1) Компоненты ОС

Планировщик процессов (process scheduler) определяет, когда и в течение какого периода времени процесс обрабатывается процессором.

Диспетчер памяти (memory manager) определяет, когда и каким образом память распределяется между процессами, и что следует предпринять, если основной памяти недостаточно.

Диспетчер ввода-вывода (input/output manager) обслуживает запросы ввода данных с аппаратных устройств и вывода на них.

Диспетчер межпроцессного взаимодействия (interprocess communication (ipc) manager) позволяет процессам взаимодействовать между собой.

Диспетчер файловой системы (file system manager) упорядочивает поименованные объекты (файлы) на запоминающих устройствах и предоставляет интерфейс для доступа к данным на этих устройствах.

Компоненты, которые выполняются независимо друг от друга, но решают свои задачи в общей области памяти, называются **потоками (thread)**.

Запрос к устройствам ввода-вывода обрабатывается **драйвером устройства (device driver)**, который является программным компонентом ОС.

Планировщик дисковых операций (disk scheduler) – отвечает за переупорядочение запросов дискового ввода/вывода для повышения производительности и сокращения времени ожидания доступа к диску.

2) Задачи ОС

Свойства операционных систем:

- Эффективность,
- Живучесть,
- Масштабируемость,
- Расширяемость,
- Мобильность,
- Защищенность,
- Интерактивность,
- Практичность.

1) Эффективная ОС обладает высокой производительностью и малым средним значением времени обработки запросов.

2) Живучая ОС – это отказоустойчивая и надежная система, не дающая сбоя в работе при ошибке отдельного приложения или компоненты аппаратуры.

3) Масштабируемая ОС способная использовать ресурсы по мере их наращивания.

4) Расширяемая ОС может адаптироваться к новым технологиям и обладает возможностью расширения для решения задач, изначально не предусмотренных при разработке данной ОС.

5) Мобильная ОС функционирует на различных конфигурациях аппаратных средств.

6) Защищенная (безопасная) ОС препятствует пользователям и программному обеспечению в получении несанкционированного доступа к услугам и ресурсам.

7) Интерактивная ОС позволяет приложениям быстро реагировать на действия пользователей и другие события в системе.

8) Практичная ОС – это система, способная удовлетворить широкому спектру пользовательских потребностей.

3) Монолитная и Многоуровневая ОС

1) Монолитная ОС (monolithic operation system) - Каждый компонент такой ОС включен в ядро и может непосредственно взаимодействовать с другими компонентами (просто выполняя вызов соответствующей функции). Ядро выполняется с неограниченными правами доступа к ресурсам. Такие ОС высокоэффективные. Однако вследствие того, что монолитные ядра объединяют все компоненты воедино, определение источника сбоев и ошибок затруднено. Также такие ОС уязвимы для ошибочного или вредительского кода.

2) Многоуровневый (layered) **подход** к построению операционных систем предложил архитектурное решение, основанное на организации компонентов, выполняющих сходные функции, в уровни. Каждый уровень системы взаимодействует с соседним, расположенным над или под ним. Более низкие уровни обслуживают более высокие, используя интерфейс, скрывающий их реализацию.

Многоуровневая ОС обладает более высокой степенью модульности. Модульность ОС обеспечивает простоту структурной организации и взаимодействия, как правило, упрощая проверку работоспособности, отладку и модифицирование. Поскольку для передачи данных с одного уровня на другой должны быть использованы промежуточные элементы, производительность системы снижается, в отличие от монолитного ядра. Так как все уровни наделены неограниченным доступом к ресурсам, многоуровневые ядра по-прежнему уязвимы для ошибочного или вредительского кода.

4) Архитектура ОС на основе микроядра. Сетевые и распределенные ОС

1) ОС на основе микроядра предоставляет лишь малый набор услуг: это необходимое условие сохранения небольших размеров ядра и обеспечения его масштабируемости. В перечень этих услуг входят низкоуровневое управление памятью, межпроцессное взаимодействие и базовые средства синхронизации для обеспечения совместного функционирования нескольких процессов. Часть компонентов выполняется вне ядра и с более низким уровнем привилегий.

Микроядра демонстрируют высокую степень модульности, что делает их расширяемыми, мобильными и масштабируемыми. Однако такая модульность достигается за счет повышения интенсивности межмодульного взаимодействия, что может привести к снижению производительности системы.

