Операционные системы

Введение в операционные системы

Введение в операционные системы

Основные определения

Определение ОС

Операционная система (ОС) – комплекс системных программ, обеспечивающий оптимальное управление ресурсами вычислительной системы в соответствии с некоторым критерием эффективности.

Критерием эффективности ОС может быть, например, пропускная способность (число выполненных задач за единицу времени) или реактивность (время реакции на некоторое событие) системы.

Вычислительная система (ВС) – это взаимосвязанная совокупность аппаратных средств вычислительной техники и программного обеспечения, предназначенная для обработки информации.

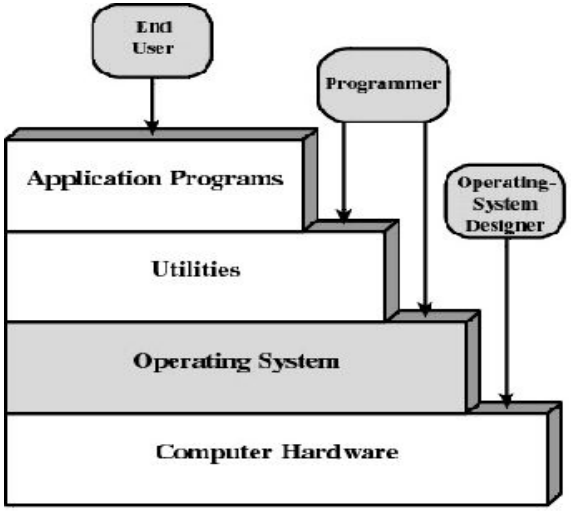
Уровни ВС

Уровень прикладных программ

Утилиты

Операционная система

Аппаратное обеспечение



Основная функция ОС

Основной функцией ОС является управление аппаратными ресурсами ВС и включает решение следующих, не зависящих от типа ресурса задач:

- планирование и удовлетворение запросов на ресурсы;
- отслеживание состояния ресурса;
- разрешение конфликтов.

Основные ресурсы ВС

- Процессорное время (процессор)
- Адресное пространство (оперативная память)
- Файлы (накопители данных)
- Внешние устройства ввода/вывода (принтеры, сетевые устройства, …)

Дополнительная функция ОС

Кроме основной функции управления ресурсами ВС, от ОС зачастую требуется решение еще одной важной задачи – предоставления программного интерфейса доступа к аппаратным ресурсам в виде некоторой виртуальной машины (программного и визуального интерфейсов), которую легче программировать и с которой легче работать, чем непосредственно с аппаратурой, составляющей реальную машину.

Мультипрограммирование: процессы и потоки

Мультипрограммирование, метод одновременно выполнения на одной ЭВМ нескольких программ, относящихся к различным задачам или различным ветвям одной и той же задачи.

- В настоящее время в большинстве ОС определены два типа единиц работы, между которыми разделяется процессор и другие ресурсы компьютера: процесс и поток.
- Процесс – абстракция, описывающая выполняющуюся программу. Для ОС процесс представляет собой единицу работы, заявку на потребление системных ресурсов. **Одним из основных ресурсов является адресное пространство процесса.**
- Поток (нить, thread) – последовательность выполнения инструкций процессора. Процесс в этом случае рассматривается ОС как заявка на потребление всех видов ресурсов, кроме одного – процессорного времени, которое ОС распределяет между потоками. Таким образом, **поток представляет собой мини-процесс, который работает в адресном пространстве породившего его процесса**.
- В простейшем случае процесс состоит из одного потока, и именно таким образом трактовалось понятие «процесс» до середины 80-х годов (например, в ранних версиях UNIX).

Мультипрограммирование: процессы и потоки

Мультипрограммирование, метод одновременно выполнения на одной ЭВМ нескольких программ, относящихся к различным задачам или различным ветвям одной и той же задачи.

- В настоящее время в большинстве ОС определены два типа единиц работы, между которыми разделяется процессор и другие ресурсы компьютера: процесс и поток.
- Процесс – абстракция, описывающая выполняющуюся программу. Для ОС процесс представляет собой единицу работы, заявку на потребление системных ресурсов. **Одним из основных ресурсов является адресное пространство процесса.**
- Поток (нить, thread) – последовательность выполнения инструкций процессора. Процесс в этом случае рассматривается ОС как заявка на потребление всех видов ресурсов, кроме одного – процессорного времени, которое ОС распределяет между потоками. Таким образом, **поток представляет собой мини-процесс, который работает в адресном пространстве породившего его процесса**.
- В простейшем случае процесс состоит из одного потока, и именно таким образом трактовалось понятие «процесс» до середины 80-х годов (например, в ранних версиях UNIX).

Введение в операционные системы

Классификация ОС

Признаки классификации

ОС могут различаться особенностями реализации внутренних алгоритмов управления основными ресурсами компьютера, особенностями использованных методов проектирования, типами аппаратных платформ, областями использования и многими другими свойствами.

Рассмотрим подробнее классификацию ОС по нескольким наиболее основным признакам:

- особенности алгоритмов управления ресурсами;
- особенности аппаратных платформ;
- особенности областей использования;
- структурная организация.

Классификация ОС

Особенности алгоритмов управления ресурсами

Поддержка многозадачности

По числу одновременно выполняемых задач ОС могут быть разделены на два класса:

- однозадачные (например, MS-DOS, MSX);
- многозадачные (ОС ЕС, UNIX, Windows 9x, NT).

Однозадачные ОС в основном выполняют функцию предоставления пользователю виртуальной машины.

Многозадачные ОС поддерживают в том или ином виде **мультипрограммирование** и управляют разделением совместно используемых ресурсов (процессор, оперативная память, файлы и пр.).

Поддержка многопользовательского режима

По числу одновременно работающих пользователей ОС делятся на:

- однопользовательские (MS-DOS, Windows 3.x);
- многопользовательские (UNIX, Windows NT).

Главным отличием многопользовательских систем от однопользовательских является **наличие средств защиты информации** каждого пользователя от несанкционированного доступа других пользователей.

Вытесняющая и не вытесняющая многозадачность

Способ распределения процессорного времени между несколькими одновременно существующими в системе задачами (процессами или потоками) в режиме мультипрограммирования во многом определяет специфику ОС.

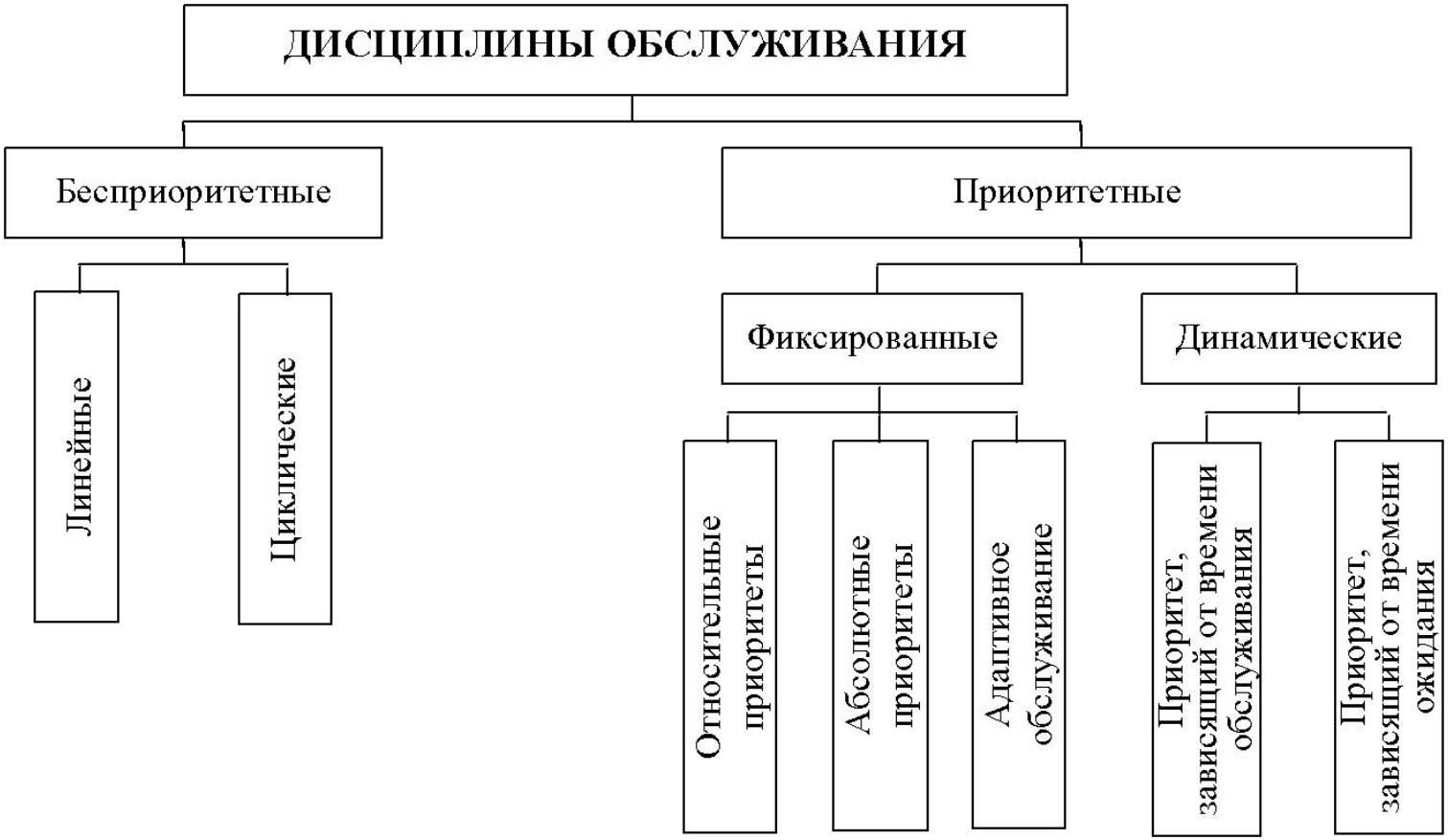
Среди множества существующих вариантов реализации многозадачности можно выделить две группы алгоритмов:

- невытесняющая (корпоративная) многозадачность (NetWare, Windows 3.x);
- вытесняющая многозадачность (Windows NT, OS/2, UNIX).

Вытесняющая и не вытесняющая многозадачность

- При невытесняющей многозадачности активный процесс (поток) выполняется до тех пор, пока он сам, по собственной инициативе, не отдаст управление ОС для того, чтобы та выбрала из очереди другой готовый к выполнению процесс (поток).
- При вытесняющей многозадачности решение о переключении процессора с одного процесса (потока) на другой принимается ОС.

Классификация дисциплин обслуживания



Классификация дисциплин обслуживания

- *Бесприоритетные ДО* – выбор из очереди производится без учета относительной важности задач и времени их обслуживания.
- *Приоритетное обслуживание* – отдельным задачам предоставляется преимущественное право перейти в состояние ВЫПОЛНЕНИЯ.
- *Фиксированные приоритеты* – являются величиной постоянной на всем жизненном цикле процесса.
- *Динамические приоритеты* – изменяются в зависимости от некоторых условий в соответствии с определенными правилами.

Поддержка многопоточности

Важным свойством операционных систем является возможность распараллеливания вычислений в рамках одного процесса.

Многопоточная ОС разделяет процессорное время не между процессами, а между их отдельными нитями (потоками).

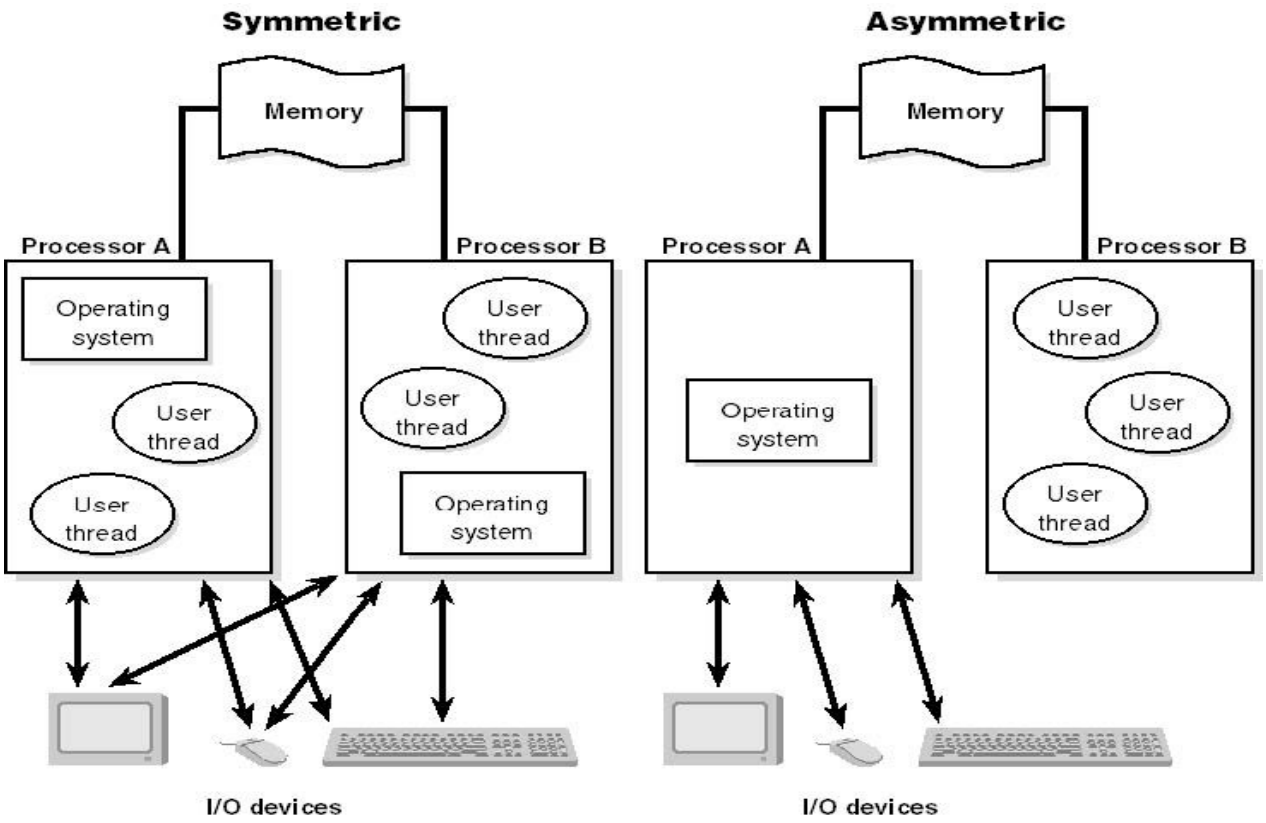
Многопроцессорная обработка

Другим важным свойством ОС является отсутствие или наличие в ней средств поддержки многопроцессорной обработки. Мультипроцессирование приводит к усложнению всех алгоритмов управления ресурсами.

В наши дни становится общепринятым введение в ОС функций поддержки многопроцессорной обработки данных. Такие функции имеются в операционных системах Solaris фирмы Sun, Windows NT-2000 фирмы Microsoft и NetWare фирмы Novell.

Многопроцессорные ОС могут классифицироваться по способу организации вычислительного процесса: асимметричные ОС и симметричные ОС.

Виды мультипроцессирования



Особенности алгоритмов управления ресурсами

Выше были рассмотрены характеристики ОС, связанные с управлением только одним типом ресурсов – процессором. Важное влияние на облик операционной системы в целом, на возможности ее использования в той или иной области оказывают особенности и других подсистем управления локальными ресурсами - подсистем управления памятью, файлами, устройствами ввода-вывода.

Классификация ОС

Особенности областей использования

Типы многозадачных ОС

Многозадачные ОС подразделяются на три типа в соответствии с использованными при их разработке критериями эффективности:

- системы пакетной обработки (например, ОС ЕС);
- системы разделения времени (UNIX, MS Windows);
- системы реального времени (QNX, RT/11).

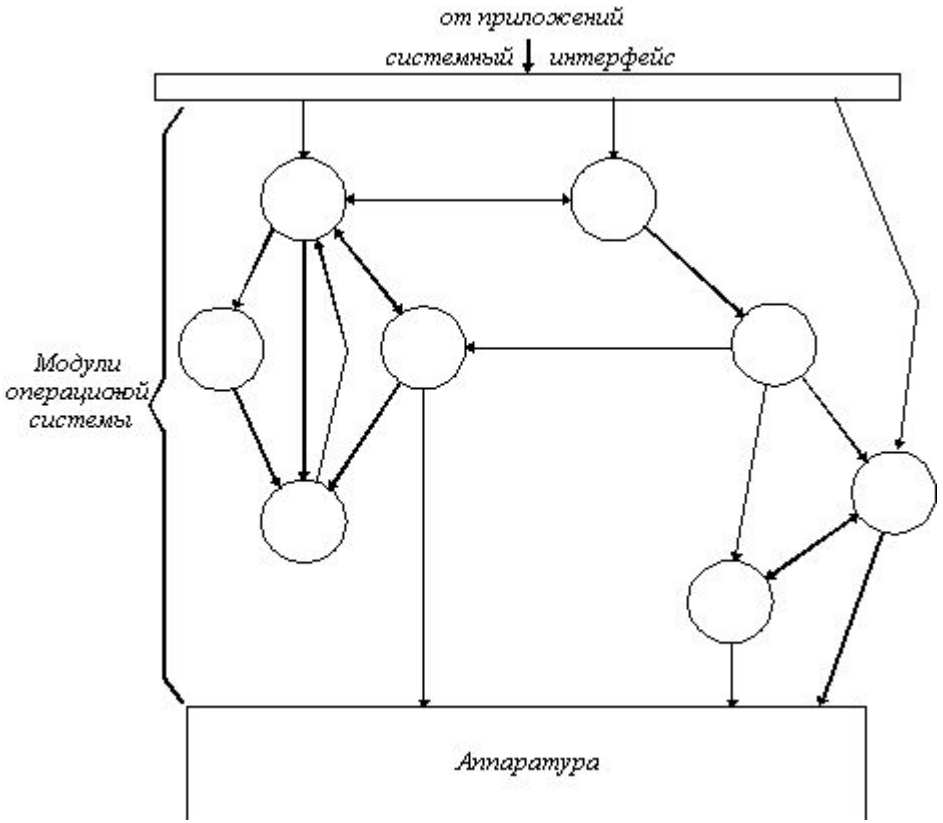
Другие системы

Некоторые операционные системы могут совмещать в себе свойства систем разных типов, например, часть задач может выполняться в режиме пакетной обработки, а часть – в режиме реального времени или в режиме разделения времени. В таких случаях режим пакетной обработки часто называют фоновым режимом.

Операционные системы

Структурная организация операционных систем

Монолитная структура

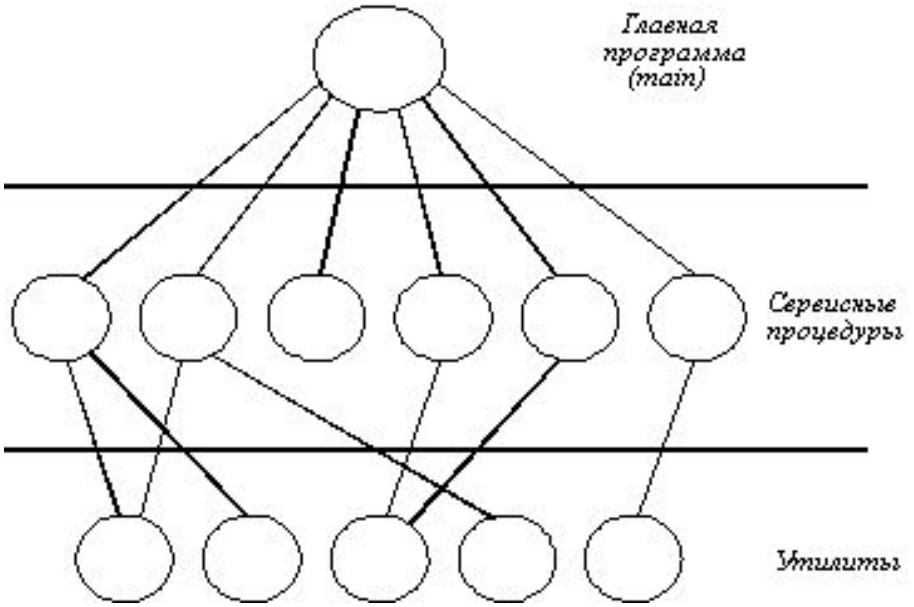


Наиболее простым и распространенным способом построения ОС является **монолитное ядро**, которое компонуется как одна программа, работающая в привилегированном режиме и использующая быстрые переходы с одной процедуры на другую, не требующие переключения из привилегированного режима в пользовательский и наоборот.

Многоуровневая структура

Развитием монолитного подхода является **многоуровневый**, когда ОС реализуется как иерархии уровней.

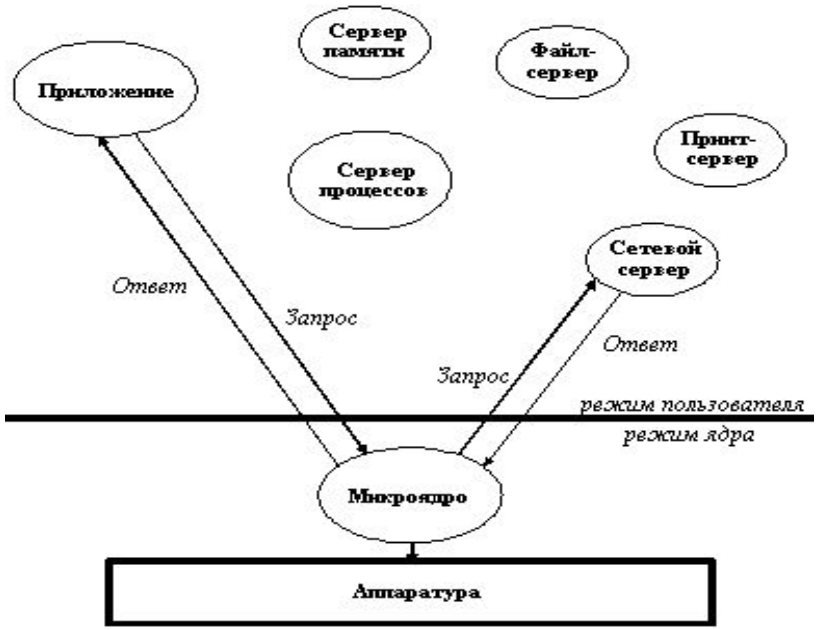
Уровни образуются группами функций ОС – файловая система, управление процессами и устройствами и т.п.



Каждый уровень может взаимодействовать только со своим непосредственным соседом – выше- или нижележащим уровнем.

Микроядерная структура

Альтернативой является построение ОС **на базе модели клиент-сервер и тесно связанной с ней концепции микроядра**. Микроядро работает в привилегированном режиме и выполняет только минимум функций по управлению аппаратурой, в то время как функции ОС более высокого уровня выполняют специализированные компоненты ОС – серверы, работающие в пользовательском режиме.



Объектно-ориентированный подход

Развитием технологии расширяемых модульных систем является **объектно-ориентированный подход**, при котором каждый программный компонент ОС является функционально изолированным от других. Основным понятием этого подхода является “объект”.

Объект – это единица программ и данных, взаимодействующая с другими объектам посредством приема и передачи сообщений. Объект может быть представлением как некоторых конкретных вещей – прикладной программы или документа, так и некоторых абстракций – процесса, события.

Программы (функции) объекта определяют перечень действий, которые могут быть выполнены над данными этого объекта. Объект-клиент может обратиться к другому объекту, послав сообщение с запросом на выполнение какой-либо функции объекта-сервера.

ООП: достоинства и недостатки

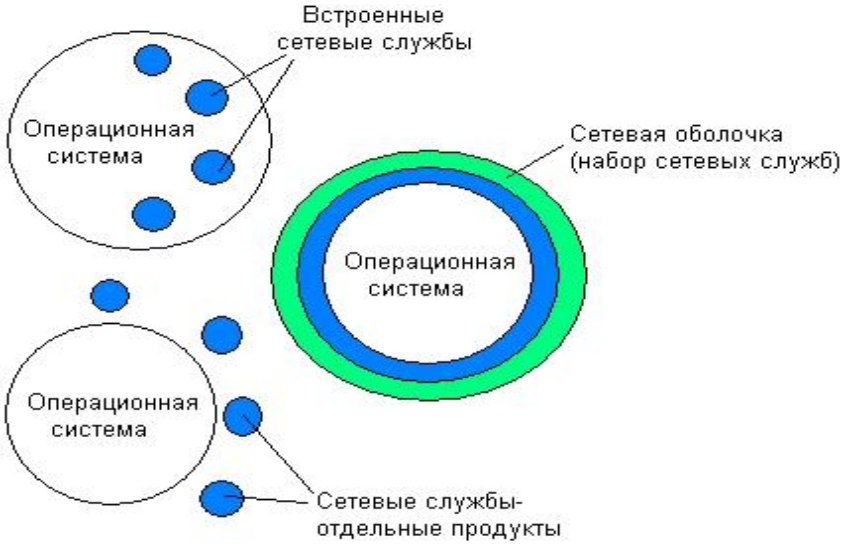
Построение ОС на базе объектно-ориентированного подхода имеет следующие достоинства:

- аккумуляция удачных решений в форме стандартных объектов и создание новых объектов на их базе с помощью механизма наследования;
- предотвращение несанкционированного доступа к данным за счет их инкапсуляции во внутренние структуры объекта;
- структурированность системы, состоящей из набора хорошо определенных объектов.

В качестве основных недостатков объектно-ориентированного похода следует выделить сложность управления объектами и как следствие более медленную работу системы.

Сетевые службы и сервисы

- сетевые службы глубоко встроены в ОС;
- сетевые службы объединены в виде некоторого набора – оболочки;
- сетевые службы производятся и поставляются в виде отдельного продукта.



Операционные системы

Эволюция операционных систем

Этапы эволюции

- 1 этап (1940-60)
  системный монитор, ранние пакетные системы
- 2 этап (1965-75)
  мультипрограммирование, пакетные ОС и ОС разделения времени
- 3 этап (1970-80) ОС мини-ЭВМ
- 4 этап (1980-90) ОС ПК
- 5 этап (1990-наст.вр.) корпоративные ОС

1 этап (1940-60)

- Середина 40-х XX-века – первые ламповые вычислительные устройства. ОС еще не появились, все задачи организации вычислительного процесса решались программистом вручную с пульта управления.
- С середины 50-х годов – новая техническая база – полупроводниковые элементы:
  - выросли технические характеристики ЭВМ: быстродействие процессоров, объемы оперативной и внешней памяти, надежность;
  - появились первые **алгоритмические языки**, и появился новый тип системного программного обеспечения – трансляторы;
  - были разработаны первые системные управляющие программы – **мониторы**.
- **Программные мониторы** – прообраз современных ОС, первые системные программы, предназначенные для управления вычислительным процессом.
- Программные мониторы предоставляли пакетный режим обслуживания на базе **язык управления заданиями**, с помощью которого программист сообщал системе и оператору, какие действия и в какой последовательности он хотел бы выполнить на ЭВМ. Типовой набор директив обычно включал признак начала отдельной работы, вызов транслятора, вызов загрузчика, признаки начала и конца исходных данных. Оператор составлял **пакет заданий**, которые в дальнейшем без его участия последовательно запускались на выполнение монитором. Кроме того, монитор был способен самостоятельно обрабатывать наиболее распространенные аварийные ситуации.
- Ранние системы пакетной обработки **значительно сократили затраты времени** на вспомогательные действия по организации вычислительного процесса и способствовали **повышению эффективности** использования компьютеров.

2 этап (1965-75)

- 1965-1975 годы переход к интегральным микросхемам, новое поколение ЭВМ – IBM/360, многопроцессорная ЭВМ для централизованных вычислений.
- Реализованы основные концепции, присущие современным ОС:
  - мультипрограммирование,
  - мультипроцессирование,
  - многотерминальный режим,
  - виртуальная память,
  - файловые системы,
  - разграничение доступа и сетевая работа.
- Мультипрограммирование было реализовано в двух вариантах – пакетная обработка и разделение времени.
- Для поддержания удаленной работы терминалов в ОС появились специальные программные модули, реализующие различные (в то время, как правило, нестандартные) протоколы связи. Поэтому эти ОС можно считать прообразом современных сетевых ОС.
- 1965-69 годы – разработка фирмами Bell Telephone Lab., General Electric и Массачусетским технологическим институтом новой многозадачной ОС – **Multics** (MULTiplexed Information and Computing Service), которая была потом переименована на UNIX.

2 этап – многотерминальные системы

- Многотерминальный режим использовался не только в системах разделения времени, но и в системах пакетной обработки. При этом не только оператор, но и все пользователи получали возможность формировать свои задания и управлять их выполнением со своего терминала. Такие ОС получили название **систем удаленного ввода заданий**.
- Для поддержки удаленной работы терминалов в ОС появились специальные программные модули, реализующие различные (как правило, нестандартные) протоколы связи. Такие ВС с **удаленными терминалами**, сохраняя централизованный характер обработки данных, в какой-то степени являлись прообразом современных компьютерных сетей, а соответствующее системное ПО – прообразом сетевых ОС.

3 этап (1970-80)

- Начало 70-х годов – первые сетевые ОС, которые в отличие от многотерминальных ОС позволяли не только рассредоточить пользователей, но и организовать распределенное хранение и обработку данных между несколькими компьютерами, связанными сетью.
- 1969 год – начало работ Министерства обороны США по объединению суперкомпьютеров оборонных и научно-исследовательских центров в единую сеть ARPANET, которая явилась отправной точкой для создания глобальной сети Интернет.
- Середина 70-х годов – широкое распространение получили мини-ЭВМ (PDP-11, Nova, HP) на базе технологии БИС, которая позволила реализовать достаточно мощные функции при сравнительно невысокой стоимости компьютера. Архитектура мини-ЭВМ была значительно упрощена по сравнению с мэйнфреймами, что нашло отражение и в их ОС. Многие функции мультипрограммных многопользовательских ОС мэйнфреймов были усечены, учитывая ограниченность ресурсов мини-компьютеров.
- ОС мини-компьютеров часто стали делать специализированными, например, только для управления в реальном времени (ОС RT-11 для PDP-11) или только для поддержания режима разделения времени (RSX-11M для PDP-11). Эти ОС не всегда были многопользовательскими, что во многих случаях оправдывалось невысокой стоимостью компьютеров.

