки и утечки памяти.

Висячая ссылка (англ. dangling pointer)

Висячая ссылка — это оставшаяся в использовании ссылка на объект, который уже удалён. После удаления объекта все сохранившиеся в программе ссылки на него становятся «висячими». Память, занимаемая ранее объектом, может быть передана операционной системе и стать недоступной, или быть использована для размещения нового объекта в той же программе. В первом случае попытка обратиться по «повисшей» ссылке приведёт к срабатыванию механизма защиты памяти и аварийной остановке программы, а во втором — к непредсказуемым последствиям.

Появление висячих ссылок обычно становится следствием неправильной оценки времени жизни объекта: программист вызывает команду удаления объекта до того, как его использование прекратится

Утечка памяти (англ. memory leak)
— процесс неконтролируемого уменьшения объёма свободной оперативной памяти (RAM) компьютера, связанный с ошибками в работающих программах, вовремя не освобождающих ненужные уже участки памяти, или с ошибками системных служб контроля памяти.
Создав объект в динамической памяти, программист может не удалить его после завершения использования. Если ссылающейся на объект переменной будет присвоено новое значение и на объект нет других ссылок, он становится программно недоступным, но продолжает занимать память, поскольку команда его удаления не вызывалась. Такая ситуация и называется утечкой памяти.

Рассмотрим следующий фрагмент кода на C++:

```
/*1*/ char *pointer = NULL;
/*2*/ for( int i = 0; i < 10; i++ ) {
/*3*/   pointer = new char[100];
/*4*/ }
/*5*/ delete [] pointer;
```

В этом примере на 3-й строке создается объект в динамической памяти. Код на 3-й строке выполняется 10 раз, причём каждый следующий раз адрес нового объекта перезаписывает значение, хранящееся в указателе pointer. На 5-й строке выполняется удаление объекта, созданного на последней итерации цикла. Однако первые 9 объектов остаются в динамической памяти, и одновременно в программе не остаётся переменных, которые бы хранили адреса этих объектов. Т.е. в 5-й строке невозможно ни получить доступ к первым 9 объектам, ни удалить их.

Сборка мусора

- В программировании **сборка мусора** (устоявшийся термин, с точки зрения русского языка правильнее «сбор мусора», англ. garbage collection, **GC**) — одна из форм автоматического управления памятью. Специальный код, называемый сборщиком мусора, периодически освобождает память, удаляя объекты, которые уже не будут востребованы приложением.

Достижимость объекта

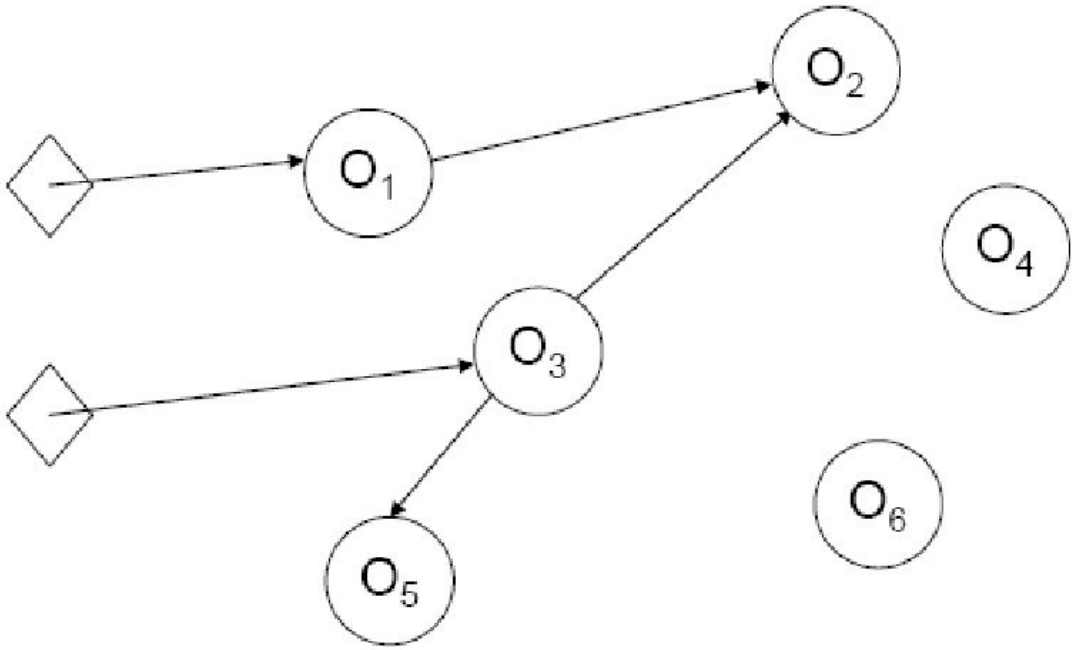
- определённое множество объектов считается достижимым изначально — корневые объекты, обычно в их число включают все глобальные переменные и объекты, на которые есть ссылки в стеке вызовов;
- любой объект, на который есть ссылка из достижимого объекта, тоже считается достижимым.

«Алгоритм пометок» (Mark and Sweep):

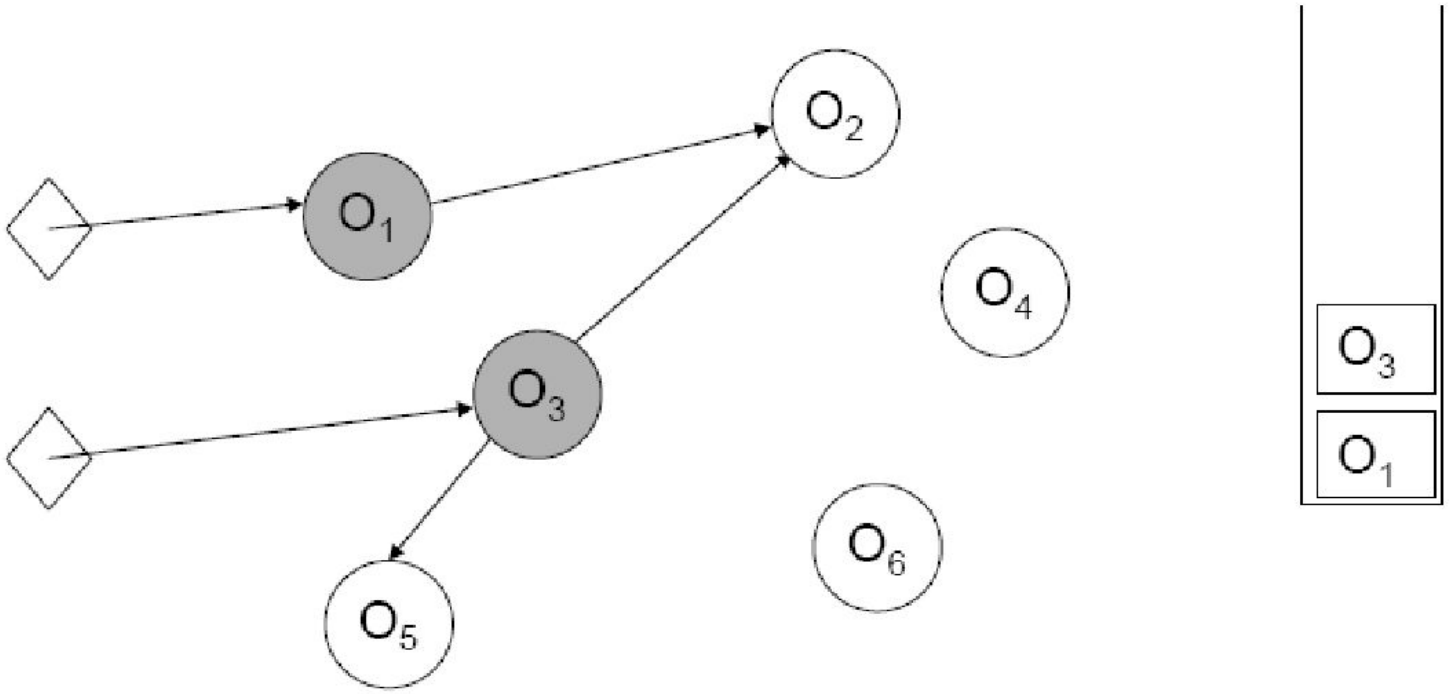
- для каждого объекта хранится бит, указывающий, достижим ли этот объект из программы или нет;
- изначально все объекты, кроме корневых, помечаются как недостижимые;
- рекурсивно просматриваются и помечаются как достижимые объекты ещё не помеченные и до которых можно добраться из корневых объектов по ссылкам;
- те объекты, у которых бит достижимости не был установлен, считаются недостижимыми.