2) Сетевая ОС предоставляет собственным процессам доступ к ресурсам, которые расположены на других независимых компьютерах компьютерной сети. Структура многих сетевых и распределенных операционных систем основана на модели «клиент-сервер». Клиентские компьютеры такой сети запрашивают ресурсы посредством соответствующего сетевого протокола. Серверы удовлетворяют запросы ресурсов.

Сетевые файловые системы являются важным компонентом сетевых операционных систем. Ярким примером сетевой файловой системы является NFS.

3) Распределенная ОС – это единая ОС, управляющая ресурсами более чем одной компьютерной системы. Распределенные системы создают иллюзию объединения многочисленных компьютеров в один мощный, таким образом, что процесс может получить доступ к любому ресурсу системы независимо от его местоположения в компьютерной сети. Распределенные операционные системы сложны в реализации.

5) Аппаратные компоненты: Материнские платы.

Материнскую плату, главную печатную плату системы, можно назвать основой компьютера. Она оснащена расширительными гнездами (слотами), к которым подключаются другие компоненты, такие как процессоры, основная память и прочие аппаратные устройства. Эти гнезда обеспечивают электрический контакт между различными аппаратными компонентами и дают пользователю возможность изменять конфигурацию аппаратных средств компьютера, подключая и отключая устройства от гнезд.

Материнские платы содержат микрокопические электрические **проводники (trace).** Они обеспечивают передачу сигналов на материнской плате. Пучок проводников образует высокоскоростной канал связи, называемой **шиной (bus).**

Состав материнских плат:

1) BIOS (Basic Input/Output System) хранит инструкции, предназначенные для инициализации и управления аппаратными средствами. Также несет ответственность за загрузку в память начального блока ОС. Этот процесс называется **начальной загрузкой**.

2) Контроллеры, которые управляют процессами передачи данных по шинам платы.

3) Набор микросхем (chipset) материнской платы – это набор контроллеров, процессоров и других аппаратных средств, интегрированных на материнской плате, определяющий возможности аппаратуры системы.

Многие мощные аппаратные компоненты подключаются к гнезам (слотам) в виде навесных плат или плат расширения (add-on card). Но большинство современных материнских плат включают микросхемы, выполняющие обработку графической информации, передачу данных по сети, а также управление RAID-накопителями. Они расположенны на плате устройства, и их называют **внутренними устройствами (on-board device).**

6) Аппаратные компоненты: процессор

Процессор (processor) – это аппаратный компонент, который выпол-няет поток команд на машинном языке. В компьютере могут присутствовать процессоры различных типов, как, например, **центральный процессор** (central processing unit, CPU), **графический сопроцессор** (graphics coproces-sor), **цифровой процессор сигналов** (digital signal processor, DSP).

Действия, которые могут выполняться процессором, определяются его набором команд. Размер команды, или **длина команды** (instruction length), зависит от архитектуры. Архитектура процессора также определят объем данных, которые могут быть обработаны за один раз (*8-16-32-64 и т.д. бит*).

Основные компоненты процессора:

1) Блок выборки команд (instruction fetch unit) загружает команды в высокоскорост-ной блок памяти, называемый регистром команд, и процессор может быстро выполнить команду.

2) Дешифратор команд (instruction decode unit) преобра-зует команду и осуществляет ввод соответствующих данных **в операционный блок** для ее выполнения.

3) Арифметико-логическое устройство (АЛУ) выполняет основные арифметические и логические операции, такие как сложение, умножение и логическое сравнение.

Интерфейс шины позволяет процессору взаимодействовать с памя-тью и другими устройствами системы.

4) Также в состав входит высокоскоростная память, называемая **кэш-памятью**, которая хранит копии данных из основной памяти. Как правило имеет небольшую емкость из-за цены. Уровень L1 самый быстрый, однако наименьший по объему и дорогой. L2 и L3 уровни менее быстрые, дешевле и больше в объеме.

5) Регистры (register) – это высокоскоростные элементы памяти, распо-ложенные в процессоре и хранящие данные, непосредственно обрабатывае-мые процессором. Прежде чем процессор начнет обработку данных, они должны быть занесены в регистр. Набор регистров процессора определяется архитектурой, и каждый ре-гистр решает определенную задачу.

7) Аппаратные компоненты: системный таймер, иерархия памяти

1) Системный таймер

Машинное время, как правило, измеряется **циклами или тактами (cycles),** их еще называют тактами синхронизации. Цикл соответствует одному полному периоду электрического сигнала, генерируемого системным синхрогенератором. Синхрогенератор определяет частоту, с которой шины передают данные и которая измеряется циклами в секунду или герцами (Гц).