4 этап (1980-90)

- Постоянное развитие версий ОС UNIX для ЭВМ различных архитектур.
- Начало 80-х годов – появление **персональных компьютеров** (ПК), которые стали мощным катализатором для бурного роста ЛВС, в результате чего поддержка сетевых функций стала для ОС ПК необходимым условием.
- Также в 80-е годы – приняты основные стандарты на коммуникационные технологии для ЛВС (например, Ethernet). Это позволило обеспечить совместимость сетевых ОС на нижних уровнях, а также стандартизовать интерфейс ОС с драйверами сетевых адаптеров.
- 1981 год – первая ОС компании Microsoft для ПК. **MS-DOS** было однопрограммной однопользовательской ОС с интерфейсом командной строки. Недостающие функции MS-DOS (например, интерфейсные и сетевые) компенсировались внешними программами. Начиная с MS-DOS v3.1 к файловой системе добавились необходимые для сетевой работы средства блокировки файлов и записей (совместная работа пользователей).
- 1983 год – первая сетевая ОС компании **Novell OS-Net** для сетей со звездообразной топологией. После выпуска фирмой IBM ПК типа PC XT, компания Novell разработала сетевую ОС **NetWare 86** для ПК.
- 1987 год – Microsoft и IBM выпустили первую многозадачную ОС **OS/2** для ПК на базе МП Intel 80286. Эта ОС поддерживала вытесняющую многозадачность, многопоточность, виртуальную память, графический пользовательский интерфейс и виртуальную машину для выполнения DOS-приложений.
- Начиная с МП Intel 80286 с поддержкой мультипрограммирования, перенос ОС UNIX на ПК, например, версия UNIX компании Santa Cruz Operation (**SCO UNIX**).

5 этап (1990 – …)

- 90-е годы – практически все ОС стали сетевыми. Сетевые функции встраиваются в ядро ОС, являясь ее неотъемлемой частью.
- Появились специализированные ОС, которые предназначены исключительно для выполнения коммуникационных задач. Например, сетевая ОС IOS компании Cisco Systems, работающая в маршрутизаторах, организует в мультипрограммном режиме выполнение набора программ, каждая из которых реализует один из коммуникационных протоколов.
- Вторая половина 90-х годов – особая поддержка со стороны ОС средств работы с Интернетом.
- Понятие корпоративная сетевая ОС. Корпоративная ОС отличается способностью хорошо и устойчиво работать в крупных сетях, которые характерны для больших предприятий, имеющих отделения в десятках городов и, возможно, в разных странах. Таким сетям органически присуща высокая степень гетерогенности программных и аппаратных средств, поэтому корпоративная ОС должна взаимодействовать с ОС разных типов и работать на различных аппаратных платформах. К настоящему времени достаточно явно определилась тройка лидеров в классе корпоративных ОС – это Novell NetWare 4-6, Microsoft Windows NT-2000, а также UNIX-системы различных производителей аппаратных платформ.

Операционные системы

Архитектура Windows NT-2000

Архитектура Windows NT-2000

Основная характеристика ОС Windows NT-2000

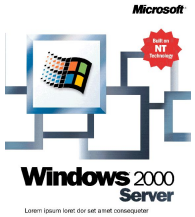
Основная характеристика Windows NT-2000

Система Windows NT-2003 не является дальнейшим развитием ранее существовавших продуктов. Ее архитектура создавалась с нуля с учетом предъявляемых к современной ОС требований:

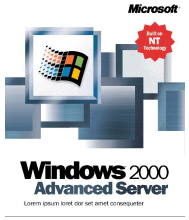
- *совместимость (compatible);*
- *переносимость (portability);*
- *масштабируемость (scalability);*
- *безопасность (security);*
- *распределенная обработка (distributed processing);*
- *надежность и отказоустойчивость (reliability and robustness);*
- *локализации (localization);*
- *расширяемость (extensibility).*

Семейство Windows 2000

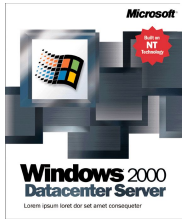
- **Windows 2000 Professional**
  - До 2-х ЦПУ, 4 ГБ ОЗУ



- **Windows 2000 Server**
  - До 4-х ЦПУ, 4 ГБ ОЗУ



- **Windows 2000 Advanced Server**
  - До 8-х ЦПУ, 8 ГБ ОЗУ
  - 2-узловая кластеризация и балансировка загрузки

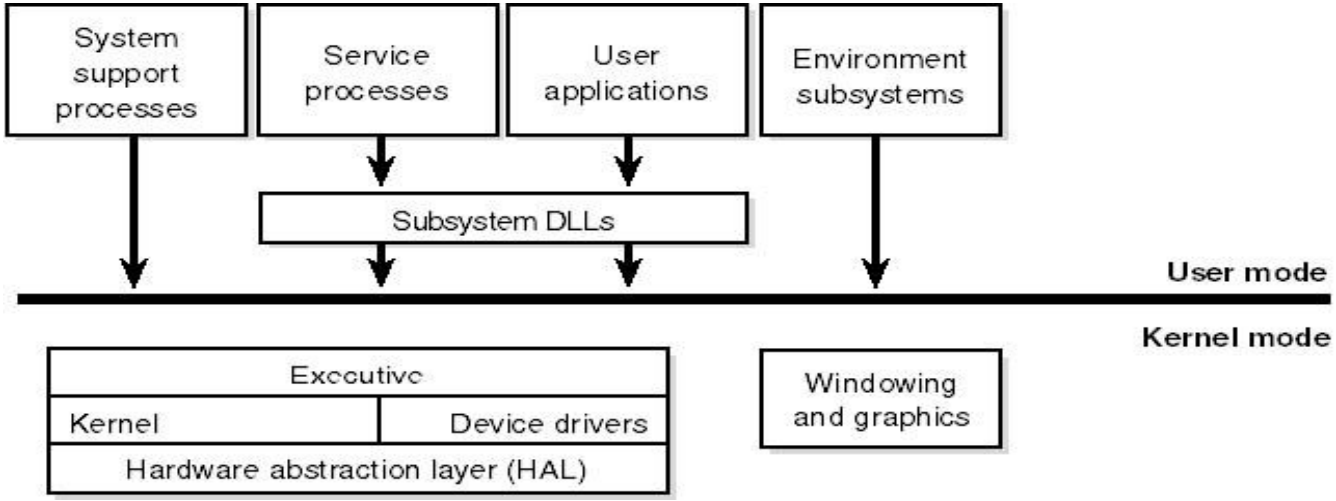


- **Windows 2000 Datacenter Server**
  - До 32-х ЦПУ, 64 ГБ ОЗУ
  - 4-узловая кластеризация

Архитектура Windows NT-2000

Архитектура ОС Windows NT-2000

Упрощенная архитектура Windows 2000



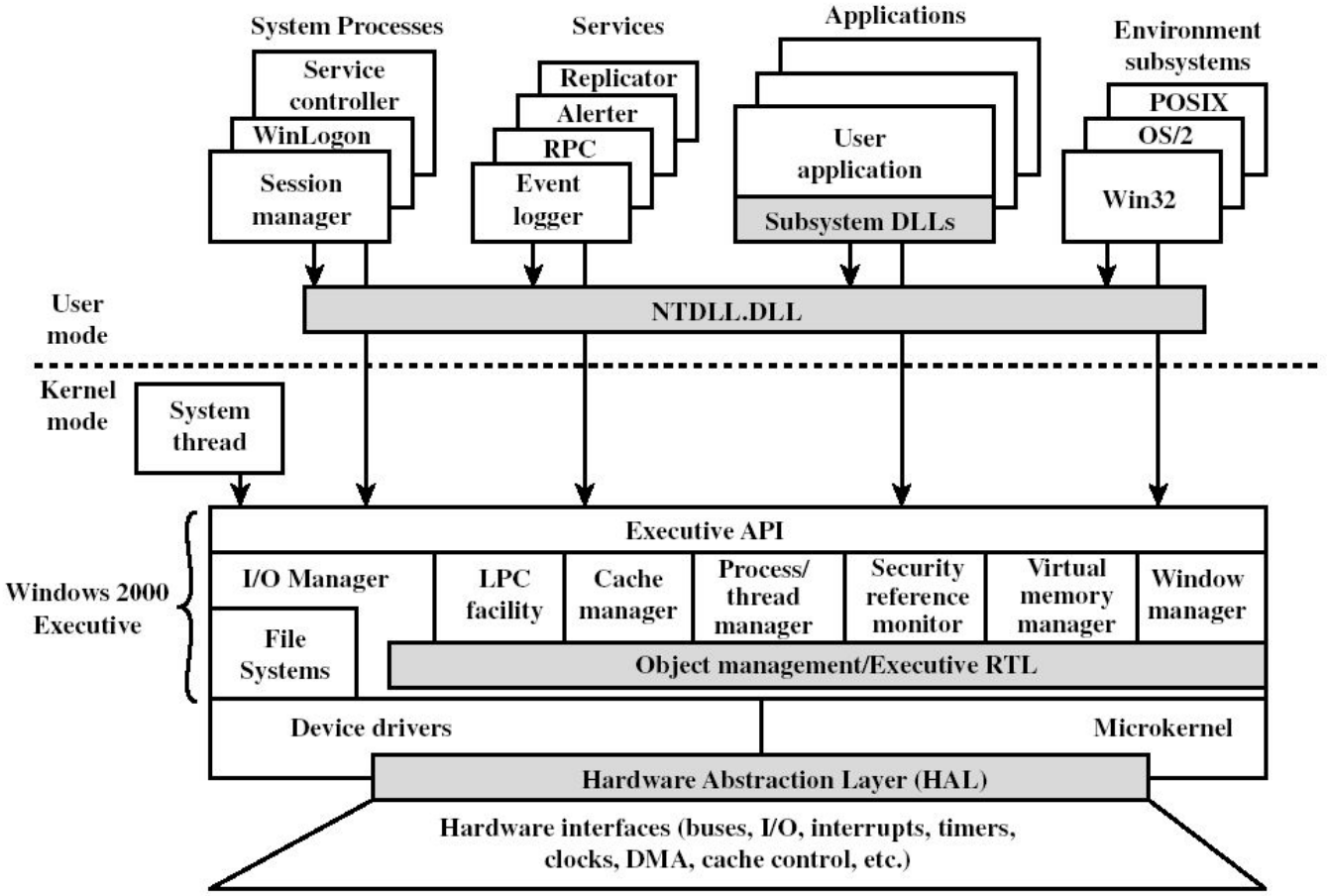
Режим ядра

- **исполняющая система NT,** которая включает управление памятью, процессами, потоками, безопасностью, вводом/выводом, межпроцессорными обменами;
- **ядро (микроядро) Windows NT** выполняет низкоуровневые функции ОС: диспетчеризация потоков, прерываний и исключений, синхронизация процессоров. Ядро также включает набор процедур и базовых объектов, используемый исполняемой частью для создания высокоуровневых конструкций;
- **уровень абстракции от оборудования (HAL – Hardware Abstraction Layer),** изолирует ядро, драйверы устройств и исполняемую часть NT от аппаратных платформ, на которых должна работать ОС. Подобный подход позволяет обеспечить переносимость Windows NT.
- **драйверы устройств** включают как файловую систему, так и аппаратные драйверы, которые транслируют пользовательские вызовы функций ввода/вывода в запросы физических устройств ввода/вывода;
- **функции графического интерфейса пользователя** работают с окнами, элементами управления и рисунками.

Пользовательский режим

- **Специальные процессы поддержки системы,** например, процесс регистрации пользователя и менеджер сессий, которые не являются службами NT.
- **Процессы сервера, которые являются службами NT** (аналог демонов в ОС Unix). Примером может быть регистратор событий (Event Logger). Многие дополнительно устанавливаемые приложения, такие как Microsoft SQL Server и Exchange Server, также включают компоненты, работающие как службы NT.
- **Подсистемы среды** представляют собой защищенные серверы пользовательского режима (user-mode), которые обеспечивают выполнение и поддержку приложений, разработанных для различного операционного окружения (различных ОС). Примером подсистем среды могут служить подсистемы Win32, Posix и OS/2 2.1.
- **Пользовательские приложения** одного из пяти типов: Win32, Windows 3.1, MS-DOS, Posix или OS/2 1.2.

Подробная архитектура Windows 2000



Исполняющая система

- **Менеджер процессов и потоков управляет процессами и потоками.** Фактически потоки и процессы поддерживаются в NT нижележащим слоем. Исполняемая часть добавляет дополнительную семантику и функции к этим объектам нижнего уровня.
- **Менеджер виртуальной памяти** использует схему управления, при которой каждый процесс получает собственное достаточно большое адресное пространство, защищенное от воздействия других процессов. Менеджер памяти также обеспечивает низкоуровневую поддержку для менеджера кэш-памяти.
- **Монитор безопасности** проводит политику обеспечения мер безопасности на локальном компьютере, охраняя системные ресурсы и выполняя процедуры аудита и защиты объектов.
- **Система ввода/вывода** использует независимый от устройств ввод/вывод и отвечает за пересылку данных соответствующим драйверам для дальнейшей обработки.
- **Менеджер кэш-памяти** улучшает производительность системы ввода/вывода файлов, размещая читаемые с диска данные в основной памяти для ускорения доступа к ним, а также откладывая на короткое время запись измененных данных на диск.
- **Менеджер объектов,** который создает, удаляет объекты и абстрактные типы данных, а также управляет ими. Объекты используются в Windows NT для представления таких ресурсов операционной системы, как процессы, потоки и объекты синхронизации.
- **LPC** (Local Procedure Call) передает сообщения между клиентским процессом и процессом сервера на том же самом компьютере. По сути, LPC – это оптимизированная версия известной процедуры удаленного вызова RPC (Remote Procedure Call).
- Широкий **набор библиотечных функций общего типа:** обработка строк, арифметические операции, преобразование типов данных, обработка структур.
- **Процедуры распределения памяти,** взаимообмен между процессами через память, два специальных типа объектов синхронизации – ресурсы и объекты fast mutex.

Ядро (микроядро) Windows NT

Ядро (Microkernel) является основой модульного строения ОС и координирует выполнение большинства базовых операций Windows NT. Программное обеспечение ядра выполняется полностью в привилегированном режиме, является неперемещаемым и невыгружаемым.

Ядро, в первую очередь, занимается планированием действий процессора. В случае если компьютер содержит несколько процессоров, ядро может выполнятся на всех процессорах (SMP) и синхронизирует их работу с целью достижения максимальной производительности системы. Ядро осуществляет диспетчеризацию потоков (threads), таким образом, чтобы максимально загрузить процессоры системы и обеспечить первоочередную обработку потоков с более высоким приоритетом.

Ядро также обеспечивает работу других базовых объектов ядра, которые используются исполняющей системой (и в некоторых случаях экспортируются в режим пользователя).

Архитектура Windows NT-2000

Объекты Windows NT-2000

Понятие объекта

В ОС Windows NT-2000 объект – это отдельный экземпляр периода выполнения (runtime instance) статически определенного типа объекта.

Тип объектов (object type), иногда называемый *классом объектов* (object class) состоит из общесистемного типа данных, функций, оперирующих экземплярами этого типа данных, и набора атрибутов.

Атрибут объекта *(object attribute)* – это поле данных внутри объекта частично определяющее его состояние.

Методы объекта (средства для манипулирования объектами) обычно считывают атрибуты объекта.

Назначение объектов

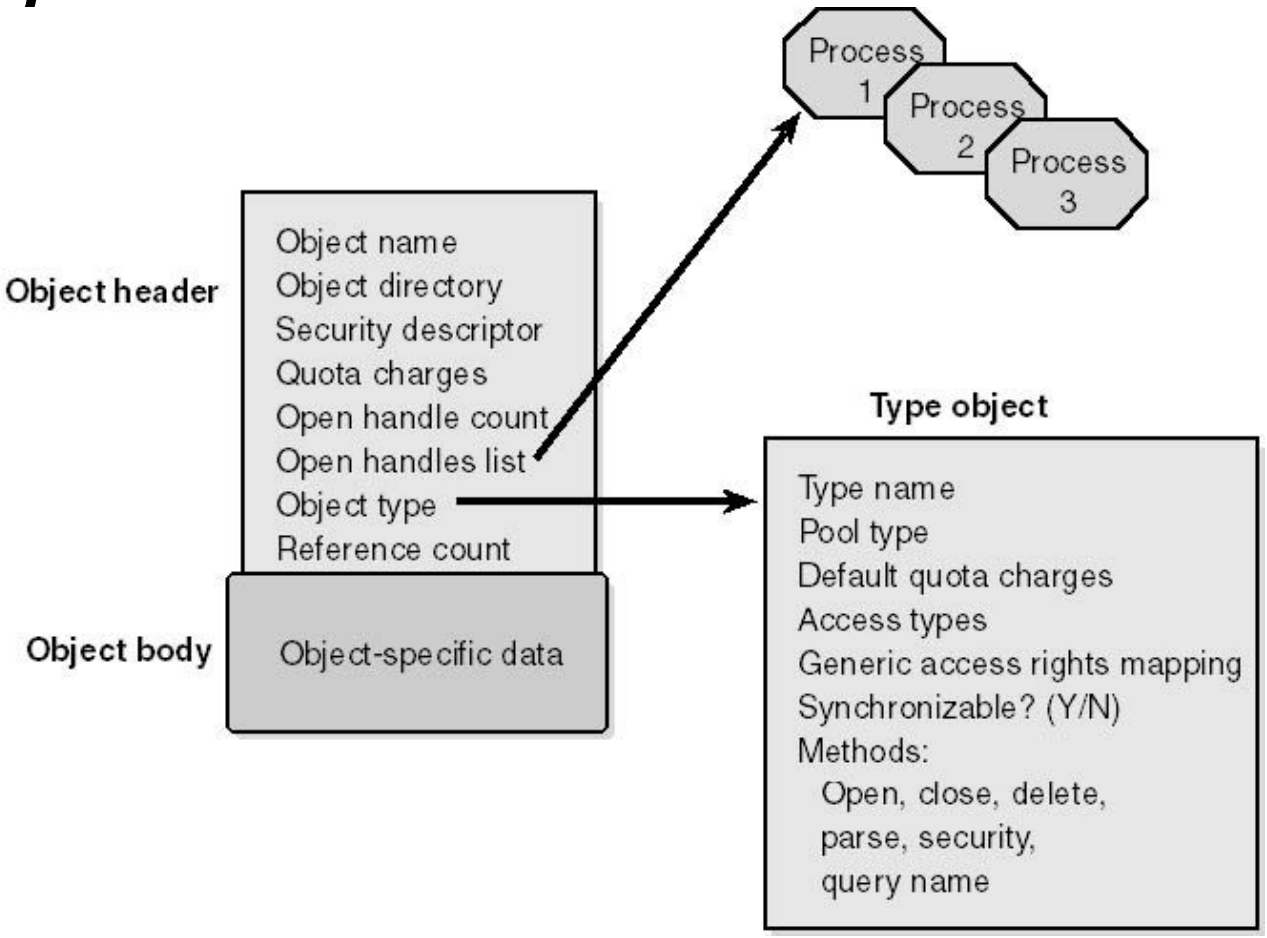
Объекты очень удобны для поддержки четырех важных функций ОС:

- присвоения понятных имен системным ресурсам;
- разделения ресурсов и данных между процессами;
- защиты ресурсов от несанкционированного доступа;
- учета ссылок (благодаря этому система узнает, когда объект больше не используется, и автоматически уничтожает его).

Типы объектов Windows 2000

- **Объекты исполнительной системы** (executive object) представляются различными компонентами исполнительной системы. Они доступны программам пользовательского режима (защищенным подсистемам) посредством базовых сервисов и могут создаваться и использоваться как подсистемами, так и исполнительной системой.
- **Объекты ядра** (kernel object) – это более примитивный набор объектов, реализованный ядром. Эти объекты невидимы коду пользовательского режима, а создаются и используются только внутри исполнительной системы.

Структура объектов Windows 2000



Структура объектов Windows 2000

Имя объекта	Делает объект видимым другим процессам для совместного использования
Каталог объектов	Обеспечивает иерархическую структуру, в которой хранятся имена объектов
Дескриптор безопасности	Определяет, кто и каким образом может использовать данный объект
Расход квоты	Задает квоту на использование ресурсов, которая списывается с процесса при открытии описателя данного объекта
Счетчик открытых дескрипторов	Подсчитывает количество открытых дескрипторов данного объекта
Список открытых дескрипторов	Содержит список процессов, открывших дескрипторы данного объекта
Временный/ постоянный статус	Указывает, можно ли уничтожить имя и освободить память объекта, если он более не используется
Режим: пользовательский/ ядра	Определяет доступность объекта в пользовательском режиме
Указатель на типовой объект	Ссылается на типовой объект, который содержит атрибуты, общие для набора однотипных объектов

Примеры объектов

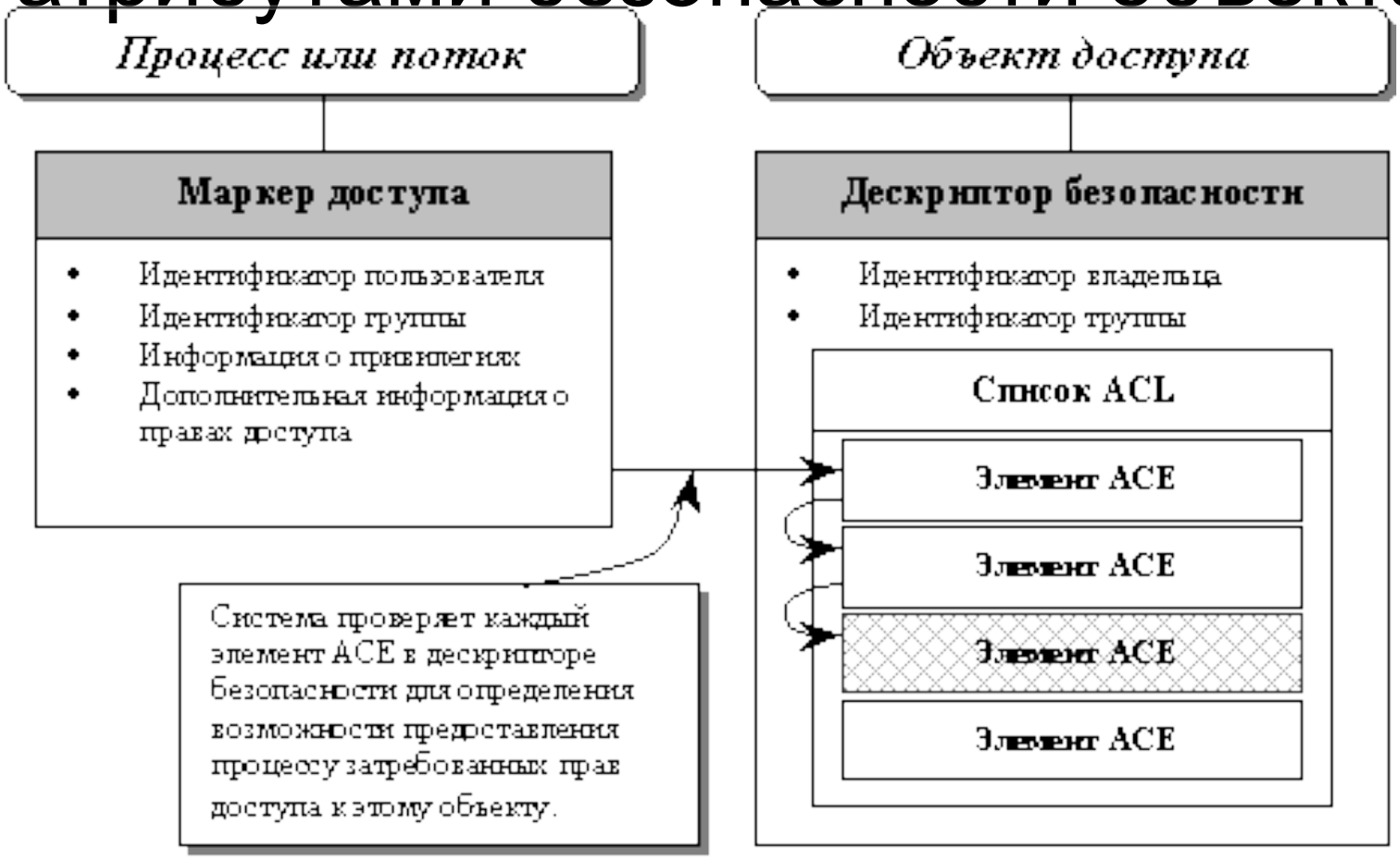
- Файл
- Регион памяти
- Поток
- Процесс
- Семафор
- Таймер

Защита объектов

ОС Windows 2000 поддерживает два вида контроля доступа к объектам:

- **управление избирательным доступом** (discretionary access control) – основной механизм контроля доступа, при котором владельцы объектов разрешают или запрещают доступ к ним для других пользователей (процессов). Когда процесс пытается использовать какой-либо объект, система проводит сравнение атрибутов безопасности, находящихся в идентификаторе безопасности маркера доступа с каждым элементом контроля доступа (access control element, ACE) в списке контроля доступа (access control list, ACL) самого объекта.
- **управление привилегированным доступом** (privileged access control) – необходим в тех случаях, когда управления избирательным доступом недостаточно. Данный метод гарантирует, что пользователь сможет обратиться к защищенным объектам, даже если их владелец недоступен.

Отношения между маркером доступа и атрибутами безопасности объекта



Операционные системы

Введение в файловые системы

Определение ФС

Файловая система – это часть ОС, назначение которой состоит в том, чтобы обеспечить пользователю удобный интерфейс при работе с данными, хранящимися на диске, и обеспечить совместное использование файлов несколькими пользователями и процессами.

Понятие ФС

В широком смысле понятие “ФС” включает:

- совокупность всех файлов на диске,
- наборы структур данных, используемых для управления файлами,
- комплекс системных программных средств, реализующих управление файлами.

Типы файлов

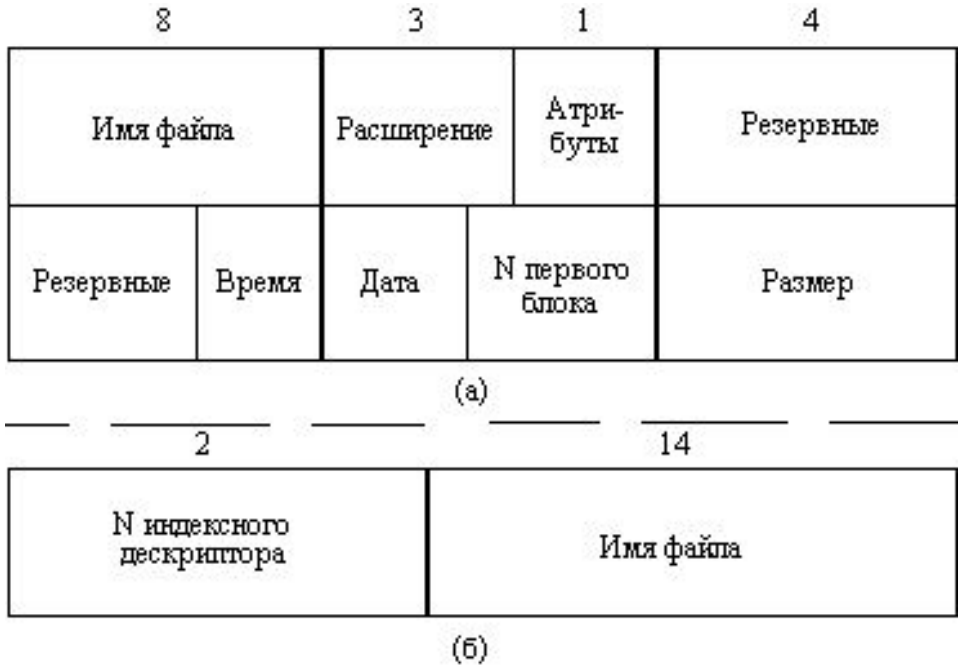
Файлы бывают разных типов:

- обычные файлы:
  - текстовые;
  - двоичные;
- специальные файлы;
- файлы-каталоги.

Атрибуты файлов

- информация о разрешенном доступе;
- пароль для доступа к файлу;
- владелец файла;
- создатель файла;
- флаги "только для чтения", "скрытый файл", "системный файл", "архивный файл", "двоичный/символьный", "временный" (удалить после завершения процесса), флаг блокировки;
- времена создания, последнего доступа и последнего изменения;
- текущий размер файла;
- максимальный размер файла.