Следует обратить внимание, что, согласно данному алгоритму, если два или более объектов ссылаются друг на друга, но ни на один из этих объектов нет других ссылок, то есть имеет место циклическая ссылка, то вся группа считается недостижимой. Эта особенность алгоритма позволяет гарантированно удалять группы объектов, использование которых прекратилось, но в которых имеются ссылки друг на друга. Такие группы часто называются «islands of isolation» (острова изоляции)

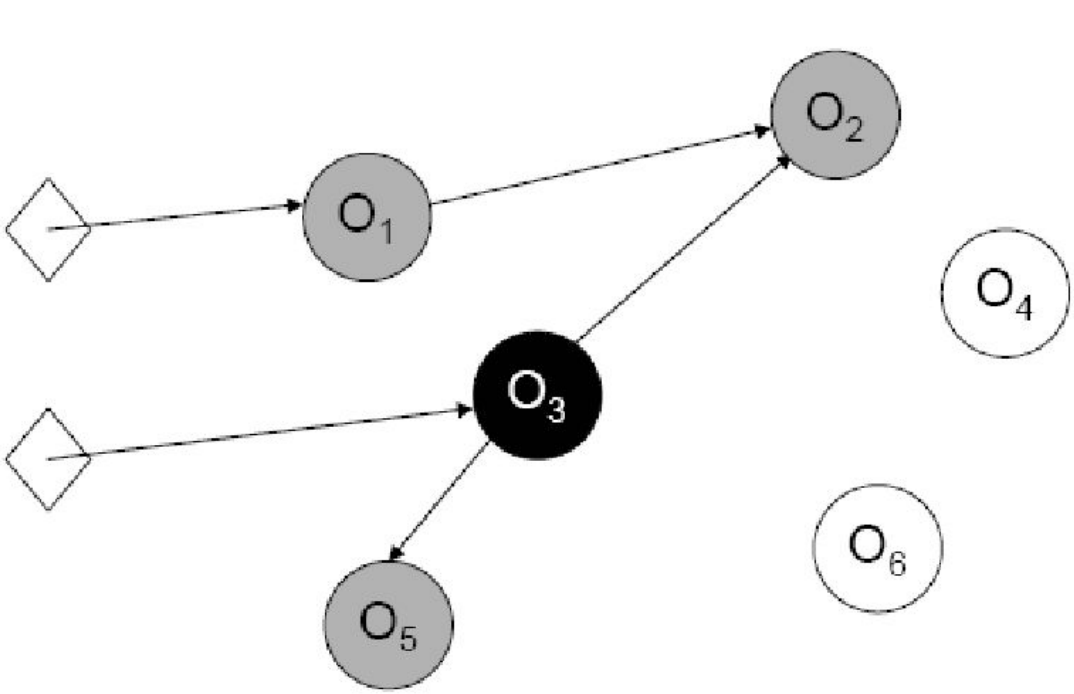
Разметка

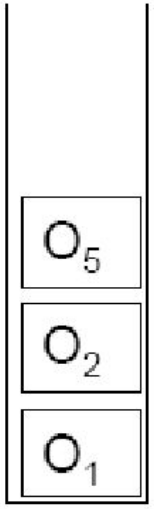


Разметка



Разметка



