Вопросы к экзамену

1. Операционные системы: определение, назначение, состав, функции.

Операционная система — это комплект программ, которые служат интерфейсом между модулями вычислительных систем и прикладными программными приложениями, а также управляют компьютерным оборудованием и процессами вычислений, эффективным распределением вычислительных мощностей среди процессов вычислений.

Операционная система — это комплекс программ, предназначенных для управления ресурсами компьютера и организации взаимодействия с пользователем.

Операционная система (ОС) — комплекс системных и управляющих программ, предназначенных для наиболее эффективного использования всех ресурсов вычислительной системы (ВС) (Вычислительная система — взаимосвязанная совокупность аппаратных средств вычислительной техники и программного обеспечения, предназначенная для обработки информации) и удобства работы с ней.

Две основные функции (назначение) ОС:

1) предоставлять пользователю некую расширенную виртуальную машину, с которой легче работать (легче программировать), чем непосредственно с аппаратурой реального компьютера или реальной сети;

2) управлять ресурсами вычислительной системы. Поэтому в специальной литературе ОС представляется всегда двояко: как расширенная виртуальная машина и как система управления ресурсами.

Функции ОС автономного компьютера обычно группируются в соответствии с типами локальных ресурсов, которыми управляет ОС. Такие группы называют подсистемами. Наиболее важные из них:

- подсистема управления процессами,
- подсистема управления памятью,
- подсистема управления файлами,
- подсистема управления внешними устройствами,
- подсистема пользовательского интерфейса,
- подсистема защиты данных и администрирования

Функции ОС:

- прием от пользователя (оператора) заданий или команд, сформулированных на соответствующих языках, и их обработка;
- загрузка в ОП программ и их исполнение;
- инициация программы (передача ей управления);
- прием и исполнение программных запросов на запуск, приостановку, остановкудругих программ; организация взаимодействия между задачами;
- идентификация всех программ и данных;
- обеспечение работы системы управления файлами и/или систем управления БД;
- обеспечение режима мультипрограммирования (многозадачности);
- планирование и диспетчеризация задач;
- обеспечение функций по организации и управлению операциями ввода/вывода;
- удовлетворение жестким ограничениям на время ответа в режиме реальноговремени (для соответствующих ОС);
- управление памятью, организация виртуальной памяти;
- организация механизмов обмена сообщениями и данными

междувыполняющимися программами;

- защита одной программы от влияния другой; обеспечение сохранности данных;
- аутентификация, авторизация и другие средства обеспечения безопасности;
- предоставление услуг на случай частичного сбоя системы;
- обеспечение работы систем программирования;
- параллельное исполнение нескольких задач.

2. Операционные системы: классификация, основные этапы развития, особенности современного этапа развития

Существует несколько классификаций ОС.

В зависимости от **способа организации вычислений**:

- Системы пакетной обработки – основной задачей является организация наибольшего количества вычислительных процессов за единицу времени. Определенные процессы объединяются в пакет, который затем обрабатывает ОС.
- Системы разделения времени – создание возможности единовременного взаимодействия с устройством сразу несколькими людьми. В порядке очереди каждый пользователь получает определенный промежуток процессорного времени.
- Системы реального времени – организация работы каждой задачи за определенный промежуток времени, присущий каждой конкретной задаче.

В зависимости от **типа ядра**:

- ОС с монолитным ядром (ядро ОС выполняет все функции, включая управление памятью, устройствами ввода-вывода и файловой системой, в одном монолитном модуле);
- ОС с микроядром (ядро ОС содержит только основные функции, такие как планирование задач, управление памятью и межпроцессное взаимодействие, а остальные функции реализуются в виде отдельных модулей, работающих в привилегированном или пользовательском режиме);
- ОС с гибридным ядром (комбинация монолитного и микроядерного подходов, где некоторые компоненты, такие как драйверы устройств, могут работать в пространстве ядра, в то время как другие функции, такие как файловая система и сетевые протоколы, могут работать в виде отдельных модулей, загружаемых по требованию).

В зависимости от **количества единовременно решаемых задач**:

- однозадачные;
- многозадачные;

В зависимости от **количества пользователей**:

- однопользовательские;
- многопользовательские.

В зависимости от **количества поддерживаемых процессоров**:

- однопроцессорные
- многопроцессорные

В зависимости от **возможности работы в компьютерной сети**:

- локальные – автономные ОС, которые не позволяют работать с компьютерными сетями;
- сетевые – ОС с поддержкой компьютерных сетей.

В зависимости от **роли в сетевом взаимодействии**:

- серверные – ОС, открывающие доступ к ресурсам сети и осуществляющие управление сетевой инфраструктурой;
- клиентские – ОС, которые имеют возможность получения доступа к ресурсам сети.

В зависимости от **типа лицензии**:

- открытые – ОС с открытым исходным кодом, который можно изучать иредактировать;
- проприетарные – ОС, связанные с определенным правообладателем и, как правило,

имеющие закрытый исходный код.

В зависимости от **сферы использования**:

- ОС мэйнфреймов – больших компьютеров;
- ОС серверов;
- ОС персональных компьютеров;
- ОС мобильных устройств;
- встроенные ОС;
- ОС маршрутизаторов.

Основные этапы развития:

Первое поколение (1945-1955 гг.)

Первое поколение компьютеров строилось преимущественно на электронных лампах. Существенная часть времени уходила на подготовку запуска программы, а сами программы выполнялись строго последовательно (такой режим работы называется **последовательной обработкой** данных).

Второе поколение (1955-1965 гг.)

Второе поколение компьютеров характеризуется использованием транзисторов, что повысило их надёжность и продлило время непрерывной работы.

Третье поколение (1965-1980 гг.)

Компьютеры **третьего поколения** использовали малые интегральные схемы, что дало им преимущество в цене и качестве по сравнению с машинами второго поколения. Самым важным достижением явилась **многозадачность**. В компьютерах предыдущего поколения вычисления останавливались на время ввода-вывода. Проблема была решена разбиением памяти на несколько частей, называемых разделами, в каждом из которых выполнялось отдельное задание. При окончании выполнения каждого текущего задания операционная система могла загружать новое задание с диска в освободившийся раздел памяти и запускать это задание. Этот технический прием называется **подкачкой данных**, или **спулингом** (spooling — английское слово, которое произошло от Simultaneous Peripheral Operation On Line, то есть совместная периферийная операция в интерактивном режиме), и его также используют для выдачи полученных данных.

Четвертое поколение (1980-2005 гг.)

Разработка БИС (большие интегральные схемы, LSI, Large Scale Integration — кремниевые микросхемы, содержащие тысячи транзисторов на одном квадратном сантиметре) привела к появлению микрокомпьютеров. Операционные системы для микрокомпьютеров принято относить к четвертому поколению.

Пятое поколение (2005 г. - по н.в.)

Широкую популярность приобрела технология **виртуализации** — представление вычислительных ресурсов, абстрагированное от аппаратной реализации.

Особенности современного этапа и перспективы развития ОС

На современном этапе развития операционных систем на передний план вышли средства обеспечения безопасности. Современным операционным системам присуща многоплатформенность, т.е. способность работать на совершенно различных типах компьютеров.

3. Компоненты архитектуры вычислительных систем, их назначение и взаимодействие.

АЛУ – выполняет арифметические и логические операции; УУ – организует процесс выполнения программ

Процессор: Среди специальных регистров процессора стоит выделить **счетчик команд**, который содержит адрес ячейки памяти со следующей выбираемой командой, указатель стека, который ссылается на вершину текущего стека в памяти и **слово состояния программы** — PSW (Program Status Word). Каждый раз при переключении задач содержимое регистров предыдущей задачи выгружается в оперативную память и загружается содержимое регистров для следующей задачи. Этот процесс называется переключением контекста. Многие современные процессоры способны одновременно выполнять более одной команды. Подобная организация работы называется **конвейером**. Процессоры с **суперскалярной** архитектурой имеют несколько исполнительных блоков, которые могут выполнять команды не в порядке их следования, что хоть и может повысить быстродействие, но усложняет операционную систему.

Идея RISC заключается в замене сложных инструкций на комбинацию простых. Конвейер инструкций Arm-Risc:

- **(Fetch) Извлечение** инструкции из памяти и увеличение счетчика команд, чтобы извлечь следующую инструкцию в следующем такте.
- **(Decode) Декодирование** инструкции — определение, что эта инструкция делает. То есть активация необходимых для выполнения этой инструкции частей микропроцессора.
- **(Execute) Выполнение** включает использование арифметико-логического устройства (АЛУ) или совершение сдвиговых операций.
- **(Memory) Доступ к памяти**, если необходимо. Это то, что делает инструкция load.
- **(Write Back) Запись результатов** в соответствующий регистр.

Второй основной составляющей любого компьютера является память. В идеале память должна быть максимально быстрой (работать быстрее, чем производится выполнение одной инструкции, чтобы работа центрального процессора не замедлялась обращениями к памяти), довольно большой и чрезвычайно дешевой.

Магнитный жесткий диск состоит из одной или нескольких металлических пластин, вращающихся со скоростью 5400, 7200, 10 000 или 15 000 оборотов в минуту.

Еще можно найти компьютеры, использующие шину PCI (Peripheral Component Interconnect), разработанную компанией Intel. Таким образом, контроллер соединяется с памятью непосредственно, то есть передача данных между центральным процессором и памятью происходит не через шину PCI.

Шина USB

Видеокарта – устройство, преобразующее цифровую информацию в форму, пригодную для дальнейшего вывода на экран монитора.

4. Задачи операционной системы по управлению и организации работы компьютера.

Выделим три важные группы функций.

1. **Виртуальная память**. Механизм виртуальной памяти используется многими операционными системами. Она позволяет создать впечатление, будто у машины больше памяти, чем есть на самом деле.

2. **Файловый ввод-вывод**.

3. **Параллелизм** (как организовано одновременное выполнение нескольких процессов, обмен информацией и синхронизация). Под процессом можно понимать работающую программу и всю информацию об ее состоянии (памяти, регистрах, счетчике команд, вводе-выводе и т. д.).

Самыми важными задачами управления компьютерным оборудованием, осуществляемыми операционной системой, являются:

- Параллельное функционирование модулей ввода, вывода информации и процессора.
- Организация кэширования данных и выполнение согласования скоростей информационного обмена.
- Разбиение модулей и информационных данных среди процессов.
- Организация удобной работы логического интерфейса между модулями и оставшейся частью системы.
- Организация поддержки различных устройств с обеспечением возможности просто их добавить.
- Режим динамической загрузки и выгрузки драйверов.
- Обеспечение поддержки набора файловых систем.

Модули, предназначенные для ввода и вывода информации, подразделяются на следующие типы:

- Модули, ориентированные на работу с блоками.
- Модули, ориентированные на работу с байтами.

Также следует упомянуть технологию, которая позволяет создавать несколько сред или выделенных ресурсов из единой физической аппаратной системы – называется **виртуализация**. Программное обеспечение, гипервизор, напрямую подключается к этой аппаратной системе и позволяет разбить ее на отдельные, безопасные среды – виртуальные машины. По идее, гипервизор должен распределять аппаратные ресурсы между виртуальными машинами так, чтобы процессы выполнялись быстрее. Физическая машина с гипервизором называется **хостом**, а виртуальные машины, которые используют ресурсы данного хоста – **гостями**. Для них ресурсами являются процессор, память, хранилище. Для получения доступа к этим ресурсам операторы управляют виртуальными экземплярами.

5. Структура ядра и его функции. Объекты ядра. Основные операции над объектами ядра.

Под архитектурой ОС обычно понимают структурную организацию ОС на основе программных модулей. Современные ОС представляют собой хорошо структурированные модульные системы.

Единой архитектуры ОС не существует, но существуют универсальные подходы к структурированию ОС:

Наиболее общим подходом к структуризации ОС является подразделение модулей на две группы:

- модули, выполняющие основные функции ОС — **ядро ОС**;
- модули, выполняющие вспомогательные функции ОС.

Модули ядра выполняют базовые функции ОС

- управление процессами;
- управление памятью;
- управление устройствами ввода-вывода.

Функции, входящие в состав ядра, можно разделить на два класса.

1 класс. Функции для решения внутрисистемных задач организации вычислительного процесса

(переключение контекстов процессов, загрузка/выгрузка страниц, обработка прерываний). Эти функции недоступны для приложений.

2 класс. Функции для поддержки приложений (доступны приложениям). Эти функции создают для приложений так называемую прикладную программную среду и образуют интерфейс прикладного программирования — API. Приложения обращаются к ядру с запросами — системными вызовами. Функции API обслуживают системные вызовы — предоставляют доступ к ресурсам системы в удобной и компактной форме, без указания деталей их физического расположения.

Объекты ядра используются системой и приложениями для управления множеством разных ресурсов: процессами, потоками, файлами и т.д. **Объект** — это коллекция данных, являющихся частью режима ядра операционной системы, которыми управляет Диспетчер объектов Windows (Windows Object Manager).

Типичные объекты режима ядра включают следующие категории объектов:

- объекты устройств;
- объекты файлов;
- символические ссылки;
- разделы реестра;
- потоки и процессы;
- объекты диспетчера ядра, такие как объекты событий и объекты мьютексов;
- объекты обратного вызова;
- объекты section.

Например, когда мы создаем объект ядра в операционной системе Windows, функция возвращает **описатель**, идентифицирующий созданный объект (HANDLE).

Создание новых объектов, или открытие по имени уже существующих, приложение может осуществить при помощи Win32-функций, таких, как **CreateFile**, **CreateSemaphore**, **OpenSemaphore** и т.д. Полученный описатель затем нужно использовать как первый параметр в различных функциях, например, **ReadFile** и **WriteFile**, для работы с реальным файлом. Во избежание утечки памяти всегда рекомендуется закрывать объект, когда в нем отпала надобность. Впрочем, по окончании работы процесса система закрывает все его объекты. Одной из таких функций является функция **CloseHandle**. Некоторые дескрипторы требуют своей функции закрытия, например, функция **FindClose** должна применяться для закрытия дескриптора, полученного от функции **FindFirstFile**.

6. Утилиты. Системные обрабатывающие программы. Библиотеки процедур. Программы дополнительных услуг.

Утилиты — это системное программное обеспечение, которое помогает поддерживать правильное и бесперебойное функционирование компьютерной системы. Они помогают операционной системе управлять, организовывать, поддерживать и оптимизировать функционирование компьютерной системы. Утилиты предоставляют доступ к возможностям (параметрам, настройкам, установкам), недоступным без их применения, либо делают процесс изменения некоторых параметров проще (автоматизируют его).

Основные функции обрабатывающих программ:

1) **перенос информации**. Перенос может выполняться между различными устройствами или в пределах одного устройства. При этом под устройствами понимаются: ОП, устройства ВП, устройства ввода-вывода;

2) **преобразование информации**. То есть после считывания информации с устройства обрабатывающая программа преобразует эту информацию, а только затем записывает ее

на это же или на другое устройство.

В зависимости от того, какая из этих двух функций является основной, обрабатывающие системные программы делятся на **утилиты** и **лингвистические процессоры**. Основной функцией утилиты является перенос информации, а основная функция лингвистического процессора – перевод описания алгоритма с одного языка на другой. Лингвистические процессоры делятся на **трансляторы** и **интерпретаторы**.

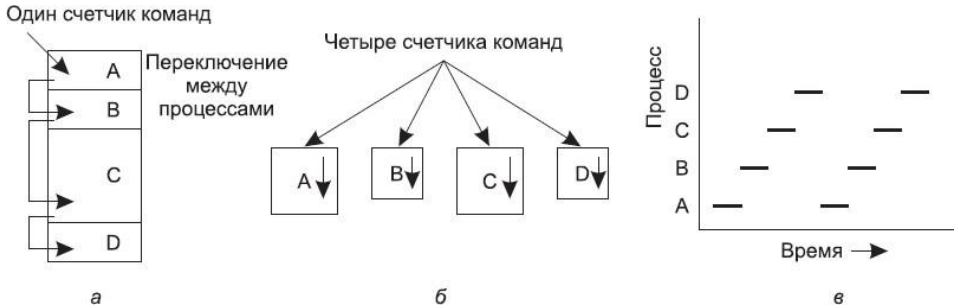
В результате работы **транслятора** алгоритм, записанный на языке программирования (исходная виртуальная программа), преобразуется в алгоритм, записанный на машинном языке. **Трансляторы** делятся на компиляторы и ассемблеры. **Интерпретатор** в отличие от транслятора не выдаёт машинную программу целиком. Выполнив перевод очередного оператора исходной программы в соответствующую совокупность машинных команд, интерпретатор обеспечивает их выполнение. Затем преобразуется тот исходный оператор, который должен выполняться следующим по логике алгоритма, и так далее.