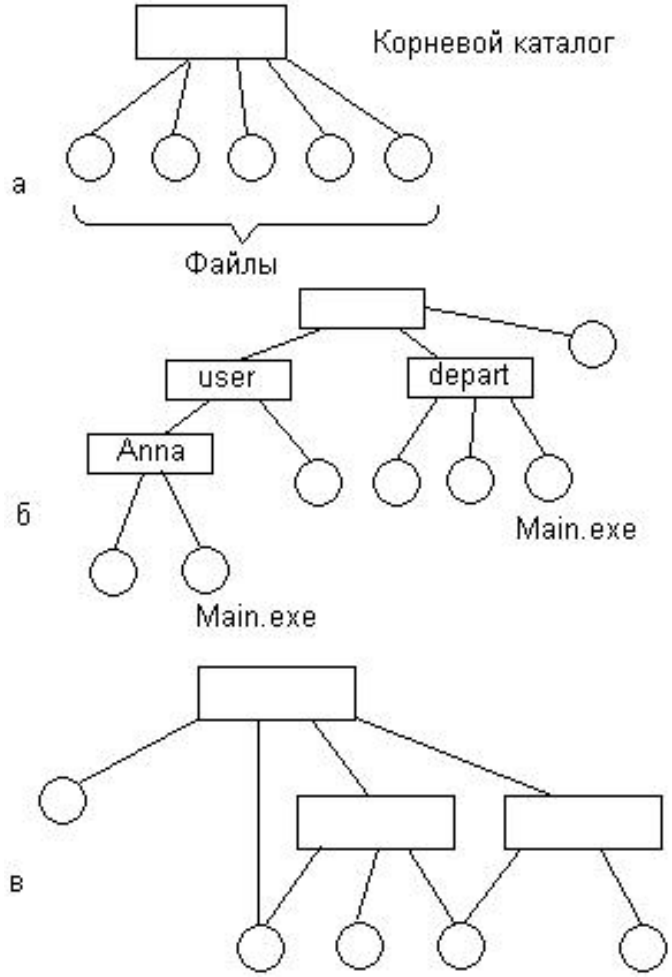
Структура каталогов



структура записи каталога MS-DOS (32 байта)

структура записи каталога ОС UNIX

Логическая организация ФС



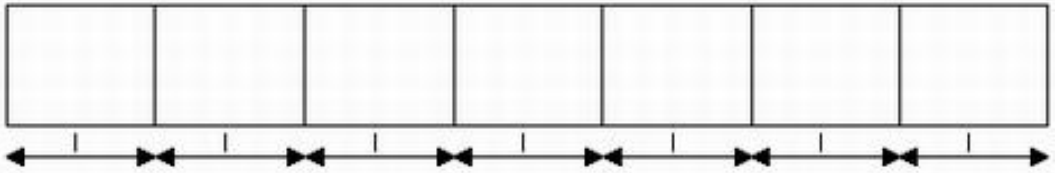
- одноуровневая

- иерархическая (дерево)

- иерархическая (сеть)

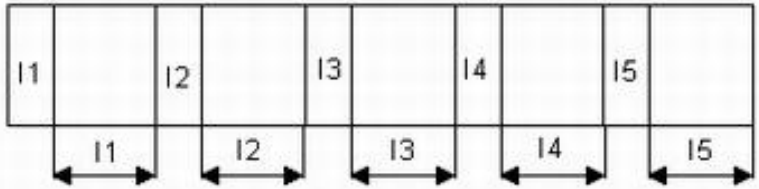
Логическая организация файла

а



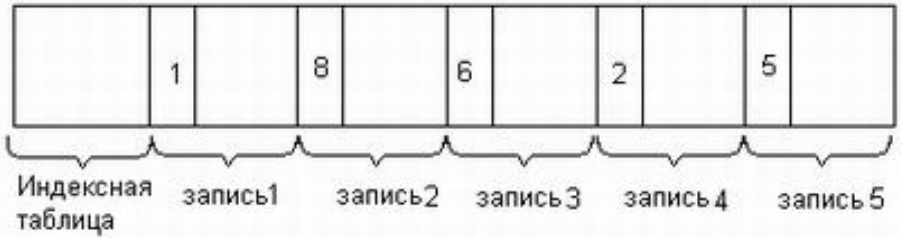
Последовательность логических записей фиксированной длины

б



Последовательность логических записей переменной длины

в



Индексная логическая организация

Индекс	1	2	3	4	5	6
Адрес	21	201	315	661	670	715

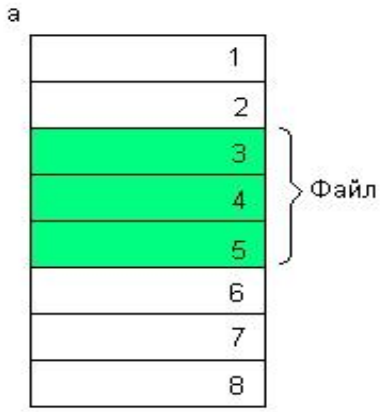
Индекс ≡ ключ

Физическая организация файла

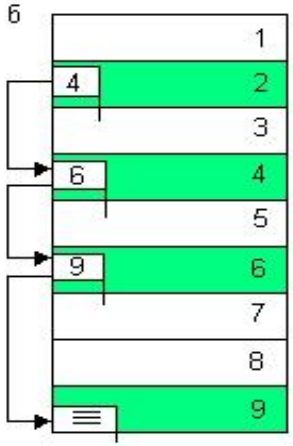
- Физическая организация файла описывает правила расположения файла на устройстве внешней памяти (например, диске).
- Файл состоит из физических записей – блоков.
- Блок – наименьшая единица данных, которой внешнее устройство обменивается с оперативной памятью.

Способы физической организа...

непрерывное размещение

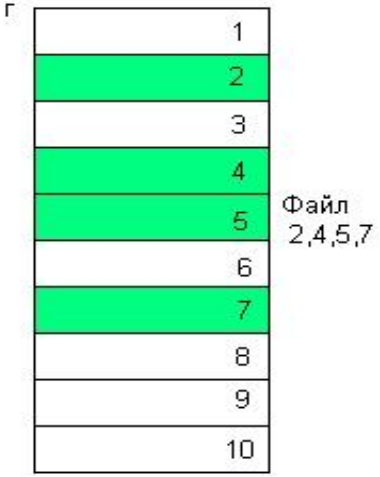


связанный список блоков



связанный список индексов

перечень номеров блоков



Права доступа к файлу

- создание файла
- уничтожение файла
- открытие файла
- закрытие файла
- чтение файла
- запись в файл
- дополнение файла
- дополнение файла
- поиск в файле
- получение атрибутов файла
- установление новых значений атрибутов
- переименование
- выполнение файла
- чтение каталога

Матрица прав доступа

Строки соответствуют всем пользователям

Столбцы соответствуют всем файлам системы

Имена файлов

Имена пользователей	modern.txt	win.exe	class.dbf	unix.ppt
kira	читать	выполнять	–	выполнять
genya	читать	выполнять	–	выполнять читать
nataly	читать	–	–	выполнять читать
victor	читать писать	–	создать	–

На пересечении строк и столбцов указываются разрешенные операции

Кэширование диска

- Перехват запросов к внешним блочным ЗУ, промежуточным программным слоем – подсистемой буферизации (ПБ).
- ПБ представляет собой буферный пул, располагающийся в ОЗУ, и комплекс программ, управляющих этим пулом по принципу кэш-памяти.
- Каждый буфер пула равен одному блоку.

Кэширование диска – запись

- сквозная
- отложенная

Общая модель ФС

Файловые системы

Работа с файлами в Windows API

Работа с томами

- Для выяснения того, какие логические диски существуют в системе, используется функция

  DWORD GetLogicalDrives(void)

  Каждый установленный бит возвращаемого значения соответствует существующему в системе логическому устройству. Например, если в системе существуют диски A:, C: и D:, то возвращаемое функцией значение равно 13(10).
- Функция

  DWORD GetLogicalDrivesStrings(DWORD cchBuffer, LPTSTR lpszBuffer)

  заполняет lpszBuffer информацией о корневом каталоге каждого логического диска в системе. В приведенном выше примере буфер будет заполнен символами

  A:\<null>C:\<null>D:\<null><null>

  параметр cchBuffer определяет длину буфера. Функция возвращает реальную длину буфера, необходимую для размещения всей информации.

Работа с томами

- Для определения типа диска предназначена функция
  UINT GetDriveType(LPTSTR lpszRootPathName)
  В качестве параметра ей передается символическое имя корневого каталога (напр. **A:**), а возвращаемое значение может быть одно из следующих:

Идентификатор	Описание
0	Тип устройства определить нельзя
1	Корневой каталог не существует
DRIVE_REMOVABLE	Гибкий диск
DRIVE_FIXED	Жесткий диск
DRIVE_REMOTE	Сетевой диск
DRIVE_CDROM	Компакт диск
DRIVE_RAMDISK	RAM диск

Работа с томами

- Для получения подробной информации о носителе используется функция **GetVolumeInformation**. Она заполняет параметры информацией об имени тома, названии файловой структуры, максимальной длине имени файла, дополнительных атрибутах тома, специфических для файловой структуры.
- Функция **GetDiskFreeSpace** сообщает информацию о размерах сектора и кластера и о наличии свободных кластеров.

Работа с каталогами и файлами

Функция	Выполняемое действие
GetCurrentDirectory	Получение текущего каталога
SetCurrentDirectory	Смена текущего каталога
GetSystemDirectory	Получение системного каталога
GetWindowsDirectory	Получение основного каталога системы
CreateDirectory	Создание каталога
RemoveDirectory	Удаление каталога
CopyFile	Копирование файла
MoveFile MoveFileEx	Перемещение или переименование файла
DeleteFile	Удаление файла

Синхронная работа с файлами

```
HANDLE CreateFile (
    LPCTSTR lpFileName,        // pointer to name of the file
    DWORD dwDesiredAccess,     // access (read-write) mode
    DWORD dwShareMode,         // share mode
    LPSECURITY_ATTRIBUTES lpSecurityAttributes, // pointer to security
                               // descriptor
    DWORD dwCreationDistribution,// how to create
    DWORD dwFlagsAndAttributes, // file attributes
    HANDLE hTemplateFile       // handle to file with attributes to copy
);
```

В случае удачи функция **CreateFile** возвращает описатель открытого файла как объекта ядра. Существенно, что в противном случае она возвращает не NULL, а INVALID_HANDLE_VALUE.

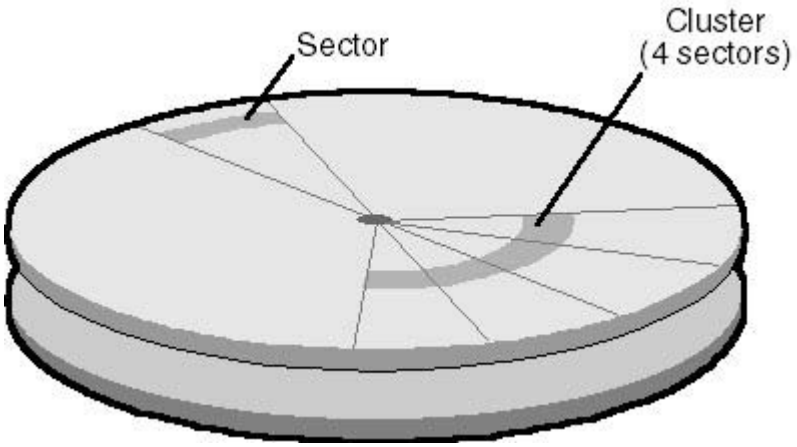
Асинхронная работа с файлами

```
BOOL ReadFile(
        HANDLE hFile,     // handle of file to read
        LPVOID lpBuffer,  // address of buffer that receives data
        DWORD nNumberOfBytesToRead,// number of bytes to read
        LPDWORD lpNumberOfBytesRead,// address of number of bytes read
        LPOVERLAPPED lpOverlapped // address of structure needed for
                                  // overlapped I/O
);
BOOL WriteFile(
        HANDLE hFile,     // handle to file to write to
        LPCVOID lpBuffer,      // pointer to data to write to file
        DWORD nNumberOfBytesToWrite, // number of bytes to write
        LPDWORD lpNumberOfBytesRead,// pointer to number of bytes written
        LPOVERLAPPED lpOverlapped // address of structure needed for
                                  //overlapped I/O
);
```

Файловые системы

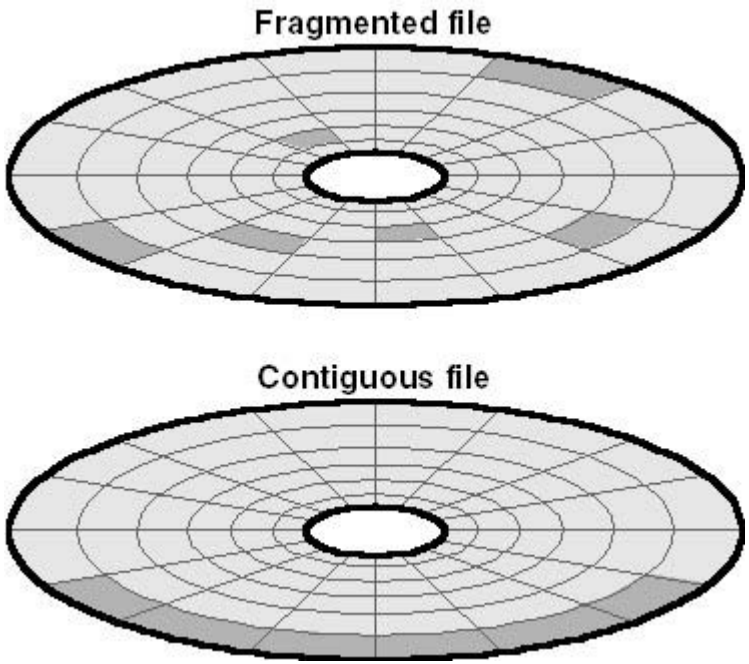
Файловые системы фирмы Microsoft

Кластеры



В ОС Microsoft Windows основной единицей хранения информации является кластер – группа смежных секторов. Число секторов в кластере всегда равно степени двойки.

Фрагментация и дефрагментация



Файл, который занимает на диске более одного непрерывного участка, называется *фрагментированным.*

Фрагментация диска - это появление на диске множества свободных участков, разделенных занятыми участками.

Дефрагментация диска - это перемещение данных на разделе, после которого, кластеры содержащие части одного файла, размещаются последовательно.

Физические и логические диски

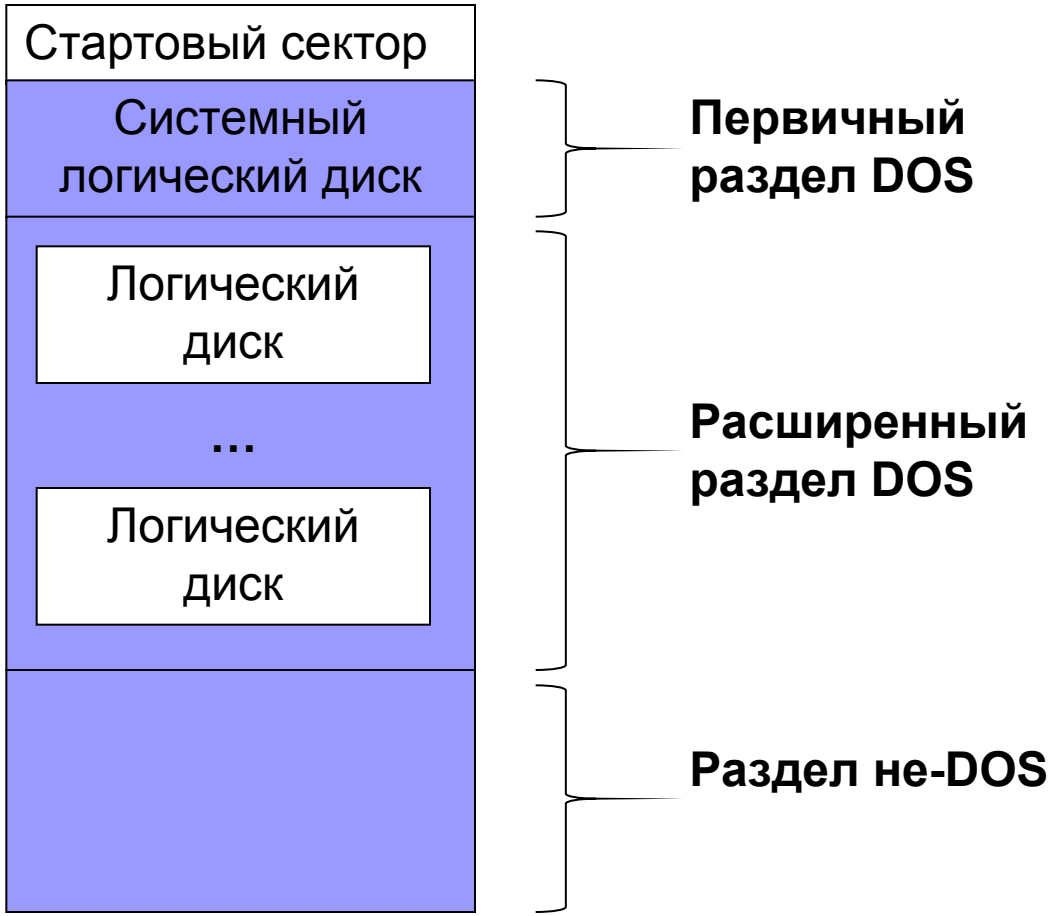
Основные причины разбиения физического диска на несколько логических:

- ограничения файловых систем на максимальный размер физического диска;
- повышение надежности файловой системы;
- поддержка нескольких ОС.

Файловые системы

Файловая система FAT
для MS DOS

Таблица разделов логического диска



Структура логического диска FAT

- **Загрузочная запись** (первый сектор диска) – служит для загрузки ОС и организация хранения данных.
- **FAT** (File Allocation Table) – таблица размещения файлов.
- **Корневой каталог** – для FAT16 512 записей о файлах и каталогах, расположенных в корне файловой системы.

Элемент каталога FAT16

Содержание	Размер (байт)
Имя файла	8
Расширение	3
Байт атрибутов	1
Зарезервировано	10
Время	2
Дата	2
Номер начального кластера	2
Размер файла	4

Логическая организация данных

Размеры разделов и кластеров FAT16 для Windows 95-2000

Размер раздела	Размер кластера
0 – 32 Мб	512 б
33 – 64 Мб	1 Кб
65 – 128 Мб	2 Кб
129 Мб – 256 Мб	4 Кб
257 Мб – 512 Мб	8 Кб
513 Мб – 1024 Мб	16 Кб
1 Гб – 2 Гб	32 Кб
2 Гб – 4 Гб	64 Кб

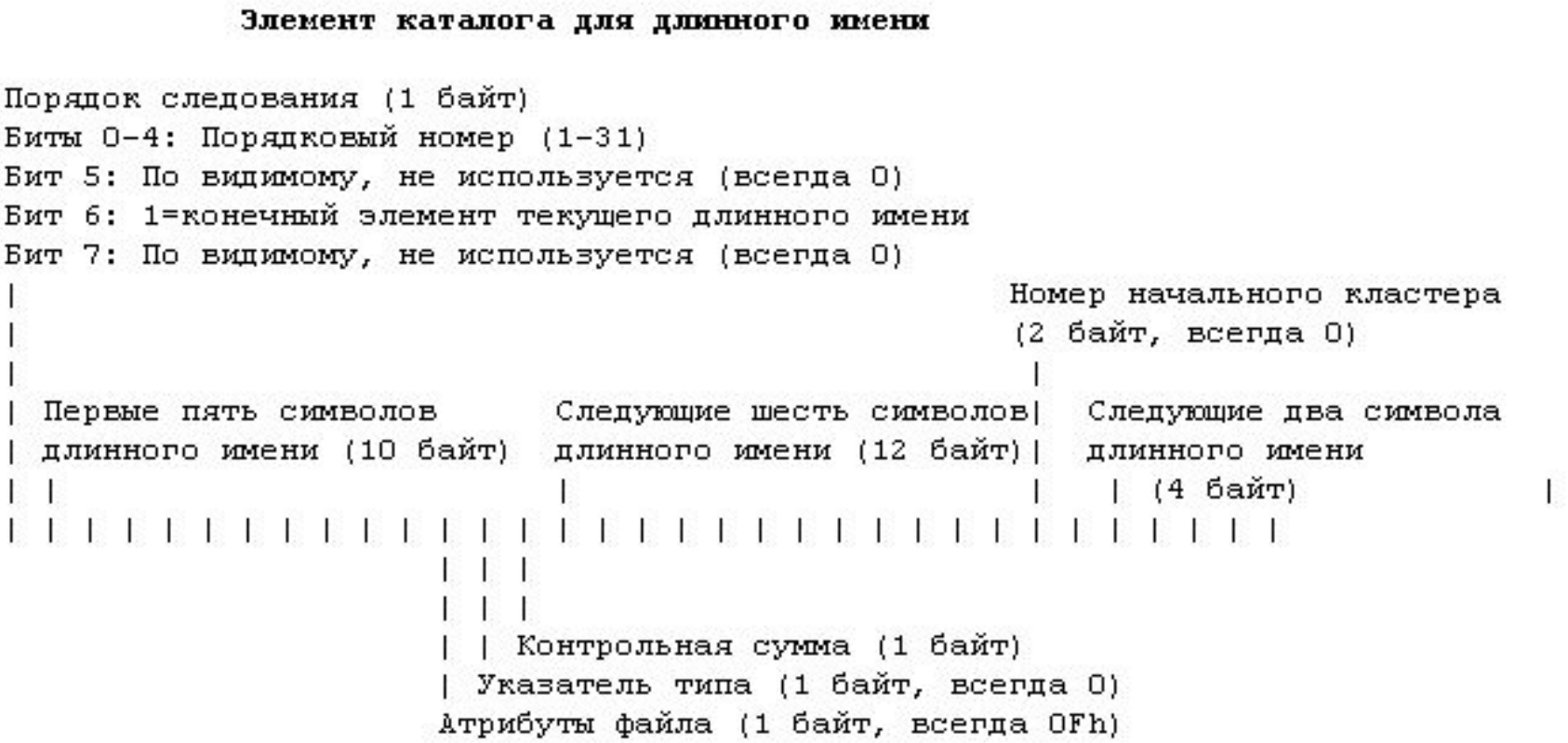
Файловые системы

Файловая система FAT32

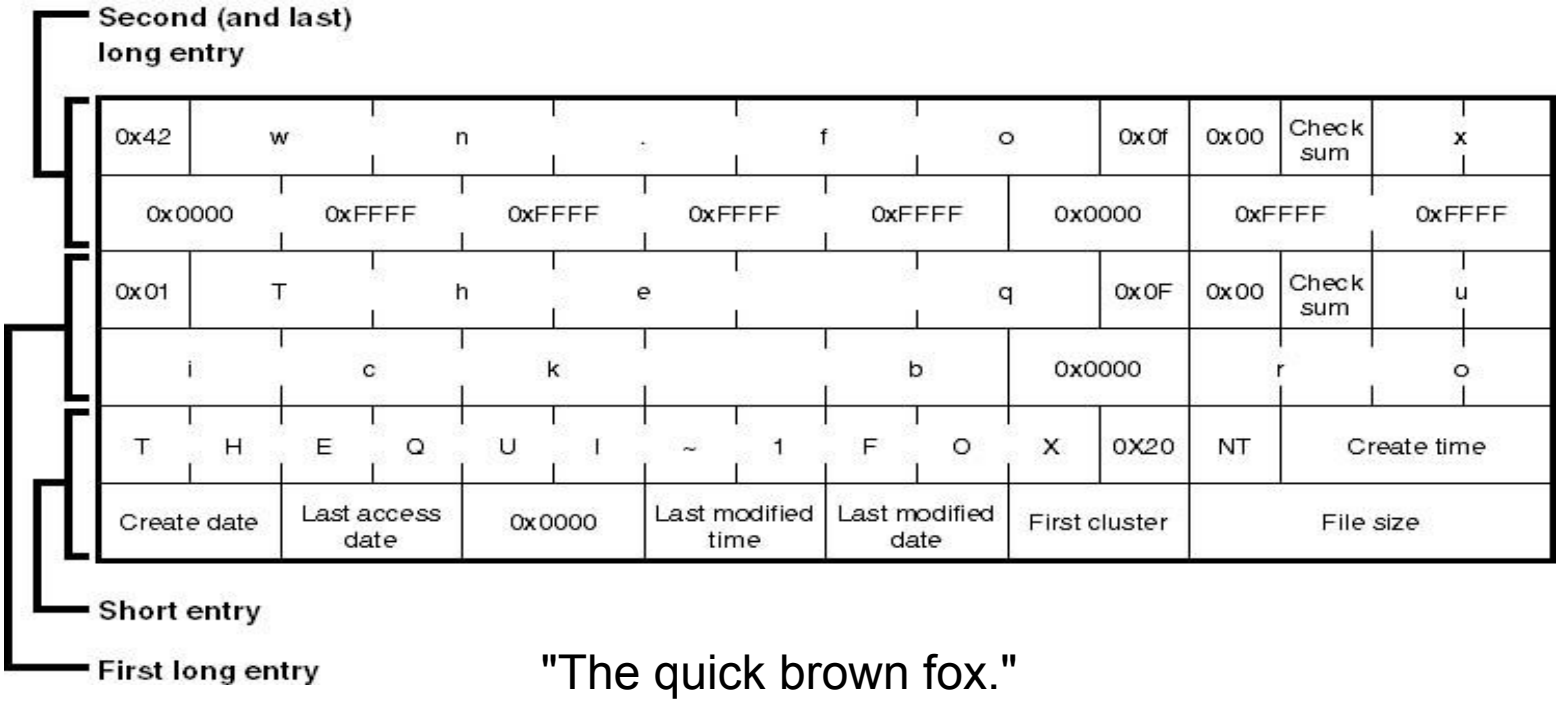
Файловая система FAT32

- FAT32 это развитие файловой системы FAT(VFAT, FAT16).
- 32-разрядная адресация кластеров – максимальное число адресуемых кластеров – 4 294 377 472.
- Поддержка больших разделов (более 4Gb), кроме этого уменьшен размер кластера на разделе.
- Поддержка длинных имен до 255 символов.
- У нее нет ограничений на число и размер расширения.
- Длинные имена (LFN) хранятся в специально отформатированных 32-байт записях, байт атрибутов у которых равен 0Fh.

Элемент каталога FAT32



Пример длинного имени



"The quick brown fox."

THEQUI~1.FOX

Проблемы длинных имен в FAT32

- Требуется больше дискового пространства
- Б**о́**льшая фрагментация (на уровне каталогов)

Сравнение FAT16 и FAT32

FAT 16	
	максимальный размер раздела ;– 2Гб (для Win2000 – 4Гб)
	при размере раздела > 512 Мб неэкономно расходует место на диске (из-за большого размера кластера);
	распознается большинством ОС, используемыми на ПК;
	позволяет уплотнять диск программой сжатия данных Drivespace;
	имеет корневой каталог фиксированного размера (512 записей).

FAT 32	
	работает чуть медленнее, чем FAT16;
	разделы до 2Тб;
	большая эффективность использования места на диске;
	нельзя уплотнить с помощью программы сжатия данных;
	старые версии DOS и многие другие ОС не "видят" разделы с форматом FAT 32;
	корневой каталог является обычным расширяемым каталогом.

Файловые системы

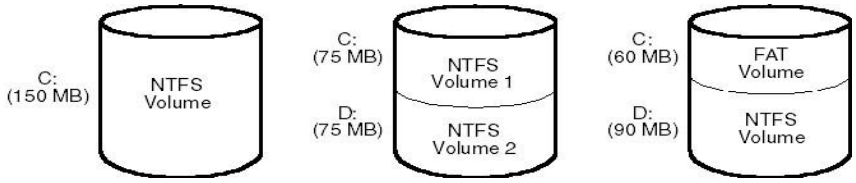
Файловая система NTFS

Краткое описание

- Разработана для быстрого выполнения стандартных файловых операций типа чтения, записи и поиска.
- Поддерживает улучшенные операции восстановления файловой системы на очень больших жестких дисках.
- Включает возможности безопасности, требуемые для файловых серверов и высококачественных персональных компьютеров в корпоративной среде.

Понятия и термины NTFS

- Структура NTFS начинается с тома (volume). Том соответствует логическому разделу на диске и создается, когда Вы форматируете диск или часть его для NTFS.
- На одном диске может находиться один или несколько томов.

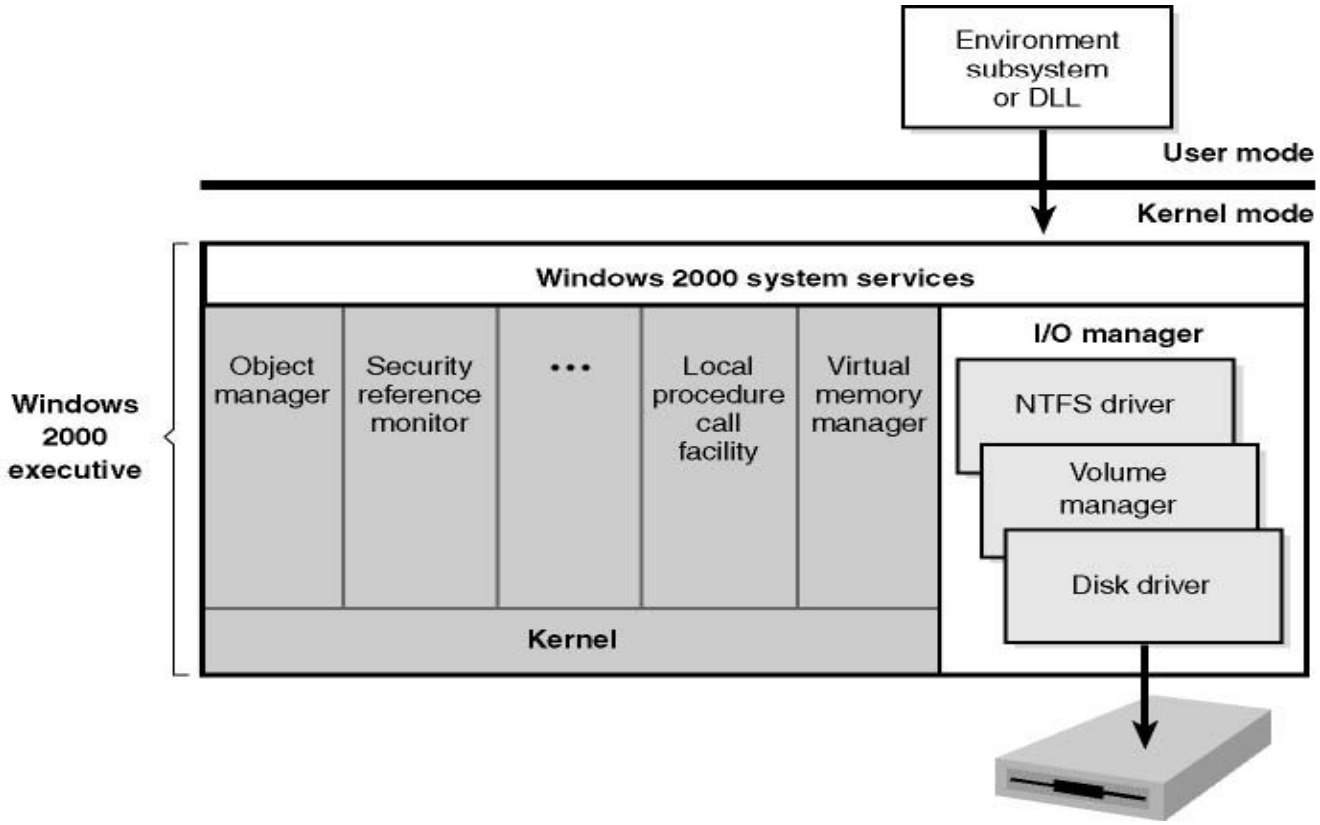


- NTFS обрабатывает каждый том независимо от других.