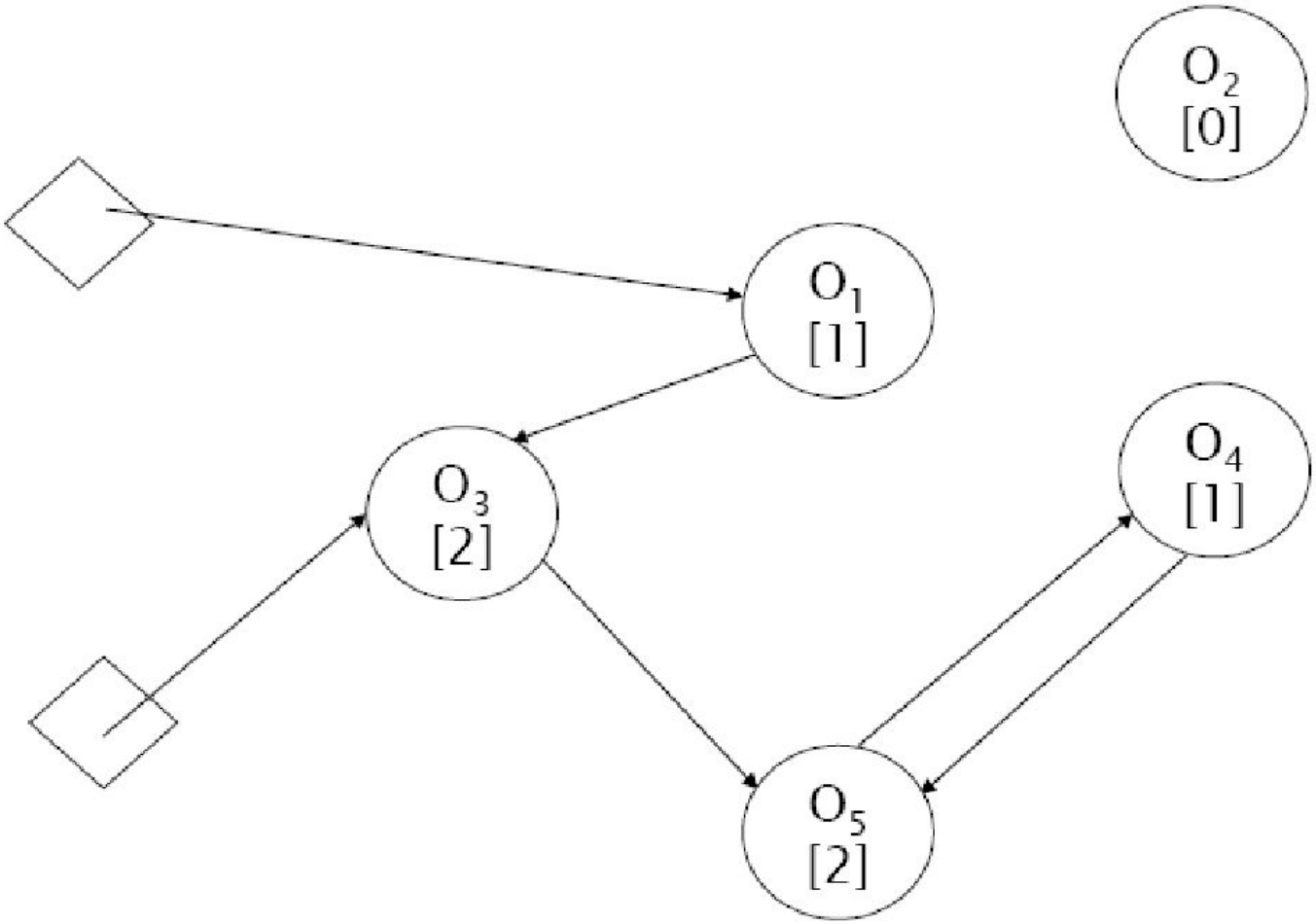


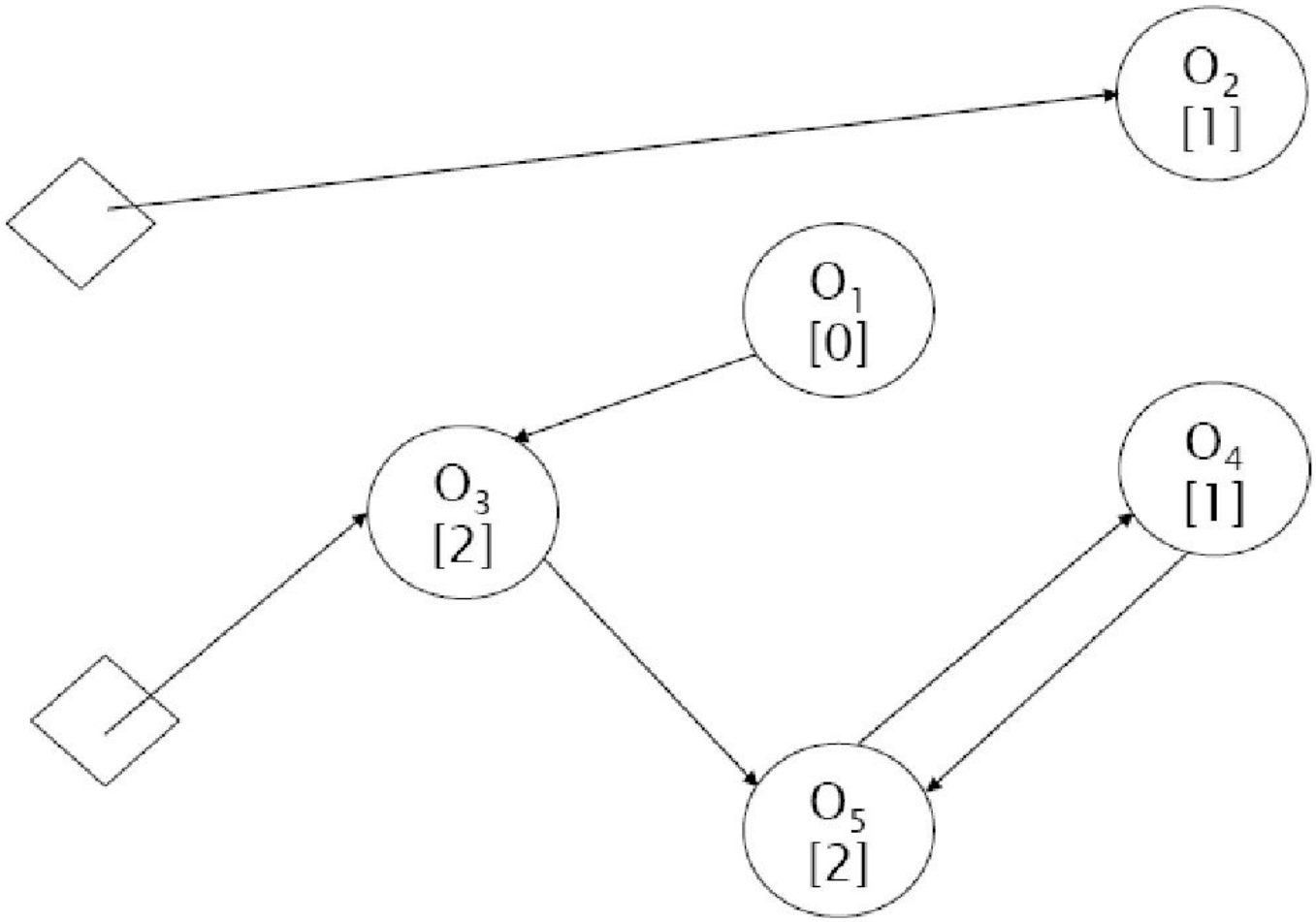
Алгоритм подсчёта ссылок

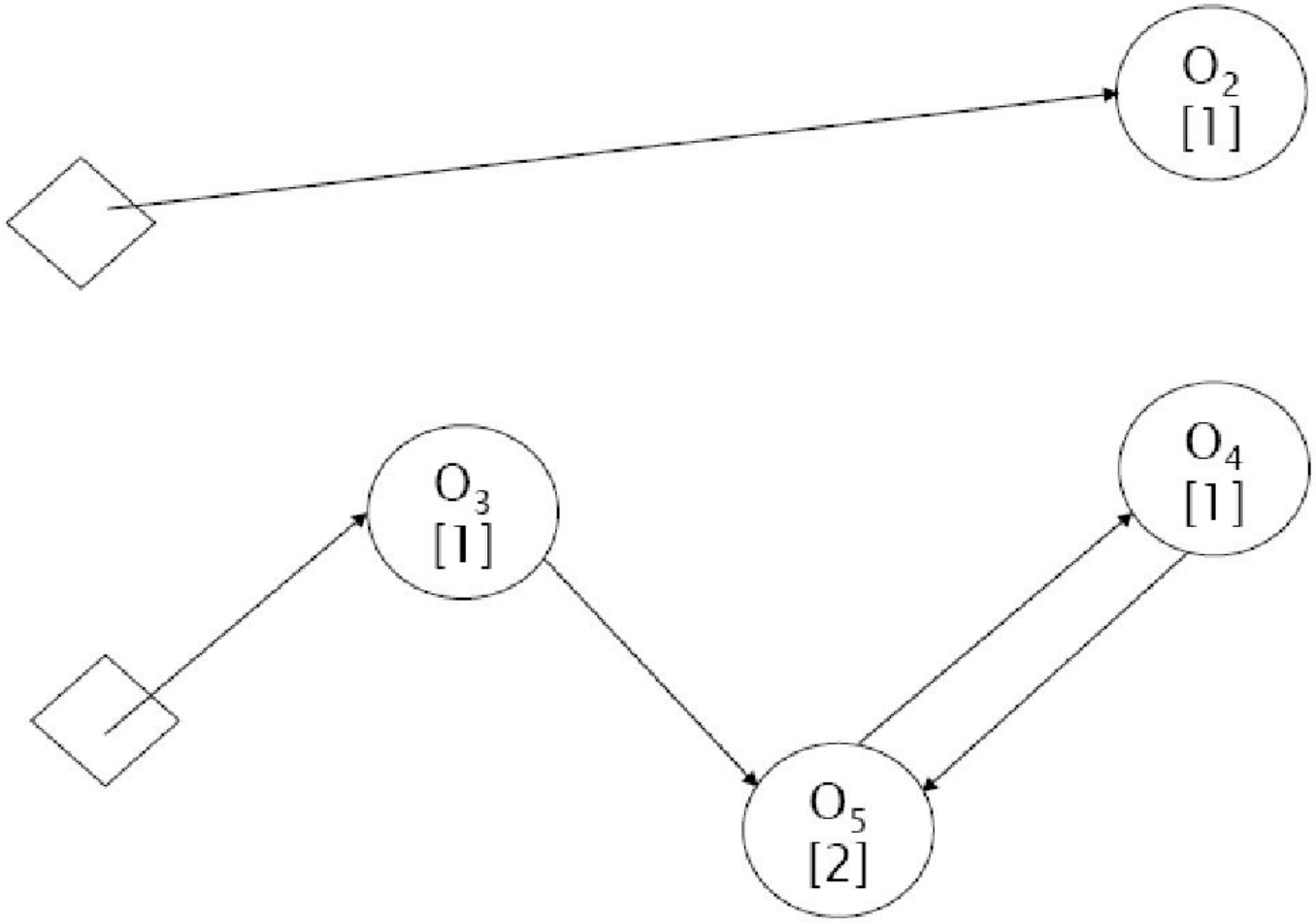
Другой вариант алгоритма определения достижимости — обычный подсчёт ссылок. Его использование замедляет операции присваивания ссылок, но зато определение достижимых объектов тривиально — это все объекты, значение счётчика ссылок которых превышает нуль. Без дополнительных уточнений этот алгоритм, в отличие от предыдущего, не удаляет циклически замкнутые цепочки вышедших из употребления объектов, сохранивших ссылки друг на друга.