Библиотеки процедур и функций различного назначения включены в категорию вспомогательных модулей операционной системы. **Библиотеки** — это набор функций, которые могут использоваться в различных программах. Библиотеки могут быть статические (библиотека привязывается к определенной программе или софт содержит данную библиотеку в своем теле.) и динамическими (библиотеки грузятся в оперативную память и используются).

Программы доп услуг: В эту категорию входит большое количество разнообразных программ: специальный вариант пользовательского интерфейса, калькулятор, некоторые игры (какие, например, поставляются в составе ОС).

7. Понятие процесса. Системные и пользовательские процессы. Операции над процессами.

Процесс — это просто экземпляр выполняемой программы, включая текущие значения счетчика команд, регистров и переменных. Концептуально укаждого процесса есть свой, виртуальный, центральный процессор. Постоянное переключение между процессами называется **мультипрограммированием**, или **многозадачным** режимом работы.



Компьютер: а — четыре программы, работающие в многозадачном режиме; б — концептуальная модель четырех независимых друг от друга последовательных процессов; в — в отдельно взятый момент активна только одна программа

Процессы, которые выполняют системный код, называются **системными** и применяются к системе в целом. Они занимаются выполнением таких служебных задач,

как распределение памяти, обмен страницами между внутренним и вспомогательным запоминающими устройствами, контроль устройств и т.п.

Пользовательские процессы выполняют собственный код и иногда обращаются к системным функциям. Выполняя собственный код, пользовательский процесс пребывает в пользовательском режиме (user mode).

Операции над процессами.

Запуск процесса. Из числа процессов, находящихся в состоянии готовность, операционная система выбирает один процесс для последующего исполнения.

Приостановка процесса. Работа процесса, находящегося в состоянии исполнение, приостанавливается в результате какого-либо прерывания.

Блокирование процесса. Процесс блокируется, когда он не может продолжать работу, не дождавшись возникновения какого-либо события в вычислительной системе.

Разблокирование процесса. После возникновения в системе какого-либо события операционной системе нужно точно определить, какое именно событие произошло

Переключение контекста

В действительности же деятельность мультипрограммной операционной системы состоит из цепочек операций, выполняемых над различными процессами, и сопровождается переключением процессора с одного процесса на другой.

8. Организация межпроцессного взаимодействия в ОС. Сигналы. Каналы. Классические проблемы межпроцессного взаимодействия.

При выполнении параллельных процессов может возникать проблема, когда каждый процесс, обращающийся к разделяемым данным, исключает для всех других процессов возможность одновременного с ним обращения к этим данным – это называется **взаимоисключением.** Ресурс, который допускает обслуживание только одного пользователя за один раз, называется критическим ресурсом. Для организации коммуникации между одновременно работающими процессами применяются средства межпроцессного взаимодействия (Interprocess Communication - IPC).

Средства локального уровня IPC привязаны к процессору и возможны только в пределах компьютера.

Удаленные IPC предоставляют механизмы, которые обеспечивают взаимодействие как в пределах одного процессора, так и между программами на различных процессорах, соединенных через сеть.

Под **высокоуровневыми IPC** обычно подразумеваются пакеты программного обеспечения, которые реализуют промежуточный слой между системной платформой и приложением.

Сигнал в операционных системах семейства Unix — асинхронное уведомление процесса о каком-либо событии, один из основных способов взаимодействия между процессами. Отдельные сигналы подразделяются на три класса:

- системные сигналы (ошибка аппаратуры, системная ошибка и т.д.);
- сигналы от устройств;
- сигналы, определенные пользователем.

Канал (pipe) представляет собой средство связи стандартного вывода одного процесса со стандартным вводом другого. После создания канала, процесс может при помощи обычного системного вызова **write**() выводить данные в него, а затем вводить их, вызывая соответственно функцию **read**(). При выполнении вызова **fork**() дескрипторы

канала наследуются процессом-"потомком". Таким образом, оба процесса получают возможность обмениваться данными.

Проблемы:

Синхронный доступ. Если все процессы считывают данные из файла, то в большинстве случае проблем не возникает. Однако, при попытке одним из процессов изменить этот файл, могут возникнуть так называемые конкурентные ситуации (race conditions).

Дисциплина доступа. Если нужно, чтобы как можно большее количество пользователей могли записывать данные, организуется так называемая очередь (по правилу «один пишет, несколько читают»). Практически организуется две очереди: одна — для чтения, другая — для записи. Такую дисциплину доступа можно организовать с помощью **барьеров** (блокировок). При этом создается общий барьер для считывателей, так как несколько процессов могут одновременно считывать данные, а также отдельный барьер для процесса-писателя. Такая организация предотвращает взаимные помехи в работе.

Голодание процессов. Организация дисциплины доступа может привести к ситуации, когда процесс будет длительно ждать своей очереди для записи данных. Поэтому иногда нужно организовывать очереди с приоритетами.

Если нельзя точно определить, какой из процессов запрашивает или возвращает свои данные в нужный компьютер первым, используется так называемое взаимодействие по модели "клиент-сервер". При этом используются один или несколько клиентов и один сервер. Клиент посылает запрос серверу, а сервер отвечает на него. После этого клиент должен дождаться ответа сервера, чтобы продолжать дальнейшее взаимодействие. Такое поведение называется **управлением потоком**. При одновременном доступе здесь также могут использоваться очереди с приоритетами.

Классический **тупик** возникает, если процесс А получает доступ к файлу А и ждет освобождения файла В. Одновременно процесс В, получив доступ к файлу В, ждет освобождения файла А. Оба процесса теперь ждут освобождения ресурсов другого процесса и не освобождают при этом собственный файл.

9. Концепция потока. Параллельное исполнение потоков. Главный поток процесса.

Поток выполнения (англ. thread — нить) — наименьшая единица обработки, исполнение которой может быть назначено ядром операционной системы. Несколько потоков выполнения могут существовать в рамках одного и того же процесса и совместно использовать ресурсы, такие как память, тогда как процессы не разделяют этих ресурсов. В частности, потоки выполнения разделяют инструкции процесса (его код) и его контекст (значения переменных, которые они имеют в любой момент времени).

Термин "параллельность" рассматривается в контексте вытесняющей многозадачности операционной системы - то есть, операционная система выделяет потоку некоторый квант времени, а затем переключается на другой поток.

Многопоточность: процесс, порождённый в операционной системе, может состоять из нескольких потоков, выполняющихся «параллельно», то есть **без предписанного порядка во времени**.

По умолчанию процесс создается с одним потоком, называемым **главным** или основным потоком. В Linux потоки реализованы с помощью системного вызова **clone**(), который как минимум в 10 раз меньше занимает времени для создания еще одного потока, чем создание еще одного процесса при помощи **fork**(). Такая скорость достигается за счет того, что многие атрибуты процесса разделяются между потоками.

В Windows процесс может породить практически неограниченное количество потоков. Для этого используется функция **CreateThread**. Процесс будет активен, пока активен хотябы один поток.

10. Диаграммы состояния потоков. Понятие контекста и переключения контекста.

Для каждого созданного потока в системе предусматриваются **три возможных его состояния**:

- состояние **выполнения**, когда код потока выполняется процессором; на однопроцессорных платформах в этом состоянии в каждый момент времени может находиться только один поток;
- состояние **готовности** к выполнению, когда поток готов продолжать свою работу и ждет освобождения ЦП;
- состояние **ожидания** наступления некоторого события; в этом случае поток не претендует на время ЦП, пока не наступит определенное событие (завершение операции ввода/вывода, освобождение необходимого потоку занятого ресурса, сигнала от другого потока), часто такие потоки называют блокированными. Изменение состояния потока происходит в результате соответствующих действий. Удобно для этих целей использовать следующую диаграмму состояний и переходов (рис. 2.2.5)

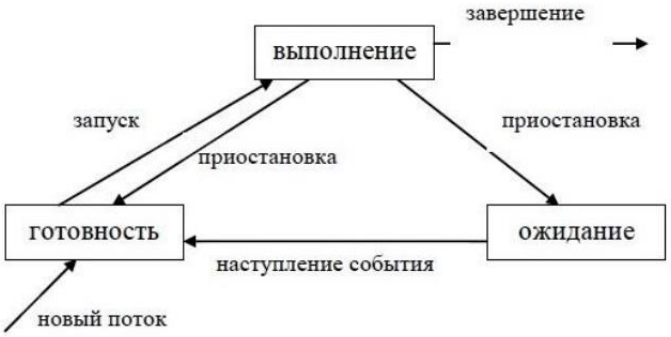


Рис. 2.2.5. Диаграмма состояний потока

Переключение контекста (context switch) — в многозадачных ОС и средах — процесс прекращения выполнения процессором одной задачи (процесса, потока, нити) с сохранением всей необходимой информации и состояния, необходимых для последующего продолжения с прерванного места, и восстановления и загрузки состояния задачи, к выполнению которой переходит процессор. В процедуру переключения контекста входит так называемое планирование задачи — процесс принятия решения, какой задаче передать управление. При переключении контекста происходит сохранение и восстановление следующей информации:

- Регистровый контекст регистров общего назначения (в том числе флаговый регистр)
- Контекст состояния сопроцессора с плавающей точкой / регистров MMX (x86)
- Состояние регистров SSE, AVX (x86)
- Состояние сегментных регистров (x86)
- Состояние некоторых управляющих регистров (например, регистр CR3, отвечающий за страничное отображение памяти процесса) (x86)

При переключении контекста в операционных системах происходит сохранение и восстановление следующей информации:

1. **Регистры процессора**:** Это включает в себя значения регистров общего назначения, указателя стека, указателя инструкций и других регистров, содержащих состояние выполнения процесса.

2. **Счетчик программы** (Program Counter, PC):** Этот регистр указывает на адрес следующей инструкции, которую нужно выполнить. При переключении контекста сохраняется текущее значение PC процесса, и затем восстанавливается значение для следующего процесса.

3. **Стек**:** Состояние стека, включая указатель стека (Stack Pointer, SP), сохраняется, чтобы можно было восстановить точку выполнения процесса.

4. **Состояние флагов**:** Флаги процессора, которые могут содержать информацию о состоянии выполнения инструкций, также сохраняются и восстанавливаются.

5. **Регистры сопроцессора** (если они используются):** В случае использования сопроцессора для выполнения вещественных операций, его состояние также сохраняется и восстанавливается.

6. **Информация о состоянии процесса**:** Возможно, дополнительные данные о состоянии процесса, такие как значения переменных и ресурсов, могут сохраняться и восстанавливаться в зависимости от конкретной реализации операционной системы.

1. **Регистры общего назначения** (General-Purpose Registers):** Это места, где процессор временно хранит числа и данные во время выполнения программы. Они используются для выполнения различных операций, таких как сложение, вычитание и хранение промежуточных результатов.

2. **Указатель стека** (Stack Pointer, SP):** Этот регистр указывает на текущий адрес в стеке, который используется для временного хранения данных в порядке Last In, First Out (LIFO). Стек часто используется для хранения временных значений и вызова функций.

3. **Указатель инструкций** (Instruction Pointer, IP):** Этот регистр содержит адрес следующей инструкции, которую процессор должен

выполнить. Когда процессор выполняет инструкцию, значение IP обновляется, указывая на следующую инструкцию.

4. **Регистры сопроцессора** (если они используются):** Эти регистры предназначены для выполнения специализированных операций, таких как вещественные вычисления. Например, если программа выполняет математические операции с десятичными числами, значения этих чисел могут храниться в регистрах сопроцессора.

11. Многозадачность в ОС. Типы многозадачности.

Многозадачность (multitasking) — свойство операционной системы или среды выполнения обеспечивать возможность параллельной (или псевдопараллельной) обработки нескольких задач. Истинная многозадачность операционной системы возможна только в распределённых вычислительных системах.

Существует **два типа многозадачности**:

- Процессная многозадачность (основанная на процессах — одновременно выполняющихся программах). Здесь программа — наименьший элемент управляемого кода, которым может управлять планировщик операционной системы. Более известна большинству пользователей (работа в текстовом редакторе и прослушивание музыки).
- Поточная многозадачность (основанная на потоках). Наименьший элемент управляемого кода — поток (одна программа может выполнять 2 и более задачи одновременно)

К **псевдопараллельной** многозадачности можно отвести следующие типы: простое переключение, совместная (или кооперативная) многозадачность и вытесняющая (или приоритетная) многозадачность.

1. **Простое переключение**:**

- Загружает в память два или более приложения, но предоставляет процессорное время только основному.

- Преимущества: Может использовать уже работающие программы без учета многозадачности. Недостатки: Ограниченное взаимодействие между программами.

2. **Совместная многозадачность (кооперативная):**

- Задача выполняется после объявления готовности отдать процессорное время другим задачам (задачи сами определяют, когда передавать управление другим задачам).

- Преимущества: Упрощает программирование, отсутствие необходимости в защите данных. Недостатки: Затруднена реализация многозадачной архитектуры ввода-вывода. Если задача не передает управление, она может заблокировать систему, оставив остальные задачи без возможности на выполнение.

3. **Вытесняющая многозадачность (приоритетная):**

- Операционная система передает управление от одной программы к другой по различным событиям (ОС сама определяет управление между задачами).

- Преимущества: Быстрый отклик на действия пользователя, полная реализация многозадачного ввода-вывода в ядре ОС. Недостатки: Требует дисциплины при написании кода, особые требования к реентерабельности и защите данных.

4. **Реентерабельность**

Реентерабельность программы означает, что её код может быть безопасно использован несколькими пользователями или процессами. Для обеспечения реентерабельности необходимо, чтобы вызываемый код не модифицировался, процедура не сохраняла информацию между вызовами, данные были уникальными для каждого пользователя, и не возвращались указатели на общие объекты. Реентерабельность важна для безопасности функций в многопоточных средах и программирования многозадачных систем, включая операционные системы.

Реентерабельность кода означает, что определенная функция или часть кода может быть повторно вызвана в то время как предыдущий вызов этой же функции ещё не завершился, и при этом ничего не "сломается" и не произойдёт конфликта данных.

Проще говоря, реентерабельный код можно представить как книгу в библиотеке. Если один человек читает книгу (вызывает функцию), другой человек может взять другой экземпляр той же книги и читать её одновременно без каких-либо проблем.

Для того чтобы функция была реентерабельной, она должна **соответствовать нескольким условиям**:

1. Не использовать глобальные или статические переменные для хранения состояния.
2. Не модифицировать свои аргументы.
3. Не вызывать другие нереентерабельные функции.

12. Иерархия, приоритеты и планирование потоков. Динамические уровни приоритетов.

1. **Иерархия и приоритеты потоков:**

- ****Иерархия**:** Потоки часто группируются по уровням приоритета в зависимости от их важности и роли. Например, системные потоки могут иметь более высокий приоритет по сравнению с прикладными.

- ****Приоритеты**:** Каждому потоку присваивается приоритет, который определяет его относительную важность и влияет на решение о том, какие потоки будут выполнены в первую очередь.

2. ****Планирование потоков**:**

«циклическое планирование»

Одним из самых старых, простых, справедливых и наиболее часто используемых считается алгоритм **циклического планирования**. Каждому

процессу назначается определенный интервал времени, называемый его **квантом**, в течение которого ему предоставляется возможность выполнения. Если процесс к завершению кванта времени все еще выполняется, то ресурс центрального процессора у него отбирается и передается другому процессу. Разумеется, если процесс переходит в заблокированное состояние или завершает свою работу до истечения кванта времени, то переключение центрального процессора на другой процесс происходит именно в этот момент.

Недостаток: Если частые переключения (квант - 4мс, а время переключения равно 1мс), то происходит уменьшение производительности.

«приоритетное планирование»

Основная идея проста: каждому процессу присваивается значение приоритетности и запускается тот процесс, который находится в состоянии готовности и имеет наивысший приоритет.