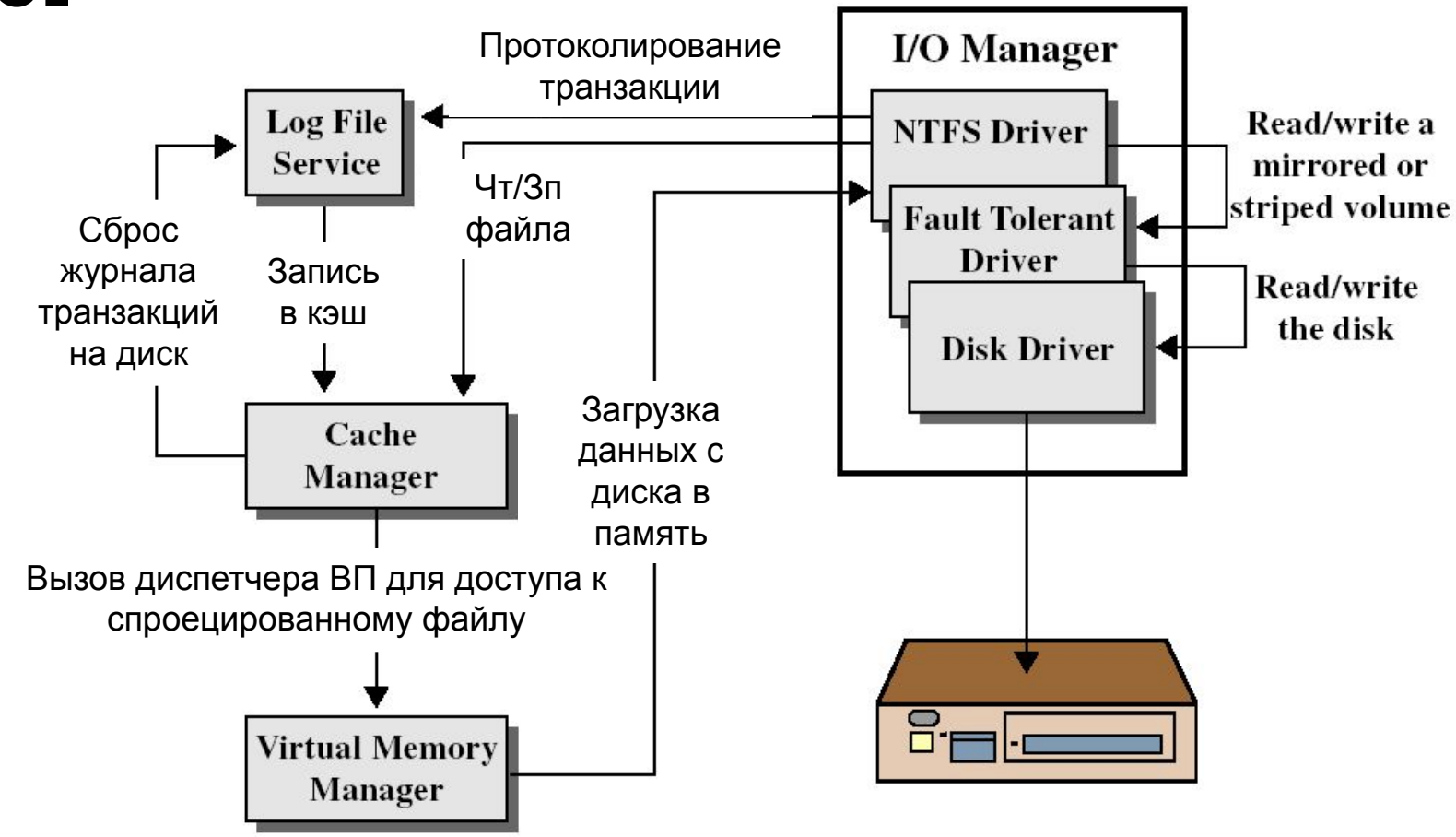
Размер кластера для NTFS

Размер логического диска	Рекомендуемый размер кластера
512 MB и менее	512 б
513 Мб – 1 Гб	1 Кб
1 Гб – 2 Гб	2 Кб
Более 2 Гб	4 Кб

NTFS и архитектура Windows 2000



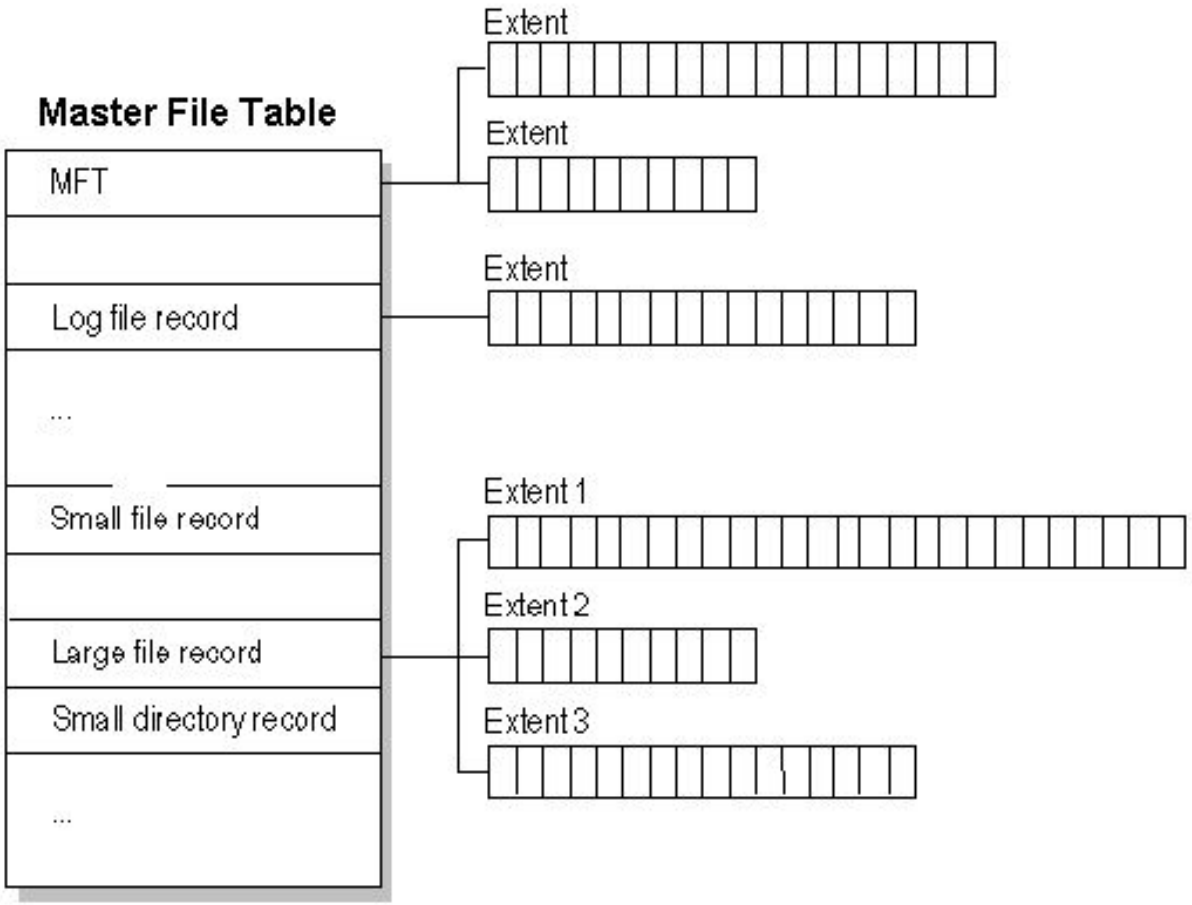
Взаимодействие NTFS со связанными компонентами



Физическая структура NTFS

NTFS поддерживает размеры кластеров - от 512 байт до 64 Кбайт

MFT и ее структура



Метафайлы

- Первые 16 файлов NTFS (метафайлы) носят служебный характер.
- Метафайлы находятся корневом каталоге NTFS диска - они начинаются с символа имени "$"
- Для метафайлов указан реальный размер - можно узнать, например, сколько ОС тратит на каталогизацию всего диска.

Перечень метафайлов (1)

$MFT	список содержимого тома NTFS
$MFTmirr	копия первой записи MFT
$LogFile	файл поддержки журналирования шагов транзакций
$Volume	служебная информация - метка тома, версия файловой системы, т.д.
$AttrDef	список стандартных атрибутов файлов на томе
$.	корневой каталог
$Bitmap	карта свободного места тома, каждый бит которой соответствует одному кластеру тома и указывает его состояние (свободен или занят)

Перечень метафайлов (2)

$BadClus	Список всех плохих кластеров тома. Кластер считается плохим, если в нем есть один плохой сектор
$Secure	База данных атрибутов безопасности. Применяется только в NTFS версии 5.0 в среде Microsoft Windows 2000
$Upcase	файл - таблица соответствия заглавных и прописных букв в имен файлов на текущем томе.
$Extend	Файл хранит расширенную информацию файловой системы NTFS версии 5.0, применяемой в среде Microsoft Windows 2000, такую как дисковые квоты, точки монтирования и т.д.

Файл и его атрибуты (1)

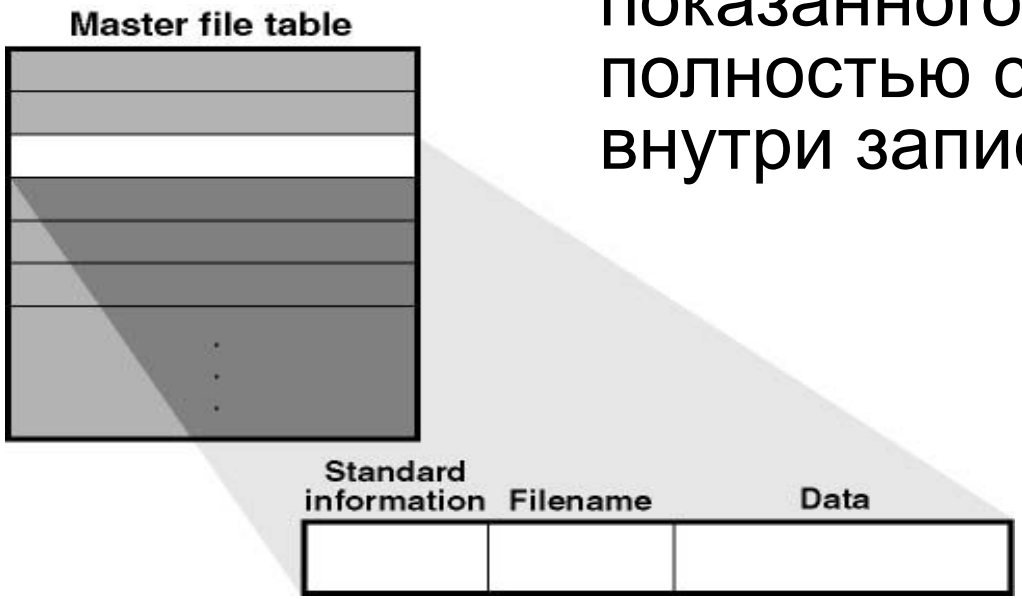
Standard Information (стандартная информация)	Стандартный атрибут. Дата и время создания и последнего изменения файла, дата и время последнего доступа к файлу, флаги доступа к файлу, а также дата и время изменения записи MFT, соответствующей данному файлу.
Attribute List (список атрибутов)	Перечисляет все другие атрибуты.
Filename (имя файла)	Имя файла или каталога. В этом атрибуте хранится имя файла или каталога, набор флагов доступа, размер файла, а также ссылка на запись MFT каталога, в котором хранится данный файл или каталог.
MS-DOS Name	Имя файла в формате 8.3
Version	Номер последней версии файла
Security Descriptor (дескриптор безопасности)	Фиксирует информацию о том, кто может обращаться к файлу, кто является его владельцем и так далее (ACL)
Data (данные)	Содержит данные файла

Файл и его атрибуты (2)

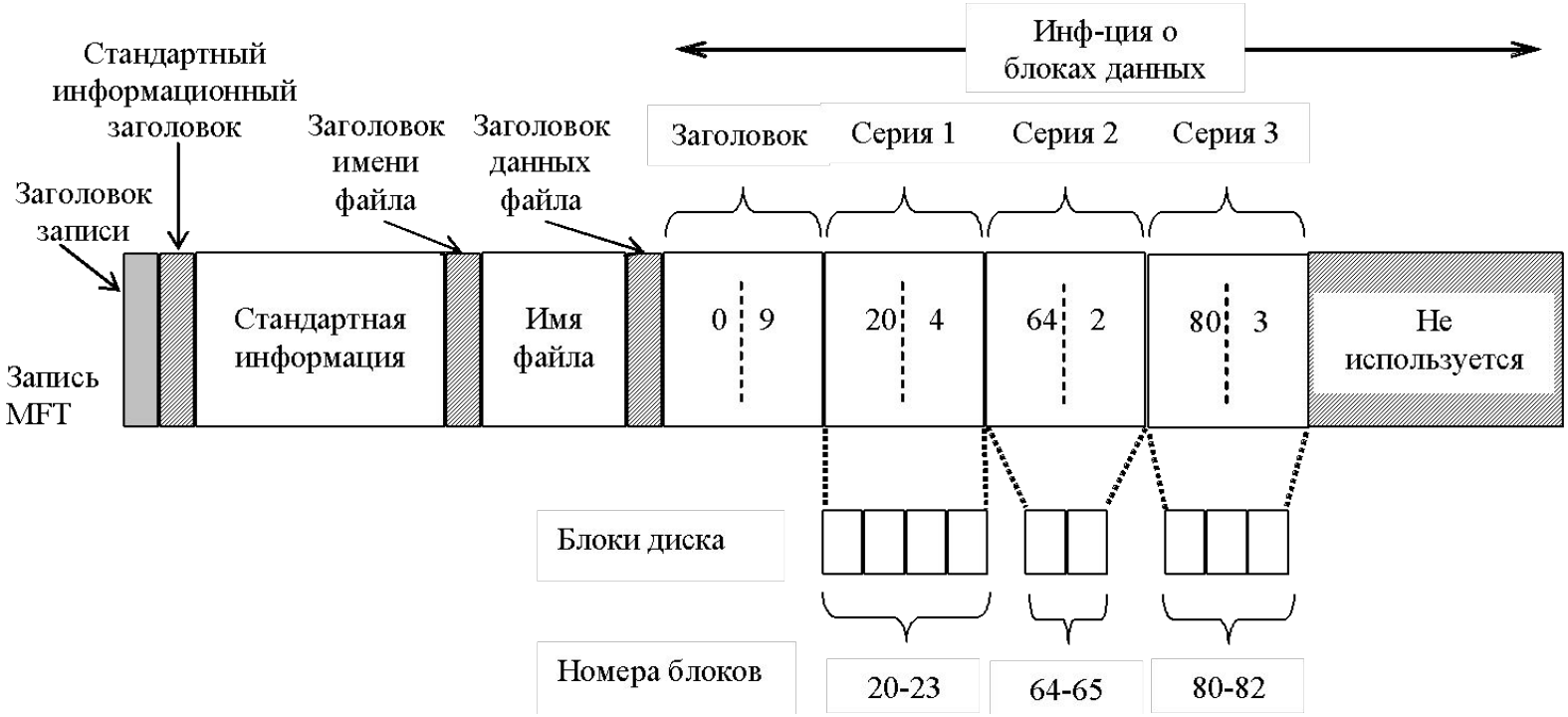
Volume Version	версия тома, используется только в системных файлах тома
Volume Information (информация о томе)	Используется только в системном файле тома и включает в частности версию и имя тома
Volume Name	метка тома
Index Root (корневой индекс)	Корневая вершина дерева типа B+, используемого для поиска файлов в каталоге. Всегда резидентный.
Index Allocation (размещение индекса)	Узлы ветвей дерева типа B+. нерезидентные части индексного списка B-дерева
External Attribute Information	номер первого кластера и количество кластеров нерезидентного атрибута
Bitmap (битовый массив)	Предоставляет информацию об использовании записей в MFT или каталоге

Непосредственное хранение файлов

Небольшие файлы и каталоги (обычно до 1500 байт или меньше), типа файла, показанного на слайде, могут полностью содержаться внутри записи MFT.



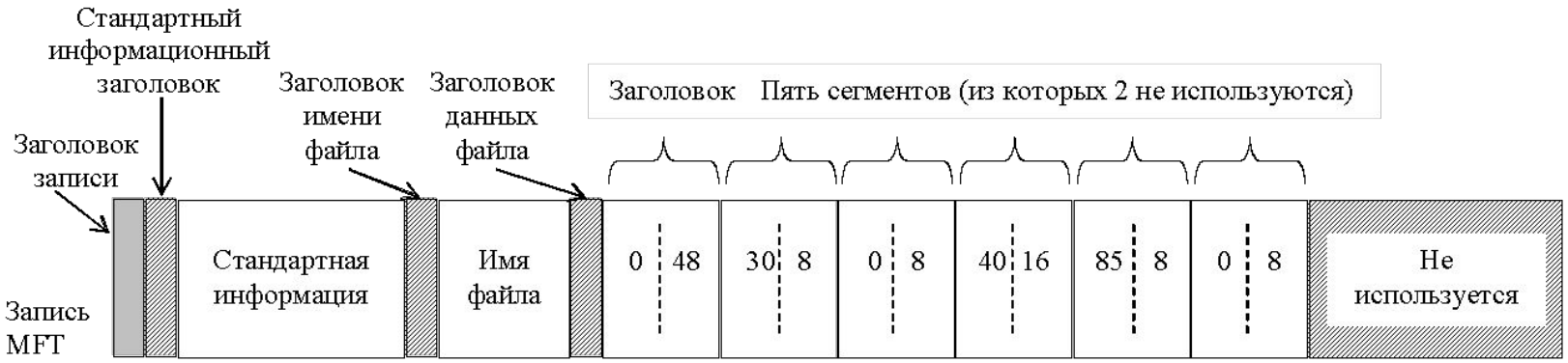
Нерезидентное хранение файлов



Запись MFT для короткого файла

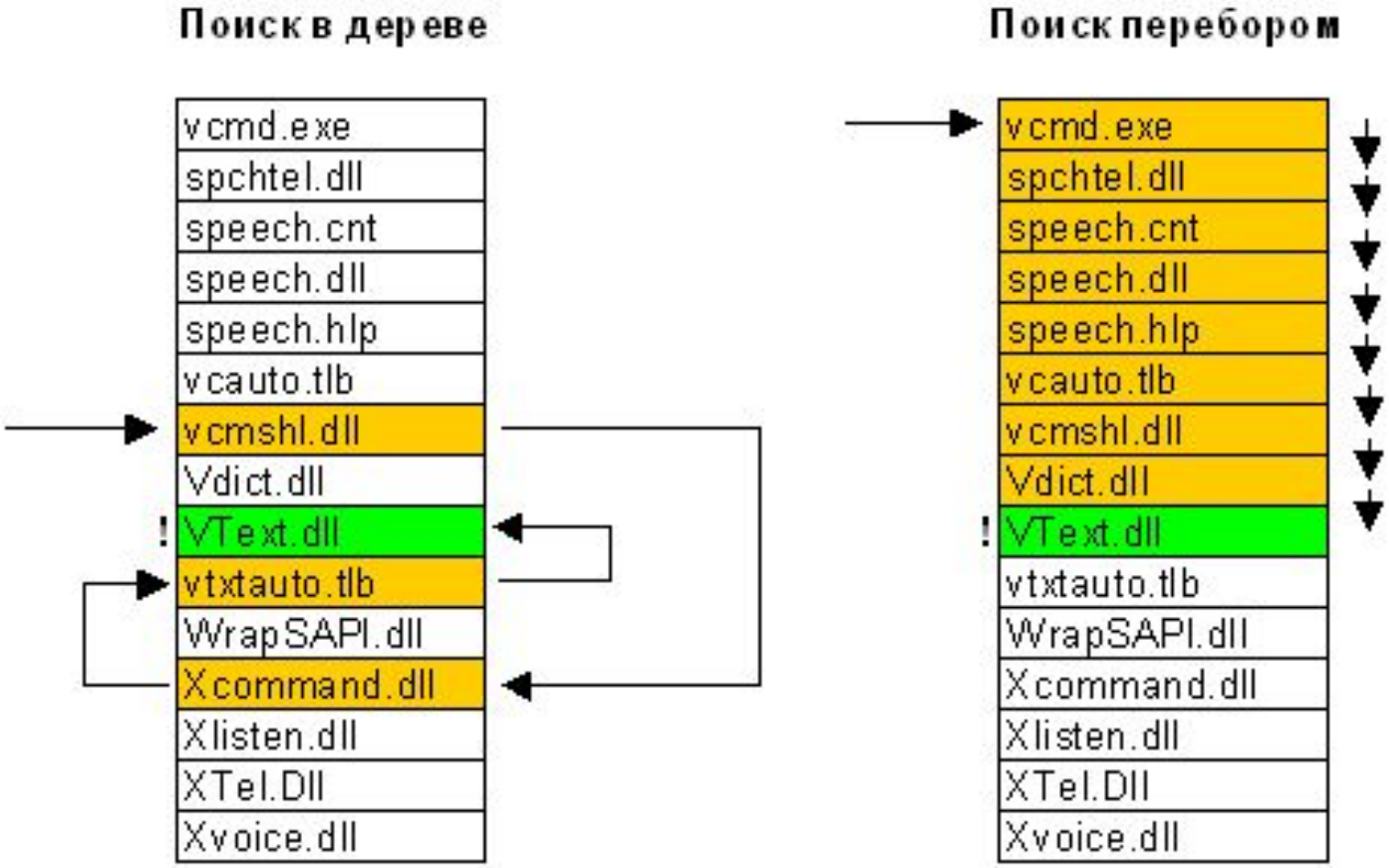
Сжатие файлов

Пример 48-блочного файла сжатого до 32 блоков

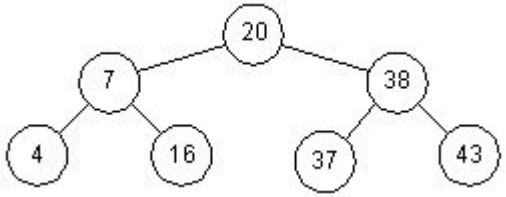


Запись MFT после сжатия файла

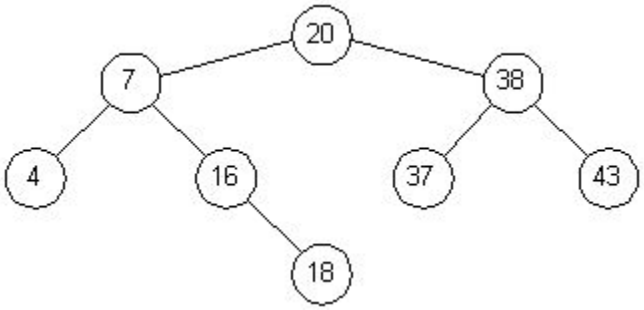
Каталоги


Поиск в дереве
vcmd.exe
spchtel.dll
speech.cnt
speech.dll
speech.hlp
vcauto.tlb
vcmshl.dll
Vdict.dll
!
VText.dll
vtxtauto.tlb
WrapSAPI.dll
Xcommand.dll
Xlisten.dll
XTel.Dll
Xvoice.dll
Поиск перебором
vcmd.exe
spchtel.dll
speech.cnt
speech.dll
speech.hlp
vcauto.tlb
vcmshl.dll
Vdict.dll
!
VText.dll
vtxtauto.tlb
WrapSAPI.dll
Xcommand.dll
Xlisten.dll
XTel.Dll
Xvoice.dll

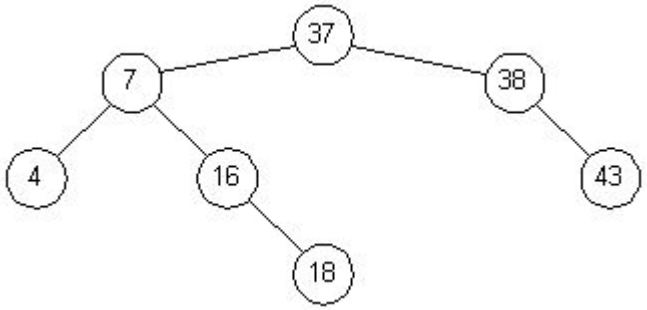
Двоичное дерево B+



Двоичное дерево

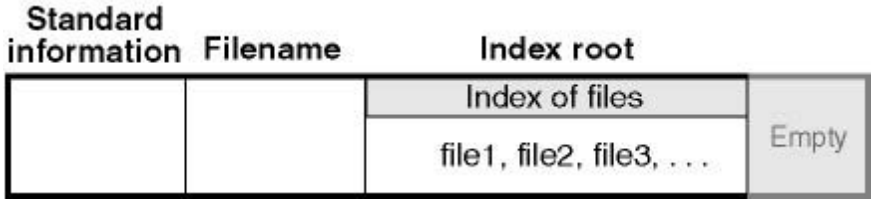


Бинарное дерево после добавления узла 18

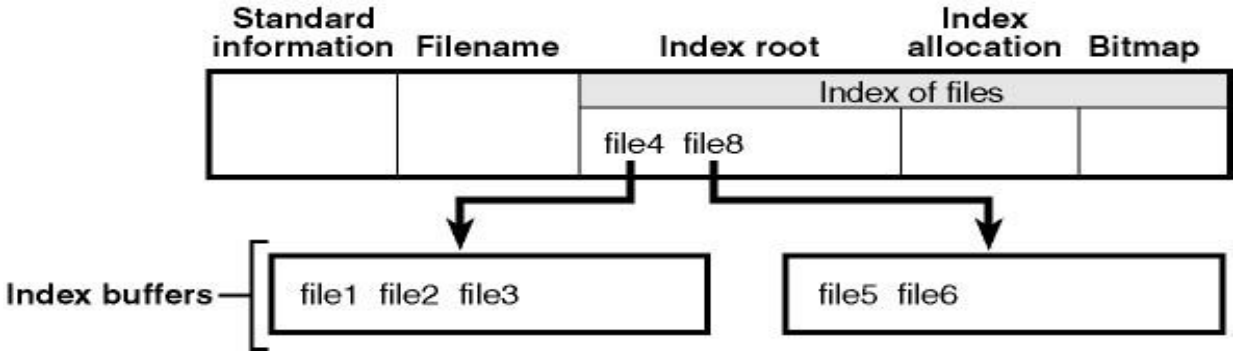


Бинарное дерево после удаления узла 20

Хранение каталога



а) запись MFT для небольшого каталога



б) запись MFT для каталога

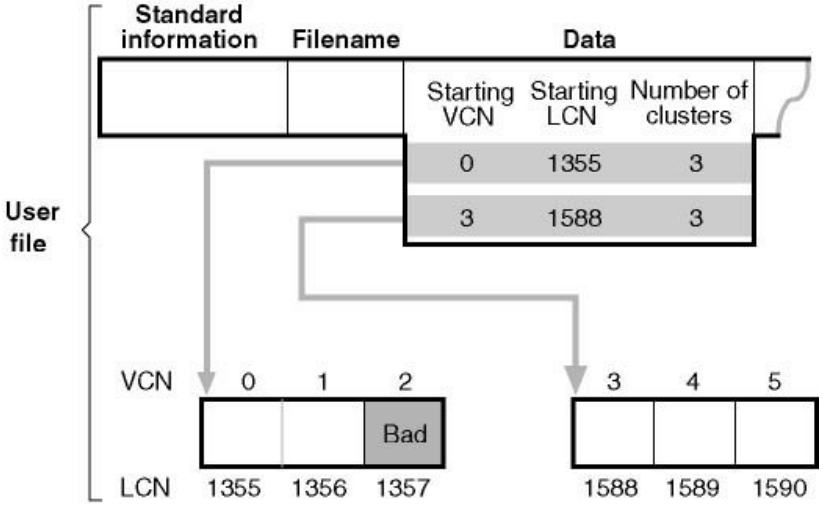
Защита целостности данных

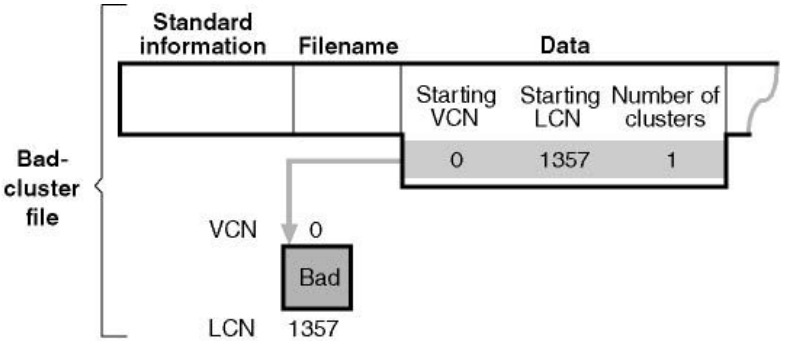
NTFS является *восстанавливаемой* ФС и поддерживает следующие технологии защиты целостности данных:

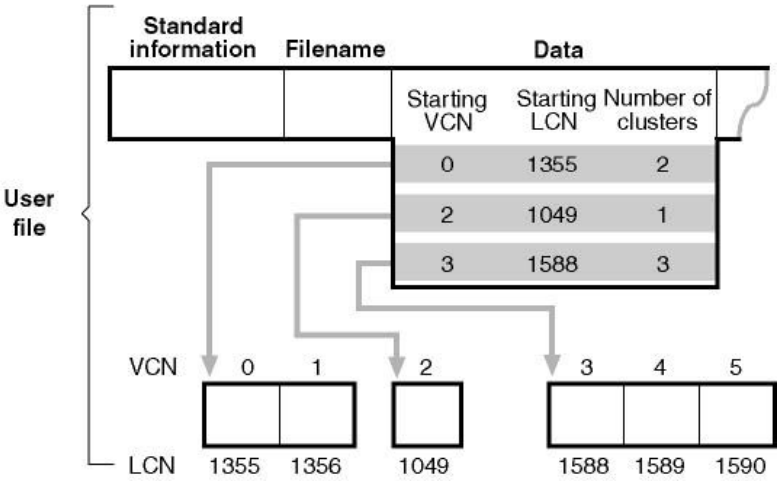
- *Горячая фиксация*
- *Механизм транзакций*

Система восстановления NTFS гарантирует корректность **файловой системы**, а не ваших данных.