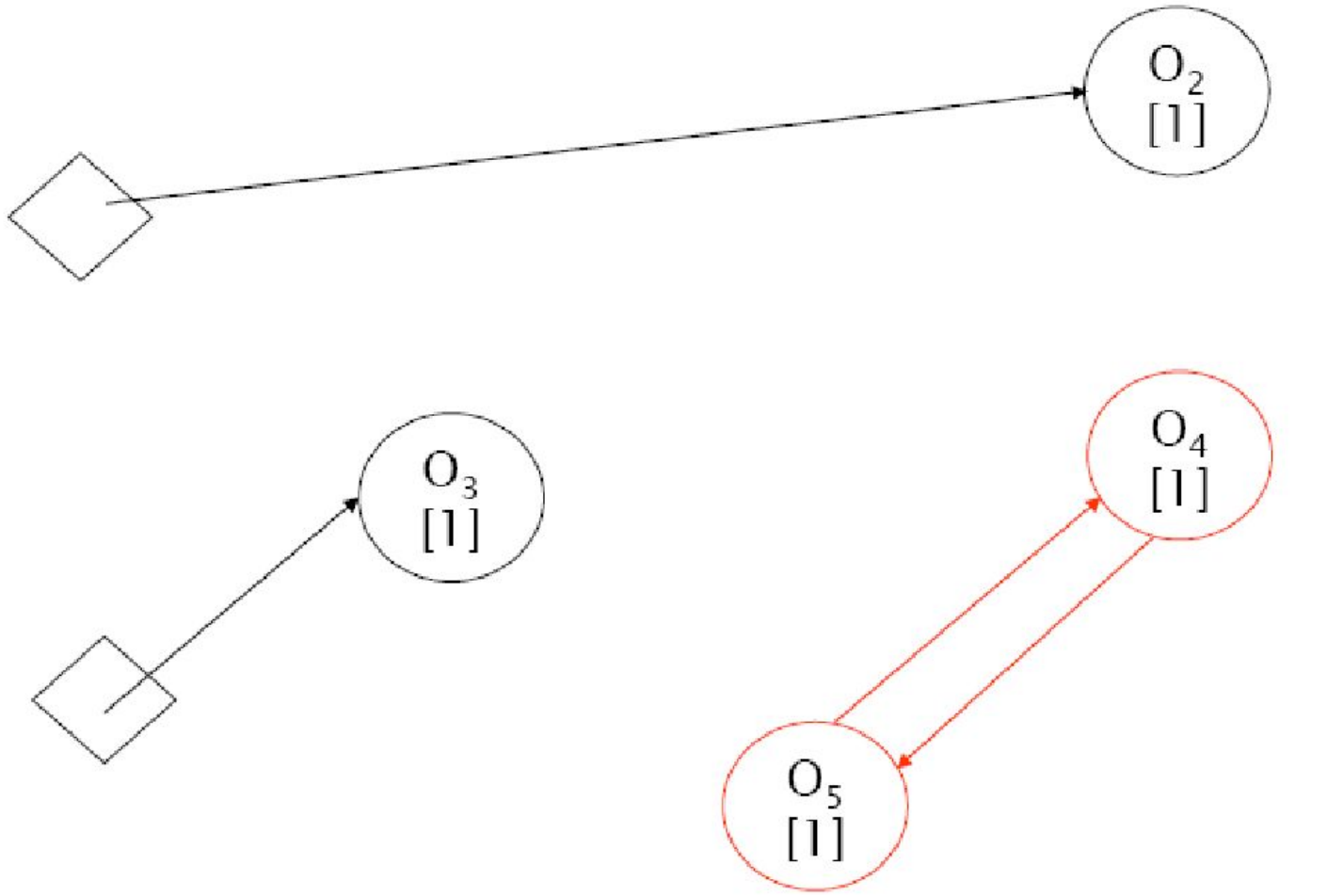
Автоматическое управление памятью: подсчет ссылок

- При создании, с каждым объектом ассоциируется счетчик ссылок на этот объект
- При модификации указателей счетчики корректируются
- Если счетчик достигает 0, объект освобождается


O2
[0]
O1
[1]
O3
[2]
O4
[1]
O5
[2]


O2
[1]
O1
[0]
O3
[2]
O4
[1]
O5
[2]


O2
[1]
O3
[1]
O4
[1]
O5
[2]



Циклический мусор

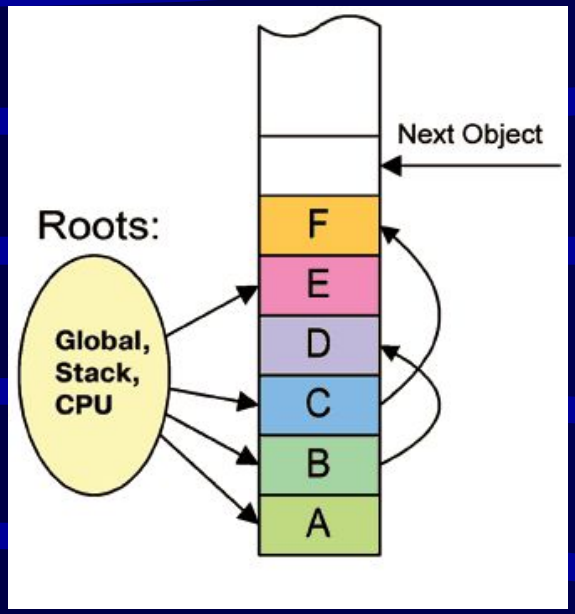
Как только определено множество *недостижимых* объектов, сборщик мусора может освободить память, занимаемую ими.

- **Неперемещающий сборщик мусора** быстро освобождает память (поскольку эта операция сводится к пометке соответствующих блоков памяти как свободных), но тратит больше времени на её выделение (поскольку память фрагментируется и при выделении необходимо найти в карте памяти нужное количество блоков подходящего размера).
- **Перемещающий сборщик** требует сравнительно больше времени на этапе сборки мусора (тратится дополнительное время на дефрагментацию памяти и изменение всех ссылок на перемещаемые объекты), зато перемещение позволяет использовать чрезвычайно простой и быстрый алгоритм выделения памяти. При дефрагментации объекты передвигаются так, чтобы разделить всю память на две большие области — занятую и свободную, и сохраняется указатель на их границу. Для выделения новой памяти достаточно лишь переместить эту границу, вернув кусок из начала свободной памяти.

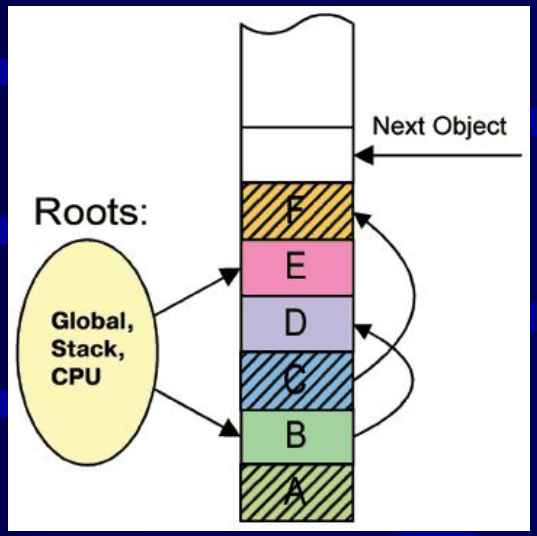
- Для обеспечения высокой скорости доступа к объектам в динамической памяти, объекты, поля которых используются совместно, перемещающий сборщик позволяет размещать в памяти недалеко друг от друга. Тогда они вероятнее окажутся в кэше процессора одновременно, что уменьшит количество обращений к относительно медленному ОЗУ.
- Многие современные сборщики мусора подразделяют все объекты на несколько поколений — серий объектов с близким временем существования. Как только память, выделенная одному из поколений, заканчивается, в этом поколении и во всех более «молодых» производится поиск недостижимых объектов. Все они удаляются, а оставшиеся переводятся в более «старое» поколение.