Даже если у персонального компьютера один владелец, на нем могут выполняться несколько процессов разной степени важности. Например, фоновому процессу, отправляющему электронную почту, должен быть назначен более низкий приоритет, чем процессу, воспроизводящему на экране видеофильм в реальном времени. Чтобы предотвратить бесконечное выполнение высокоприоритетных процессов, планировщик должен понижать уровень приоритета текущего выполняемого процесса с каждым сигналом таймера (то есть с каждым его прерыванием). Если это действие приведет к тому, что его приоритет упадет ниже приоритета следующего по этому показателю процесса, произойдет переключение процессов. Можно выбрать и другую альтернативу: каждому процессу может быть выделен максимальный квант допустимого времени выполнения. Когда квант времени будет исчерпан, шанс запуска будет предоставлен другому процессу, имеющему наивысший приоритет.

«кооперативное планирование»

Это такой алгоритм планирования, при котором процесс получает столько процессорного времени, сколько он считает нужным. Таким образом, все процессы делят процессорное время, периодически передавая управление следующей задаче.

- **Динамические уровни приоритетов**:** Некоторые системы поддерживают динамическое изменение приоритетов в зависимости от активности и состояния потоков. Например, при потреблении процессорного времени поток может повысить свой приоритет для более частого выполнения.

Цель планирования потоков вполне очевидна — определение порядка выполнения потоков в условиях внешней или внутренней многозадачности. Однако способы достижения этой цели существенно зависят от типа ОС. Рассмотрим сначала принципы планирования для универсальных ОС. Для таких ОС нельзя заранее предсказать, сколько и какие потоки будут запущены в каждый момент времени и в каких состояниях они будут

находиться. Поэтому планирование должно выполняться динамически на основе сбора и анализа информации о текущем состоянии вычислительной системы.

Динамические уровни приоритетов в Windows:**

В Windows каждый поток обладает динамическим приоритетом, который система может автоматически изменять для обеспечения отзывчивости и предотвращения голодания потоков. Динамический приоритет изначально совпадает с базовым приоритетом, и его система может повышать или понижать в зависимости от различных условий. Повышение приоритета происходит, например, при перемещении процесса на передний план, получении входных данных или выполнении условий ожидания. После повышения динамического приоритета система постепенно его уменьшает с течением времени. Этот механизм направлен на предотвращение инверсии приоритета и гарантирование эффективного использования процессорного времени.

13. Синхронизация и взаимоблокировка ресурсов. Механизмы синхронизации.

В некоторых операционных системах процессы, работающие совместно, могут сообща использовать некое общее хранилище данных. Каждый из процессов может считывать из общего хранилища данных и записывать туда информацию. Это хранилище представляет собой участок в основной памяти (возможно, в структуре данных ядра) или файл общего доступа. Местоположение совместно используемой памяти не влияет на суть взаимодействия и возникающие проблемы. Ситуации, в которых два (и более) процесса считывают или записывают данные одновременно и конечный результат зависит от того, какой из них был первым, называются состояниями состязания.

Критическая область — часть программы, в которой есть обращение к совместно используемым данным. Соответственно, **критический ресурс** — тот ресурс, к которому осуществляется одновременный доступ

Механизмы синхронизации:**

1. **Критические секции**:**

- Области кода, где доступ к общим данным ограничивается для одного потока в определенный момент времени, чтобы предотвратить конфликты.
- Предоставляют мьютекс для защиты от одновременного доступа нескольких потоков.

2. **Мьютексы**:**

- Обеспечивают механизм блокировки, позволяющий только одному потоку получить доступ к ресурсу или критической секции.
- Предотвращают конфликты и обеспечивают монопольный доступ к ресурсу.

3. **Семафоры**:**

- Имеют счетчик, контролирующий количество потоков, которым разрешен доступ к ресурсу.
- Подходят для сценариев с ограниченным числом доступных ресурсов.

4. **События**:**
- Используются для сигнализации между потоками.
- Могут быть использованы для синхронизации, где один поток ждет наступления события, а другой его сигнализирует.

5. **Барьеры**:**
- Используются для синхронизации группы потоков, ожидающих друг друга перед выполнением следующего этапа работы.

Пример барьера: Рассмотрим ситуацию, когда два процесса параллельно работают с общим массивом данных. Один процесс пытается читать данные, а другой пытается их изменить. Без использования барьеров может возникнуть ситуация, когда чтение и запись происходят в произвольном порядке, что может привести к непредсказуемым результатам.

С использованием барьеров можно организовать так, чтобы чтение завершилось до того, как начнется запись, и наоборот. Таким образом, барьеры гарантируют согласованность выполнения операций и предотвращают возможные конфликты в работе параллельных процессов.

6. **Атомарные операции**:**
- Гарантируют неделимость выполнения операции, предотвращая переключение контекста между потоками.
- Обеспечивают безопасное выполнение операций над общими данными без необходимости блокировки.

Простыми словами, атомарные операции в операционных системах - это такие действия, которые выполняются целиком и неделимо, без вмешательства других операций. Иными словами, они либо выполняются полностью, либо вообще не выполняются, и никакие другие операции не могут вмешаться посередине.

Пример атомарной операции: Рассмотрим операцию увеличения значения в памяти на 1. Если эта операция атомарная, то она либо полностью увеличит значение на 1, либо вообще не изменит его. Даже если несколько процессов пытаются одновременно увеличить значение, результат будет корректным, и не произойдет "перетирания" изменений друг друга.

14. Взаимоблокировка ресурсов в многозадачных системах. Решение задачи взаимоблокировки ресурсов.

Взаимоблокировка в группе процессов возникает в том случае, если каждый процесс из этой группы ожидает события, наступление которого зависит исключительно от другого процесса из этой же группы.

для возникновения ресурсных взаимоблокировок **должны выполняться четыре условия**:

1. **Условие взаимного исключения**. Каждый ресурс либо выделен в данный момент только одному процессу, либо доступен.

2. **Условие удержания и ожидания**. Процессы, удерживающие в данный момент ранее выделенные им ресурсы, могут запрашивать новые ресурсы.

3. **Условие невыгружаемости**. Ранее выделенные ресурсы не могут быть принудительно отобраны у процесса. Они должны быть явным образом высвобождены тем процессом, который их удерживает.

4. **Условие циклического ожидания**. Должна существовать кольцевая последовательность из двух и более процессов, каждый из которых ожидает высвобождения ресурса, удерживаемого следующим членом последовательности.

2.4.6. **Решение задачи взаимоблокировки ресурсов**

Чаще всего для борьбы с взаимными блокировками используются четыре стратегии:

1. **Игнорирование проблемы**. Может быть, если вы проигнорируете ее, она проигнорирует вас.

2. **Обнаружение и восстановление**. Дайте взаимоблокировкам проявить себя (не блокировать сразу же), обнаружьте их и выполните необходимые действия.

3. **Динамическое уклонение от них за счет тщательного распределения ресурсов**.

4. **Предотвращение за счет структурного подавления одного из четырех условий, необходимых для их возникновения (одного из 4 которые описаны выше).**

15. Компьютерное время. Ожидаемые таймеры.

Время Windows — это количество времени в миллисекундах, прошедшее с момента последнего запуска системы. Этот формат в первую очередь предназначен для обеспечения обратной совместимости с 16-разрядной версией Windows. Чтобы обеспечить успешное выполнение приложений, предназначенных для 16-разрядной версии Windows, функция GetTickCount возвращает текущее время Windows.

Хотя система внутренне использует время в формате UTC, в приложениях обычно отображается **местное время** (local), которое является датой и временем суток для вашего часового пояса. Поэтому, чтобы обеспечить правильные результаты, необходимо знать, ожидает ли функция получать время в формате UTC или местное время, а также возвращает ли функция время в формате UTC или местное время.

Объект таймера ожидания — это объект синхронизации, состояние которого по достижении указанного срока устанавливается в значение Signaled. Существует два типа таймеров ожидания, которые можно создать: сброс вручную и синхронизация. Таймер любого типа также может быть периодическим.

Таймер **сброса вручную** - Таймер, состояние которого остается сигнальным до вызова SetWaitableTimer, чтобы установить новое время выполнения.

Таймер **синхронизации** - Таймер, состояние которого остается сигнальным до тех пор, пока поток не завершит операцию ожидания в объекте таймера.

Периодический таймер - Таймер, который повторно активируется каждый раз, когда истечет указанный период, пока таймер не будет сброшен или отменен.

Периодический таймер — это либо периодический таймер сброса вручную, либо периодический таймер синхронизации. Поток использует функцию CreateWaitableTimer или CreateWaitableTimerEx для создания объекта таймера. Поток создания указывает, является ли таймер таймером сброса вручную или таймером синхронизации. Создающий поток может указать имя объекта таймера. Потоки в других процессах могут открывать дескриптор для существующего таймера, указывая его имя в вызове функции OpenWaitableTimer. Любой поток с дескриптором объекта таймера может использовать одну из функций ожидания для ожидания, пока состояние таймера будет задано как сигнальное. Поток вызывает функцию SetWaitableTimer для активации таймера.

****Компьютерное время в программировании:****

1. ****Системное Время:****

- **Определение времени:** Компьютеры поддерживают внутренние часы, отслеживающие текущее системное время.
- **RTC (Real-Time Clock):** Часы в реальном времени, хранящие время даже при выключенном компьютере.

2. **Работа с Временем в Программах**:**
- **API и Библиотеки:** Языки программирования предоставляют API и библиотеки для работы с временем (например, `time` в Python, `java.time` в Java).
- **Типы данных времени:** Обычно используются структуры данных, представляющие момент времени, интервалы и т. д.

3. **Форматы Времени**:**
- **UNIX Timestamp:** Количество секунд, прошедших с 1 января 1970 года (эпоха) в UNIX-подобных системах.
- **Дата и Время в Календарной Нотации:** Представление даты и времени в удобочитаемой форме (например, "2023-01-01 12:00:00").

4. **Операции с Временем**:**
- **Арифметика Времени:** Добавление или вычитание интервалов времени.
- **Сравнение Времени:** Операции сравнения для определения порядка событий.
- **Форматирование и Разбор Времени:** Преобразование времени в строковый формат и обратно.

5. **Часовые Пояса и Летнее/Зимнее Время**:**
- **TimeZone:** Учет различий в часовых поясах.
- **DST (Daylight Saving Time):** Учет переходов на летнее/зимнее время.

6. **Таймеры и Задержки**:**
- **Таймеры ОС:** Использование системных таймеров для планирования событий.
- **Задержки:** Ожидание выполнения определенного условия или прошествия времени.

7. **Системные Часы и Синхронизация**:**
- **NTP (Network Time Protocol):** Протокол синхронизации времени через сеть.
- **Системные События:** Обновление системного времени при включении, сетевых событиях и др.

16. Управление памятью: адресное пространство процесса, организация памяти, основные механизмы управления памятью, концепция рабочего множества.

Адресное пространство — это набор адресов, который может быть использован процессом для обращения к памяти.

Организация памяти и основные механизмы управления памятью:

1. **Иерархия памяти:**

- Компьютеры используют иерархию памяти с разными типами и характеристиками, включая кэш-память (для быстрого доступа к памяти), оперативную память (для среднего по скорости доступа к памяти) и дисковые накопители (для медленного доступа к памяти).

2. **Менеджер памяти:**

- Часть операционной системы, управляющая иерархией памяти, называется менеджером или диспетчером памяти.
- Задачи менеджера памяти включают выделение и освобождение памяти, отслеживание используемых областей и обеспечение эффективного использования ресурсов.

3. **Физические адреса:**

- Реальные адреса, используемые для выбора микросхем физической памяти.
- Организованы в виде последовательности 8-разрядных байтов.

4. **Модели доступа к памяти:**

- Сплошная ("плоская") модель памяти, сегментированная модель памяти, модель режима реального адреса.
- Различные модели использовались в различных поколениях компьютеров.

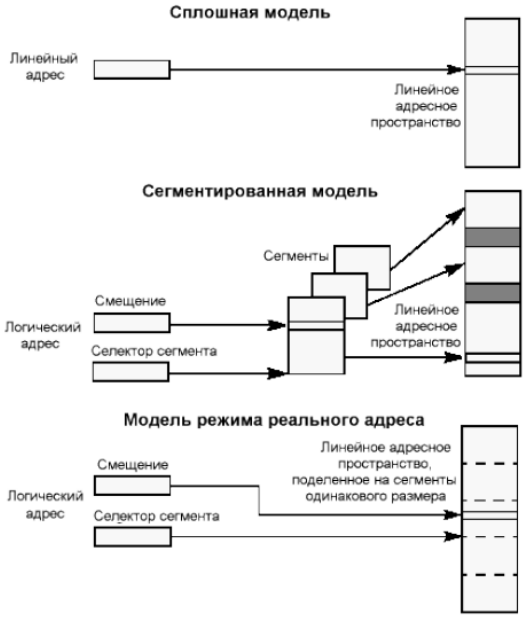


Рис. 3.1.1. Три модели доступа к памяти

5. **Организация физической памяти**:**

- Методы распределения памяти предоставляют программистам возможность эффективного использования компьютерной системы.

- Примеры включают ОЗУ (RAM), ПЗУ (ROM), и различные варианты размещения системных компонентов.

Рис. 3.1.2. Три простых способа организации памяти при наличии операционной системы и одного пользовательского процесса (существуют и другие варианты)

6. **Замена данных (свопинг):**

- Когда операционная система сохраняет содержимое памяти на диске перед загрузкой новой программы.

- Используется в простых встроенных устройствах и была распространена в ранних компьютерах.

7. **Адресное пространство:**

- Абстракция для памяти, позволяющая одновременное размещение в памяти нескольких приложений без взаимных помех.

- Понятие адресного пространства обеспечивает изоляцию программ.

Как операционные системы управляют памятью компьютера в случае, когда ей не хватает для всех выполняемых процессов. Есть два основных подхода: свопинг и виртуальная память.

1. **Свопинг:** Процессы полностью загружаются в память, выполняются некоторое время, а затем выгружаются на диск. Это позволяет освободить оперативную память для других процессов. Однако с течением времени может возникнуть фрагментация памяти.
2. **Виртуальная память:** Позволяет программам запускаться даже если они занимают только часть оперативной памяти. Операционная система следит за использованием памяти, используя битовые матрицы или списки свободного пространства.

Основная задача операционной системы в управлении памятью — создание абстракции для удобного моделирования и управления ресурсами памяти в компьютерной системе.

Концепция рабочего набора (набора) в контексте управления памятью:

Рабочий набор процесса:

- У каждой программы имеется собственное адресное пространство, которое разбивается на участки, называемые страницами

- Рабочий набор - Набор страниц в виртуальном адресном пространстве процесса, находящихся в физической памяти на текущий момент.

- Включает только страничные выделения памяти, исключая ресурсы, не поддерживающие подкачку.

- **Обработка ошибок страницы**:**

- Если процесс обращается к странице, не входящей в рабочий набор, происходит ошибка страницы.

- Обработчик ошибки системной страницы пытается устранить ошибку, читая содержимое страницы из резервного хранилища (если ошибка жесткой страницы).

- **Ошибки мягкой страницы**:**

- Возникают, когда страница уже находится в памяти другого процесса, в состоянии перехода или при первом обращении к новой виртуальной странице.

- **Удаление страниц из рабочего набора:**

- Процессы могут уменьшать или очищать рабочий набор, вызывая соответствующие функции.

- Диспетчер памяти также может обрезать страницы из рабочего набора для создания дополнительной доступной памяти.

- **Страницы перехода**:**

- Страницы, удаленные из рабочих наборов процессов, становятся страницами перехода.

- Сохраняются в оперативной памяти до повторного обращения или перепрофилирования.

- **Запись грязных страниц**:**

- Грязные переходные страницы, измененные после последней записи на диск, могут быть записаны в их резервное хранилище.

- **Размер рабочего набора процесса**:**

- Каждый процесс имеет минимальный и максимальный размер рабочего набора, влияющий на поведение подкачки виртуальной памяти.

- **Интерфейс для получения и изменения размера рабочего набора**:**

- Функции GetProcessMemoryInfo, GetProcessWorkingSetSizeEx и SetProcessWorkingSetSizeEx используются для получения и управления размером рабочего набора.