Горячая фиксация







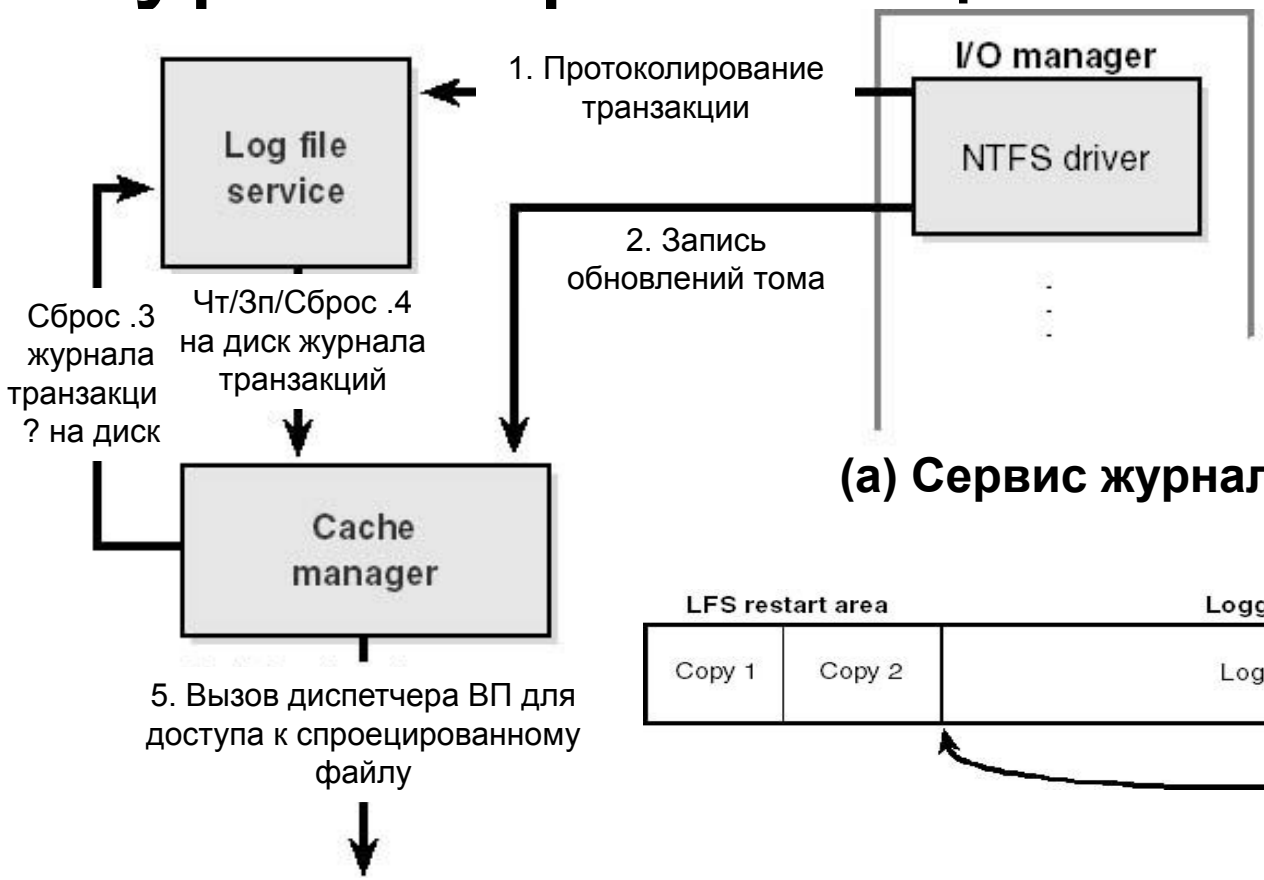
а) MFT-запись файла с плохим кластером;

б) исправленная MFT-запись файла;

Механизм транзакций

- Восстанавливаемость ФС в NTFS обеспечивается при помощи техники обработки транзакций, называемой протоколированием (logging).
- В состав средств протоколирования NTFS входят два важных компонента:
  - журнал транзакций (log file) – это системный файл, создаваемый командой Format.
  - сервис журнала операций (log file service, LFS) – набор системных процедур, которые NTFS использует для доступа к журналу транзакций.

Журнал транзакций

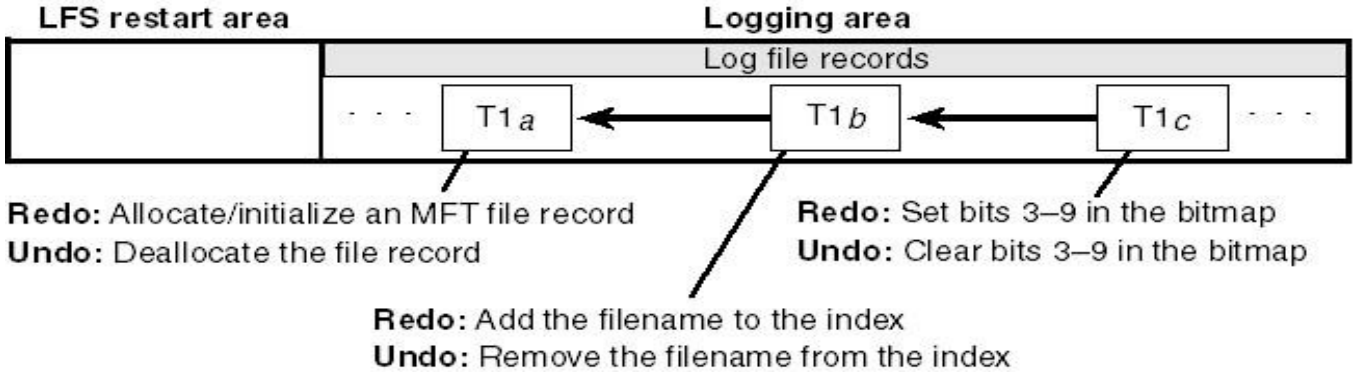


(а) Сервис журнала транзакций

(б) Журнал транзакций

Записи модификации

- создание файла
- удаление файла
- расширение файла
- урезание файла
- установка файловой информации
- переименование файла
- изменение прав доступа к файлу



Структура записи модификации

- **Информация для повтора (redo info)**

  как вновь применить к то́му одну подоперацию полностью запротоколированной транзакции, если сбой системы произошел до того, как транзакция была переписана из кэша на диск.

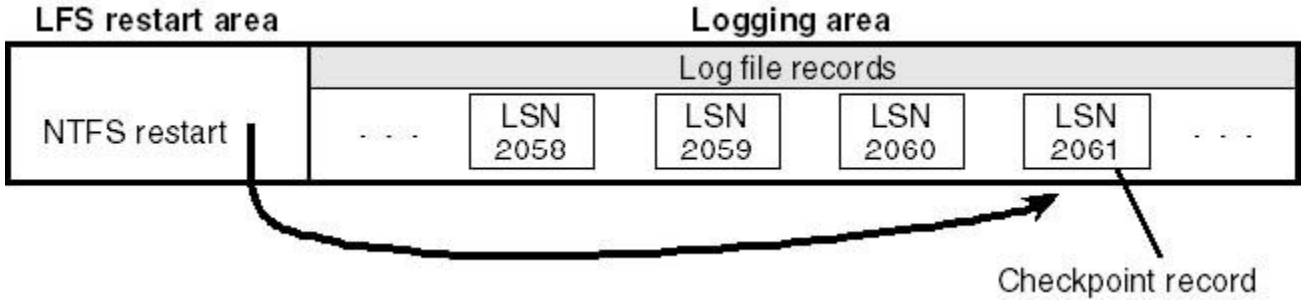
- **Информация для отмены (undo info)**

  как устранить изменения, вызванные одной подоперацией транзакции, которая в момент сбоя была запротоколирована лишь частично.

Запись контрольной точки

Периодически (5 сек.) NTFS помещает в журнал транзакций **записи контрольной точки**:

- Запись контрольной точки помогает NTFS определить, какая обработка необходима для восстановления тома, если сбой произошел “сразу” после помещения этой записи в журнал.
- LSN контрольной точки записывается в область рестарта



Таблицы восстановления

- **Таблица транзакций** (transaction table) предназначена для отслеживания транзакций, которые были начаты, но еще не подтверждены. Их надо удалить в процессе восстановления.
- В **таблицу измененных страниц** (dirty page table) записывается информация о том, какие страницы кэша содержат изменения структуры файловой системы, еще не записанные на диск. Эти данные в процессе восстановления должны быть сброшены на диск.

Процесс восстановления

При восстановлении тома NTFS загружает журнал транзакций в оперативную память и выполняет три прохода:

- **анализ;**
- **повтор транзакций;**
- **отмена транзакций.**

Безопасность в NTFS

- Защита файлов NTFS на объектном уровне – Security Reference Monitor определяет, имеет ли пользователь необходимые права для вызова какого-либо из этих методов.
- Шифрование файлов с помощью специального драйвера EFS (Encrypting File System).

Фрагментация файлов в NTFS

- NTFS полностью не предотвращает фрагментацию
- NTFS снижает возможность возникновения фрагментации (например, в многозадачном режиме)
- NTFS снижает отрицательное влияние фрагментации на быстродействие

Файловые системы

NTFS vs. FAT

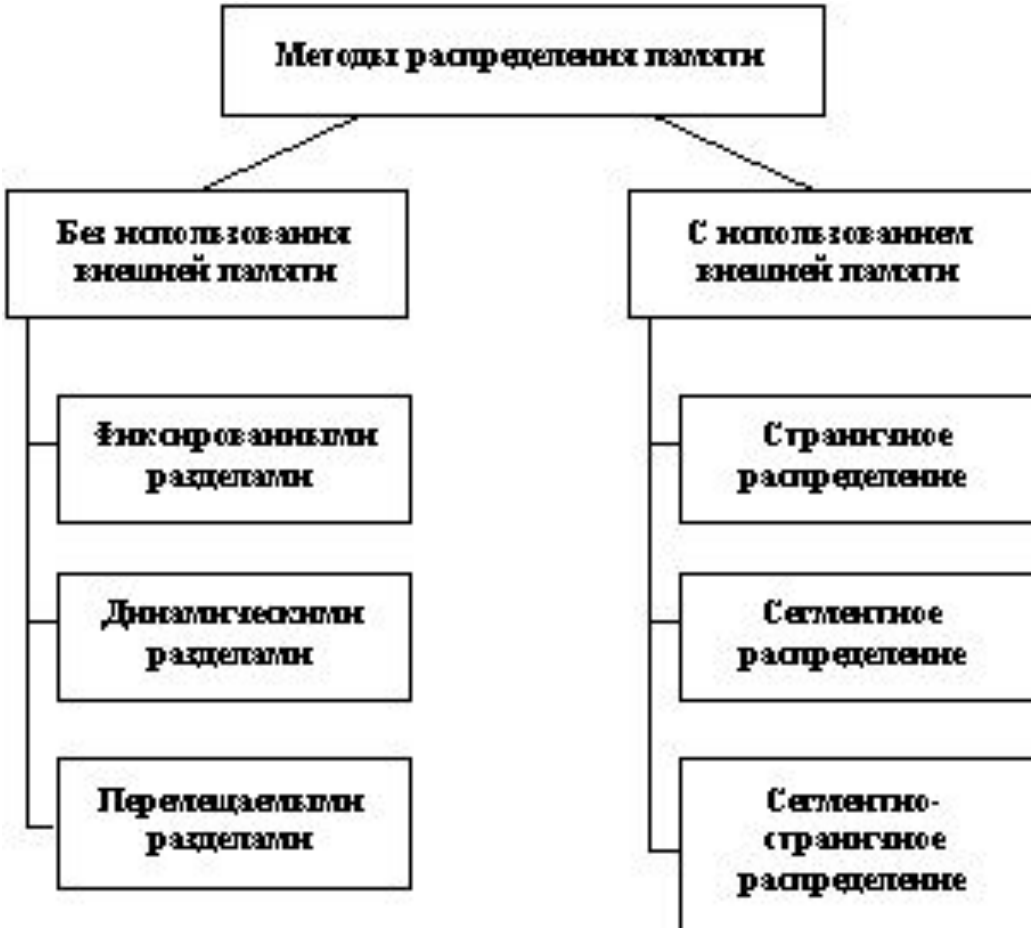
Итоги

Параметр	Лидер	Аутсайдер
Поиск данных файла	FAT16, NTFS	FAT32
Поиск свободного места	NTFS	FAT32
Работа с каталогами и файлами	NTFS(*)	

Операционные системы

Основы управления памятью

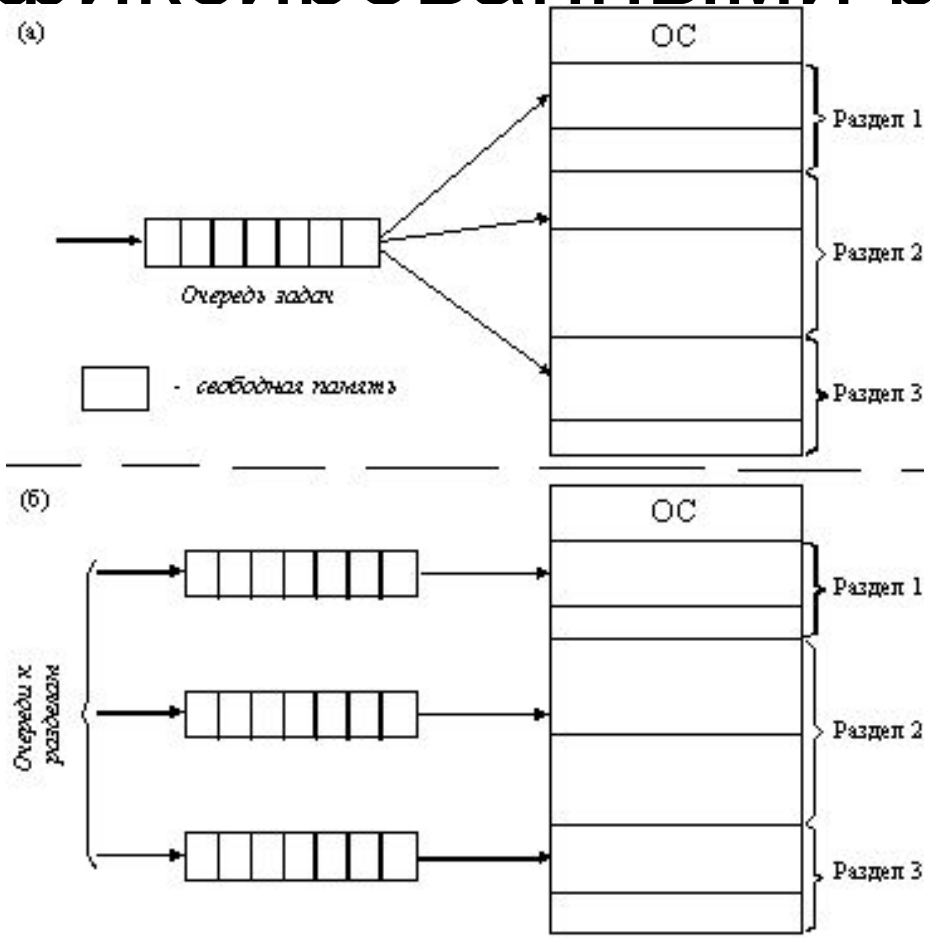
Методы управления памятью



Основы управления памятью

Методы распределения памяти без использования дискового пространства

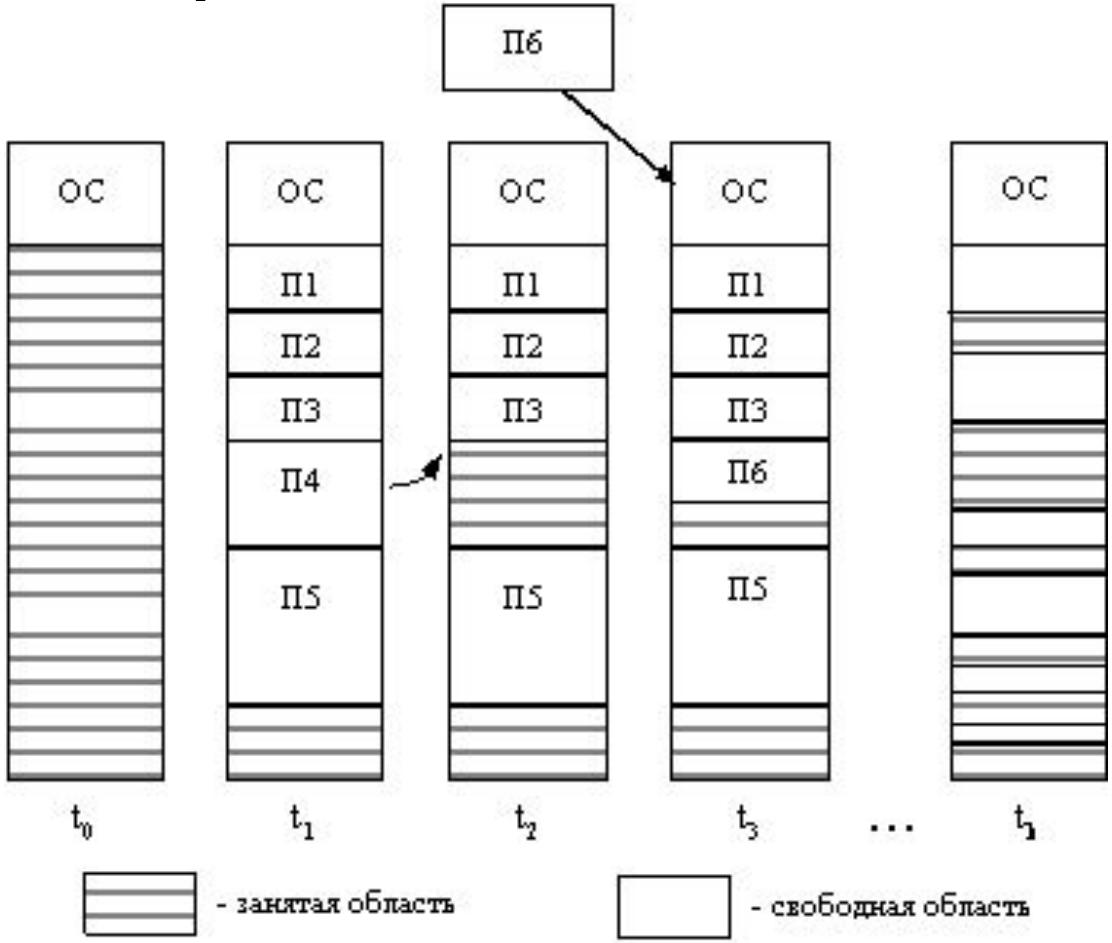
Распределение памяти фиксированными разделами



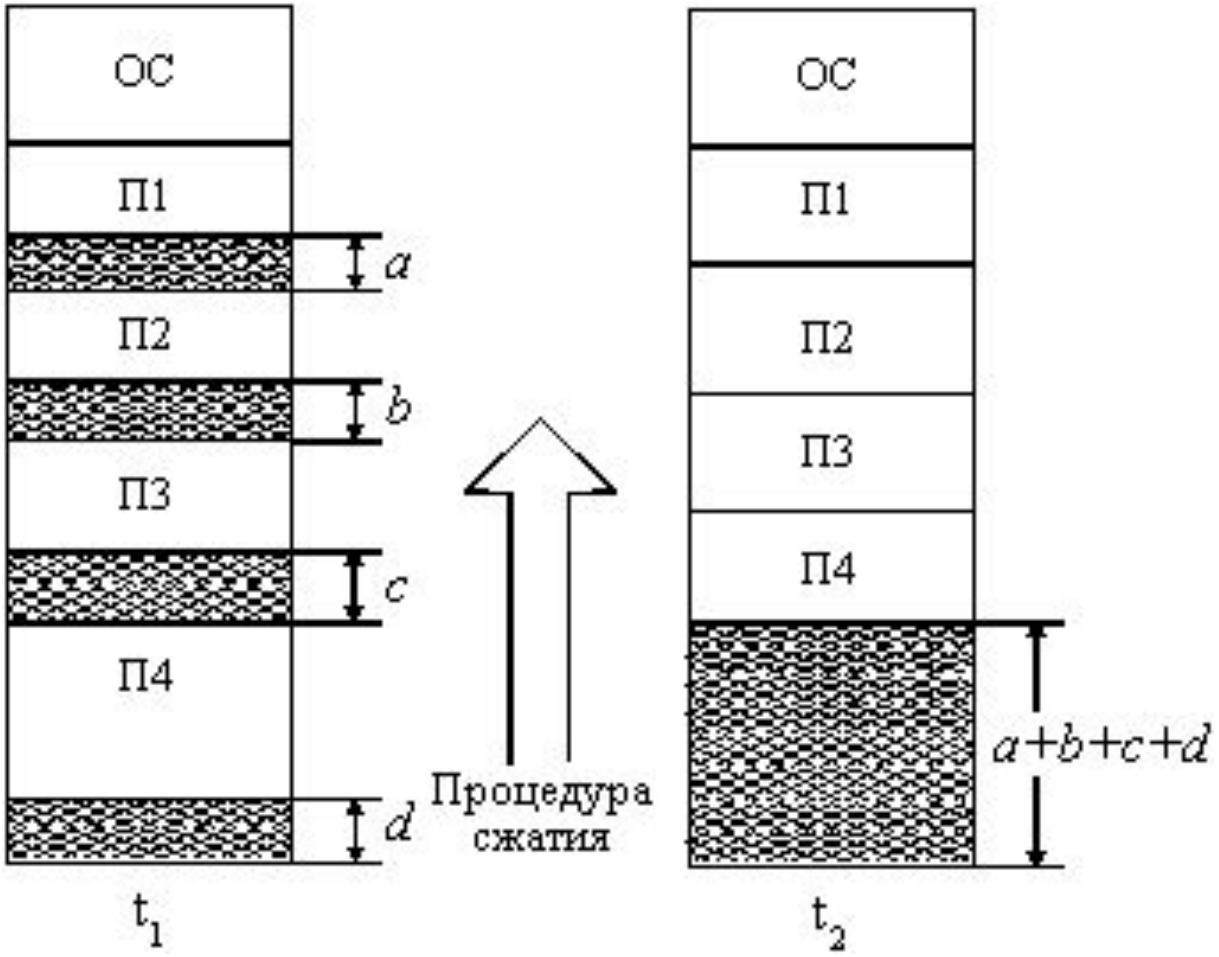
а - с общей очередью;

б - с отдельными очередями

Распределение памяти разделами переменной величины



Перемещаемые разделы



Основы управления памятью

Методы распределения памяти с использованием дискового пространства

Понятие виртуальной памяти

Виртуальная память (ВП) – это совокупность программно-аппаратных средств, позволяющих использовать программы, размер которых превосходит имеющуюся оперативную память.

Для этого менеджер ВП решает следующие задачи:

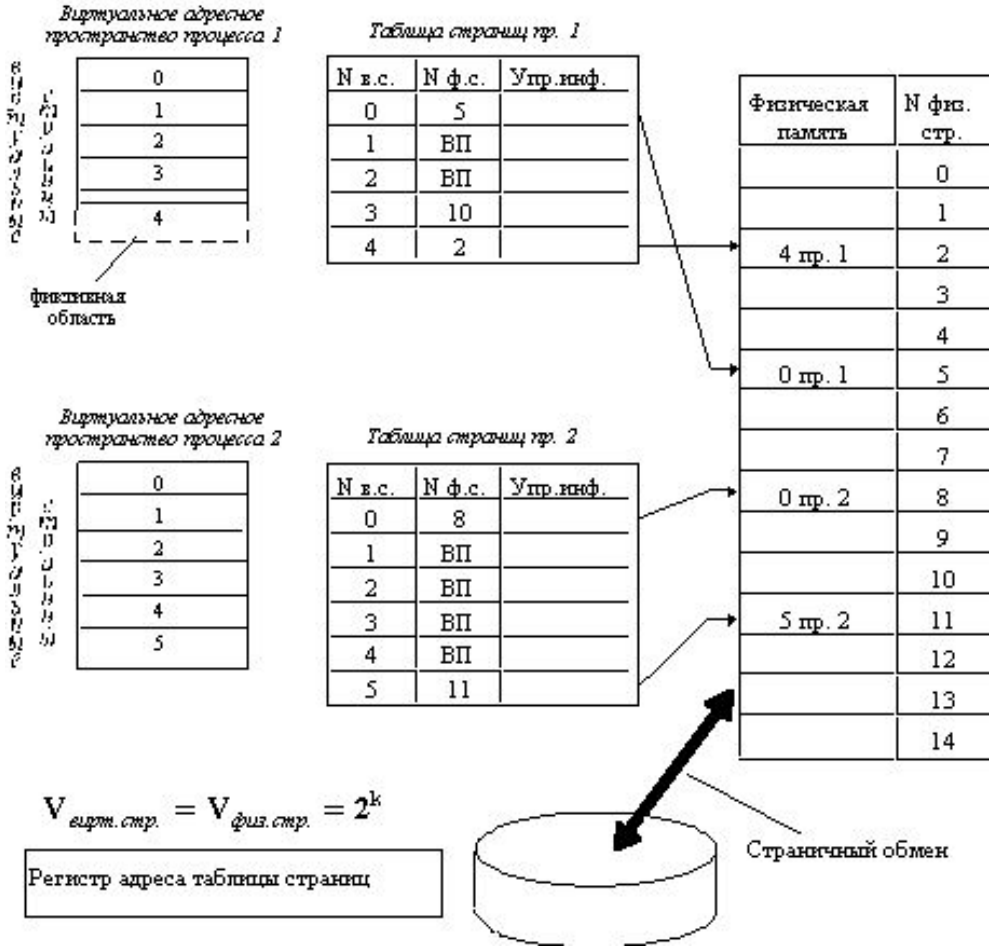
- размещает данные в запоминающих устройствах разного типа, например, часть программы в оперативной памяти, а часть на диске;
- перемещает по мере необходимости данные между запоминающими устройствами разного типа, например, подгружает нужную часть программы с диска в оперативную память;
- преобразует виртуальные адреса в физические.

Физические и виртуальные адреса

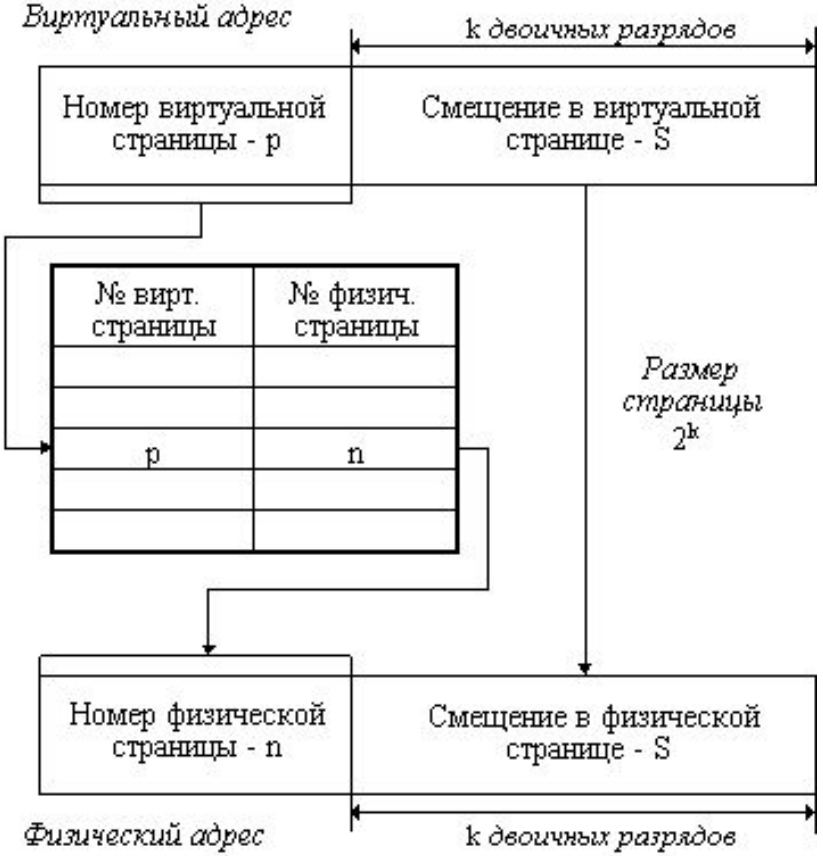
Суть концепции виртуальной памяти заключается в том, что адреса, к которым обращается выполняющийся процесс, отделяются от адресов, реально существующих в первичной памяти:

- адреса, на которые делает ссылки выполняющийся процесс, называются *виртуальными адресами (ВА).* Диапазон ВА, к которым может обращаться выполняющийся процесс, называется *пространством виртуальных адресов* V этого процесса.
- адреса, которые существуют в первичной памяти, называются *реальными* (или физическими) *адресами (ФА)*. Диапазон ФА, существующих в конкретной ЭВМ, называется *пространством реальных адресов* R этой ВС.