Процессоры и другие устройства используют **производные частоты (derived speed)**, умножая или разделяя частоту системной шины. Например, процессор с частотой 2 ГГц и частотой системной шины 200 МГц для генерирования внутреннего синхросигнала использует множитель 10.

2) Иерархия памяти.

Регистры – это самый быстрый и наиболее дорогой тип памяти в системе, они работают на одинаковой с процессором скорости. Скорость кэш памяти измеряется ее задержкой – интервалом времени, необходимым для передачи данных. Задержка, как правило, измеряется в наносекундах или циклах процессора.

Следующая ступень иерархии - основная память (main memory), которая также называется оперативной (real memory). характеризуется задержкой от десяти до сотни циклов процессора.

Регистры, кэш-память и основная память являются энергозависимой запоминающей средой (volatile media): информация, которая хранится в этой среде, теряется при выключении электропитания.

DVD, CD, HDD и ленточные накопители относятся к наименее дорогим и наиболее медленным устройствам хранения данных в компьютерных системах. Задержка HDD, измеряется в миллисекундах.

8) Аппаратные компоненты: основная память, прямой доступ к памяти

1) Основная память (random access memory, RAM)) реализуется в виде энергозависимой памяти с произвольной выборкой. Выражение «произвольная выборка» означает, что процессы могут обращаться к ячейкам с данными в любом порядке. Напротив, к ячейкам с данными **на последовательном запоминающем устройстве** приходится обращаться последовательно.

Динамическое ОЗУ (dynamic RAM), требует наличия схемы обновления (регенерации) для периодического чтения содержимого ячеек во избежание потери данных. Эта процедура не является обязательной для **статического ОЗУ (static RAM, SRAM)**, которое не нуждается в регенерации для сохранения данных.

Пропускная способность (bandwidth) – объем информации, который можно передать в единицу времени.

2) Прямой доступ к памяти (direct memory access, DMA) позволяет передавать блоки данных непосредственно с устройств ввода/вывода в основную память и обратно, что освобождает процессор для выполнения программных инструкций. Канал прямого доступа к памяти использует контроллер ввода/вывода для управления передачей данных между устройствами ввода/вывода и основной памятью. Для уведомления процессора контроллер ввода/вывода генерирует прерывание по завершении процесса обмена. Прямой доступ к памяти значительно повышает производительность тех систем, которые выполняют большое количество операций ввода/вывода.

9) Аппаратные компоненты: шины, периферийные устройства

1) Шина – это совокупность проводников, при помощи которых осуществляется передача информации между аппаратными компонентами компьютера. Как правило, шина состоит из шины данных (data bus), которая передает данные, и адресной шины (address bus), которая определяет получателя или источник этих данных.

Порт (port) - это шина, соединяющая только два устройства. Шина, которая совместно используется несколькими устройствами для выполнения операций ввода/вывода, также называется каналом ввода/вывода (Input/Output channel).

В каждый конкретный момент времени может осуществляться только одна операция обращения к определенному модулю памяти. Во избежание конфликта между двумя сигналами на шине, аппаратное устройство, называемое контроллером, определяет уровень привилегий каждого обращения к основной памяти, и, чаще всего, каналам предоставляется преимущество перед процессорами.

2) Периферийное устройство – это любое аппаратное устройство компьютера, которое не является обязательным для выполнения программных инструкций. К таким устройствам относится множество устройств ввода/вывода различных типов (например, принтеры, сканеры и мыши), сетевые устройства (сетевые интерфейсные карты и модемы), а также запоминающие устройства (компакт-диски, универсальные цифровые диски и жесткие диски). Те периферийные устройства, которые находятся внутри системного блока, считаются внутренними Остальные периферийные устройства – внешними. Мыши и клавиатуры являются примерами устройств посимвольного ввода (character device) - они передают данные по одному символу за сеанс обмена.

Последовательные порты (serial port) передают данные по одному биту. Параллельные порты (parallel port) передают данные по несколько битов за раз. Универсальная последовательная шина (universal serial bus, USB) являются популярным высокоскоростным последовательным интерфейсом. К современным интерфейсам относится интерфейс Serial ATA (SATA), также несколько беспроводных интерфейсов, включая Bluetooth и IEEE 802.11b/g/n/ac.

10) Поддержка функций операционных систем аппаратными средствами. Пользовательский режим ОС, режим ядра.

Операционные системы зачастую предусматривают несколько различных исполнительных режимов или режимов работы (execution mode).