Концепция рабочего набора играет ключевую роль в управлении виртуальной памятью, обеспечивая эффективное использование физической памяти и обеспечивая доступность данных для процессов.

17. Классификация запоминающих устройств. Иерархия памяти. Оперативные и постоянные запоминающие устройства.

Запоминающее устройство — носитель информации, предназначенный для записи и хранения данных. В основе работы запоминающего устройства может лежать любой физический эффект, обеспечивающий приведение системы к двум или более устойчивым состояниям.

Классификация запоминающих устройств по устойчивости записи и возможности перезаписи:

- **Постоянные** (ПЗУ), содержание которых не может быть изменено конечным пользователем (например, BIOS). ПЗУ в рабочем режиме допускает только считывание информации.
- **Записываемые** (ППЗУ), в которые конечный пользователь может записать информацию только один раз (например, CD-R).
- **Многократно перезаписываемые** (ПППЗУ) (например, CD-RW).
- **Оперативные** (ОЗУ) — обеспечивают режим записи, хранения и считывания информации в процессе её обработки. Быстрые, но дорогие ОЗУ (SRAM) строят на триггерах, более медленные, но более дешёвые разновидности ОЗУ — динамические ЗУ (DRAM) строят на элементах, состоящих из ёмкости (конденсатора) и полевого транзистора, используемого в качестве ключа разрешения записи-чтения. В обоих видах ЗУ информация исчезает после отключения от источника питания (например, тока).

По **типу доступа** ЗУ делятся на:

- устройства с последовательным доступом (например, магнитные ленты).
- устройства с произвольным доступом (RAM) (например, оперативная память).
- устройства с прямым доступом (например, жесткие магнитные диски).
- устройства с ассоциативным доступом (специальные устройства, для повышения производительности БД)

Классификация запоминающих устройств **по геометрическому исполнению**:

- дисковые (магнитные диски, оптические, магнитооптические);
- ленточные (магнитные ленты, перфоленты);
- барабанные (магнитные барабаны);
- карточные (магнитные карты, перфокарты, флэш-карты, и др.)
- печатные платы (карты DRAM).

Классификация запоминающих устройств **по физическому принципу**:

- перфорационные (перфокарта; перфолента);
- с магнитной записью (ферритовые сердечники, магнитные диски, магнитные ленты, магнитные карты);
- оптические (CD, DVD, HD-DVD, Blu-ray Disc);
- использующие эффекты в полупроводниках (флэш-память) и другие

Иерархия памяти в простых словах:

В компьютере есть разные виды памяти, и они разделены на уровни. В самом верху находятся **очень быстрые регистры процессора**. Они могут быстро предоставить данные, но их не так много.

После регистров идет **кэш первого уровня** — это небольшая, но очень быстрая память. Если процессор не находит нужные данные здесь, он смотрит в **кэш второго уровня**, который чуть больше и немного медленнее. Так продолжается до **кэша третьего уровня**, который уже весьма велик по объему, но не так быстр.

Если данные не найдены в кэше, процессор обращается к **оперативной памяти**. Она медленнее, но вмещает больше информации. Можно представить оперативную память как кэш для локального диска, который, в свою очередь, может рассматриваться как кэш для данных с удаленных серверов.

В конечном итоге, существует иерархия с разными уровнями памяти, от быстрой и маленькой до медленной и большой. Каждый уровень старается хранить копии данных, чтобы процессор мог получить к ним доступ быстро, минимизируя ожидание.

Простыми словами и вкратце **Оперативные и постоянные запоминающие устройства**:

ПЗУ (Постоянное Запоминающее Устройство):**

ПЗУ - это место в компьютере, где хранятся важные программы, например, управление процессором, программы для запуска и выключения компьютера, а также тестовые программы для проверки работы различных блоков. Он также содержит информацию о том, где находится операционная система.

В ПЗУ находятся:

- программа управления работой процессора;
- программа запуска и останова компьютера;
- программы тестирования устройств, проверяющие при каждом включении компьютера правильность работы его блоков;
- программы управления дисплеем, клавиатурой, принтером, внешней памятью;
- информация о том, где на диске находится операционная система.

ОЗУ (Оперативное Запоминающее Устройство):**

ОЗУ - это место, где компьютер хранит данные и команды, которые ему нужны в данный момент для выполнения операций. ОЗУ передает данные процессору, обычно через кэш-память. Каждая ячейка в ОЗУ имеет свой уникальный адрес. Это временное хранилище, и его емкость может быть до 64 ГБ. ОЗУ устанавливается в виде модулей, таких как DIMM, и использует разные поколения DDR-памяти, которые отличаются по быстродействию и другим характеристикам.

Различные поколения DDR-модулей отличаются электрически и физически (разное количество контактов). Чтобы избежать их повреждения, положения ключа в них отличается (рис. 3.1.12).

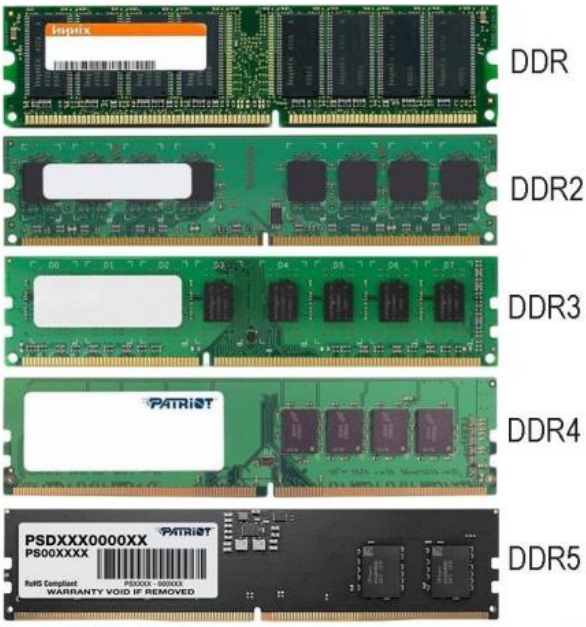


Рис. 3.1.12. Сравнение форм-фактора различных поколений DDR-памяти

С каждым поколением DDR-памяти увеличивается быстродействие и другие характеристики. Сравнительная таблица приведена на рис. 3.1.13.

18. Виртуальная память. Структуризация адресного пространства виртуальной памяти. Задачи управления виртуальной памятью

Виртуальная память

Виртуальная память - это механизм, который позволяет программам использовать больше памяти, чем физически доступно на компьютере. Он делает это, создавая абстракцию адресного пространства, которая отличается от физической памяти. Каждому процессу предоставляется собственное виртуальное адресное пространство, которое изолировано от других процессов.

Физические адреса являются конкретными и окончательными — без трансляции, без подкачки, без проверки привилегий. Вы выставляете их на шину и всё: выполняется чтение или запись. Однако в современной операционной системе программы используют абстракцию — виртуальное адресное пространство. Каждая программа пишется в такой модели, что она выполняется одна, всё пространство принадлежит ей, код использует адреса логической памяти, которые должны быть преобразованы в физические адреса до того, как будет выполнен доступ к памяти

Логический адрес на x86 состоит из двух частей: селектора сегмента и смещения внутри сегмента. Процесс трансляции включает два шага:

- учёт сегментного селектора и переход от смещения внутри сегмента к некоторому линейному адресу;
- перевод линейного адреса в физический.

Структуризация адресного пространства виртуальной памяти

(В этом пункте мб хуйню написал)

Страничная память — способ организации виртуальной памяти, при котором виртуальные адреса отображаются на физические постранично.

Виртуальная память делится на страницы. Размер размера страницы задается процессором и обычно на x86-64 составляет 4 KiB. Это означает, что управление памятью в ядре выполняется с точностью до страницы. Когда вам понадобится новая память, ядро предоставит вам одну или несколько страниц. При освобождении памяти вы вернёте одну или несколько страниц

Физическая память также поделена на страницы.

AMD решила, что в первых реализациях архитектуры фактически при трансляции адресов будут использоваться только младшие 48 бит виртуального адреса.

Кроме того, спецификация AMD требует, что старшие 16 бит любого виртуального адреса, биты с 48-го по 63-й, должны быть копиями бита 47. Если это требование не выполняется, процессор будет вызывать исключение. Адреса, соответствующие этому правилу, называются «канонической формой»

Ставится задача транслировать 48-битный виртуальный адрес в физический. Она решается аппаратным обеспечением — блоком управления памятью (memory management unit, MMU). Этот блок является частью процессора. Чтобы транслировать адреса, он использует структуры данных в оперативной памяти, называемые таблицами страниц.

Пусть для определённости размер страницы равен 4 КБ. Значит, младшие 12 битов адреса кодируют смещение внутри страницы и не изменяются, а старшие биты используются для определения адреса начала страницы

Задачи управления виртуальной памятью: задача размещения, задача перемещения, задача преобразования адресов, задача замещения

Виртуальная память 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.
- Операционные системы используют различные алгоритмы для определения, какие страницы данных должны быть выгружены из оперативной памяти, чтобы освободить место для других данных (LRU)

Все эти действия выполняются автоматически

19. Подкачка. Алгоритмы замещения страниц. Куча (heap). Стек.

Подкачка

Файл подкачки или виртуальная память — это способ системы виртуальной памяти увеличить оперативную память, когда ее не хватает для совершения операций. Система автоматически задействует файл подкачки, когда приложениям не хватит системной памяти ОЗУ

Подкачка в Linux:

В Linux подкачка реализуется через механизм swap-раздела (или файла).

Если система обнаруживает, что в оперативной памяти недостаточно места для хранения всех активных процессов, она начинает перемещать части неиспользуемых данных на swap-раздел.

Swap-раздел может быть создан на жестком диске, и он предоставляет дополнительное пространство для хранения страниц памяти, которые временно не нужны.

Когда данные необходимы, они могут быть восстановлены из swap-раздела обратно в оперативную память.

Подкачка в Windows:

В Windows подкачка реализуется с использованием файла подкачки (paging file).

Аналогично Linux, если оперативной памяти недостаточно, Windows перемещает неиспользуемые данные на файл подкачки.

Файл подкачки обычно создается при установке системы и представляет собой зарезервированное место на жестком диске.

Windows использует алгоритмы замещения страниц для определения, какие данные следует перемещать на файл подкачки и какие - обратно в оперативную память.

Алгоритмы замещения страниц

Алгоритмы замещения страниц — это алгоритмы, используемые операционной системой для принятия решения о том, какие страницы памяти следует выгрузить на долгосрочное хранилище (например, в файл подкачки) при нехватке оперативной памяти

Оптимальный алгоритм замещения страниц:

Это идеальный алгоритм, который всегда выбирает страницу для замещения, которая не будет использована в течение самого долгого времени. Однако он трудно реализуем из-за невозможности предвидения будущего использования страниц.

Использование битов состояния в алгоритмах исключения страниц:

Многие алгоритмы используют биты состояния (например, бит "использования" или "модификации") для принятия решения о том, какие страницы следует замещать. Например, бит "использования" может помочь определить, насколько часто страница была доступна.

Алгоритм исключения давно использовавшейся страницы:

Замещает страницу, которая давно не использовалась. Таким образом, предполагается, что давно не используемые страницы будут менее актуальными.

Алгоритм «первой пришла, первой и ушла» (FIFO):

Замещает страницу, которая находится в памяти дольше всего. Использует принцип очереди: первая загруженная страница первой выгружается.

Алгоритм «второй шанс»:
Также известен как алгоритм Часов. Имеет очередь страниц, и если страница получает "второй шанс" (то есть она снова используется), она не выгружается. В противном случае, она может быть выбрана для замещения.

Алгоритм «часы»:
Это улучшенная версия алгоритма «второй шанс», который использует круговой список страниц, аналогичный циферблату часов. Страницы, получившие "второй шанс", не выгружаются, а их бит "второго шанса" сбрасывается.

Алгоритм замещения наименее востребованной страницы (LRU):
Замещает страницу, которая была использована реже всего. Этот метод оценивает частоту использования каждой страницы и выбирает ту, которая реже всего использовалась.

Алгоритм нечастого востребования (NFU - Not Frequently Used):
Замещает страницу, которая редко запрашивается. Подсчитывает частоту использования каждой страницы и выбирает ту, которая использовалась наименее часто.

Алгоритм старения (Aging):
Этот метод представляет собой модификацию алгоритма «второй шанс», где каждая страница имеет свой счетчик старения. При каждом тике счетчик уменьшается, и страницы с более низкими значениями заменяются.

Алгоритм «рабочий набор»:
Основан на идее, что программы обычно используют ограниченное количество страниц в определенный момент времени. Система следит за "рабочим набором" страниц для каждого процесса и пытается поддерживать их в оперативной памяти.

Алгоритм WSClock (Working Set Clock):
Это улучшение алгоритма «второй шанс», которое также учитывает временные интервалы, в течение которых страница не использовалась, и её "возраст". Использует понятие рабочего множества, подобного алгоритму «рабочий набор».

Куча

Куча — это сегмент памяти, для которого не устанавливается постоянный размер перед компиляцией и который может динамически управляться программистом. Думайте о куче как о «свободном пуле» памяти, который вы можете использовать при запуске приложения. Размер кучи приложения определяется физическими ограничениями вашей оперативной памяти (оперативной памяти) и обычно намного больше размера стека

Каждый процесс имеет кучу по умолчанию, предоставляемую системой
Частная куча — это блок одной или нескольких страниц в адресном пространстве вызывающего процесса. После создания частной кучи процесс использует такие функции, как HeapAlloc и HeapFree, для управления памятью в этой куче

HeapCreate - Создание частного объекта кучи
HeapAlloc - Выделение указанного количества байтов из частной кучи
HeapFree - Освобождение ранее выделенного блока памяти
HeapDestroy - Уничтожение частного объекта кучи
HeapSize, HeapReAlloc, HeapValidate - Предоставление информации о размере памяти, перераспределение блока и проверка допустимости памяти

Стек

Стек — это сегмент памяти, в котором данные, такие как локальные переменные и вызовы функций, добавляются и/или удаляются по принципу «последним пришел — первым вышел» (LIFO).

Стек также известен как очередь LIFO (последним пришел — первым ушел), поскольку значения извлекаются из стека в порядке, обратном тому, в котором они помещаются в него

На текущую вершину стека указывает регистр esp. Стек «растет» вниз, от верхних адресов памяти к младшим, поэтому значения, недавно помещенные в стек, располагаются по адресам памяти над указателем esp

20. Типы устройств ввода/вывода. Обработка внешних прерываний. Синхронный и асинхронный ввод/вывод.

Типы устройств ввода/вывода

Устройства ввода-вывода можно условно разделить на две категории: блочные устройства и символьные устройства. К блочным относятся такие устройства, которые хранят информацию в блоках фиксированной длины, у каждого из которых есть собственный адрес. Вся передача данных ведется пакетами из одного или нескольких целых (последовательных) блоков. Важным свойством блочного устройства является то, что оно способно читать или записывать каждый блок независимо от всех других блоков. Среди наиболее распространенных блочных устройств жесткие диски, приводы оптических дисков и флеш-накопители USB.

Другой тип устройств ввода-вывода — символьные устройства. Они выдают или воспринимают поток символов, не относящийся ни к какой блочной структуре. Они не являются адресуемыми и не имеют никакой операции позиционирования. В качестве символьных устройств могут рассматриваться терминалы, принтеры, сетевые интерфейсы, мыши (в качестве устройства-указателя) и множество других устройств, не похожих на дисковые устройства.

Обработка внешних прерываний

На аппаратном уровне прерывания работают следующим образом [2]. Когда устройство ввода-вывода заканчивает свою работу, оно инициирует прерывание (при условии, что прерывания разрешены операционной системой). Для этого устройство выставляет сигнал на выделенную устройству специальную линию шины. Этот сигнал

распознается микросхемой контроллера прерываний, расположенной на материнской плате. Контроллер прерываний принимает решение о дальнейших действиях

При отсутствии других необработанных запросов прерывания контроллер прерываний обрабатывает прерывание немедленно. Если прерывание уже обрабатывается, и в это время приходит запрос от другого устройства по линии с более низким приоритетом, то новый запрос просто игнорируется.

Для обработки прерывания контроллер выставляет на адресную шину номер устройства, требующего к себе внимания, и устанавливает сигнал прерывания на соответствующий контакт процессора.