СБОРЩИК МУСОРА C#

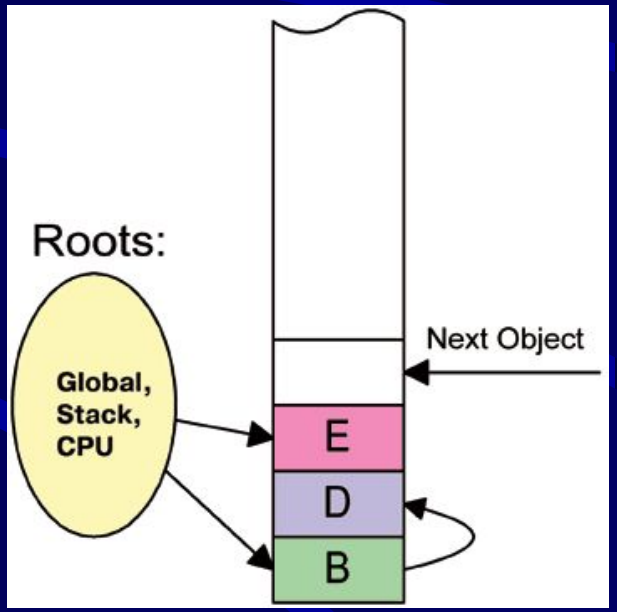
- Алгоритм работы сборщика мусора (garbage collector, GC), являющегося частью CLR, подробно описан в книге Джефри Рихтера «Applied Microsoft .NET Framework Programming».
- Для хранения объектов CLR использует хип, подобный хипу C++, за тем важным исключением, что хип CLR не фрагментирован.
- Выделение объектов производится всегда один за другим в последовательных адресах памяти, что позволяет существенно повысить производительность всего приложения. Когда память заканчивается, в дело вступает процесс, основная задача которого сводится к тому, чтобы освободить место в хипе путём дефрагментации неиспользуемой памяти.
- Первое, что делает GC во время сборки мусора – это принимает решение о том, что все выделенные блоки памяти в программе - это как раз и есть мусор. Далее начинается поиск «живых» указателей на объекты. Microsoft называет эти указатели «roots».
- Найдя «живой» указатель, сборщик мусора помечает объект, на который этот указатель указывает, как всё ещё используемый программой и запрашивает информацию о его структуре, чтобы в свою очередь просканировать уже этот объект на наличие указателей на другие объекты. И так далее, пока все указатели в программе не будут обработаны.



Начальное состояние



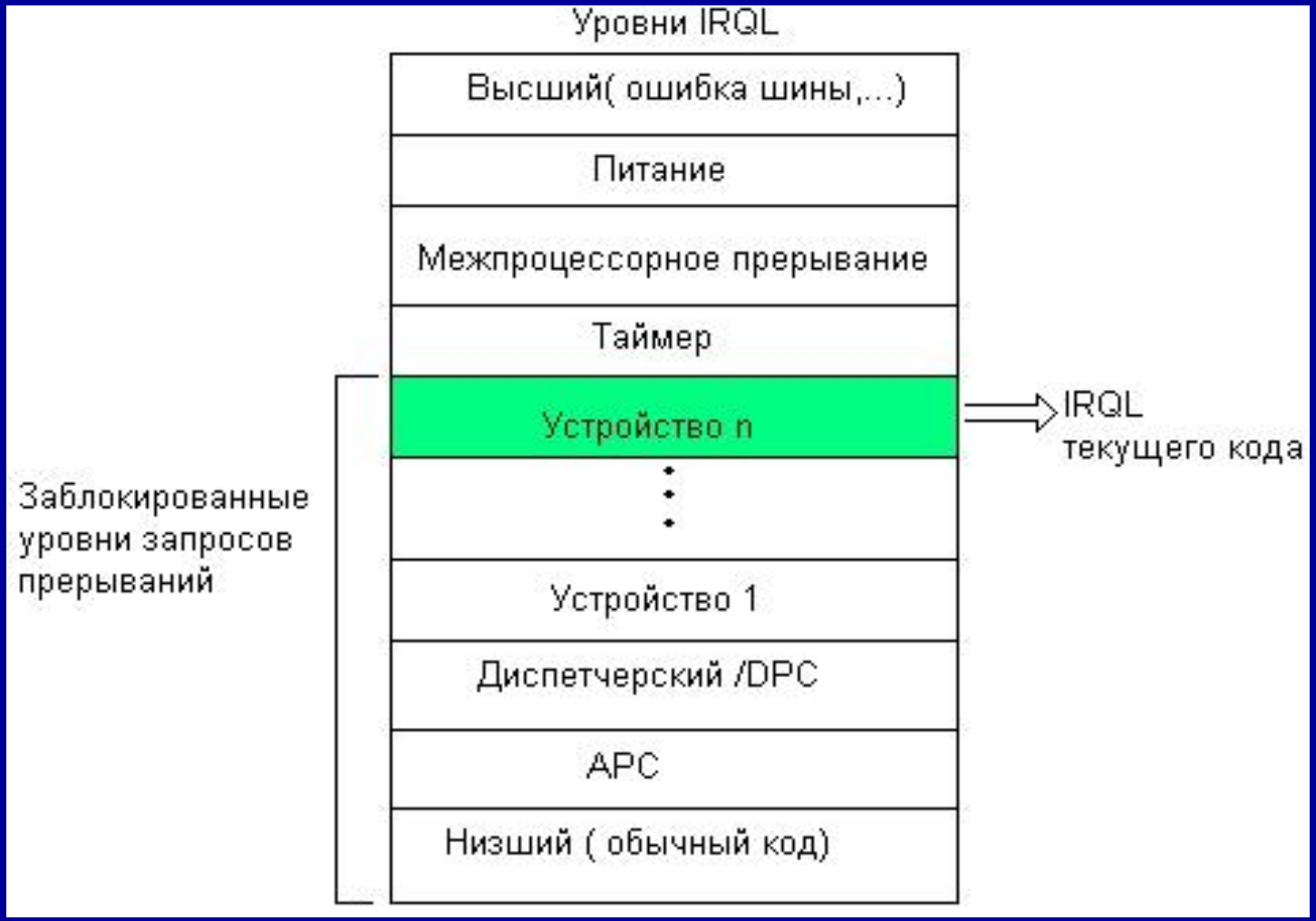
Heap после завершения работы с некоторыми объектами



Heap после сборки мусора

Алгоритмы работы GC построены во многом исходя из правил, полученных статистическим и опытным путём.
В частности, одно из таких правил утверждает, что только что созданные «молодые» объекты имеют наиболее короткое время жизни, а живущие уже давно будут жить ещё долго. Именно в соответствии с этим правилом в CLR существуют понятие **поколений** (generations).
Объекты, выжившие после первой сборки мусора и дефрагментации, объявляются первым поколением. В следующий раз, те объекты из первого поколения, которые опять смогли выжить, перемещаются во второе поколение и уже больше не трогаются, выжившие объекты из нулевого поколения перемещаются в первое.
Например, **объекты, пережившие две сборки мусора, остаются в хипе навсегда и GC не занимается их дефрагментацией**.

Диспетчер прерываний Windows NT/2000/XP (Trap Handler) работает с программной моделью прерываний, единой для всех аппаратных платформ



Уровни запросов прерываний

- Прерывания обслуживаются в соответствии с их приоритетом. Windows 2000 использует схему приоритетов прерываний, известную под названием уровни запросов прерываний (interrupt request levels, IRQL).
- 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.

Организация контроля доступа в Windows

- Для разделяемых ресурсов в Windows NT применяется общая модель объекта, который содержит такие характеристики безопасности, как набор допустимых операций, идентификатор владельца, список управления доступом. Объекты в Windows NT создаются для любых ресурсов в том случае, когда они являются или становятся разделяемыми — файлов, каталогов, устройств, секций памяти, процессов. Характеристики объектов в Windows NT делятся на две части — общую часть, состав которой не зависит от типа объекта, и индивидуальную, определяемую типом объекта.