Страничное распределение

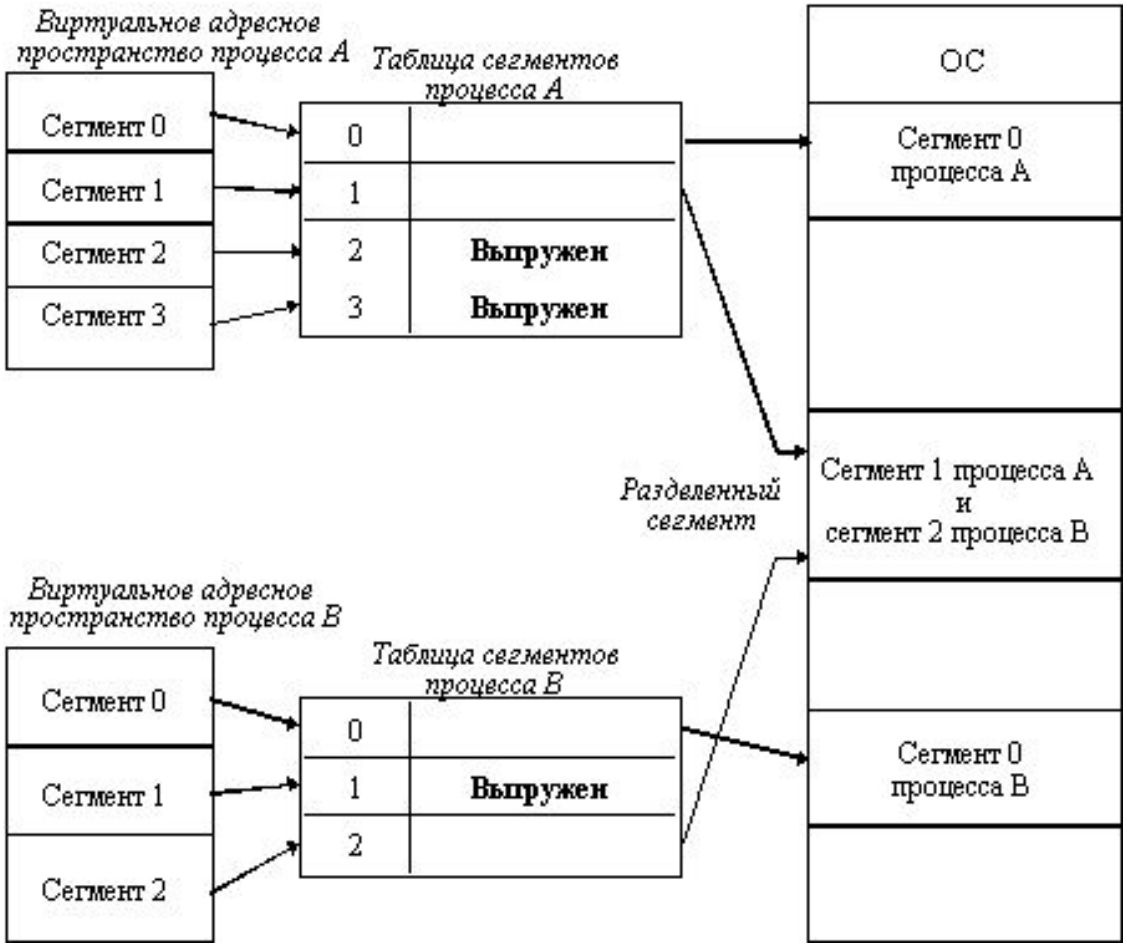


Страничное распределение: преобразования ВА в ФА

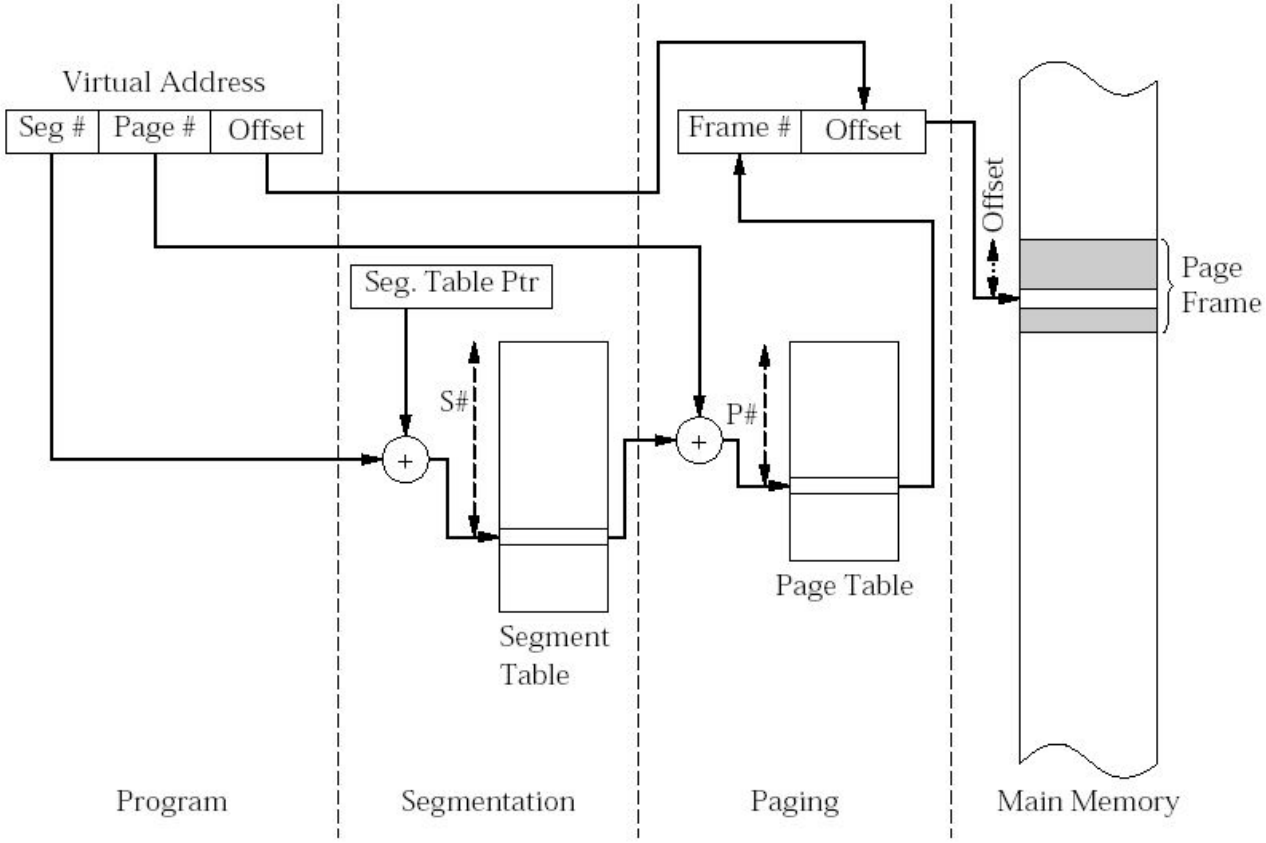


1. на основании начального адреса таблицы страниц, номера виртуальной страницы и длины записи в таблице страниц определяется адрес нужной записи в таблице,
2. из этой записи извлекается номер физической страницы,
3. к номеру физической страницы присоединяется смещение.

Сегментное распределение



Странично-сегментное распределение



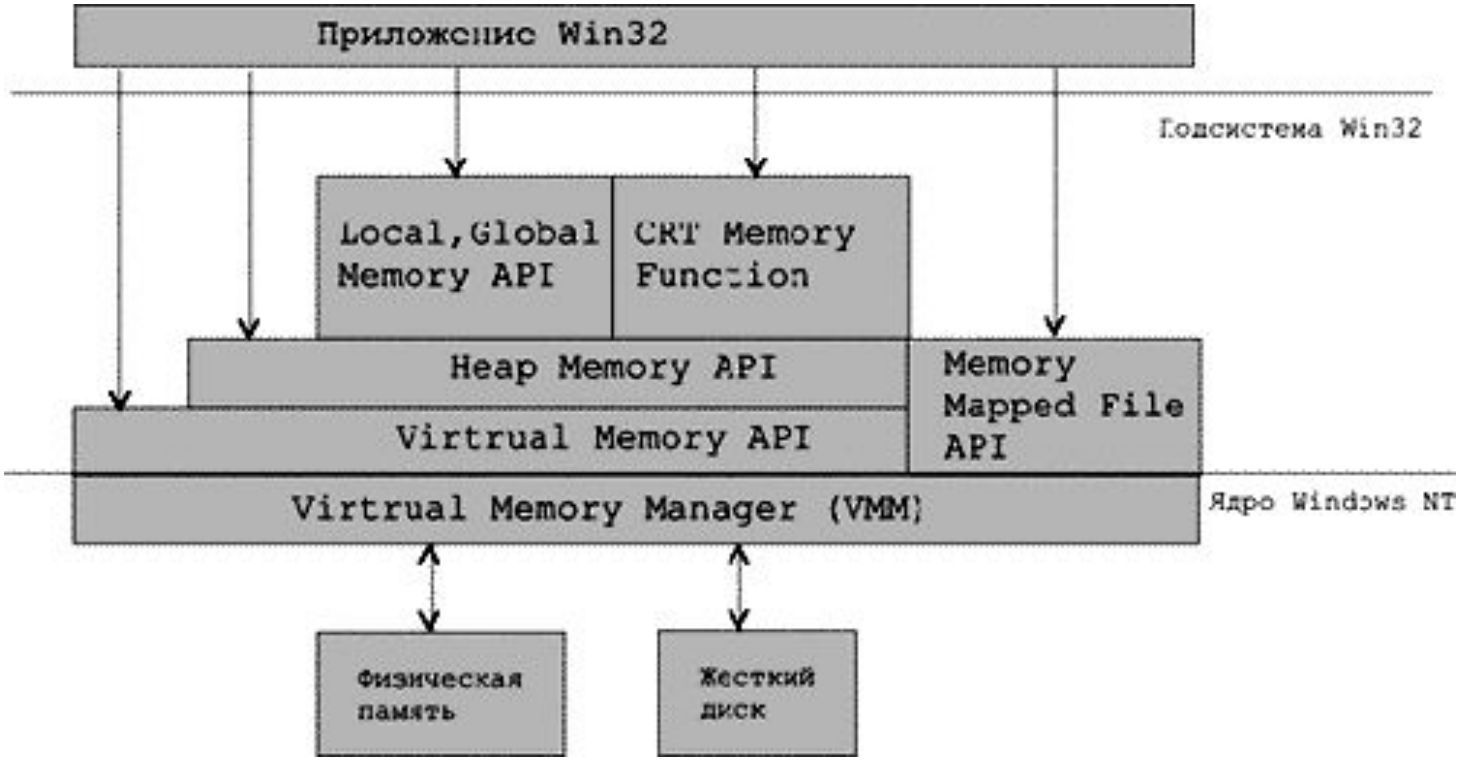
Алгоритмы замещения страниц (свопинга)

- Замещение случайной страницы
- **FIFO** (First In First Out) – замещение первой использованной страницы
- **LRU** (Least Recently Used) – замещение дольше всех неиспользовавшихся страниц
- **NRU** (Not Recently Used) или clock – замещение не использовавшихся в последнее время страницы
- **LFU** (Least Frequently Used) – замещение наименее часто используемых страниц

Операционные системы

Архитектура памяти в Win32 API. Общие принципы

Архитектура API управления памятью



Менеджер виртуальной памяти

VMM (Virtual Memory Manager)

- управление виртуальным адресным пространством процесса;
- разделение памяти между процессами;
- защита виртуальной памяти одного процесса от других процессов.

Адресное пространство процесса

Windows NT 4.0

Windows NT 4.0 Enterprise

Средства защиты памяти

- **Отдельное адресное пространство для каждого процесса.** Аппаратура запрещает процессу доступ к физическим адресам другого процесса.
- **Два режима работы**: режим ядра, в котором процессам разрешен доступ к системным данным, и пользовательский режим, в котором это запрещен.
- **Страничный механизм защиты.** Каждая виртуальная страница имеет набор признаков, который определяет разрешенные типы доступа в пользовательском режиме и в режиме ядра.
- **Объектно-ориентированная защита памяти.** Каждый раз, когда процесс открывает указатель на секцию, монитор ссылок безопасности проверяет, разрешен ли доступ процесса к данному объекту.

Операционные системы

Управление виртуальной памятью в Win32

Каталог страниц и свопинг

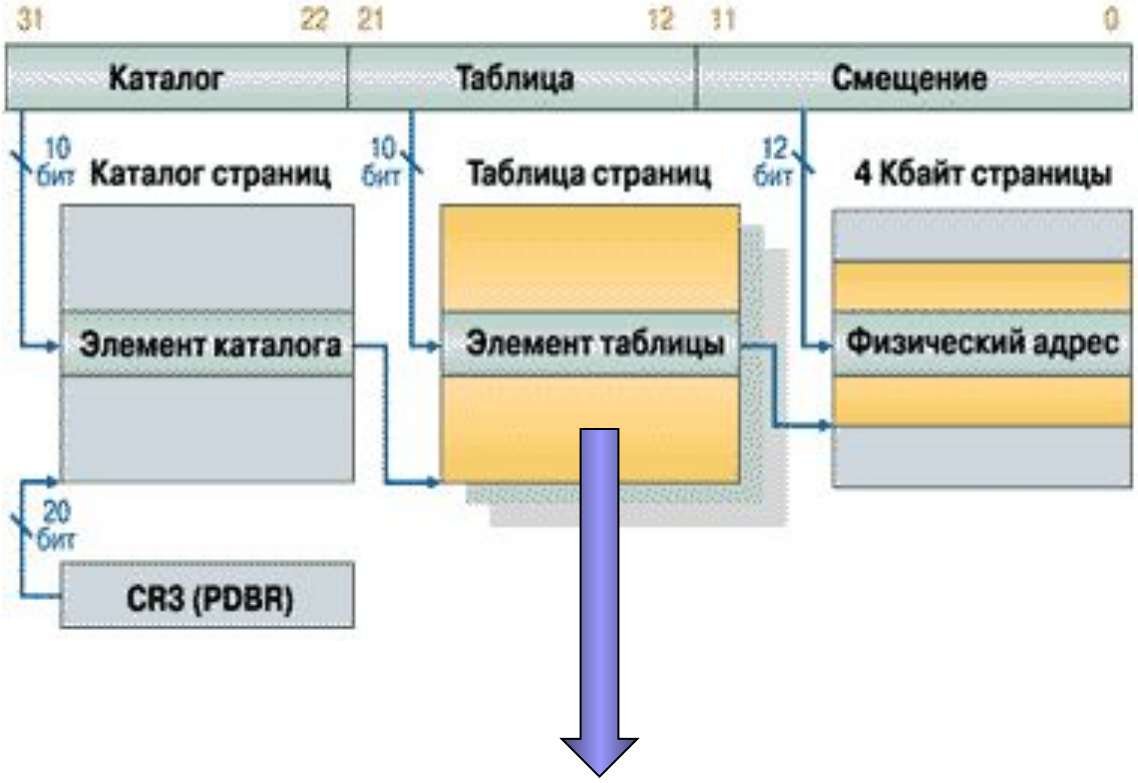
Каждому процессу назначается свой каталог страниц. Именно поэтому адресное пространство каждого процесса изолировано, что очень хорошо с точки зрения защиты процессов друг от друга.

Для того, чтобы обеспечить все линейное адресное пространство процесса физическими ячейками памяти. Windows NT-2000 применяет свопинг (swapping).

Организацией свопинга занимается VMM. При генерации системы на диске образуется специальный файл свопинга, куда записываются те страницы, которым не находится места в физической памяти. Процессы могут захватывать память в своем 32-битном адресном пространстве и, затем, использовать ее. Страница может иметь различные состояния.

VMM использует алгоритм LRU (Least Recently Used) – замещение дольше всех неиспользовавшихся страниц.

Страничное преобразование



Элемент таблицы страниц (**P**age **T**able **E**lement)

Элемент таблицы страниц

- Защита – Win32 API поддерживает три допустимых значения: *PAGE_NOACCESS, PAGE_READONLY u PAGE_READWRITE*.
- Базовый физический адрес страницы в памяти.
- Pagefile – индекс используемого файла подкачки (один из 16 возможных в системе файлов).
- State – состояние страницы в системе:
  - T (Transition) – отмечает страницу как переходную;
  - D (Dirty) – страница, в которую была произведена запись;
  - P (Present) – страница присутствует в ОП или находится в файле подкачки.

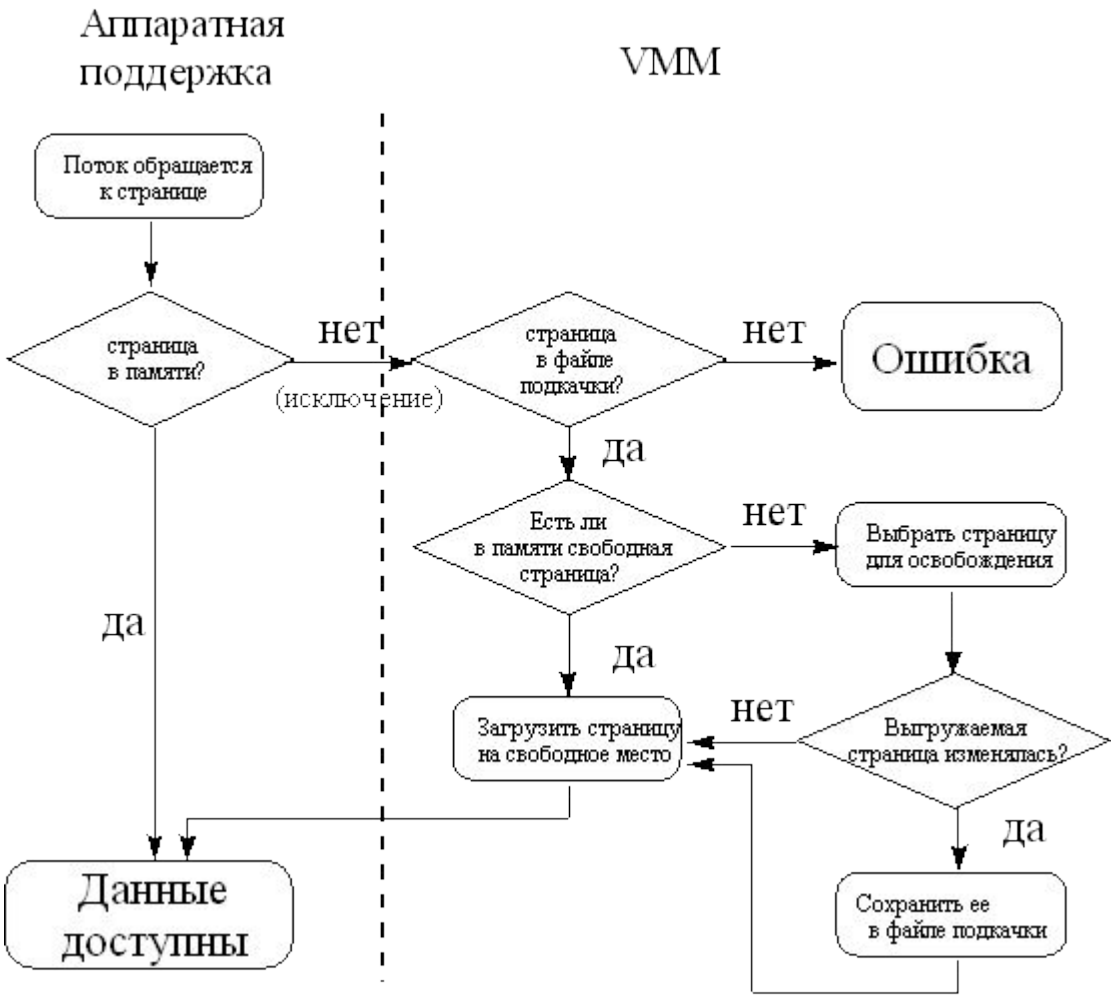
Отдельные состояния страниц

- Valid – страница используется процессом. Она реально существует в ОП и помечена в PTE как присутствующая в рабочем множестве процесса (P=1, T=0).
- Modified – содержимое страницы было изменено (D=1). В PTE страница помечена как отсутствующая (P=0) и переходная (T=1).
- Standby – содержимое страницы не изменялось (D=0). В PTE страница помечена как отсутствующая (P=0) и переходная (T=1).
- Free – страница, на которую не ссылается ни один PTE. Страница свободна, но подлежит обнулению, прежде чем будет использована.
- Zeroed – свободная и обнуленная страница, пригодная к непосредственному использованию любым процессом.
- Bad – страница, которая вызывает аппаратные ошибки и не может быть использована ни одним процессом.

Реализация свопинга

С понятием свопинга связаны три стратегии:

- *выборка* (fetch);
- *размещение* (placement);
- *замещение* (replacement).



Архитектура памяти в Win32 API

Организация «статической» виртуальной памяти

Работа приложений с виртуальной памятью

Блок адресов в адресном пространстве процесса может находиться в одном из трех состояний

- Выделен (committed) – блоку адресов назначена физическая память либо часть файла подкачки.
- Зарезервирован (reserved) – блок адресов помечен как занятый, но физическая память не распределена.
- Свободен (free) – блок адресов не выделен и не зарезервирован.

Функции API для работы виртуальной памятью

- **VirtualAlloc**
- ***VirtualFree***
- ***VirtualQuery***
- ***VirtualLock***
- ***VirtualUnlock***
- ***VirtualProtect***
- ***VirtualProtectEx***

Архитектура памяти в Win32 API

Организация «динамической» виртуальной памяти

Кучи (heaps)

Кучи (heaps) – это динамически распределяемые области данных.

- HANDLE GetProcessHeap(void) – для получения дескриптора кучи по умолчанию;
- LPVOID HeapAlloc(HANDLE hHeap, DWORD dwFlags, DWORD dwSize) – для выделения из кучи блока памяти заданного размера и возвращения указателя;
- LPVOID HeapReAlloc(HANDLE hHeap, DWORD dwFlags, LPVOID lpOldBlock, DWORD dwSize) – для изменения размера выделенного блока памяти с возможностью перемещения блока при необходимости;
- BOOL HeapFree(HANDLE hHeap, DWORD dwFlags, LPVOID lpMem) – для освобождения выделенного блока памяти кучи.

Создание новых куч

- для защиты друг от друга различных структур данных;
- для повышения эффективности управления памятью;
- для уменьшения рабочего множества процесса;
- для повышения эффективности работы многонитевых приложений.

HANDLE HeapCreate (DWORD dwFlags, DWORD dwInitialSize, DWORD dwMaximumSize).

Архитектура памяти в Win32 API

Файлы, проецируемые в память

Проецируемые файлы

“Как и виртуальная память, проецируемые файлы позволяют резервировать регион адресного пространства и передавать ему физическую память. Различие между этими механизмами состоит в том, что в последнем случае физическая память не выделяется из системного страничного файла, а берется из файла, уже находящегося на диске. Как только файл спроецирован в память, к нему можно обращаться так, как будто он в нее целиком загружен.”

(Джеффри Рихтер. Windows для профессионалов.)

Применение проецируемых файлов

Этот механизм имеет три применения в Win32:

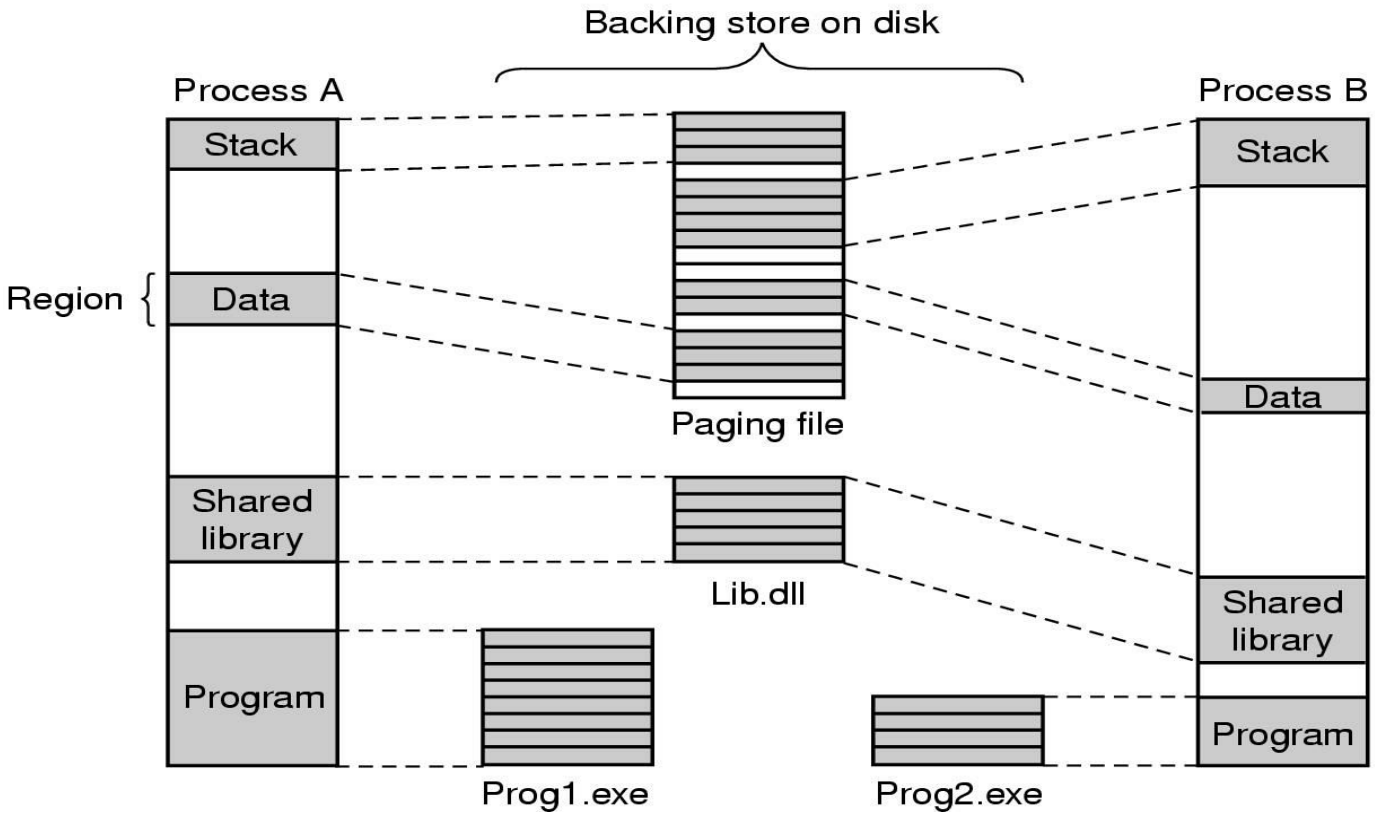
- Для запуска исполняемых файлов (EXE) и динамически связываемых библиотек (DLL).
- Для работы с файлами.
- Для одновременного использования одной области данных двумя процессами.

Запуск исполняемых файлов и DLL

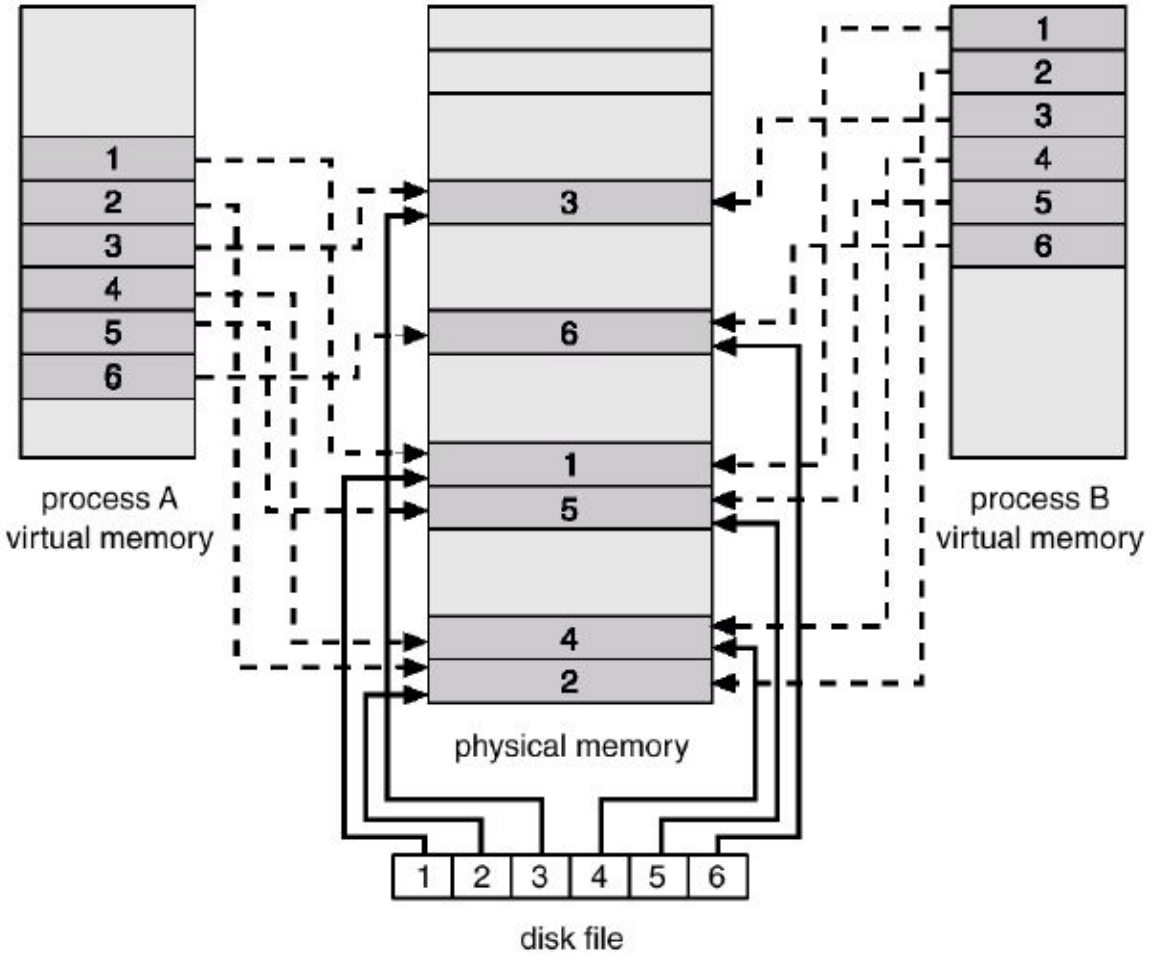
При исполнении функции ***CreateProcess*** система обращается к VMM для выполнения следующих действий

1. Создать адресное пространство процесса (размером 4Gb).
2. Резервировать в адресном пространстве процесса регион размером, достаточным для размещения исполняемого файла. Начальный адрес региона определяется в заголовке EXE-модуля. Обычно он равен 0x00400000.
3. Отобразить исполняемый файл на зарезервированное адресное пространство.
4. Таким же образом отобразить на адресное пространство процесса необходимые ему динамически связываемые библиотеки. Информация о необходимых библиотеках находится в заголовке EXE-модуля. Желательное расположение региона адресов описано внутри библиотеки.