Этот сигнал заставляет процессор приостановить текущую работу и начать выполнять обработку прерывания. Номер, выставленный на адресную шину, используется в качестве индекса в таблице, называемой вектором прерываний

Вскоре после начала своей работы процедура обработки прерываний подтверждает получение прерывания, записывая определенное значение в порт контроллера прерываний. Это подтверждение разрешает контроллеру издавать новые прерывания. Благодаря тому, что центральный процессор откладывает выдачу подтверждения до момента, когда он уже готов к обработке нового прерывания, удается избежать ситуации состязаний при появлении почти одновременных прерываний от нескольких устройств

Аппаратура всегда, прежде чем начать процедуру обработки прерывания, сохраняет определенную информацию. Сохраняемая информация и место ее хранения широко варьируются в зависимости от центрального процессора. Как минимум сохраняется счетчик команд, что позволяет продолжить выполнение прерванного процесса. Другая крайность представляет собой сохранение всех программно доступных регистров и большого количества внутренних регистров центрального процессора.

Синхронный и асинхронный ввод/вывод

Асинхронный ввод-вывод (I/O) используется для оптимизации производительности приложений. В отличие от синхронного ввода-вывода, где приложение блокируется во время операции, асинхронный подход позволяет приложению продолжать выполнение, пока операция I/O выполняется.

По умолчанию все файловые дескрипторы в Unix-системах создаются в «блокирующем» режиме.

Существует два различных, но взаимодополняющих способа устранить блокировки:

• неблокирующий режим ввода-вывода

• мультиплексирование с помощью системного API, такого как select либо epoll

Файловый дескриптор помещают в “неблокирующий” режим, добавляя флаг O_NONBLOCK к существующему набору флагов дескриптора с помощью fcntl:

```
/* Добавляем флаг O_NONBLOCK к дескриптору fd */
const int flags = fcntl(fd, F_GETFL, 0);
fcntl(fd, F_SETFL, flags | O_NONBLOCK);
```

С момента установки флага дескриптор становится неблокирующим.

Такой подход работает, но имеет свои минусы:

• Если данные приходят очень медленно, программа будет постоянно просыпаться и тратить впустую процессорное время

• Когда данные приходят, программа, возможно, не прочитает их сразу, т.к. выполнение приостановлено из-за nanosleep

• Увеличение интервала сна уменьшит бесполезные траты процессорного времени, но увеличит задержку обработки данных

• Увеличение числа файловых дескрипторов с таким же подходом к их обработке увеличит долю расходов на проверки наличия данных

Для решения этих проблем операционная система предоставляет мультиплексирование ввода-вывода.

Существует несколько мультиплексирующих системных вызовов:

• Вызов select существует во всех POSIX-совместимых системах, включая Linux и MacOSX

• Группа вызовов epoll_* существует только на Linux

• Группа вызовов kqueue существует на FreeBSD и других *BSD

Все три варианта реализуют единый принцип: делегировать ядру задачу по отслеживанию прихода данных для операций чтения/записи над множеством файловых дескрипторов. Все варианты могут заблокировать выполнение, пока не произойдёт какого-либо события с одним из дескрипторов из указанного множества

Все мультиплексирующие системные вызовы, как правило, работают независимо от режима файлового дескриптора (блокирующего или неблокирующего). Программист может даже все файловые дескрипторы оставить блокирующими, и после select либо epoll возвращённые ими дескрипторы не будут блокировать выполнение при вызове read или write, потому что данные в них уже готовы.

Группа вызовов epoll является наиболее развитым мультиплексером в ядре Linux и способна работать в двух режимах:

• level-triggered - похожий на select упрощённый режим, в котором файловый дескриптор возвращается, если остались непрочитанные данные

• edge-triggered - файловый дескриптор с событием возвращается только если с момента последнего возврата epoll произошли новые события (например, пришли новые данные)

21. Файловые системы. Файлы и директории. Управление внешней памятью.

Файловые системы

Файловая система (file system) — порядок, определяющий способ организации, хранения и именования данных на носителях информации в компьютерах, а также в другом электронном оборудовании: цифровых фотоаппаратах, мобильных телефонах и т. п. Файловая система определяет формат содержимого и способ физического хранения информации, которую принято группировать в виде файлов. Конкретная файловая система определяет размер имен файлов (и каталогов), максимальный возможный размер файла и раздела, набор атрибутов файла. Некоторые файловые системы предоставляют сервисные возможности, например, разграничение доступа или шифрование файлов

Основные понятия:

Суперблок:

Критическая информация о компоновке файловой системы.

Хранится в первом блоке каждой группы блоков.

Определяет размер файловой системы, максимальное число файлов, объем свободного пространства и другие параметры.

i-узел (inode):

Описывает один файл, каталог или устройство.

Идентифицируется уникальным порядковым номером.

Содержит информацию о типе файла, правах доступа, размере, времени доступа и других атрибутах.

Элемент каталога (dentry):

Представляет собой элемент каталога и поддерживает перемещение по путям.

Кэшируется в dentry_cache для улучшения производительности.

Файл (file):

Представляет собой открытый файл в памяти.

Создается при системном вызове open и поддерживает операции чтения, записи, блокировки и другие.

Файловые системы в Linux:

Ext4 и XFS: Наиболее широко используемые файловые системы в Linux.

VFS (Virtual File System): Абстракция файловой системы в ядре Linux, определяет основные абстракции и разрешенные операции.

Выбор файловой системы: При выборе учитываются факторы, такие как масштабируемость, стабильность и целостность данных.

Операции файловой системы:

read_inode, sync_fs: Операции для работы с суперблоком.

create, link: Операции для работы с элементами каталога.

read, write: Операции для работы с файлами в памяти.

Принцип работы:

Суперблок и i-узлы: Хранят важную информацию о файловой системе, размещаются на диске и в памяти.

Dentry и кэш: Обеспечивают кэширование элементов каталога для улучшения производительности.

File: Предоставляет абстракцию открытого файла и поддерживает операции над ним.

EXT2 (Second Extended File System):

Особенности:

Использовалась вплоть до середины 2000-х.

Поддерживала файлы размером до 2 ТБ.

Отсутствие журналирования — восстановление после сбоев могло быть долгим.

Ограничения:

Не имела механизмов для быстрого восстановления после сбоев.

EXT3 (Third Extended File System):

Особенности:

Развитие EXT2 с добавлением журналирования.

Журналирование улучшало надежность и скорость восстановления.

Поддерживала файлы размером до 2 ТБ.

Журналирование:

Запись изменений в журнал перед их фактической записью на диск.

Уменьшение времени восстановления после сбоев.

EXT4 (Fourth Extended File System):

Особенности:
Продолжает развитие EXT3 с дополнительными улучшениями.
Поддерживает файлы размером до 16 ТБ.
Более эффективное управление пространством и быстрым восстановлением.
FAT (File Allocation Table):

Особенности:
Применяется в системах Windows.
Простая структура с таблицей выделения пространства.
Поддерживает файлы до 4 ГБ.
Ограничения:
Ограниченная поддержка прав доступа и журналирования.
NTFS (New Technology File System):

Особенности:
Применяется в системах Windows, заменяя FAT.
Поддержка развитых функций, таких как журналирование, шифрование, и сжатие файлов.
Поддерживает большие файлы и тома.
Безопасность:
Поддержка механизмов безопасности, контроля доступа и аудита.

Файлы и директории

Файлы и Директории в Linux:

Файлы:

Последовательность байтов произвольной длины, содержащая информацию.
Не различаются по типу (текстовые, двоичные).
Имена файлов ограничены 255 символами, за исключением символа NUL.
Принято использовать расширения, но не обязательно для системы.
Директории:

Группируют файлы для удобной организации.
Иерархическая структура, корневой каталог - /.
Имена путей: абсолютные (от корня) и относительные (от текущего каталога).
Рабочий каталог (working directory) позволяет использовать относительные пути.
Символы и Расширения:

Соглашения для имен файлов: "prog.c" - программа на языке C, "prog.py" - на Python.
Максимальная длина расширения произвольна, файлы могут иметь несколько расширений.
Программы могут ожидать определенные расширения, но это не регламентируется ОС.
Ссылки (link):

Жесткие (hard) и символьные (soft).

Представляют записи каталога, указывающие на существующие файлы.

Жесткие ссылки - физически указывают на одни и те же данные, символьные - на путь.

Структуры Каталогов:

При создании каталога создаются записи "." (текущий каталог) и ".." (родительский).

Имя файла с точкой в начале считается скрытым, не отображается по умолчанию.

Домашний Каталог:

Хранит собственные данные пользователя и настройки программ.

Полный путь хранится в переменной окружения HOME.

Знак тильды ~ заменяет полный путь, например, ~/file1 эквивалентен /home/user1/file1.

Отображение Скрытых Файлов:

Скрытые файлы начинаются с точки в начале имени файла.

Опция -a или --all в команде ls для отображения скрытых файлов.

Директории и Пути:

Символ / разделяет имена каталогов в путях (например, /usr/ast/x).

Абсолютные пути указывают от корневого каталога, относительные - от рабочего каталога.

Домашний Каталог:

Предназначен для хранения собственных данных пользователя и личных настроек.

Обозначается тильдой ~ и хранится в директории /home.

Правила Именования:

Используются все ASCII-символы, кроме NUL.

Длина имени файла ограничена 255 символами.

Управление внешней памятью.

NFS (Network File System):

Основные принципы:

Предоставляет доступ к общим файловым системам для различных клиентов и серверов.

Клиенты и серверы могут быть в одной локальной сети или работать удаленно через Глобальную сеть.

Экспорт и Монтаж:

Каждый сервер NFS экспортирует свои каталоги для доступа клиентам.

Файл /etc/exports хранит список экспортируемых каталогов.

Клиенты монтируют удаленные каталоги, интегрируя их в свою файловую систему.

Примеры Монтирования:

Клиенты монтируют каталоги серверов в своих локальных каталогах.

Различные имена файла на различных клиентах из-за разных точек монтирования.

Точка монтирования - локальная для клиента, сервер не знает о ее местоположении.

Поддерживаемые Системные Вызовы:

Поддерживаются большинство системных вызовов Linux, за исключением open и close.

Отсутствие необходимости открывать и закрывать файлы перед чтением.

Операция lookup:

Запрос на сервере для поиска файла и возврата описателя файла.

Необходим для чтения файла, не требует предварительного открытия.

Преимущества Схемы:

Нет необходимости хранить информацию о открытых файлах на сервере.

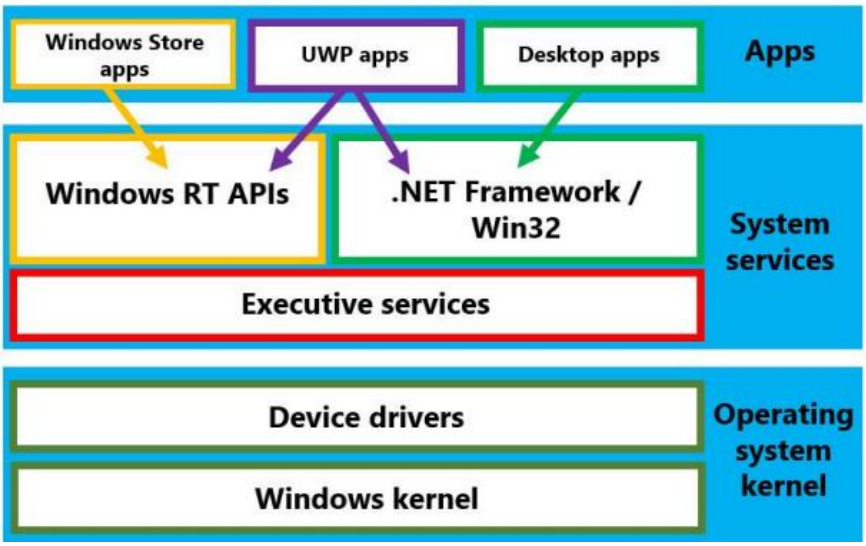
Перезагрузка сервера не приводит к потере информации о файлах.

Серверы, не поддерживающие информацию состояния, называются серверами без состояния.

22. Принципы организации и структура ОС Windows. Обзор версий Windows. Методы инсталляции ОС Windows.

Принципы организации и структура ОС Windows

Архитектура операционной системы Windows включает ядро операционной системы, системные службы и приложения



Ядро операционной системы

На самом низком уровне операционной системы ядро операционной системы состоит из самого ядра Windows и драйверов устройств низкого уровня. Ядро отвечает за прием запросов операционной системы от системных служб. Затем ядро преобразует эти запросы в инструкции для аппаратного обеспечения компьютера, включая центральный процессор (ЦП), память и аппаратные устройства. При запуске операционной системы сначала инициализируются ядро и связанные с ним низкоуровневые драйверы устройств, а затем службы операционной системы.

Системные службы

Службы операционной системы являются частью операционной системы, а не компонентами, которые вы устанавливаете после развертывания операционной системы.

Кроме того, службы операционной системы функционируют без каких-либо действий со стороны пользователя. Фактически они начинаются до того, как пользователь войдет в систему. И службы операционной системы, и драйверы устройств являются программным обеспечением. Однако разница между ними заключается в том, что драйверы устройств напрямую взаимодействуют с аппаратными устройствами или компонентами. Как правило, системная служба взаимодействует с другими программными компонентами операционной системы

Понимание приложений

На верхнем уровне операционной системы приложения работают путем взаимодействия с пользователем компьютера, а на нижнем уровне — путем интеграции со службами операционной системы

Отличия службы от приложения

Служба – процесс, работающий, даже когда никто не зарегистрирован в системе.

Сравнение служб и драйверов

Информация о службах и драйверах хранится в одном и том же разделе реестра. Различить их можно по параметрам Start и Type

Реестр Windows

Реестр — это база данных, в которой Windows хранит параметры конфигурации пользователя и компьютера. Всякий раз, когда вы вносите изменения в конфигурацию Windows, это изменение фиксируется в реестре

Реестр Windows организован иерархически

Обзор версий Windows

Windows 10/Выпуск Windows 11	Аудитория	Доступность
Home	Индивидуальное домашнее использование	Каждый
Pro	Малый и средний бизнес, продвинутые пользователи	Каждый
Pro для рабочих станций	Пользователи с повышенными требованиями к производительности и хранилищу	Каждый
Enterprise	Крупные коммерческие организации	Доступно для корпоративной лицензии. Корпоративное лицензирование Microsoft, Соглашение Microsoft Enterprise, Microsoft Store для образовательных учреждений или программа Microsoft Cloud Solution Provider.
Enterprise LTSC	Крупные корпоративные организации с ограничительными требованиями к изменениям	Корпоративное лицензирование Microsoft, Соглашение Microsoft Enterprise или программа поставщика облачных решений Microsoft.
Pro Education	Сравнимо с Pro для сотрудников школы, администраторов, учителей и учащихся.	Доступно для клиентов с корпоративной лицензией для учебных заведений.

Education	Сравнимо с Enterprise для школьного персонала, администраторов, учителей и учащихся.	Доступно для клиентов с корпоративной лицензией для учебных заведений.
IoT Core/ Enterprise	Устройства стационарного назначения и бытовые устройства	Доступно через дистрибьюторов Windows IoT

Серверные версии Windows:

Последняя LTSC-версия: Windows Server 2022, также поддерживается Windows Server 2019.

Безопасность:

Windows Server 2022 обеспечивает многоуровневую безопасность и гибридные возможности с Azure.

Встроенные возможности защищенного ядра снижают риск от угроз безопасности.

Защищенное Соединение:

Улучшенное защищенное соединение с использованием быстрых и защищенных зашифрованных соединений HTTPS и стандартного шифрования SMB AES 256.

Редакции Windows Server 2022:
Standard: Основная функциональность, исключая Hotpatching и SDN. Storage Replica ограничен до 2TB и 2 автоматически лицензируемых машин.
Datacenter: Поддерживает все функции, кроме Hotpatching и SMB over QUIC.
Datacenter: Azure Edition: Премиальная редакция, поддерживается в Azure, предлагает дополнительные функции, такие как горячее обновление и SMB через QUIC.