- Все объекты хранятся в древовидных иерархических структурах, элементами которых являются объекты-ветви (каталоги) и объекты-листья (файлы). Для объектов файловой системы такая схема отношений является прямым отражением иерархии каталогов и файлов. Для объектов других типов иерархическая схема отношений имеет свое содержание, например, для процессов она отражает связи «родитель-потомок», а для устройств отражает принадлежность к определенному типу устройств и связи устройства с другими устройствами, например SCSI-контроллера с дисками.

- Проверка прав доступа для объектов любого типа выполняется централизованно с помощью монитора безопасности (Security Reference Monitor), работающего в привилегированном режиме.
- Для системы безопасности Windows NT характерно наличие большого количества различных предопределенных (встроенных) субъектов доступа — как отдельных пользователей, так и групп. Так, в системе всегда имеются такие пользователи, как Administrator, System и Guest, а также группы Users, Administrators, Server Operators, Everyone и др.
- Смысл этих встроенных пользователей и групп состоит в том, что они наделены некоторыми правами. При добавлении нового пользователя администратору остается только решить, к какой группе или группам отнести этого пользователя. Конечно, администратор может создавать новые группы, а также добавлять права к встроенным группам для реализации собственной политики безопасности.

Windows поддерживает три класса операций доступа, которые отличаются типом субъектов и объектов, участвующих в этих операциях.

- Разрешения (permissions) — это множество операций, которые могут быть определены для субъектов всех типов по отношению к объектам любого типа: файлам, каталогам, принтерам, секциям памяти и т. д.
- Права (user rights) — определяются для субъектов типа группа на выполнение некоторых системных операций: установку системного времени, архивирование файлов, выключение компьютера и т. п. В этих операциях участвует особый объект доступа — операционная система в целом. Некоторые права у встроенной группы являются также встроенными — их у данной группы нельзя удалить.
- Возможности пользователей (user abilities) определяются для отдельных пользователей на выполнение действий, связанных с формированием их операционной среды, например изменение состава главного меню программ.

- При входе пользователя в систему для него создается так называемый токен доступа (access token), включающий идентификатор пользователя и идентификаторы всех групп, в которые входит пользователь. В токене также имеются: список управления доступом (ACL) по умолчанию, который состоит из разрешений и применяется к создаваемым процессом объектам; список прав пользователя на выполнение системных действий.

- Все объекты, включая файлы, потоки, события, даже токены доступа, когда они создаются, снабжаются дескриптором безопасности. Дескриптор безопасности содержит список управления доступом — ACL (Access Control List). Владелец объекта, обычно пользователь, который его создал, обладает правом избирательного управления доступом к объекту и может изменять ACL объекта, чтобы позволить или не позволить другим осуществлять доступ к объекту. Говорят, что список ACL состоит из элементов управления доступом (Access Control Element, ACE), при этом каждый элемент соответствует одному идентификатору. Список ACL с добавленным к нему идентификатором владельца называют характеристиками безопасности.

Разрешения на доступ к каталогам и файлам

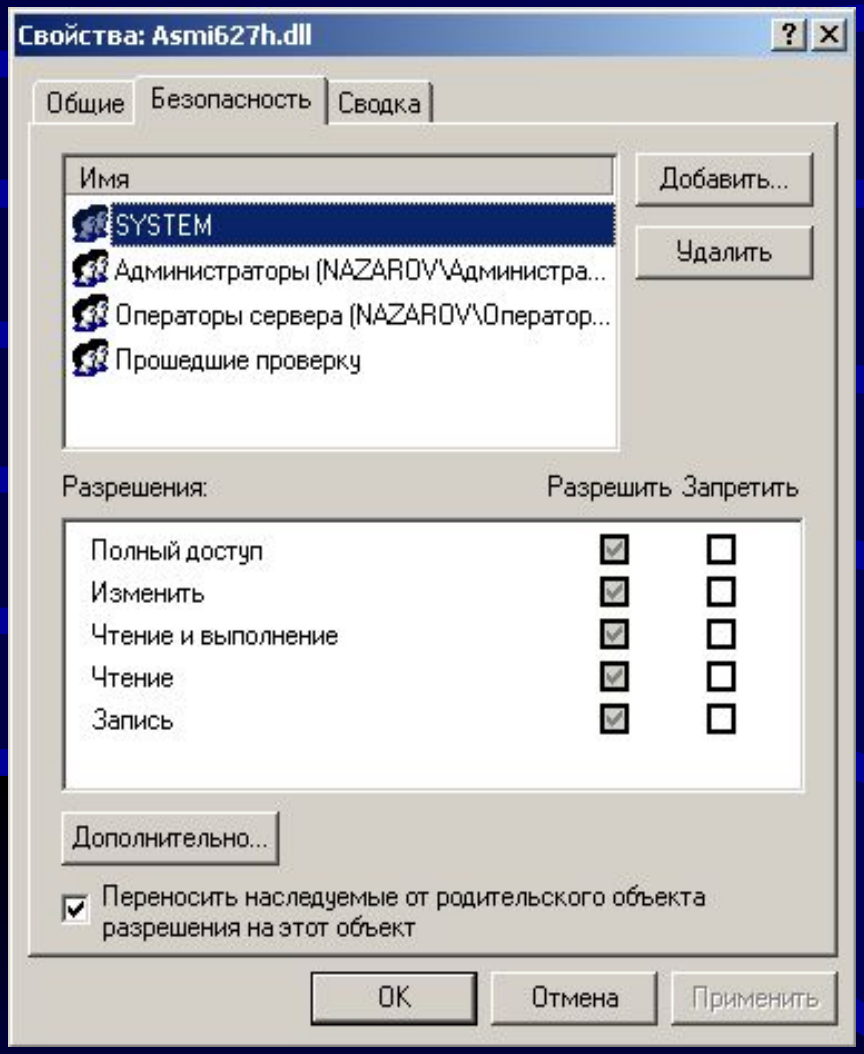
- Доступ к каталогам и файлам контролируется за счет установки соответствующих разрешений.
- Разрешения в Windows NT бывают индивидуальные и стандартные. Индивидуальные разрешения относятся к элементарным операциям над каталогами и файлами, а стандартные разрешения являются объединением нескольких индивидуальных разрешений.