Запуск EXE-файлов и DLL-библиотек



Одновременное использование одной области данных двумя процессами



Файлы данных, проецируемые в память

Проецирование файла данных в память:

- Создается объект ядра “файл”. Для создания объекта “файл” используется функция ***CreateFile***.
- С помощью функции ***CreateFileMapping*** создается объект ядра “проецируемый файл”. При этом используется дескриптор файла, возвращенный функцией CreateFile.
- Производится отображение объекта “проецируемый файл” или его части на адресное пространство процесса. Для этого применяется функция **MapViewOfFile**.

Завершение проецирования файла данных:

- Выполняется открепление файла от адресного пространства процесса с помощью функции ***UnmapViewOfFile***.
- Выполняется уничтожение объектов “файл” и “проецируемый файл” с помощью функции ***CloseHandle***.

Операционные системы

Управление центральным процессором и объединение ресурсов

Управление центральным процессором…

Процессы и потоки

Основные понятия

- Задание – набор процессов, управляемых как единое целое, с общими квотами и лимитами
- **Процесс** – контейнер для ресурсов
- **Поток** – сущность планируемая ядром
- Волокно – облегченный поток, управляемый полностью в пространстве пользователя

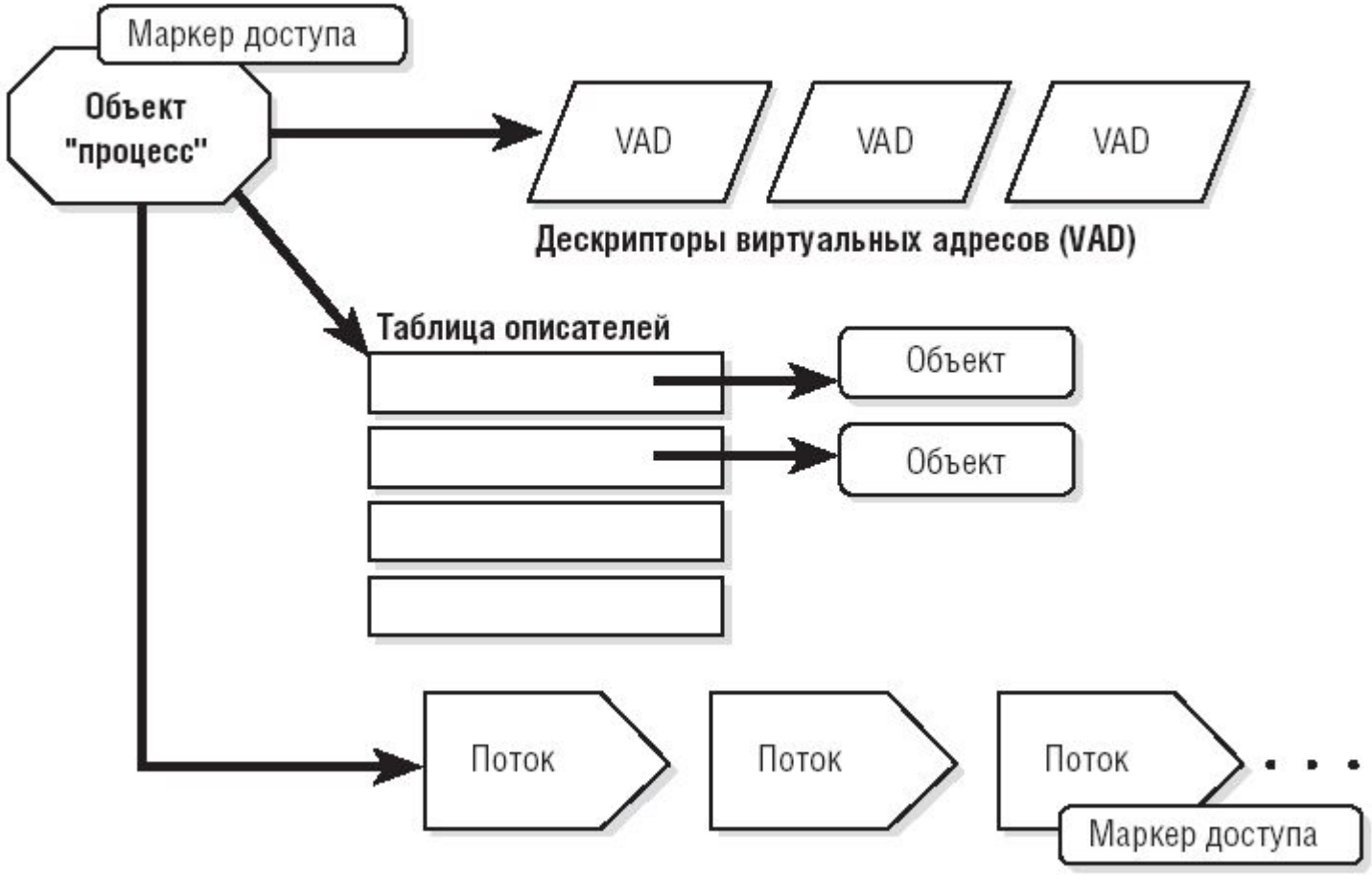
Процессы

Процесс – это совокупность системных ресурсов, задействованная для выполнения определенной работы.

Понятие "процесс" включает следующее:

- исполняемый код;
- собственное адресное пространство, которое представляет собой совокупность виртуальных адресов, которые может использовать процесс;
- ресурсы системы, такие как файлы, семафоры и т.п., которые назначены процессу операционной системой;
- хотя бы одну выполняемую нить.

Процесс и его ресурсы



Атрибуты процесса в Windows NT-2000

- Идентификатор процесса – уникальное значение, которое идентифицирует процесс в рамках операционной системы.
- Закрытое *виртуальное адресное пространство* – диапазон адресов виртуальной памяти, которым может пользоваться процесс.
- Исполняемую программу – начальный код и данные, проецируемые на виртуальное адресное пространство процесса.
- Список открытых дескрипторов различных системных ресурсов – семафоров, файлов и других объектов, доступных всем потокам в данном процессе.
- Токен доступа – исполняемый объект, содержащий информацию о безопасности и идентифицирующий пользователя, группы безопасности и привилегии, сопоставленные с процессом.
- Базовый приоритет – основа для исполнительного приоритета нитей процесса.
- Процессорная совместимость – набор процессоров, на которых могут выполняться нити процесса.
- Предельные значения квот ресурсов (см. Job).
- Время исполнения – общее количество времени, в течение которого выполняются все нити процесса.
- Список потоков процесса (как минимум один поток).

Нити (thread)

Нить (поток) – это непрерывная последовательность инструкций, выполняющих определенную функцию.

Для выполнения нити необходимы две вещи: системное время и адресное пространство.

Нить не имеет собственного адресного пространства и получает доступ к адресному пространству процесса-родителя.

Атрибуты нити в Windows NT-2000

- Идентификатор клиента – уникальное значение, которое идентифицирует нить при ее обращении к серверу.
- Контекст нити – информация, которая необходима ОС для того, чтобы продолжить выполнение прерванной нити. Контекст нити содержит текущее состояние регистров, стеков и индивидуальной области памяти.
- Два стека, один из которых используется потоком при выполнении в режиме ядра, а другой – в пользовательском режиме;
- Локальная память потока (thread local storage, TLS) – закрытая область памяти, используемую подсистемами, библиотеками исполняющих систем (runtime libraries) и DLL;
- Динамический приоритет – значение приоритета нити в данный момент.
- Базовый приоритет – нижний предел динамического приоритета нити.
- Процессорная совместимость нитей – перечень типов процессоров, на которых может выполняться нить.
- Время выполнения нити – суммарное время выполнения нити в пользовательском режиме и в режиме ядра.
- Состояние предупреждения – флаг, который показывает, что нить должна выполнять вызов асинхронной процедуры.
- Счетчик приостановок – текущее количество приостановок выполнения нити.

Задание (job) в Windows 2000

- В Windows 2000 в модель процессов введено новое расширение – *задания* (jobs).
- Задания предназначены в основном для того, чтобы группами процессов можно было оперировать и управлять как единым целым.
- Объект-задание позволяет устанавливать определенные атрибуты и накладывать ограничения на процесс или процессы, сопоставленные с заданием.

Квоты и лимиты для заданий (job)

- квоты (базовые и расширенные базовые ограничения):
  - максимальное количество процессов;
  - суммарное время центрального процессора (для каждого процесса и для задания в целом);
  - максимальное количество используемой памяти (для каждого процесса и для задания в целом);
- базовые ограничения по пользовательскому интерфейсу;
- ограничения, связанные с защитой.

Волокна (fibers) в Windows 2000

- Введены в Windows 2000 для переноса существующих серверных приложений из UNIX.
- Реализованы на уровне кода пользовательского режима.
- Процессорное время между волокнами распределяется по пользовательскому алгоритму.

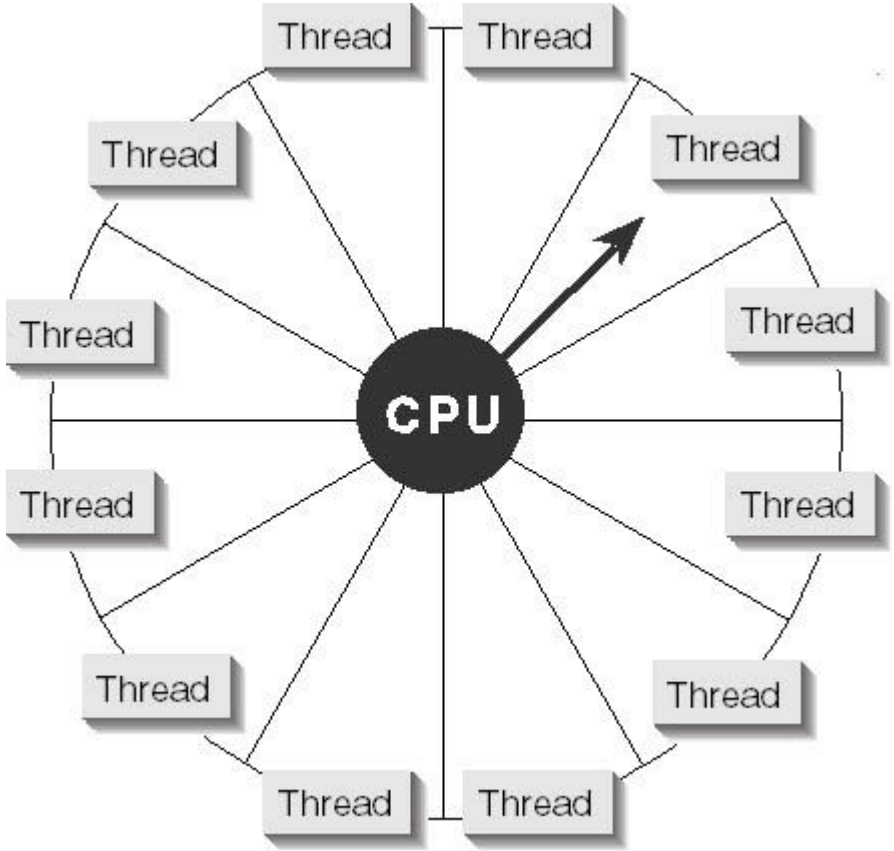
Управление центральным процессором…

Планирование загрузки процессорного времени

Системный планировщик в Windows NT-2000

- Вытесняющая мультипоточность.
- Квантование времени.
- Приоритетный режим обслуживания:
  - абсолютные приоритеты;
  - динамические приоритеты.

Квантование времени



Операционная система выделяет потокам кванты времени по принципу карусели

Приоритетный режим обслуживания

- Разработчик ПО может использовать приоритеты от 1 до 31.
- Нулевой приоритет зарезервирован для потока обнуления страниц.
- Поток наследует приоритет процесса, породившего его.
- ОС Windows NT 4.0 предоставляет 4 класса приоритетов: Realtime, High, Normal и Idle.
- ОС Windows 2000: еще 2 дополнительных класса приоритетов – Below Normal и Above Normal.
- Относительный приоритет потока: idle, lowest, below normal, normal (обычный), above normal, highest и time-critical.

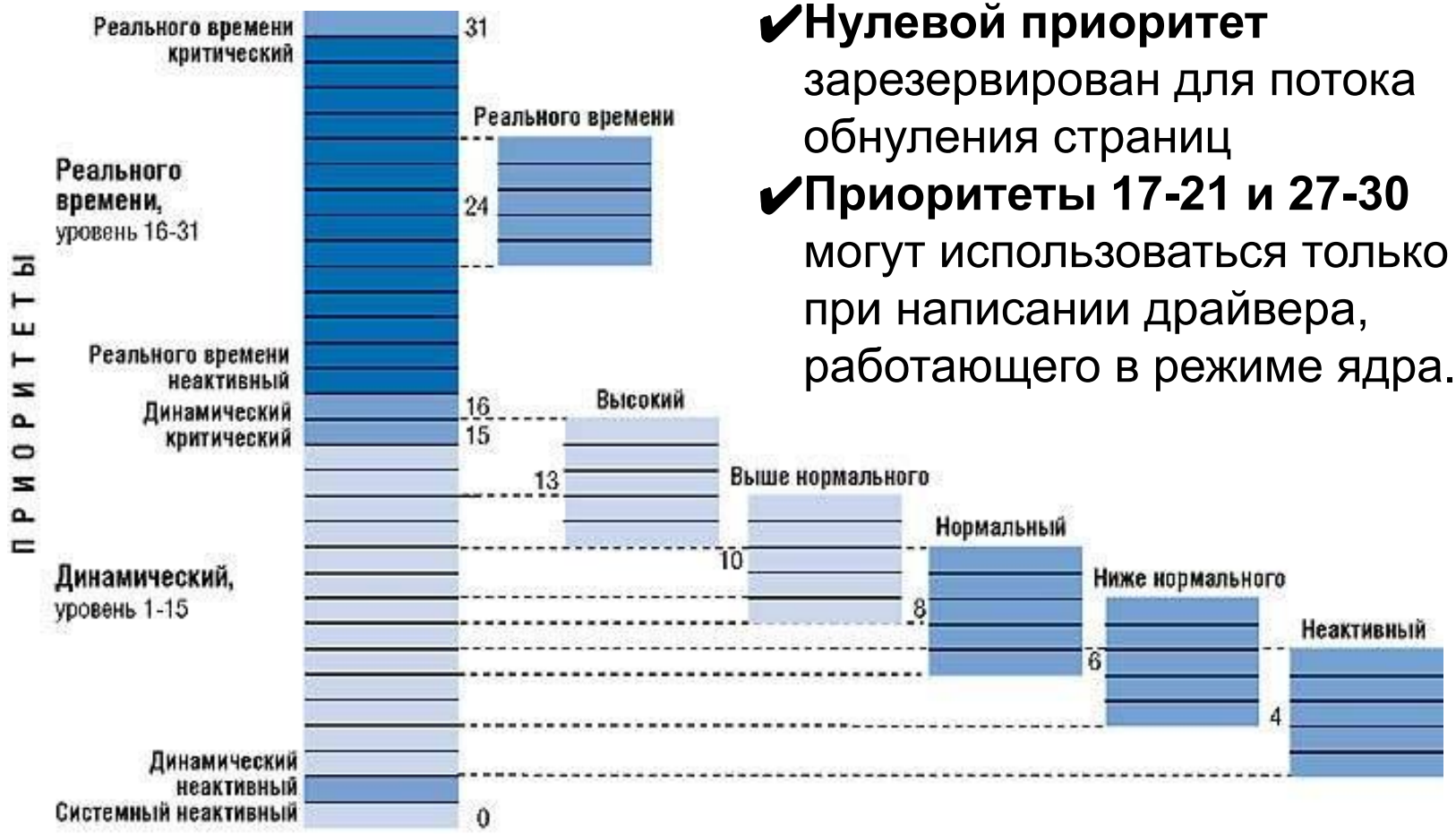
Классы приоритета процессов

Класс приоритета	Описание
Realtime (24)	Потоки в этом процессе обязаны немедленно реагировать на события, обеспечивая выполнение критических по времени задач. Такие потоки вытесняют даже компоненты ОС. Процессы этого класса нельзя запустить, если пользователь не имеет привилегии Increase Scheduling Priority (администратор и пользователь с расширенными полномочиями).
High (13)	Потоки в этом процессе тоже должны немедленно реагировать на события, обеспечивая выполнение критических по времени задач. Этот класс присвоен, например, Task Manager.
Above normal (10)	*Класс приоритета, промежуточный между normal и high.*
Normal (8)	Потоки в этом процессе не предъявляют особых требований к выделению им процессорного времени (*99%* приложений).
Below normal (6)	*Класс приоритета, промежуточный между normal и idle.*
Idle (4)	Потоки в этом процессе выполняются, когда система не занята другой работой. Этот класс приоритета обычно используется для утилит, работающих в фоновом режиме, экранных заставок и пр.

Относительные приоритеты потоков

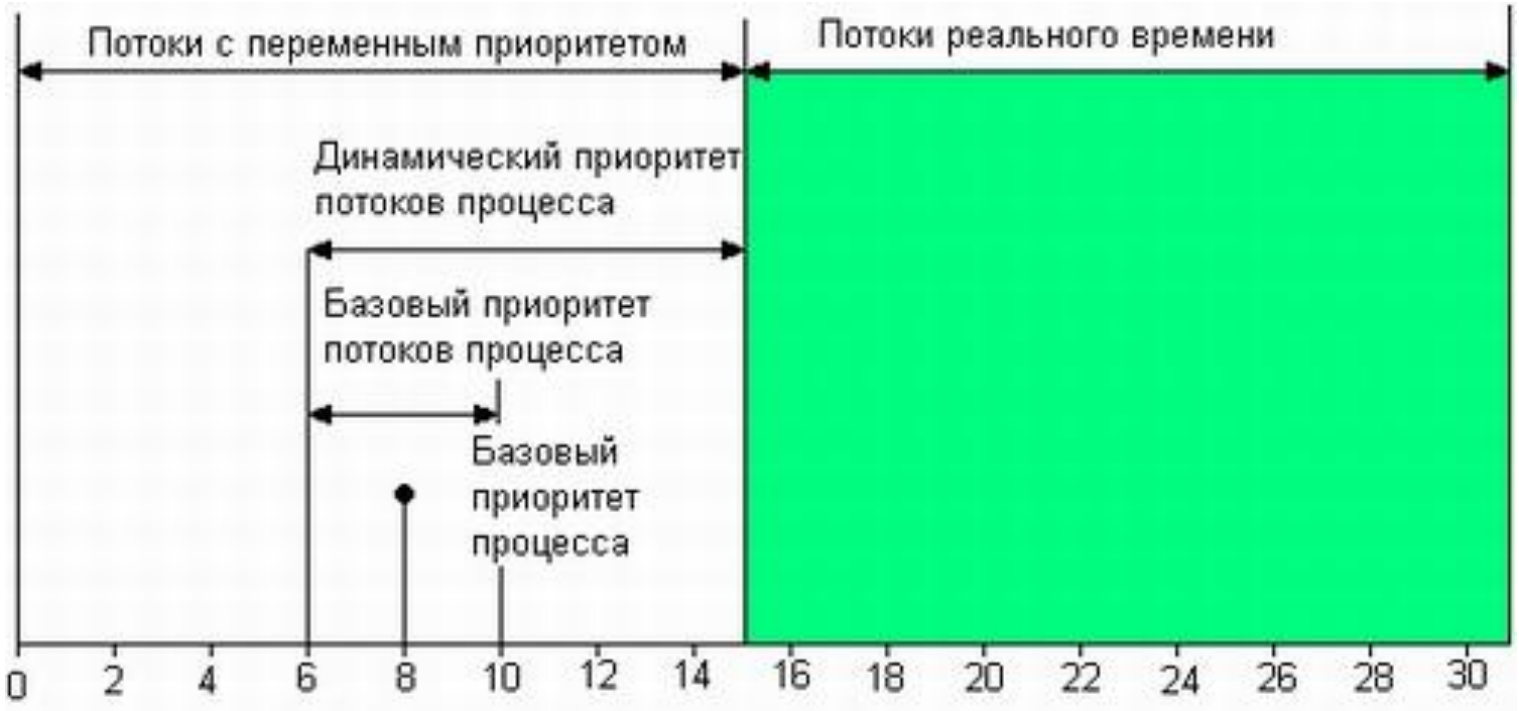
Относительный приоритет потока	Описание
Time-critical	Поток выполняется с приоритетом 31 в классе real-time и с приоритетом 15 в других классах
Highest	Поток выполняется с приоритетом на два уровня выше обычною для данного класса
Above normal	Поток выполняется с приоритетом на один уровень выше обычного для данного класса
Normal	Поток выполняется с обычным приоритетом процесса для данного класса
Below normal	Поток выполняется с приоритетом на один уровень ниже обычного для данного класса
Lowest	Поток выполняется с приоритетом на два уровня ниже обычного для данного класса
Idle	Поток выполняется с приоритетом 16 в классе real-time и с приоритетом 1 в других классах

Иллюстрация по приоритетам для Windows 2000



- ✔**Нулевой приоритет** зарезервирован для потока обнуления страниц
- ✔**Приоритеты 17-21 и 27-30** могут использоваться только при написании драйвера, работающего в режиме ядра.

Динамические приоритеты



В ходе выполнения нити ее приоритет (1-15) может меняться – механизм *адаптивного планирования.*

Принципы адаптивного планирования

- Если поток полностью исчерпал свой квант, то его приоритет понижается на некоторую величину.
- Приоритет потоков, которые перешли в состояние ожидания, не использовав полностью выделенный им квант, повышается.
- Приоритет не изменяется, если поток вытеснен более приоритетным потоком.
- Повышение приоритета для “голодающих” потоков.

“Голодающие” потоки

Пример: Представьте, что поток с приоритетом 4 готов к выполнению, но не может получить доступ к процессору из-за того, что его постоянно занимают потоки с приоритетом 8.

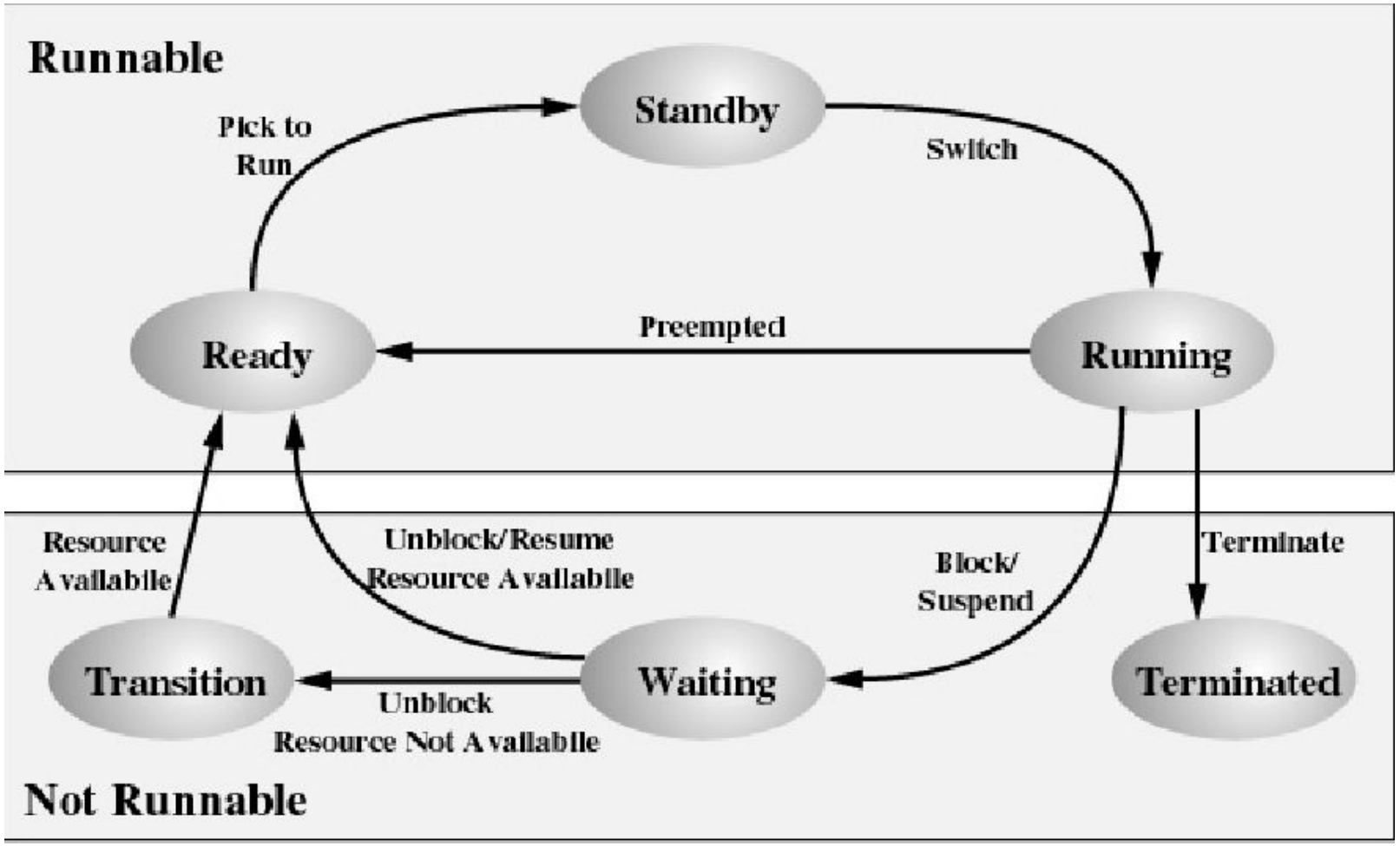
Это типичный случай "голодания" потока с более низким приоритетом. Обнаружив такой поток, не выполняемый на протяжении уже трех или четырех секунд, система поднимает его приоритет до 15 и выделяет ему двойную порцию времени. По его истечении потоку немедленно возвращается его базовый приоритет.

Алгоритм планировщика

Нить освобождает процессор, если:

- блокируется, уходя в состояние ожидания;
- завершается;
- исчерпан квант;
- в очереди готовых появляется более приоритетная нить.

Граф состояний нити для Windows 2000



Основы управления процессами

API Win32 для создания и завершения процессов

Создание процесса

BOOL **CreateProcess** (

PCTSTR pszApplicationName, *// имя исполняемого файла*

PTSTR pszCommandLine, *// командная строка*

PSECURITY_ATTRIBUTES psaProcess, *//*

PSECURITY_ATTRIBUTES psaThread, *// атрибуты защиты потоков*

BOOL bInheritHandles, *// наследование дескрипторов*

DWORD fdwCreate, *// флаги*

PVOID pvEnvironment, *// блок памяти, хранящий строки переменных окружения*

PCTSTR pszCurDir, *// текущий диск и каталог для процесса*

PSTARTUPINFO psiStartInfo, *// используется Windows-функциями при создании нового процесса*

PPROCESS_INFORMATION ppiProcInfo *// инициализируемая структура*

);

Флаги потоков

DEBUG_PROCESS DEBUG_ONLY_THIS_PROCESS

CREATE_SUSPENDED

DETACHED_PROCESS CREATE_NEW_CONSOLE
CREATE_NO_WINDOW

CREATE_BREAKAWAY_FROM_JOB

IDLE_PRIORITY_CLASS

BELOW_NORMAL_PRIORITY_CLASS

NORMAL_PRIORITY_CLASS

ABOVE_NORMAL_PRIORITY_CLASS

HIGH_PRIORITY_CLASS

REALTIME_PRIORITY_CLASS

Завершение процесса

- входная функция первичного потока возвращает управление (рекомендуемый способ);
- один из потоков процесса вызывает функцию *ExitProcess* (нежелательный способ);
- поток другого процесса вызывает функцию *TerminateProcess* (тоже нежелательно);
- все потоки процесса умирают по своей воле (большая редкость).

Функция ExitProcess

Процесс завершается, когда один из его потоков вызывает *ExitProcess:*

VOID ExitProcess(UINT fuExitCode);

Эта функция завершает процесс и заносит в параметр *fuExitCode* код завершения процесса.