Методы инсталляции ОС Windows

Ручная установка с использованием установочного носителя:
Установка системы вручную с использованием установочного носителя.
Применяется, когда нет подходящего образа для конкретной аппаратной платформы или при небольшом количестве установок.
Автономная (необслуживаемая) установка:
Используется файл ответов, предоставляющий ответы на вопросы, задаваемые во время ручной установки.
Позволяет автоматизировать установку с заданными параметрами.

Установка, сопровождаемая производителем:
Некоторые производители поставляют носители автономной установки, упрощающие процесс для конечного пользователя.

Создание копий или образов систем:
Полезно при развертывании нескольких идентичных компьютеров или серверов.
Создание образов системы с помощью инструментов Microsoft, таких как WDS, Sysprep, или сторонних разработчиков.

Служба развертывания Windows (WDS):
Позволяет развертывать установочные образы и специально настроенные образы на серверах и настольных компьютерах.

Диспетчер настроек системного центра (SCCM):
Средство развертывания и управления образами систем с возможностью централизованного управления.

Инструментальный набор развертывания Microsoft (MDT):
Обеспечивает средства для создания образов, управления драйверами и приложениями, а также автоматизации процесса развертывания.

23. ОС Windows: организация рабочей среды пользователя, работа с учетными записями пользователей и групп, работа с профилями пользователей.

Организация рабочей среды пользователя

Рабочая среда пользователя состоит из настроек рабочего стола, например, цвета экрана, настроек мыши, размера и расположения окон, из настроек процесса обмена информацией по сети и с устройством печати, переменных среды, параметров реестра и набора доступных приложений

Для управления средой пользователя предназначены следующие средства Windows:

Сценарий Входа в Сеть:

Командные файлы (.bat) или исполняемые файлы (.exe), выполняемые при входе пользователя в сеть.

Могут включать команды для создания сетевого соединения или запуска приложений.

Профили Пользователей:

Хранят настройки рабочей среды пользователя.

Сервер Сценариев Windows (WSH):

Независим от языка и предназначен для работы на 32-разрядных платформах Windows. Включает в себя ядро сценариев VBScript и JScript. Позволяет выполнение сценариев на рабочем столе Windows или в консоли команд. Большинство настроек пользовательской среды управляются с помощью групповых политик.

Работа с учетными записями пользователей и групп

Локальные учетные записи пользователей определяются локально на устройстве и могут назначаться только на этом устройстве

Учетные записи локальных пользователей по умолчанию — это встроенные учетные записи, которые создаются автоматически при установке операционной системы.

Учетная запись локального администратора по умолчанию — это учетная запись пользователя для системного администрирования.

Учетная запись администратора имеет полный контроль над файлами, каталогами, службами и другими ресурсами на локальном устройстве. Учетная запись администратора может создавать других локальных пользователей, назначать права пользователей и назначать разрешения. Учетная запись администратора может в любое время управлять локальными ресурсами, изменив права и разрешения пользователя.

Гостевая учетная запись позволяет случайным или разовым пользователям, у которых нет учетной записи на компьютере, временно войти на локальный сервер или клиентский компьютер с ограниченными правами пользователя.

Учетные записи локальных пользователей по умолчанию и созданные вами учетные записи локальных пользователей находятся в папке «Пользователи».

Работа с профилями пользователей

Система создает профиль пользователя при первом входе пользователя на компьютер. При последующих входах система загружает профиль пользователя, а затем другие системные компоненты настраивают среду пользователя в соответствии с информацией в профиле

Типы профилей пользователей:

Локальные профили: Хранятся на локальном жестком диске компьютера.

Перемещаемые профили: Копия локального профиля на сервере, загружается при входе в сеть.

Обязательные профили: Используются администраторами для указания параметров, изменения не сохраняются.

Временные профили: Выдаются при ошибке загрузки профиля, удаляются после каждого сеанса.

Элементы профиля пользователя:

Куст реестра: Файл NTuser.dat, загружается при входе, содержит настройки и конфигурацию реестра.

Папки профилей в файловой системе: Хранят файлы профиля, такие как документы, настройки приложений.

Преимущества профилей пользователей:

Сохранение параметров между входами пользователя.

Настроенный рабочий стол для каждого пользователя при совместном использовании компьютера.

Уникальность параметров в профиле для каждого пользователя.

Изменения в профиле одного пользователя не влияют на других пользователей.

24. Системный реестр ОС Windows, его назначение и использование.

Реестр — это база данных, в которой Windows хранит параметры конфигурации пользователя и компьютера. Всякий раз, когда вы вносите изменения в конфигурацию Windows, это изменение фиксируется в реестре

Реестр Windows организован иерархически. На верхнем уровне имеется пять кустов реестра, которые представляют собой отдельный набор связанных параметров, структурированных как серия ключей, подразделов и значений: HKEY_CLASSES_ROOT, HKEY_CURRENT_USER, HKEY_LOCAL_MACHINE, HKEY_USERS и HKEY_CURRENT_CONFIG.

- HKEY_CLASSES_ROOT: Ассоциации файлов и приложений для открытия файлов.
- HKEY_CURRENT_USER (HKCU): Информация о конфигурации текущего пользователя.
- HKEY_LOCAL_MACHINE (HKLM): Важные параметры конфигурации компьютера.
- HKEY_USERS: Информация о конфигурации всех пользователей, включая текущего.
- HKEY_CURRENT_CONFIG: Информация о текущем профиле оборудования.

Для поддержания структуры базы данных аналогичные настройки хранятся в папках и подпапках, известных как ключи и подразделы. Это упрощает ссылку на определенное значение реестра. Вы можете указать путь, объявив соответствующий куст, ключ, подразделы и значение:

HKCU\Control Panel\Desktop\Wallpaper: Хранит обои рабочего стола пользователя.

HKLM\Software\Microsoft\Windows\CurrentVersion\Run: Содержит программы для автозапуска.

Значения определяют поведение операционной системы и хранятся в разделах и подразделах. Существует много типов значений, в зависимости от типа данных, которые каждое из них хранит.

REG_BINARY. Необработанные двоичные данные. Эти значения обычно отображаются в шестнадцатеричном формате. Информация об оборудовании часто хранится в значениях REG_BINARY.

REG_DWORD. 4-байтовые числа (32-битное целое число). Многие значения, связанные с драйверами устройств и службами, хранятся в значениях REG_DWORD. Например, значения START и TYPE для драйверов устройств всегда определяются в значениях типа REG_DWORD.

REG_SZ. Текстовая строка фиксированной длины.

REG_EXPAND_SZ. Текстовая строка переменной длины.

REG_MULTI_SZ. Несколько строковых значений.

25. ОС Windows: планирование и назначение разрешений NTFS.

1. Введение:

NTFS (New Technology File System) - файловая система, используемая в операционных системах Windows.

Разрешения NTFS определяют доступ к файлам и папкам на уровне файловой системы.

2. Понимание Разрешений:

Разрешения определяют, кто может что делать с файлами и папками (чтение, запись, исполнение).

Каждый объект (файл или папка) имеет ACL (Access Control List), содержащий список разрешений.

3. Пользователи и Группы:

Разрешения могут быть назначены пользователям или группам пользователей.

Группы упрощают управление, так как разрешения могут быть назначены целой группе.

4. Права и Роли:

Права: Определяют общие действия, такие как изменение системного времени.

Роли: Задают специфические задачи, такие как администратор или пользователь.

5. Процесс Назначения Разрешений:

Выбор Объекта: Выберите файл или папку, для которой вы хотите настроить разрешения.

Свойства и Закладка "Безопасность": Правый клик по объекту, выберите "Свойства" и перейдите на закладку "Безопасность".

Выбор Пользователя или Группы: Нажмите "Изменить" и выберите нужного пользователя или группу.

6. Работа с Разрешениями:

Разрешения по Умолчанию: Определяют доступ к новым объектам до изменения разрешений.

Наследование Разрешений: Разрешения могут быть унаследованы от родительских объектов.

7. Основные Разрешения:

Полный доступ (Full Control): Полные права на чтение, запись, изменение, удаление и выполнение.

Чтение и Исполнение (Read & Execute): Позволяет просматривать и выполнять файлы, но не изменять.

Запись (Write): Позволяет создавать и изменять файлы, но не удалять или исполнять.

8. Пример Назначения Разрешений:

Задача: Назначить группе "Редакторы" разрешение на изменение файлов в папке "Проекты".

Действия:

Выберите папку "Проекты".

Перейдите на закладку "Безопасность".

Нажмите "Изменить" и добавьте группу "Редакторы" с разрешением "Изменение".

9. Проверка Разрешений:

Проверяйте разрешения, удостоверьтесь, что необходимые группы и пользователи имеют доступ.

10. Завершение:

Разрешения NTFS важны для безопасности и управления файлами в Windows.

Регулярно аудитите разрешения для поддержания актуальности и безопасности.

26. Средства автоматической настройки в ОС Windows.

– Командное консольное окно.
– BAT-файлы.
– PowerShell.
– WMI.
– ETW.
– Групповые политики.
– Административные шаблоны.
– Файловые системы.
– Динамические диски.
– Управление сжатием на дисках NTFS.
– Шифрование файлов.
– Управление дисковыми квотами.
– Использование Консоли восстановления.

Командное консольное окно

В Windows есть две оболочки командной строки: командная оболочка **cmd** и **PowerShell**.

Каждая оболочка — это программная программа, которая обеспечивает прямую связь между оператором и операционной системой или приложением, предоставляя среду для автоматизации ИТ-операций.

Командная оболочка **cmd** была первой оболочкой, встроенной в Windows, для автоматизации повседневных задач, с пакетными (.bat или .cmd) файлами.

PowerShell был разработан для расширения возможностей командной оболочки для выполнения команд PowerShell, называемых **командлетами**. Командлеты похожи на команды Windows, но предоставляют более расширяемый язык сценариев.

Команды Windows и командлеты PowerShell можно запускать в PowerShell, но командная оболочка может выполнять только команды Windows, а не командлеты PowerShell.

(также тут можно написать, как открыть cmd или PowerShell, а также написать примеры команд)

BAT-файлы

BAT-файл (Batch-файл)текстовый файл, содержащий команды системы, которые могут выполняться последовательно. Расширение .cmd или .bat.

Комментарии начинаются с REM (remark)

Строки, начинающиеся с @, не выводятся на экран.

Допускаются программные конструкции, например:

for %%i **in** (1 1 5) **do** *echo* %%i (%% переменные в циклах, а % переменные вне цикла) (**in** (1-нач знач 1-шаг 5-конечное знач)

if a *equ* 0 *echo* 0 **else** *echo* notzero (*equ- =)*

Для вызова другого скрипта используется команда call

Для создания переменной используется команда set (set a=0)

PowerShell — расширяемое средство автоматизации от Microsoft с открытым исходным кодом.(выпущен августе 2016 года.)

- Основан на .NET Framework и интегрирован с ним
- Интеграция с COM (Component Object Model), WMI(Windows Management Instrumentation) и ADSI (Active Directory Service Interfaces)
- Командлеты
- доступ к источникам данных, таким как файловая система или реестр Windows
- имеет собственную расширяемую справку, доступную через **командлет Get-Help.**

WMI

Инструментарий управления Windows **(WMI)** — это инфраструктура для управления данными и операциями в операционных системах Windows. Он представляет собой реализацию стандарта WBEM, ориентированного на создание универсального интерфейса мониторинга и управления различными системами.

В основе структуры данных в WBEM **лежит Common Information Model (CIM),** реализующая ООП-подход к представлению компонентов системы.

wbemtest.exe — графическая утилита для взаимодействия со структурой WMI на локальном или удаленном компьютере.

wmic.exe — консольная утилита для взаимодействия со структурой WMI на локальном или удаленном компьютере.

Event Tracing for Windows (ETW) — это служба, которая позволяет получать события от одного или нескольких поставщиков событий в режиме реального времени или из файла *.etl за некоторый временной период.

Архитектура ETW включает в себя 4 элемента (рис. 4.2.1):

- поставщики событий (providers)
- потребители событий (consumers)
- контроллеры ETW (controllers)
- сессии ETW (event tracing sessions)

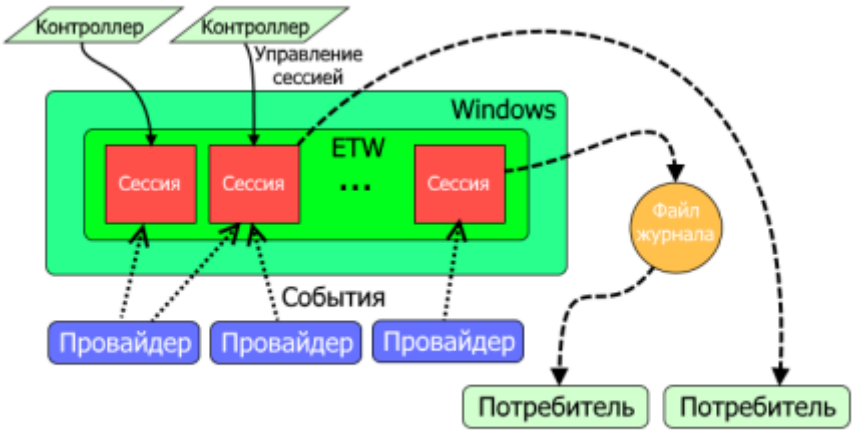


Рис. 4.2.1. Архитектура ETW

Групповая политика — важный элемент любой среды Microsoft Active Directory (AD). Её основная **цель** — дать ИТ-администраторам возможность централизованно управлять пользователями и компьютерами в домене.
Групповая политика **состоит из** набора политик, называемых **объектами групповой политики (GPO)**.
При создании домена AD автоматически создаются два объекта групповой политики:
• **политика домена** по умолчанию устанавливает базовые параметры для всех пользователей и компьютеров в домене в трех плоскостях: политика паролей, политика блокировки учетных записей и политика Kerberos;
• **политика контроллеров домена** по умолчанию устанавливает базовые параметры безопасности и аудита для всех контроллеров домена в рамках домена
Объект групповой политики не действует, пока не будет связан с контейнером Active Directory, например, сайтом, доменом или подразделением.

Административные шаблоны (файлы) в формате XML, написанные на различных языках, которые задают параметры групповых политик, основанные на значениях реестра. Эти параметры отображаются и редактируются в локальном редакторе групповых политик на компьютере. Административные шаблоны позволяют администраторам устанавливать и изменять настройки безопасности, поведения и другие параметры на компьютере или сети через групповые политики.

Два вида административных шаблонов:

• **ADMX** — не зависящий от языка файл установки, который указывает количество и тип параметров политики, расположение по категориям согласно отображению файла в редакторе локальных групповых политик.

• **ADML** — файл установки на определенном языке, который предоставляет связанные с языком сведения для ADMX-файла. позволяет параметру политики отображаться в редакторе локальных групповых политик на нужном языке. Вы можете добавлять новые языки, добавляя новые ADML-файлы на нужных языках.

Файловая система – это способ организации и хранения файлов на жестком диске компьютера.

Основным типов является **NTFS** (New Technology File System). NTFS обладает высокой степенью надежности, поддерживает шифрование(технологии EFS

(Encrypting File System)) и права доступа к файлам и папкам. Кроме того, NTFS позволяет работать с файлами размером до 16 терабайт. (стоит по умолч в Windows)

Другим типом явл **FAT** (File Allocation Table). FAT поддерживает работу с файлами менее 4 гигабайт и не обеспечивает такую высокую степень надежности, как NTFS. Однако FAT более совместима с различными операционными системами и устройствами.

Динамические диски предоставляют функции, которых нет у базовых дисков. Вы можете создавать тома, охватывающие несколько дисков, и отказоустойчивые тома. На каждом динамическом диске компьютера хранится копия базы данных динамического диска.

Следующие операции можно выполнять только на динамических дисках:

- Создание и удаление составных, чередующихся и зеркальных томов.
- Расширьте простой том до несмежного пространства или составного тома.
- Удалите зеркало из зеркального тома.
- Восстановление зеркальных томов.
- Повторно активируйте отсутствующий или отключенный диск

Особенности:

- Вы не можете преобразовать базовый диск в динамический диск, если на диске не имеется хотя бы 1 МБ неиспользуемого пространства из-за базы данных диспетчера логических дисков.
- Вы не можете преобразовать динамический диск в базовый без потери данных.
- Вы не можете использовать Windows PowerShell для управления динамическими дисками

Управление сжатием на дисках NTFS

два типа сжатия:

- NTFS-сжатие на уровне файловой системы,
- Сжатые ZIP-папки (Compressed Folders).