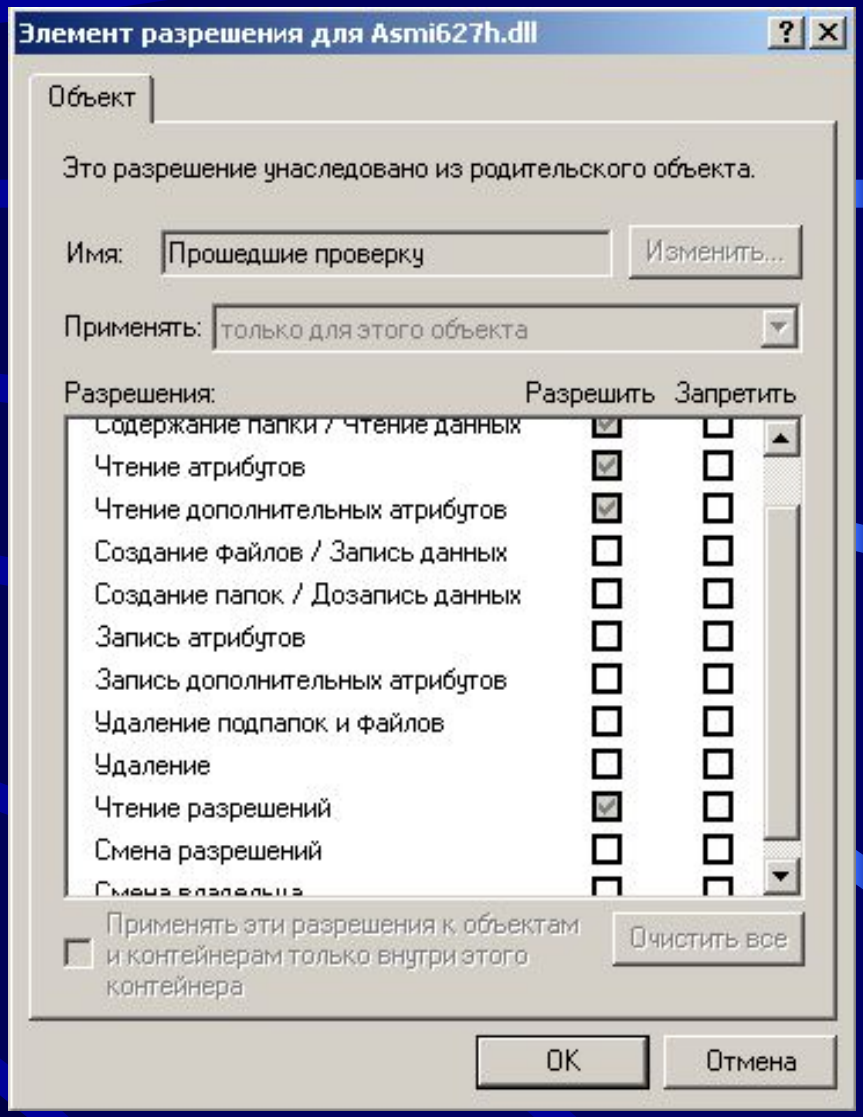
Разрешение	Для каталога	Для файла
Read (R)	Чтение имен файлов и каталогов, входящих в данный каталог, а также атрибутов и владельца каталога	Чтение данных, атрибутов, - имени владельца и разрешений файла
Write (W)	Добавление файлов и каталогов, изменение атрибутов каталога, чтение владельца и разрешений каталога	• Чтение владельца и разрешений файла, изменение атрибутов файла, изменение и добавление данных файла
Execute (X)	Чтение атрибутов каталога, выполнение изменений в каталогах, входящих в данный каталог, чтение имени владельца и разрешений каталога	Чтение атрибутов файла, имени владельца и разрешений. Выполнение файла, если он хранит код программы
Delete (D)	Удаление каталога	Удаление файла
Change Permission (P)	Изменение разрешений каталога	Изменение разрешений файла
Take Ownership (O)	Стать владельцем каталога	Стать владельцем файла

Для файлов в Windows определено четыре стандартных разрешения: No Access, Read, Change и Full Control, которые объединяют индивидуальные разрешения

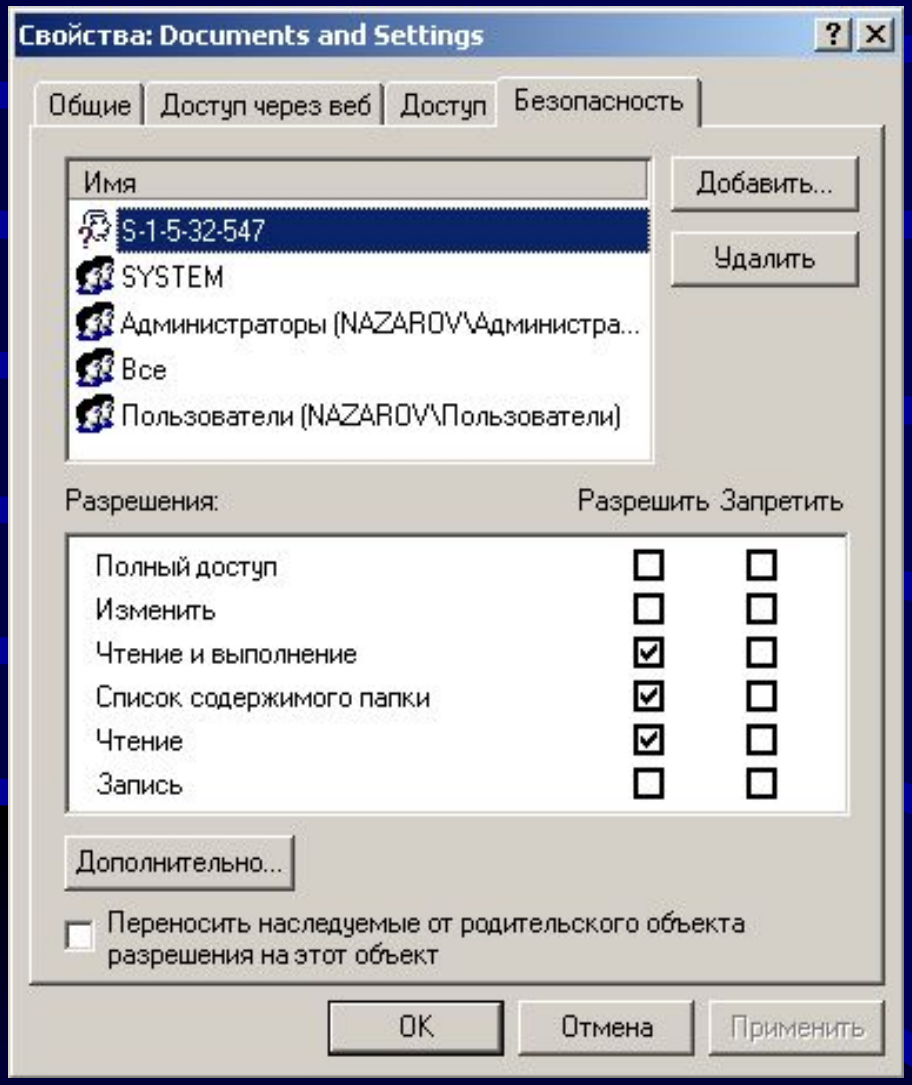
Стандартное разрешение	Индивидуальные разрешения
No Access	Ни одного
Read	RX
Change	RWXD
Full Control	Все