Функция TerminateProcess

Вызов функции *TerminateProcess* тоже завершает процесс:

BOOL TerminateProcess(HANDLE hProcess, UINT fuExitCode);

Параметр *bProcess* идентифицирует описатель завершаемого процесса, а в параметре *fuExitCode* возвращается код завершения процесса.

Когда все потоки процесса “уходят”

Обнаружив, что в процессе не исполняется ни один поток, операционная система немедленно завершает его.

При этом код завершения процесса приравнивается коду завершения последнего потока.

Управление динамическими приоритетами потоков процесса

```
BOOL SetProcessPriorityBoost(
    HANDLE hProcess, // дескриптор процесса
    BOOL DisablePriorityBoost // состояние
    //форсированного приоритета
);
BOOL GetProcessPriorityBoost(
    HANDLE hProcess,          // дескриптор процесса
    PBOOL pDisablePriorityBoost // состояние
    //форсированного приоритета
);
```

Основы управления процессами

API Win32 для управления потоками

Создание потока

```
HANDLE CreateThread (
   PSECURITY_ATTRIBUTES psa,
   SIZE_T cbStack,
   PTHREAD_START_ROUTINE pfnStartAddr,
   PVOID pvParam,
   DWORD dwCreate,
   PDWORD pdwThreadID
);
```

Установка приоритета

Поток создается с приоритетом потока **THREAD_PRIORITY_NORMAL**. Используйте функции GetThreadPriority и SetThreadPriority, чтобы получать и установить приоритетное значение потока.

```
BOOL SetThreadPriority(
    HANDLE hThread, // дескриптор потока
    int nPriority   // уровень приоритета потока
);
```

Приоритеты потоков

Приоритет	Назначение
THREAD_PRIORITY_ABOVE_NORMAL	Приоритет на 1 пункт выше класса приоритета.
THREAD_PRIORITY_BELOW_NORMAL	Приоритет на 1 пункт ниже класса приоритета.
THREAD_PRIORITY_HIGHEST	Приоритет на 2 пункта выше класса приоритета.
THREAD_PRIORITY_IDLE	Базовый приоритет 1 для процессов **IDLE_PRIORITY_CLASS**, **BELOW_NORMAL_PRIORITY_CLASS**, **NORMAL_PRIORITY_CLASS**, **ABOVE_NORMAL_PRIORITY_CLASS** или **HIGH_PRIORITY_CLASS** и уровень базового приоритета 16 для процессов **REALTIME_PRIORITY_CLASS**.
THREAD_PRIORITY_LOWEST	Приоритет на 2 пункта ниже класса приоритета.
THREAD_PRIORITY_NORMAL	Нормальный приоритет класса приоритета.
THREAD_PRIORITY_TIME_CRITICAL	Базовый приоритет 15 для процессов **IDLE_PRIORITY_CLASS**, **BELOW_NORMAL_PRIORITY_CLASS**, **NORMAL_PRIORITY_CLASS**, **ABOVE_NORMAL_PRIORITY_CLASS** или **HIGH_PRIORITY_CLASS** и уровень базового приоритета 31 для процессов **REALTIME_PRIORITY_CLASS**.

Завершение потока

- функция потока возвращает управление (рекомендуемый способ);
- поток самоуничтожается вызовом функции *ExitThread* (нежелательный способ);
- один из потоков данного или стороннего процесса вызывает функцию *TerminateThread* (нежелательный способ);
- завершается процесс, содержащий данный поток (тоже нежелательно).

Функция ExitThread

Поток можно завершить принудительно, вызвав:

VOID ExitThread(DWORD dwExitCode);

В параметр *dwExitCode* Вы помещаете значение, которое система рассматривает как код завершения потока.

Функция TerminateThread

Вызов этой функции также завершает поток:

BOOL TerminateThread(HANDLE hThread,
DWORD dwExitCode);

В параметр *dwExitCode* помещается код завершения потока. После того как поток будет уничтожен, счетчик пользователей его объекта ядра "поток” уменьшится.

Если завершается процесс

Функции *ExitProcess* и *TerminateProcess* принудительно завершают потоки, принадлежащие завершаемому процессу.

Действия при завершении потока

- Освобождаются все описатели User-объектов, принадлежавших потоку.
- Код завершения потока меняется со STILL_ACTIVE на код, переданный в функцию *ExitThread* или *TerminateThread.*
- Объект ядра "поток" переводится в свободное состояние.
- Если данный поток является последним активным потоком в процессе, завершается и сам процесс.
- Счетчик пользователей объекта ядра "поток" уменьшается на 1.

BOOL GetExitCodeThread(HANDLE hThread, PDWORD pdwExitCode);

Управление динамическими приоритетами потока

```
BOOL SetThreadPriorityBoost(
    HANDLE hThread,           // дескриптор потока
    BOOL DisablePriorityBoost // состояние
    //форсирования приоритета
);
BOOL GetThreadPriorityBoost(
    HANDLE hThread,            // дескриптор потока
    PBOOL pDisablePriorityBoost // состояние форсажа
    //приоритета
);
```

Управление потоками

Флаг CREATE_SUSPENDED

DWORD ResumeThread(HANDLE hThread); - **возобновить поток**

DWORD SuspendThread(HANDLE hThread); - **“заморозка” потока**

Засыпание и переключение потоков

VOID **Sleep** (
  DWORD dwMilliseconds
); **- “заморозить” себя на определенное время**

BOOL **SwitchToThread**();

Определение периодов выполнения потока

```
BOOL GetThreadTimes(
   HANDLE hThread,
   PFILETIME pftCreationTime,
   PFILETIME pftExitTime,
   PFILETIME pftKernelTime,
   PFILETIME pftUserTime
);
```

Операционные системы

Межпроцессное взаимодействие

Виды межпроцессного взаимодействия (IPC)

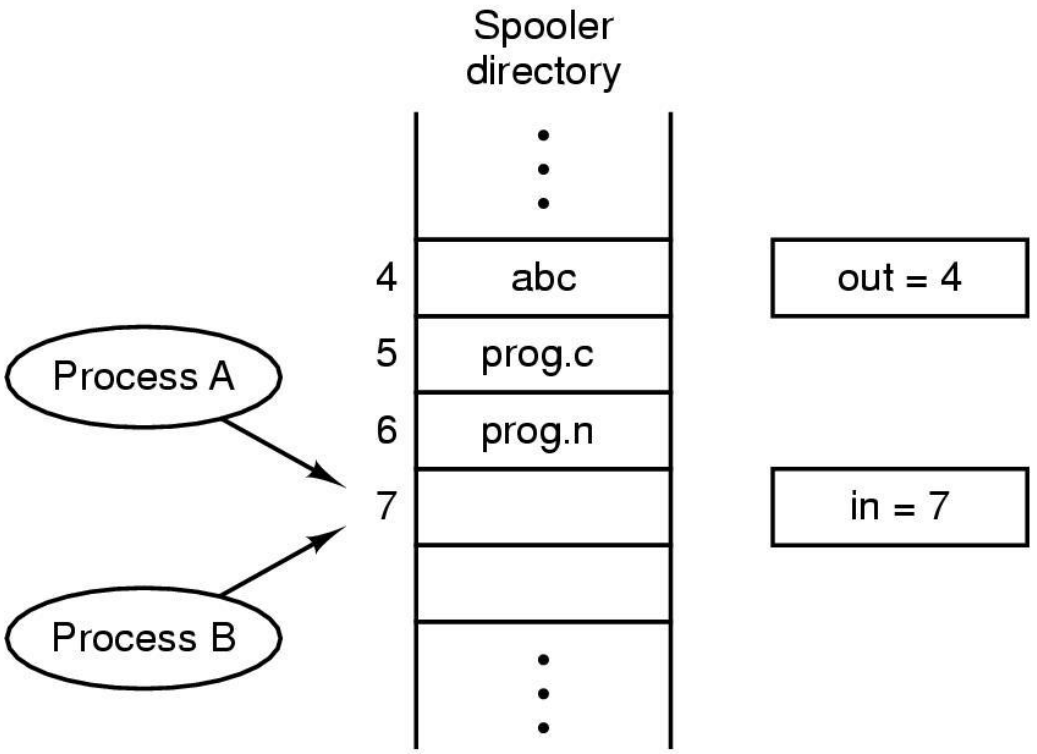
- Передача информации от одного процесса другому
- Предотвращение критических ситуаций
- Синхронизация процессов

Межпроцессное взаимодействие

Предотвращение критических ситуаций и средства синхронизации процессов

Возникновение гонок (состязаний)

Два процесса хотят получить доступ к общей памяти в одно и тоже время.



Критические секции

- Критическая секция – это часть программы, результат выполнения которой может непредсказуемо меняться, если переменные, относящиеся к этой части программы, изменяются другими потоками в то время, когда выполнение этой части еще не завершено.
- Во всех потоках, работающих с критическими данными, должна быть определена критическая секция.
- В разных потоках критическая секция состоит в общем случае из разных последовательностей команд.

Условия исключения гонок

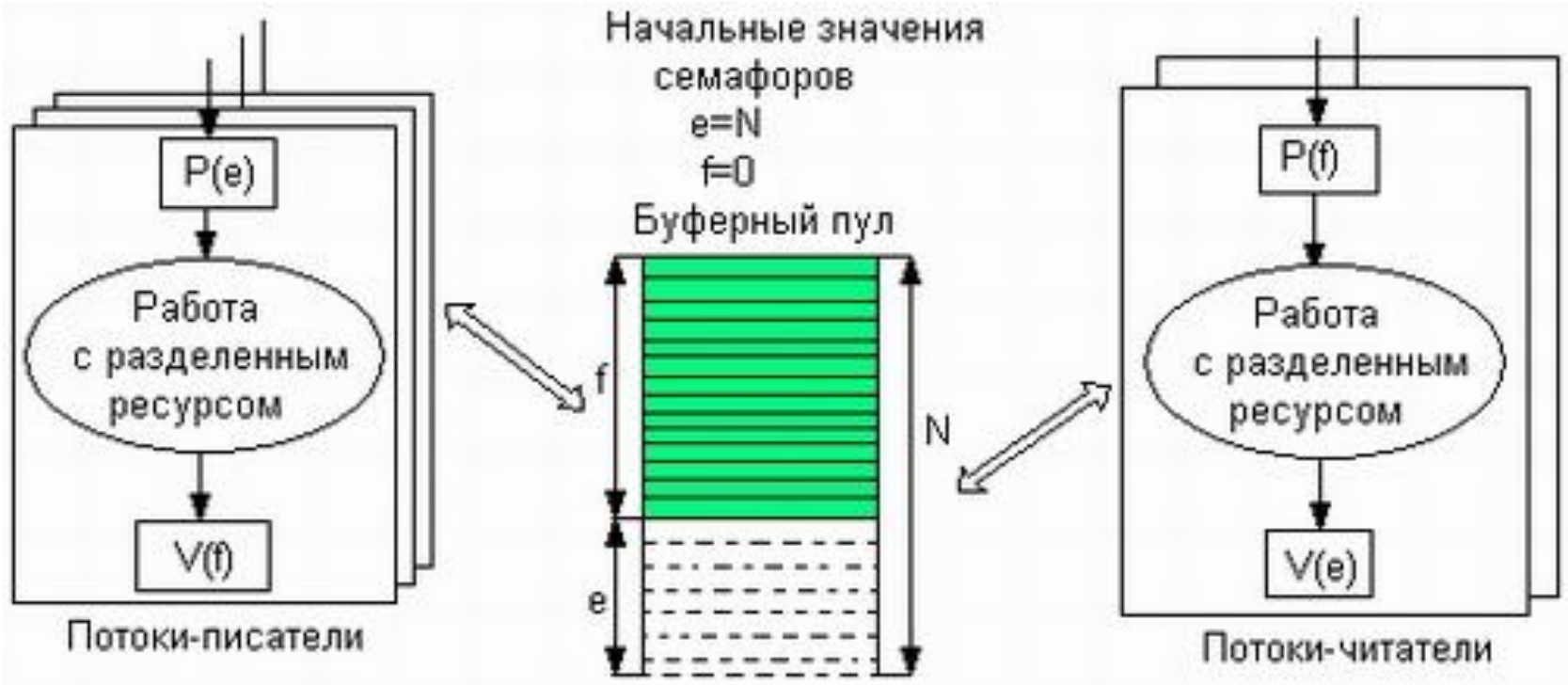
- Два процесса не должны одновременно находиться в критической секции
- В программе не должно быть предположений о скорости или количестве процессоров
- Процесс вне критической секции не может блокировать другие процессы
- Должна быть невозможна ситуация, когда процесс вечно ждет попадания в критическую секцию

Семафоры

Семафор - неотрицательная целая переменная S >= 0, которая может изменяться и проверяться только посредством двух неделимых примитивов:

- V(S): переменная S увеличивается на 1 единым неделимым действием. К переменной S нет доступа другим потокам во время выполнения этой операции.
- P(S): уменьшение S на 1, если это возможно. Если S=0 и невозможно уменьшить S, оставаясь в области целых неотрицательных значений, то в этом случае поток, вызывающий операцию P, ждет, пока это уменьшение станет возможным. Успешная проверка и уменьшение также являются неделимой операцией.

Задача о читателях и писателях



e — число пустых буферов, и f — число заполненных буферов

Мьютексы

- **Мьютекс** – переменная, которая может находиться в одном из двух состояний: блокированном или неблокированном.
- Если процесс хочет войти в критическую секцию – он вызывает примитив блокировки мьютекса. Если мьютекс не заблокирован, то запрос выполняется и процесс попадает в критическую секцию, иначе процесс попадает в очередь ожидания.

Межпроцессное взаимодействие

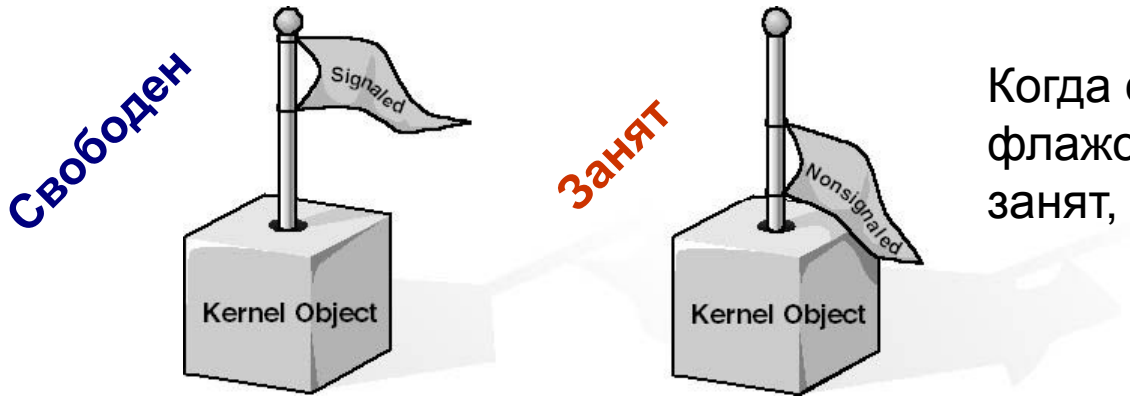
Синхронизация потоков с использованием объектов ядра Windows 2000

Синхронизация потоков

- **Синхронизация** означает способность потока приостанавливать свое исполнение и ждать, пока не завершится выполнение некоторой операции другим потоком.

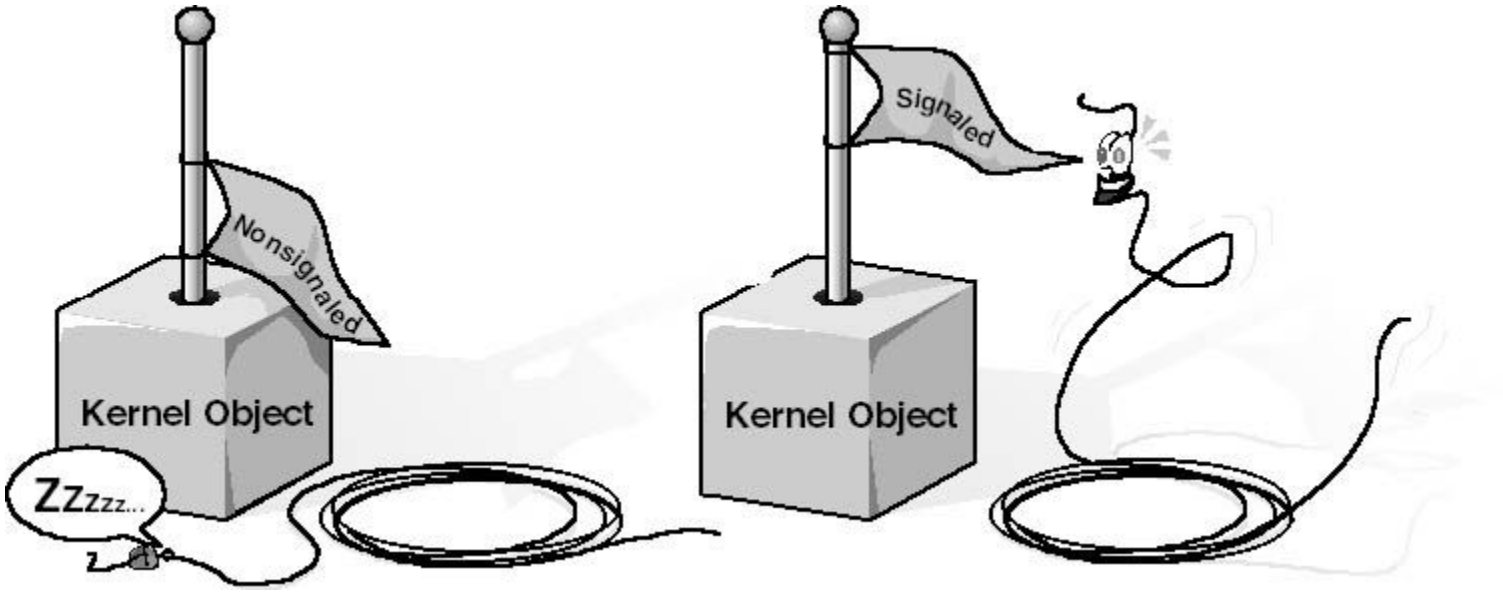
Объекты синхронизации и их состояния

- процессы
- потоки
- задания
- файлы
- консольный ввод
- уведомления об изменении файлов
- события
- ожидаемые таймеры
- семафоры
- мьютексы



Когда объект свободен, флажок поднят, а когда он занят, флажок опущен.

Спящие потоки



Функции ожидания

Ожидание одного объекта

```
DWORD WaitForSingleObject(
    HANDLE hObject,
    DWORD dwMilliseconds
);
```

Ожидание нескольких объектов

```
DWORD WaitForMultipleObjects(
    DWOHD dwCount, // кол-во
    CONST HANDLE* phObjects,
    BOOL fWaitAll,
    DWORD dwMilliseconds
);
```

Бесконечное ожидание объекта:

```
WaitForSingleObject(hProcess, INFINITE);
```

Объекты синхронизации

- События
- Ожидаемый таймер
- Семафор
- Мьютекс

События

События – самая примитивная разновидность объектов ядра. Они содержат счетчик числа пользователей (как и все объекты ядра) и две булевы переменные: одна сообщает тип данного объекта-события, другая – его состояние (свободен или занят).

Создание события

```
HANDLE CreateEvent(
   PSECURITY_ATTRIBUTES psa,
   BOOL fManualReset,
   BOOL fInitialState, // свободен / занят
   PCTSTR pszName );

HANDLE OpenEvent(
   DWORD fdwAccess,
   BOOL fInhent,
   PCTSTR pszName);
```

Управление событием

- Перевод события в свободное состояние:
  BOOL **SetEvent**(HANDLE hEvenet);
- Перевод события в занятое состояние:
  BOOL **ResetEvent**(HANDLE hEvent);
- Освобождение события и перевод его обратно в занятое состояние:
  BOOL **PulseEvent**(HANDLE hEvent);

Ожидаемые таймеры

Ожидаемые таймеры (waitable timers) – это объекты ядра, которые самостоятельно переходят в свободное состояние в определенное время или через регулярные промежутки времени.

Создание ожидаемого таймера

```
HANDLE CreateWaitableTimer(
   PSECURITY_ATTRIBUTES psa,
   BOOL fManualReset,
   PCTSTR pszName  // имя объекта );

HANDLE OpenWaitableTimer(
   DWORD dwDesiredAccess,
   BOOL bInheritHandle,
   PCTSTR pszName // имя объекта );
```

Управление ожидаемым таймером

```
BOOL SetWaitableTimer(
    HANDLE hTimer,
    const LARGE_INTEGER *pDueTime,
    LONG lPeriod,
    PTIMERAPCROUTINE
    pfnCompletionRoutine,
    PVOID pvArgToCompletionRoutine,
    BOOI fResume );

BOOL CancelWaitableTimer(HANDLE hTimer);
```

Создание семафора

```
HANDLE CreateSemaphore(
   PSECURITY_ATTRIBUTE psa,
   LONG lInitialCount, // начальное значение
   LONG lMaximumCount, // “объем”
   PCTRTR pszName);

HANDLE OpenSemaphore(
   DWORD fdwAccess,
   BOOL bInhentHandle,
   PCTSTR pszName);
```

Управление семафором

Поток получает доступ к ресурсу, вызывая одну из *Wait*-функций и передавая ей описатель семафора.

BOOL **ReleaseSemaphore**(
HANDLE hSem,
LONG lReleaseCount,
PLONG plPreviousCount);

Создание мьютекса

```
HANDLE CreateMutex(
   PSECURITY_ATTRIBUTES psa,
   BOOL fInitialOwner,
   PCTSTR pszName);

HANDLE OpenMutex(
   DWORD fdwAccess,
   BOOL fInheritHandle,
   PCTSTR pszName);
```

Управление мьютексом

Поток получает доступ к ресурсу, вызывая одну из *Wait*-функций и передавая ей описатель мьютекса.

BOOL **ReleaseMutex**(HANDLE hMutex);

Межпроцессное взаимодействие

Передача информации

Методы передачи информации

Сообщения WM_COPYDATA	Лучший способ пересылки блока данных из одной программы в другую.
Анонимные каналы (Anonymous pipes)	Полезны для организации прямой связи между двумя процессами на одном ПК.
Именованные каналы (Named pipes)	Полезны для организации прямой связи между двумя процессами на одном ПК или в сети.
Почтовые ячейки (mailslots)	Полезны для организации связи одного процесса со многими на одном ПК или в сети.
Гнезда (sockets)	Полезны для организации пересылки данных в гетерогенных средах.
Вызов удаленных процедур RPC	Слишком сложен, чтобы использовать его для простых пересылок данных.
Разделяемая память	Непросто выделить вне DLL.
Файлы отображаемой памяти	Обеспечивают одновременный доступ к объектам файла отображения из нескольких процессов.

Сообщение WM_COPYDATA

Отправитель:

```
COPYDATASTRUCT cds;
cds.cbData = (DWORD) nSize;  // Размер буфера
cds.lpData = (PVOID) pBuffer; // Буфер с данными
SendMessage (hWndTarget, WM_COPYDATA,
    (WPARAM) hWnd, (LPARAM) &cds);
```

Получатель:

```
PCOPYDATASTRUCT pcds = (PCOPYDATASTRUCT)
    lParam;
PBYTE pBuffer = (PBYTE) pcds -> lpData;
```

Анонимные каналы

Анонимные каналы не имеют имен.

Не пригодны для обмена через сеть.

Главная цель – служить каналом между родительским и дочерним процессом или между дочерними процессами.

Односторонний обмен.

Не возможен асинхронный обмен.

Анонимные каналы

BOOL **CreatePipe**(
PHANDLE *hReadPipe*,
PHANDLE *hWritePipe*,
LPSECURITY_ATTRIBUTES *lpPipeAttributes*,
DWORD *nSize*
);

ReadFile
WriteFile

Передача дескрипторов

- Установить параметр bInheritable структуры SECURITY_ATTRIBUTES в TRUE, чтобы дескрипторы могли наследоваться.
- Вызов функции CreateProcess с параметром bInheritHandles = TRUE
- Передача дескрипторов (командная строка, сообщения…)
- Вызов функции DuplicateHandle

Создание 2-ого дескриптора

BOOL **DuplicateHandle**(
HANDLE *hSourceProcessHandle*,
HANDLE *hSourceHandle*,
HANDLE *hTargetProcessHandle*,
LPHANDLE *lpTargetHandle*,
DWORD *dwDesiredAccess*,
BOOL *bInheritHandle*,
DWORD *dwOptions*
);

NPFS

- **Named Pipe File System** является виртуальной файловой системой, которая управляет каналами **named pipes**.
- Каналы named pipes относятся к классу файловых объектов (API Win32).
- RPC реализован как надстройка над NPFS;
- Канал представляет собой виртуальное соединение, по которому передается информация от одного процесса к другому.
- Канал может быть однонаправленным или двунаправленным (дуплексным).

Работа с именованными каналами

- Серверный процесс создает канал на локальном компьютере с помощью функции программного интерфейса Win32 **CreateNamedPipe**.
- Серверный процесс активизирует канал при помощи функции **ConnectNamedPipe**, после чего к каналу могут подключаться клиенты.
- Далее производится подключение к каналу \\computer_name\pipe\pipe_name посредством вызова функции **CreateFile**.

Создание именованного канала

```
HANDLE CreateNamedPipe (
    LPCTSTR lpName,
    DWORD dwOpenMode,
    DWORD dwPipeMode,
    DWORD nMaxInstances,
    DWORD nOutBufferSize,
    DWORD nInBufferSize,
    DWORD nDefaultTimeOut,
    LPSECURITY_ATTRIBUTES
    lpSecurityAttributes
);
```

Подключение к именованному каналу

```
BOOL ConnectNamedPipe (
   HANDLE hNamedPipe,
   LPOVERLAPPED lpOverlapped
);

BOOL DisconnectNamedPipe (
   HANDLE hNamedPipe
);
```

Обмен данными по именованному каналу

```
BOOL ReadFile/WriteFile (
   HANDLE hFile,
   LPVOID lpBuffer,
   DWORD nNumberOfBytesToRead,
   LPDWORD lpNumberOfBytesRead,
   LPOVERLAPPED lpOverlapped
);
```

Работа с каналом и ее завершение

- После установления виртуального соединение серверный процесс и клиентский процесс могут обмениваться информацией при помощи пар функций **"ReadFile"** и **"WriteFile"**.
- При помощи одного и того же канала сервер может одновременно обслуживать нескольких клиентов. Для этого серверный процесс может создать N-ное количество экземпляров канала, вызвав N-ное количество раз функцию **"CreateNamedPipe"** (при этом в каждом вызове должно быть указано одно и то же имя канала).
- Клиентский процесс может отключиться от канала в любой момент с помощью функции **"CloseHandle"**. Серверный процесс может отключить клиента в любой момент с помощью функции **"DisconnectNamedPipe"**.

Почтовые ящики (MailSlots)

Аналогичны именованным каналам, но предоставляют более простой и однонаправленный интерфейс. Процесс-сервер может завести почтовый ящик и дать ему имя, глобальное в сети. Любой клиент может с помощью операций работы с файлами отправить данные в этот ящик. Сервер, по мере необходимости, может читать переданные ему данные. Кроме этого, возможно широковещательная передача информации клиентом всем серверам домена.

Mailslot является одним из механизмов, предназначенных для осуществления обмена данными между процессами (IPC). При этом процессы могут быть запущены как на одной ПЭВМ (локально), так и разных ПЭВМ, включённых в одну ЛВС (удалённо).

- Для открытия канала, созданного на другом компьютере в сети, необходимо указать имя в формате
- \\ИмяКомпьютера\mailslot\[Путь]ИмяКанала
- Можно открыть канал для передачи информации сразу всем компьютерам указанного домена. Для этого формируется имя
- \\ИмяДомена\mailslot\[Путь]ИмяКанала
- Для передачи сообщения всем компьютерам первичного домена имя задается в форме
- *\mailslot\[Путь]ИмяКанала

Создание почтового ящика Mailslot на “сервере”

```
HANDLE CreateMailslot (
    LPCTSTR lpName, // имя
    DWORD nMaxMessageSize, // максимальный размер
    DWORD lReadTimeout, // интервал-тайм аута чтения
    LPSECURITY_ATTRIBUTES  lpSecurityAttributes
     // информация о безопасности
);
```

Создание клиента почтового ящика

```
HANDLE hSlot =
    CreateFile(("\\\\computername\\mailslot\\messngr",
    GENERIC_WRITE, FILE_SHARE_READ, NULL,
    OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL,
    NULL);

if (hSlot != INVALID_HANDLE_VALUE)  {
    char buf = "From\0\To\0Message\0";
    uint cb = sizeof(buf);
    WriteFile(hSlot, buf, cb, &cb, NULL);
    …
}
```

В **MSDN** написано, что если клиент открывает слот прежде чем слот был создан сервером, то он получит **INVALID_HANDLE_VALUE**

Получение информации о почтовом ящике

```
BOOL GetMailslotInfo (
    HANDLE hMailslot, // указатель на слот
    LPDWORD lpMaxMessageSize, // максимальный размер
    LPDWORD lpNextSize, // размер следующего
    LPDWORD lpMessageCount, // количество сообщений
    LPDWORD lpReadTimeout // тайм-аут
);
```

Изменение свойств ящика

BOOL SetMailslotInfo(HANDLE *hMailslot*, DWORD *lpReadTimeout*);

Динамически компонуемые библиотеки (**D**ynamic **L**ink **L**ibrary)

Если два приложения используют одну библиотеку, то они разделяют все глобальные переменные этой библиотеки. В действительности, глобальные переменные, как и вся библиотека, отображаются на адресные пространства разных процессов. Этот метод не привносит никакой новой функциональности по сравнению с отображением проецируемых файлов и, поэтому, его использование не рекомендуется.

Раздел с общими данными в DLL

```
#pragma data_seg(".shared")
//Общие данные
#pragma data_seg()
```

UsersDll.def:

```
LIBRARY "UsersDll" SECTIONS .shared
   READ WRITE SHARED
```