Сжатие NTFS – это архивирование на уровне файловой системы NTFS, выполняется оно прозрачно драйвером файловой системы. NTFS сжатие может применяться к файлам, папкам и дискам целиком. Для этого каждому сжатому объекту присваивается специальный атрибут сжатия (compression state), который указывает, сжат файл или нет.

– Сжатие NTFS возможно только на разделах с файловой системы NTFS(Zip и в NTFS и в FAT)

– При доступе к сжатому файлу или папке осуществляется прозрачная декомпрессия, т.е. пользователь не видит различий между сжатыми и обычными файлами

– Скорость доступа к сжатому файлу ниже, т.к. систем требуется некоторое время на его распаковку

– Сжатие более слабое, чем при использовании ZIP-папок, но скорость его выполнения гораздо выше

Шифрование файлов

Windows поддерживает две технологии шифрования:

- **EFS** — на уровне файлов и папок;
- **BitLocker** — на уровне томов.

Encrypting File System (**EFS**) — система шифрования данных, реализующая шифрование на уровне файлов (кроме «домашних» версий Windows). Предоставляет возможность «прозрачного шифрования»,т.е
EFS работает, шифруя каждый файл с помощью алгоритма симметричного шифрования, зависящего от версии операционной системы и настроек. При этом используется случайно сгенерированный ключ для каждого файла, называемый File Encryption Key (FEK)

BitLocker — это функция безопасности Windows, которая обеспечивает шифрование для целых томов, устраняя угрозы кражи или раскрытия данных с потерянных, украденных или неправильно списанных устройств. TPM работает с BitLocker, чтобы убедиться, что устройство не было изменено, пока система находится в автономном режиме.

Дисковые квоты позволяют контролировать сколько места используют пользователи на файловой системе серверов и рабочих станций.
ОС Windows Server поддерживает два типа квотирования:

- квотирование на базе File Server Resource Manager (дисковые квоты FSRM);
- NTFS квоты.

С помощью дисковых квот Windows вы можете ограничить максимальный размер файлов и папок каждого пользователя так, чтобы он не превысил установленного лимита и не занял своими данными весь диск.

Использование консоли восстановления
Среда восстановления Windows 10 позволяет производить операции, имеющие отношение к восстановлению работоспособности системы: использовать **особые варианты загрузк**и (например, безопасный режим или отключение проверки цифровой подписи драйверов), **выполнять сброс ОС** или **автоматическое восстановление загрузчика Windows 10** и т.д.

27. Основные понятия системы UNIX. Пользователи системы, атрибуты пользователя. Файловая структура ОС.

ОС UNIX базируется на работе с двумя основными объектами - **файлами** и **процессами**.

Файлы хранят данные пользователя, а **процессы** определяют функциональность системы.

Работа с файлами осуществляется через чтение и запись специальных файлов, а выполнение программ включает считывание исполняемого кода из файла в память.

Операционная система UNIX организована как пирамида, где аппаратное обеспечение находится в основании, а операционная система управляет аппаратурой и предоставляет интерфейс системных вызовов для программ.

Рис. 4.3.1. Уровни операционной системы Unix

Программы используют системные вызовы для создания и управления процессами, файлами и другими ресурсами.

Графический интерфейс пользователя основан на оконной системе X, которая управляет визуальным представлением на экране.

Операционная система также включает стандартные программы, такие как командный процессор, компиляторы, редакторы, и поддерживает графические интерфейсы пользователя через X Windowing System.

Пользователи системы

Прежде чем клиент сможет начать работу с ОС UNIX, он должен стать пользователем системы, т.е. получить имя, пароль и ряд других атрибутов **Пользователем** является объект, который обладает определенными правами и может запускать на выполнение программы и владеть файлами. Пользователями могут быть отдельные клиенты, удаленные компьютеры или группы пользователей с одинаковыми правами и функциями. В системе существует один пользователь, обладающий неограниченными правами это **суперпользователь** или **администратор** системы (обычно с именем **root**).

Система различает пользователей по идентификатору пользователя UID.

Группа — список пользователей, имеющих сходные задачи. Каждая группа имеет уникальное имя, а система различает группы по групповому идентификатору (GID).Информация о пользователях обычно хранится в специальном файле: */etc/passwd*, о группах — */etc/group*. (файлы доступены только для чтения, писать может только админ)

Атрибуты пользователя

все атрибуты пользователя хранятся в файле /etc/passwd

Каждая строка файла является записью конкретного пользователя и имеет следующий формат:

name:passwd-encod:UID:GID:comments:home-dir:shell

всего семь полей (атрибутов), разделенных двоеточиями.

• name: Регистрационное имя пользователя, используемое при входе в систему.

• passwd-encod: Закодированный пароль пользователя. Алгоритм кодирования предотвращает декодирование пароля. В некоторых системах пароль хранится в отдельном файле, а в поле passwd-encod может быть символ 'x' или '!'.

• UID: Идентификатор пользователя, внутреннее представление пользователя в системе. Суперпользователь (администратор) имеет UID=0.

• GID: Идентификатор первичной группы пользователя, соответствует идентификатору в файле /etc/group.

• comments: Полное "реальное" имя пользователя или дополнительная информация.

• home-dir: Домашний каталог пользователя, место, где пользователь оказывается при входе в систему.

• shell: Имя программы, используемой в качестве командного интерпретатора при входе пользователя. Обычно это /bin/sh, /bin/csh или /bin/ksh.

Файловая структура ОС.

В начале развития Linux использовалась файловая система MINIX 1, но ограничения по длине имен и размеру файлов привели к разработке улучшенной файловой системы ext. Затем появилась файловая система **ext2**, ставшая основной для Linux.

Файл в системе Linux — это последовательность байтов произвольной длины (от 0 до некоторого максимума), содержащая произвольную информацию.

Не делается различия между текстовыми (ASCII) файлами, двоичными файлами и любыми другими типами файлов. Значение битов в файле целиком определяется владельцем файла. Имена файлов ограничены 255 символами. В именах файлов разрешается использовать все ASCII-символы, кроме символа NULL.

Для удобства файлы могут группироваться в **каталоги**. Каталоги хранятся на диске в виде файлов. Каталоги могут содержать подкаталоги, что приводит к иерархической файловой системе. Корневой каталог называется /

Таблица 4.3.1. Некоторые важные каталоги, существующие в большинстве систем Linux

Каталог	Содержание
bin	Двоичные (исполняемые) программы
dev	Специальные файлы для устройств ввода-вывода
etc	Разные системные файлы
lib	Библиотеки
usr	Каталоги пользователей

Существует два способа указания имени файла в системе Linux: абсолютный путь, начинающийся от корневого каталога, и относительный путь, указывающий относительно текущего рабочего каталога.

Стандарт POSIX обеспечивает гибкий механизм блокировки файлов, позволяя процессам блокировать как единичные байты, так и целые файлы за одну неделимую операцию. Стандарт определяет два типа блокировок: с монополизацией (эксклюзивные) и без монополизации (общие). Если часть файла имеет блокировку без монополизации, то можно повторно установить такую же блокировку, но блокировка с монополизацией будет отвергнута. Если область файла заблокирована с монополизацией, то любые попытки блокировки этой области будут отвергнуты

до снятия блокировки. Успешная установка блокировки требует доступности каждого байта в заблокированной области.

28. ОС UNIX: особенности процессов, сигналы, обработка сигналов.

Основными активными сущностями в системе Linux являются **процессы**. Каждый процесс выполняет одну программу и изначально получает один поток управления. Иначе говоря, у процесса есть один счетчик команд, который отслеживает следующую исполняемую команду. Linux позволяет процессу создавать дополнительные потоки (после того, как он начинает выполнение).

Системный вызов **fork** создает точную копию исходного процесса, называемого родительским процессом (parent process). У родительского и у дочернего процессов есть собственные (приватные) образы памяти. Если родительский процесс впоследствии изменяет какие-либо свои переменные, то эти изменения остаются невидимыми для дочернего процесса (и наоборот).

fork возвращает дочернему процессу число 0, а родительскому — отличный от нуля PID дочернего процесса. **getpid**

Типы процессоров:

Системные процессы - часть ядра, всегда в оперативной памяти. Не имеют соответствующих исполняемых файлов, запускаются особым образом при инициализации ядра. Примеры: shed, vhand, bdfflush, kmadaemon, и init.

Демоны (daemons) фоновые процессы, работающие в фоновом режиме и обслуживающие различные системные задачи или услуги. Они часто запускаются при загрузке системы и работают независимо от активности конкретного пользователя. Типичным демоном является ***cron***. Он просыпается раз в минуту, проверяя, не нужно ли ему что-то сделать. Есть работа-выполняет, нету-спит до след проверки. Позволяет планировать в системе Linux активность на минуты, часы, дни и месяцы вперед.

Прикладные процессы - все остальные, выполняющиеся в системе, часто порождаемые пользовательским сеансом. Пример: команды, запущенные пользователем. Время их выполнения ограничено сеансом работы.

Сигналы — это способ информирования процесса со стороны ядра о происшествии некоторого события. Один процесс может посылать сигнал другому через системный вызов kill.

Обработка сигналов в Linux осуществляется через установку обработчиков сигналов. Когда процесс получает сигнал, ядро отправляет управление к обработчику, который был предварительно установлен для этого сигнала. Процесс может выбрать различные действия при получении сигнала:

- **Игнорирование (SIG_IGN):** Процесс может проигнорировать сигнал, если установлен обработчик SIG_IGN (ignore).
- **По умолчанию (SIG_DFL)**: Если не установлен пользовательский обработчик, применяется действие по умолчанию (default). Например завершение процесса.
- **Пользовательский обработчик (функция):** Процесс может установить свою собственную функцию-обработчик. Когда сигнал поступает, управление передается этой функции.

29. Основные принципы функционирования Linux. Основные компоненты Linux. Дистрибутивы Linux. Файловая система Linux.

Linux имеет низкие аппаратные требования в текстовом режиме, что позволяет работать на машинах с 486 процессором и 16 MB RAM. Он организован по принципу клиент-сервер, где компьютеры в сети взаимодействуют, создавая единый компьютер. Пользователи работают через виртуальные терминалы, переключаясь<Alt> + <F1...F12>. Один компьютер может одновременно выполнять роль сервера и рабочей станции.

Linux - это ядро и ядро каждого дистрибутива Linux. Программное обеспечение ядра Linux поддерживается группой людей, возглавляемой Линусом Торвальдсом. Версию используемого ядра Linux можно узнать с помощью команды **uname**

ОС Linux обладает различными компонентами и возможностями:

– *Планировщик*: Управляет приоритетами процессов.

– *Файл подкачки:* Используется для хранения неактивных процессов на диске при переполнении оперативной памяти.

– *Модули:* Позволяют динамически добавлять поддержку новых устройств, что помогает сократить размер ядра.

– *Файловые системы:* Поддерживают различные типы файловых систем, включая ext3, ReiserFS, VFAT и NTFS.

– *Механизмы защиты:* Используются для установки разрешений на чтение, запись и выполнение файлов, обеспечивая безопасность.

– *Инструменты администрирования*: управления пользователями, дисками, сетями и другими ресурсами.

– *Серверные возможности:* Позволяют использовать Linux в роли сервера, предоставляя различные службы, такие как веб-серверы, почтовые серверы и другие.

– *Система управления пакетами:* Позволяет автоматизировать установку, обновление и удаление программного обеспечения, используя различные форматы пакетов, такие как RPM, DEB, и другие.

Мы можем распределить **дистрибутивы Linux** на три группы:

Enterprise Grade Linux Дистрибутивы этой группы предназначены для развертывания в крупных организациях с использованием оборудования предприятия.(Red Hat Enterprise Linux, CentOS, SUSE Linux Enterprise Server, Debian GNU/Linux , Ubuntu LTS)

Особенности:

- Стабильность и долгосрочная поддержка.
- Ориентированы на надежность и безопасность.
- Медленное внедрение новых технологий.
- Предоставляют более старые, но стабильные версии программного обеспечения.

Consumer Grade Linux Дистрибутивы этой группы больше ориентированы на малый бизнес или домашних пользователей и любителей.(Fedora, Ubuntu non-LTS, openSUSE)

Особенности:

- Ориентированы на новое оборудование и современные драйверы.
- Регулярные выпуски с обновлениями.
- Более быстрое внедрение новых технологий.

Experimental and Hacker Linux Дистрибутивы этой группы используют самые современные технологии. (Arch, Gentoo)

Особенности:

- Использование самых последних технологий.
- Скользящая модель выпуска.
- Предоставляют возможность для тестирования будущих функций.
- Разработаны для пользователей, готовых к возможным проблемам и изменениям.

Файловая система в ОС Linux как и в большинстве других систем имеет иерархическую (древовидную) структуру. Все объекты являются файлами, в том числе и директории для организации доступа к файлам.

Существуют следующие типы файлов: обычные файлы, каталоги, символьные ссылки, блочные устройства, символьные устройства, сокеты, каналы. Тип выводится **ls -l.**

всегда есть только один корневой каталог, который называется /

Абсолютные и относительные пути

Имя может содержать любые символы (в том числе и кириллицу), кроме / ? < > * " | Максимальная длина имени файла — 254 символа

Точка **.** в начале имени файла делает его **скрытым**, то есть, он не показывается в выводе команды ls.

Команда **pwd** (Print Working Directory) возвращает полный путь к текущей директории командной оболочки (хранится в переменной PWD) Для смены текущей директории используется команда **cd** (Change Directory)

Домашний каталог (домашняя папка, домашняя директория) – предназначен для хранения собственных данных пользователя Linux и личных настроек для программ. Полный путь к домашнему каталогу хранится в переменной окружения **HOME**, Для обычных пользователей домашний каталог находится в директории **/home**

FHS (Filesystem Hierarchy Standard) – стандарт файловой системы. В FHS все файлы и каталоги находятся внутри корневого каталога, даже если они расположены на различных физических носителях. В корневой директории должны быть следующие директории:

Определенные в стандарте FHS директории:

/	Корневой каталог, содержащий всю файловую иерархию
/bin	Основные утилиты, необходимые как в однопользовательском режиме, так и при обычной работе всем пользователям
/boot	Файлы ядра
/dev	Файлы устройств
/etc	Общесистемные конфигурационные файлы
/home	Содержит домашние папки пользователей. Часто размещается на отдельном разделе
/lib	Основные библиотеки, необходимые для работы программ из /bin и /sbin
/media	Точки монтирования для сменных носителей, таких как CD-ROM, DVD-ROM
/mnt	Содержит временно монтируемые файловые системы
/opt	Дополнительное прикладное программное обеспечение
/proc	Виртуальная файловая система, представляющая состояние ядра операционной системы и запущенных процессов в виде файлов
/root	Домашняя папка пользователя root
/run	Временные данные выполняющихся процессов
/sbin	Основные системные программы для администрирования и настройки системы
/srv	Данные для сервисов, предоставляемых системой
/tmp	Временные файлы
/usr	Вторичная иерархия для данных пользователя, содержит большинство пользовательских приложений и утилит, используемых в многопользовательском режиме.
/var	Изменяемые файлы, такие как файлы регистрации, временные почтовые файлы, файлы спулеров

30. ОС Linux: управление процессами, выполнение задач в фоновом режиме, изменение приоритетов выполняющихся программ.

Процессы существуют в иерархии: после загрузки ядра в память запускается первый процесс (init или systemd), который, в свою очередь, запускает другие процессы, которые, опять же, могут запускать другие процессы.

Каждый раз, когда пользователь вводит команду, запускается программа и генерируется один или несколько процессов. Каждый процесс имеет уникальный идентификатор (**PID**) и идентификатор родительского процесса (**PPID**).

Команда **top** динамически отображает все запущенные процессы

Иерархию процессов команда **pstree**.

Завершить процесс **kill**.

Для выполнения команды в **фоновом режиме** достаточно добавить в конце символ амперсанда **&**.