Разрешения на доступ к файлам

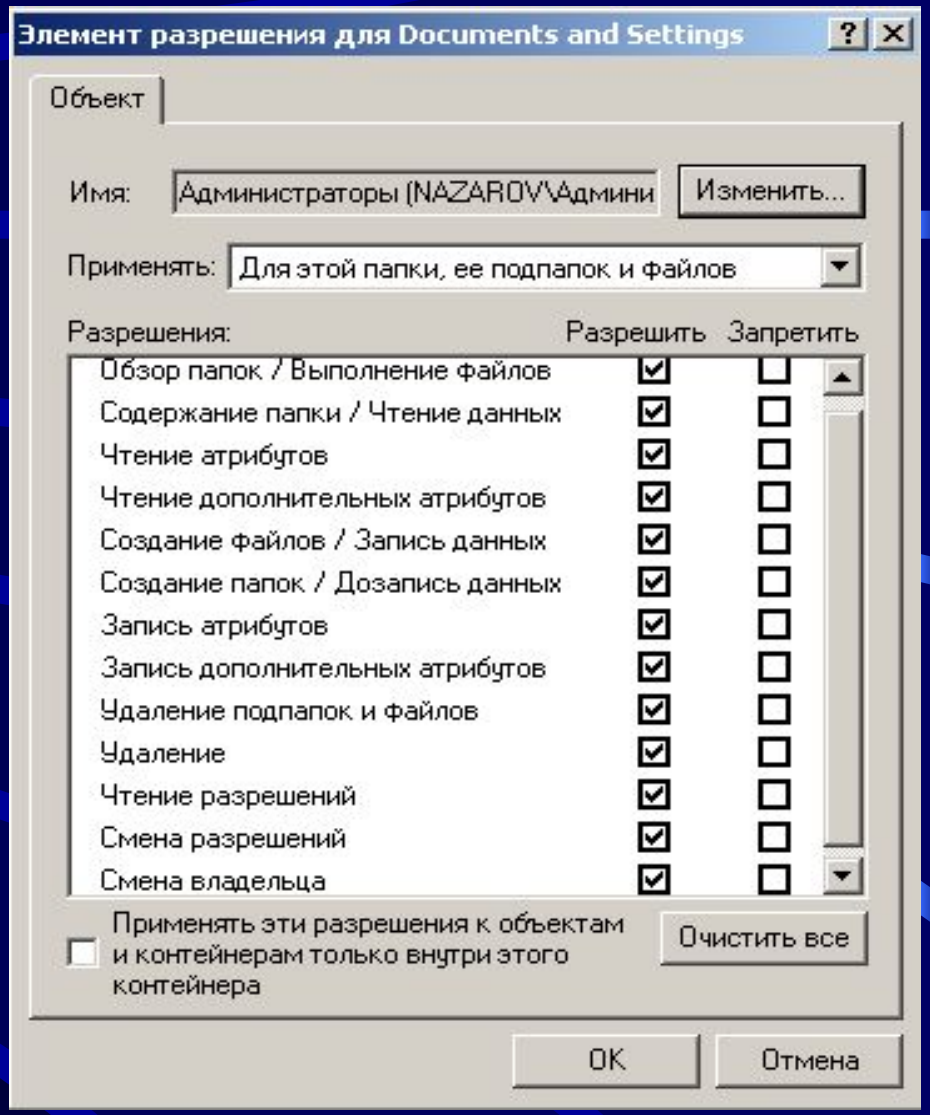




Разрешения на доступ к каталогам



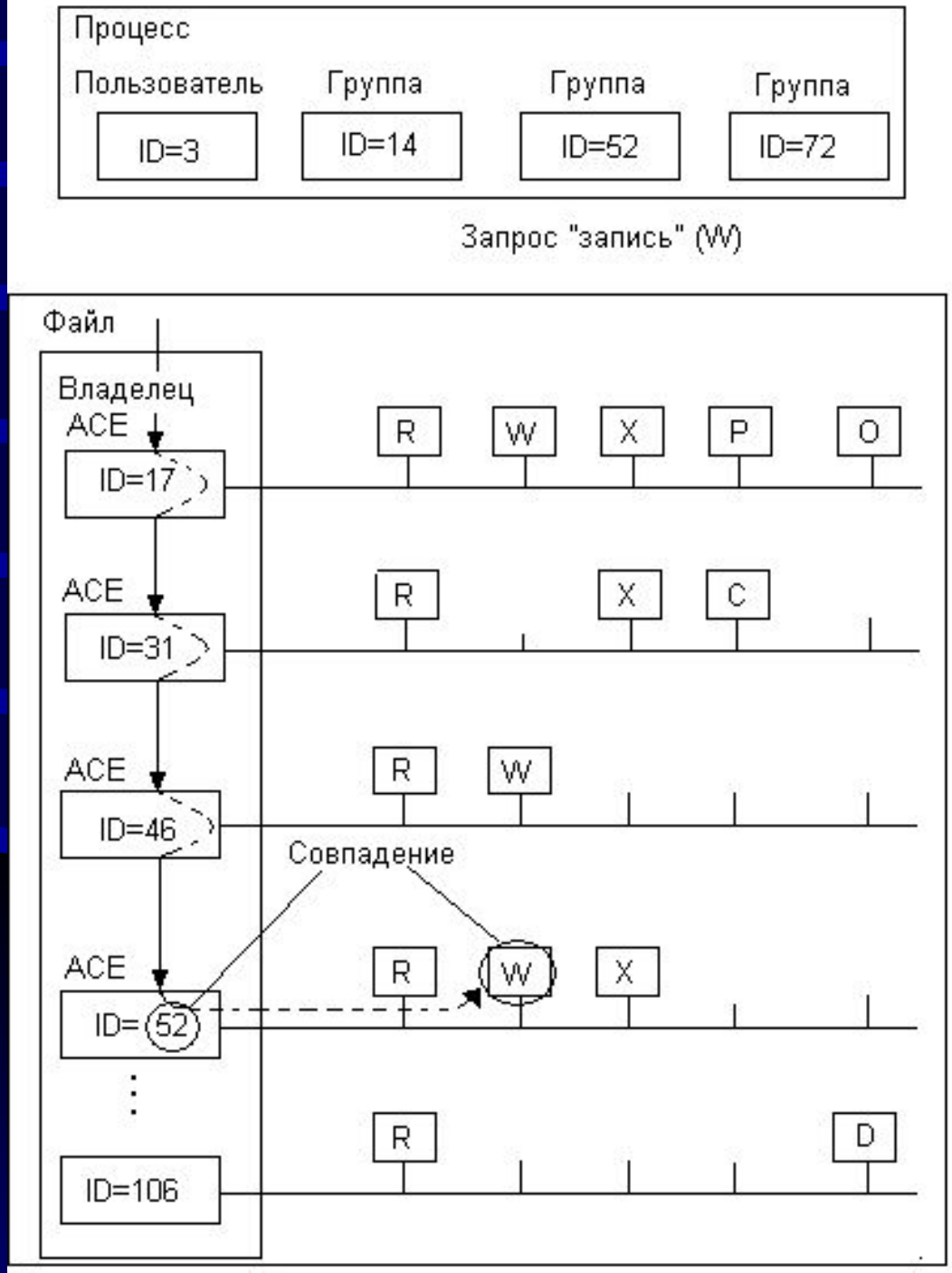
Стандартные разрешения



Специальные разрешения

Существует ряд правил, которые определяют действие разрешений.

- Пользователи не могут работать с каталогом или файлом, если они не имеют явного разрешения на это или же они не относятся к группе, которая имеет соответствующее разрешение.
- Разрешения имеют накопительный эффект, за исключением разрешения No Access, которое отменяет все остальные имеющиеся разрешения.


Процесс
Пользователь
Группа
Группа
Группа
ID=3
ID=14
ID=52
ID=72
Запрос "запись" (W)
Файл
Владелец
ACE
ID=17
R
W
X
P
O
ACE
ID=31
R
X
C
ACE
ID=46
R
W
Совпадение
ACE
ID= 52
R
W
X
R
D
ID=106

- В приведенном на рисунке примере процесс, который выступает от имени пользователя с идентификатором 3 и групп с идентификаторами 14, 52 и 72, пытается выполнить операцию записи (W) в файл. Файлом владеет пользователь с идентификатором 17. Операционная система, получив запрос на запись, находит характеристики безопасности файла (на диске или в буферной системной области) и последовательно сравнивает все идентификаторы процесса с идентификатором владельца файла и идентификаторами пользователей и групп в элементах ACE. В данном примере один из идентификаторов группы, от имени которой выступает процесс, а именно 52, совпадает с идентификатором одного из элементов ACE. Так как пользователю с идентификатором 52 разрешена операция чтения (признак W имеется в наборе операций этого элемента), то ОС разрешает процессу выполнение операции.

- Различают два основных подхода к определению прав доступа.
- **Избирательный доступ** имеет место, когда для каждого объекта сам владелец может определить допустимые операции с объектами. Этот подход называется также произвольным (от discretionary — предоставленный на собственное усмотрение) доступом, так как позволяет администратору и владельцам объектов определить права доступа произвольным образом, по их желанию. Между пользователями и группами пользователей в системах с избирательным доступом нет жестких иерархических взаимоотношений, то есть взаимоотношений, которые определены по умолчанию и которые нельзя изменить. Исключение делается только для администратора, по умолчанию наделяемого всеми правами.
-

- **Мандатный доступ** (от mandatory — обязательный, принудительный) — это такой подход к определению прав доступа, при котором система наделяет пользователя определенными правами по отношению к каждому разделяемому ресурсу (в данном случае файлу) в зависимости от того, к какой группе пользователь отнесен. От имени системы выступает администратор, а владельцы объектов лишены возможности управлять доступом к ним по своему усмотрению. Все группы пользователей в такой системе образуют строгую иерархию, причем каждая группа пользуется всеми правами группы более низкого уровня иерархии, к которым добавляются права данного уровня.
- Мандатная модель разграничения доступа предполагает назначение объекту метки (грифа) секретности, а субъекту – уровня допуска. Доступ субъектов к объектам в мандатной модели определяется на основании правил «*не читать выше*» и «*не записывать ниже*». Использование мандатной модели, в предотвращает утечку конфиденциальной информации, но снижает производительность компьютерной системы.