Работая в фоновом режиме, команда все равно продолжает выводить сообщения в терминал, из которого была запущена. Для этого она использует потоки stdout и stderr, которые можно закрыть при помощи следующего синтаксиса:

```
$ command >  /dev/null 2>&1 &
```

Узнать состояние всех остановленных и выполняемых в фоновом режиме задач в рамках текущей сессии терминала можно при помощи утилиты **jobs -l.**

В любое время можно вернуть процесс из фонового режима на передний план **fg**. Если в фоновом режиме выполняется несколько программ, следует указывать номер. Например: **$ fg %1**

Для завершения фонового процесса применяют команду **kill** с номером программы

Утилита **nice** — программа, предназначенная для запуска процессов с изменённым приоритетом nice.

Приоритет **nice** (целое число) процесса используется планировщиком процессов ядра ОС при распределении процессорного времени между процессам

nice – значение любезности, чем меньше, тем выше приоритет

значение nice – минимальное значение приоритета =лучшее значение = самый высокий приоритет.

nice – диапазон приоритетов [-20, 19], default = 0;

Чтобы установить значение nice ниже нуля, требуются права суперпользователя

Для того, чтобы изменить приоритет у существующего процесса (т.е. такого процесса, который ранее был уже запущен), необходимо воспользоваться командой:

```
$ renice [значение приоритета] -p [id процесса]
```

31. Понятие безопасности ОС. Основные угрозы безопасности ОС. Методы и защитные механизмы операционных систем.

Понятие безопасности операционной системы (ОС):

Безопасность в контексте операционных систем охватывает механизмы и стратегии, направленные на защиту информации и обеспечение нормальной работы системы. Для разграничения, термин "безопасность" будет использоваться для общих вопросов, а "защитные механизмы" — для конкретных мер, используемых ОС.

Таблица 5.1.1. Задачи и угрозы безопасности

Задачи	Угрозы
Конфиденциальность	Незащищенность данных
Целостность	Подделка данных
Доступность	Отказ от обслуживания

Первое свойство — конфиденциальность (confidentiality) — направлено на сохранение секретности данных

Второе свойство — целостность (integrity) — означает, что пользователи, не обладающие необходимыми правами, не должны иметь возможности изменять какие-либо данные без разрешения их владельцев.

Третье свойство — доступность (availability) — означает, что никто не может нарушить работу системы и вывести ее из строя

Классификация угроз **по цели** атаки:

- несанкционированное чтение информации;
- изменение информации;
- уничтожение информации;
- полное или частичное разрушение операционной системы.

Классификация угроз **по принципу воздействия** на операционную систему:

- использование известных (легальных) каналов получения информации
- использование скрытых каналов получения информации
- создание новых каналов получения информации с помощью программных закладок.

Классификация угроз **по характеру** воздействия на ОС:

- активное воздействие – несанкционированные действия злоумышленника в системе;

• пассивное воздействие – несанкционированное наблюдение злоумышленника за процессами, происходящими в системе

Операционная система **может подвергнуться следующим типичным атакам**:

• *сканирование файловой системы.* Злоумышленник просматривает файловую систему компьютера и пытается прочесть (или скопировать) все файлы подряд.

• *подбор пароля.* несколько методов подбора паролей:

- тотальный перебор;
- тотальный перебор, оптимизированный по статистике встречаемости символов или с помощью словарей;
- подбор пароля с использованием знаний о пользователе (его имени, фамилии, даты рождения, номера телефона и т. д.);

• *кража ключевой информации*. Носитель с ключевой информацией (смарткарта, Touch Memory и т. д.) может быть просто украден;

• *сборка мусора*. Во многих операционных системах удаленная пользователем информация не физически уничтожается, а лишь помечается как удаленная, что предоставляет возможность злоумышленникам восстановить и просмотреть данные.

• *превышение полномочий*. Злоумышленник, используя ошибки в программном обеспечении ОС получает полномочия, превышающие те, которые ему предоставлены в соответствии с политикой безопасности.

• *программные закладки.*

• *жадные программы* – это программы, преднамеренно захватывающие значительную часть ресурсов компьютера, в результате чего другие программы не могут выполняться или выполняются крайне медленно. Запуск жадной программы может привести к краху операционной системы.

Существуют ***два основных подхода*** *к созданию защищенных операционных систем*: **фрагментарный**, где защитные функции добавляются отдельно после создания основной системы, и **комплексный**, где защитные функции внедряются в архитектуру системы на этапе проектирования, обеспечивая их тесное взаимодействие и интеграцию.

Основные административные меры защит.

• Постоянный контроль корректности функционирования ОС, особенно ее подсистемы защиты

• Организация и поддержание адекватной политики безопасности.

•Осведомление пользователей о необходимости соблюдения мер безопасности при работе с ОС и контроль за соблюдением этих мер.

• Регулярное создание и обновление резервных копий программ и данных ОС.

• Постоянный контроль изменений в конфигурационных данных и политике безопасности ОС.

32. Механизмы безопасности в операционных системах семейства Windows.

• Идентификаторы безопасности (SID).

• Маркеры защиты.

• Списки управления доступом (ACL).

• Доменные службы Active Directory.

• Виды групп пользователей.

Идентификатор безопасности (SID) — это уникальное значение переменной длины, используемое для идентификации доверенного лица. Каждая учетная запись имеет уникальный идентификатор безопасности, выданный центром сертификации. Каждый раз, когда пользователь входит в систему, система получает идентификатор безопасности для этого пользователя из базы данных и помещает его в маркер доступа для этого пользователя.

Таблица 5.2.1. Функции Windows API для работы с идентификаторами безопасности

Функция	Описание
AllocateAndInitializeSid	Выделяет и инициализирует идентификатор безопасности с указанным количеством дополнительных учетных данных
ConvertSidToStringSid	Преобразует идентификатор безопасности в формат строки, подходящий для отображения, хранения или транспорта
ConvertStringSidToSid	Преобразует идентификатор безопасности строкового формата в допустимый функциональный идентификатор безопасности
CopySid	Копирует идентификатор безопасности источника в буфер
EqualPrefixSid	Проверяет два значения префикса SID на равенство. Префикс SID — это весь идентификатор безопасности, за исключением последнего значения подчиненного
EqualSid	Проверяет два идентификатора безопасности на равенство. Они должны точно совпадать, чтобы считаться равными
FreeSid	Освобождает ранее выделенный идентификатор безопасности с помощью функции **AllocateAndInitializeSid**
GetLengthSid	Извлекает длину идентификатора безопасности
GetSidIdentifierAuthority	Извлекает указатель на центр идентификатора для идентификатора sid
GetSidLengthRequired	Извлекает размер буфера, необходимого для хранения идентификатора безопасности с указанным числом вложенных учетных данных
GetSidSubAuthority	Извлекает указатель на указанную вложенную проверку подлинности в идентификаторе безопасности
GetSidSubAuthorityCount	Извлекает количество вложенных учетных данных в идентификаторе безопасности
InitializeSid	Инициализирует структуру SID
IsValidSid	Проверяет допустимость идентификатора безопасности, проверяя, что номер редакции находится в пределах известного диапазона и что количество вложенных авторов меньше максимального
LookupAccountName	Извлекает идентификатор безопасности, соответствующий указанному имени учетной записи
LookupAccountSid	Извлекает имя учетной записи, соответствующее указанному идентификатору безопасности

Маркер доступа представляет собой объект, содержащий информацию о контексте безопасности процесса или потока. Он включает удостоверение и привилегии учетной записи пользователя, связанной с процессом или потоком. После успешной аутентификации пользователя при входе в систему, система создает маркер доступа. Каждый процесс, выполняемый от имени данного пользователя, обладает копией этого маркера доступа. Маркер доступа используется системой для идентификации пользователя при взаимодействии с защищенными объектами или при выполнении системных задач, требующих привилегий.

Маркеры доступа содержат следующие сведения:

- Идентификатор безопасности (SID) для учетной записи пользователя

• Идентификаторы безопасности для групп, членом которых является пользователь

• Идентификатор безопасности входа, который идентифицирует текущий сеанс входа в систему.

• Список привилегий, которыми пользовались пользователи или группы пользователей.

• Идентификатор безопасности владельца • Идентификатор безопасности для основной группы и тд

Таблица 5.2.2. Функции для управления маркерами доступа

Функция	Описание
AdjustTokenGroups	Изменяет сведения о группе в маркере доступа.
AdjustTokenPrivileges	Включает или отключает привилегии в маркере доступа. Он не предоставляет новые привилегии и не отменяет существующие.
CheckTokenMembership	Определяет, включен ли указанный идентификатор безопасности в указанном маркере доступа.
CreateRestrictedToken	Создает новый маркер, который является ограниченной версией существующего маркера. Ограниченный маркер может иметь отключенные идентификаторы БЕЗОПАСНОСТИ, удаленные привилегии и список ограниченных идентификаторов БЕЗОПАСНОСТИ.
DuplicateToken	Создает новый токен олицетворения, дублирующий существующий маркер.
DuplicateTokenEx	Создает новый основной маркер или маркер олицетворения, который дублирует существующий маркер.
GetTokenInformation	Извлекает сведения о маркере.
IsTokenRestricted	Определяет, содержит ли маркер список ограничивающих идентификаторов безопасности.
OpenProcessToken	Извлекает дескриптор основного маркера доступа для процесса.
OpenThreadToken	Извлекает дескриптор маркера доступа олицетворения для потока.
SetThreadToken	Назначает или удаляет токен олицетворения для потока.
SetTokenInformation	Изменяет владельца маркера, основную группу или DACL по умолчанию

Дескриптор безопасности содержит сведения о безопасности, связанные с защищаемым объектом.

Списки управления доступом (ACL) состоят из записей управления доступом (ACE). Каждый ACE в ACL идентифицирует доверенного лица и указывает права доступа, разрешенные, запрещенные или регистрируемые для этого доверенного лица.

Дескриптор безопасности для защищаемого объекта может содержать два типа списков управления доступом: **DACL** и **SACL**

Список управления доступом (DACL) на уровне пользователей определяет, кому разрешен или запрещен доступ к объекту. При запросе доступа система проверяет записи ACE в DACL объекта, определяя, следует ли предоставлять

доступ. Если DACL отсутствует, полный доступ предоставляется всем пользователям. Если DACL не содержит соответствующих записей ACE, доступ к объекту отклоняется. Система проверяет ACE последовательно, пока не найдет разрешающую запись или отклонит доступ.

Системный список управления доступом (SACL) позволяет регистрировать попытки доступа к объекту, создавая записи аудита в журнале событий безопасности. Каждая запись ACE в SACL указывает типы попыток доступа, которые приведут к созданию записи в журнале событий. SACL может фиксировать попытки доступа при успехе и/или при неудаче.

> Чтобы определить права доступа доверенного лица к объекту, выполните следующие действия.
>
> 1. Вызовите функцию **GetSecurityInfo** или **GetNamedSecurityInfo**, чтобы получить указатель на DACL объекта.
> 2. Вызовите функцию **GetEffectiveRightsFromAcl**, чтобы получить права доступа, предоставляемые DACL указанному доверенному лицу.
>
> Функция **GetAuditedPermissionsFromAcl** позволяет проверить SACL, чтобы определить права доступа, проверенные для указанного доверенного лица или для любых групп, членом которых является это доверенное лицо. Проверяемые права указывают типы

Active Directory (AD) является службой каталогов, предоставляющей иерархическую структуру для хранения и предоставления сведений об объектах в сети. В контексте AD DS (Active Directory Domain Services), эта служба хранит информацию о пользователях, группах, серверах и других ресурсах в сети. Администраторы и авторизованные пользователи могут получать доступ к этим данным для управления и использования в сети

Active Directory также включает следующие **компоненты**:

- *Схема* — набор правил, который определяет классы объектов и атрибутов, содержащихся в каталоге, ограничения и ограничения экземпляров этих объектов, а также формат их имен.
- *Глобальный каталог*, содержащий сведения о каждом объекте в каталоге. Это позволяет находить сведения о каталоге независимо от того, какой домен в каталоге фактически содержит данные.
- *Механизм запроса и индекса*, чтобы объекты и их свойства могли быть опубликованы и найдены сетевыми пользователями или приложениями.
- *Служба репликации*, которая синхронизирует данные каталога по сети. Все контроллеры домена в домене участвуют в репликации и содержат полную копию всех сведений о каталоге для своего домена. Любые изменения данных каталога реплицируются в домене на все контроллеры домена

(и еще дохуя инфы которую вы можете прочесть в лк 5_2)

Active Directory имеет два типа групп:

- **Группы безопасности**. Используйте для назначения разрешений общим ресурсам.
- **Группы рассылки**: создание списков рассылки электронной почты.

Каждая группа имеет область, определяющий степень применения группы в дереве домена или лесу. Область группы определяет, где можно предоставить разрешения сети для группы.

Active Directory определяет следующие три области для групп:

• Универсальная • Глобальная • Локальная в домене

33. Механизмы безопасности в операционных системах семейства Linux.

В операционной системе GNU/Linux существуют следующие типы файлов:

- обычные файлы (-) все файлы с данными
- каталог (d) тип файла, данными которого является список имен других файлов и каталогов, вложенных в данный катало
- символьные ссылки (l) файл, в данных которого содержится адрес другого файла по его имени (а не индексному дескриптору).
- блочные устройства (b)
- символьные устройства (c) файлы устройств предназначены для обращения к аппаратному обеспечению компьютера (д
- сокеты (s)
- каналы (p)

Каждый тип имеет собственное обозначение одним символом.

В Linux у каждого файла и каждого каталога есть **два владельца:** *пользователь* и *группа*. владельцы устанавливаются при создании файла или каталога.
Чтобы увидеть текущие назначения владельца, вы можете использовать команду **ls-l.** Эта команда показывает пользователя и группу-владельца.
Команда **chown** позволяет сменить владельца (команда **chgrp** – только группу-владельца).

При создании файла система сохраняет идентификатор владельца (user) и идентификатор основной группы владельца (user).

```
user@UBUNTU1:~/mydocs$ ls -la
drwxrwxr-x  3 user user  4096 Oct  8 17:18 .
drwxr-xr-x 16 user user  4096 Oct  8 17:18 ..
-rw-rw-r--  1 user user  2451 Oct  8 17:18 ant.txt
drwxrwxr-x  2 user user  4096 Oct  8 17:18 birds
-rw-rw-r--  1 user user 12306 Oct  8 14:45 hare.txt
-rw-rw-r--  1 user user  2351 Oct  8 14:33 sheep.txt
-rw-rw-r--  1 user user  2476 Oct  8 14:30 wolf.txt
```

Значение разрешений для файлов:

- **r** – содержимое файла может быть прочитано
- **w** – содержимое файла может быть изменено
- **x** – файл может быть выполнен как команда

Значение разрешений для каталогов:

- **r** – содержимое каталога (имена файлов) может быть перечислено
- **w** – любой файл в каталоге может быть создан или удален
- **x** – к содержимому каталога возможен доступ (в зависимости от разрешений на файлы)

Управление разрешениями

Команда **chmod** позволяет изменять разрешения двумя способами:

- символьным chmod a+x cat.txt
- числовым chmod 750 animals

Некоторые опции:

-R, --recursive изменять файлы и директории рекурсивно
-v, --verbose выводить диагностику о каждом обрабатываемом файле
-c, --changes выводить диагностику только для изменяемых файлов

В операционной системе Linux базовые права для директории равны 0777 (rwxrwxrwx), а для файла 0666 (rw-rw-rw) По умолчанию umask 0002 используется для обычного пользователя. С этой маской права по умолчанию для директории равны 775, а для файла – 664.

атрибуты файла — это свойства метаданных, которые описывают поведение файла. Например, атрибут может указывать, сжат ли файл, или указывать, можно ли удалить файл. Вы можете просмотреть атрибуты файла с помощью команды **lsattr**: lsattr todo.txt.
Одно из распространенных применений **chattr** — установка неизменяемого флага для файла или каталога, чтобы пользователи не могли удалить или переименовать файл.

Управление свойствами файлов

В Linux каждый файл имеет довольно много свойств, например, права доступа устанавливаются трижды (для владельца, группы и всех прочих), метки времени также бывают трёх разных видов (время создание, доступа и изменения).
У каждого файла доступны следующие метки времени:

- Доступ
- Модифицирован
- Изменён
- Создан

Посмотреть метки времени папки можно также с помощью команды **stat**:
stat /путь/до/папки
С помощью команды **touch** можно изменить три метки времени файла или папки:
• время доступа • время модификации • время изменения статуса