Для пользовательских приложений, подмножество команд, которые может выполнять пользователь в **пользовательском режиме (user mode)**, исключает, например, прямое выполнение команд ввода/вывода.

Операционная система обычно работает в привилегированном режиме (также известном, как супервизорный режим или **режим ядра (kernel mode)**); она имеет доступ ко всем командам в наборе команд машины. При работе в режиме ядра процессор может выполнять привилегированные команды и обращаться к ресурсам для выполнения задач от имени процессов.

Наличие нескольких режимов позволяет организовать доступ **по принципу наименьшего уровня привилегий (principle of least privilege)** — каждый пользователь должен быть наделен минимальным достаточным уровнем привилегий и прав доступа, которые необходимы ему для выполнения его задач.

11) Поддержка функций операционных систем аппаратными средствами. Защита памяти, управление памятью.

Защита памяти (memory protection), позволяющая пресечь получение процессами доступа к памяти, не выделенной для этих процессов (как, например, к памяти других пользователей или к памяти операционной системы), реализуется посредством использования специальных регистров процессора, которые могут быть изменены только привилегированными командами. Процессор проверяет значения этих регистров для обеспечения невозможности получения процессами доступа к памяти, выделенной для других процессов. В системах, не использующих виртуальную память, система может помешать процессам в получении доступа к другим областям памяти посредством **ограничительных регистров (bounds register)**, которые определяют начало и конец области памяти, выделенной для процесса.

В состав большинства процессоров входят аппаратные средства, преобразующие виртуальные адреса, к которым обращаются процессы, в соответствующие адреса основной памяти. **Системы с виртуальной памятью** позволяют программам обращаться к адресам, которые необязательно должны относиться к ограниченному диапазону действительных (физических) ад-ресов (real address), доступному в основной памяти. Используя аппаратные средства, операционная система динамически преобразует во время рабочего цикла команды виртуальные адреса процессов в реальные адреса.

Виртуальная память также облегчает программирование в системах с разделением времени, так как процессам не нужно знать фактическое местоположение своих данных в основной памяти.

12) Поддержка функций операционных систем аппаратными средствами. Прерывания и исключения

Процессор сообщает операционной системе о таких событиях, как ошибки при выполнении программ и изменения в состоянии устройств (например, получение сетевого пакета или завершение выполнения дисковой операции ввода/вывода). Вместо механизма **опроса (polling)**, большая часть устройств посылает сигнал, называемый **прерыванием (interrupt)**, процессору, если происходит какое-либо событие. Операционная система может реагировать на изменения в состоянии устройств, извещая процессы, ожидающие эти события.

Исключения (exception) — это прерывания, которые генерируются в ответ на ошибки, как, например, сбой в работе аппаратных средств, логические ошибки. Процессор, как правило, обращается к операционной системе, чтобы та определила, каким образом необходимо отреагировать на ошибку. Если же все-таки ситуация приводит к аварийному завершению работы, операционная система может сделать это достаточно мягко, сводя к минимуму объем утраченных резуль-татов работы. Процессы могут регистрировать программы-обработчики осо-бых ситуаций в операционной системе. Когда операционная система полу-чает исключение соответствующего типа, она обращается к обработчику ис-ключительных ситуаций процесса, приведшего к исключению, для выполне-ния необходимых операций.

13) Поддержка функций операционных систем аппаратными средствами. Таймеры и часы. Начальная загрузка ОС. Plug-and-Play

1) Интервальный таймер (interval timer) периодически генерирует сиг-налы прерывания, которые заставляют процессор переключаться на опера-ционную систему. Последняя, как правило, использует интервальные таймеры, чтобы не допустить монополизации процессора отдельным процессом.

Часы истинного времени (time-of-day clock) позволяют компьютеру следить за временем «внешнего мира», как правило, с точностью до тысячных или миллионных долей секунды. Некоторые из ча-сов истинного времени питаются от специальной аккумуляторной батареи, что позволяет им работать, даже если компьютер отключен от внешнего источника питания.

2) Начальная загрузка ОС

При включении компьютера BIOS инициализирует аппаратуру системы, затем пытается загрузить в основную память команды из специальной области, которая находится на вторичном запоминающем устройстве (например HDD, CD, DVD) и называется **загрузочным сектором (boot sector)**. Данный процесс называется начальной загрузкой.

Процессору необходимо выполнить эти команды, которые загружают компоненты операционной системы в память, инициализируют регистры процессора и подготавливают систему к запуску пользовательских приложений.

Intel разработала расширенный интерфейс встроенного программного обеспечения (Extensible Firmware Interface, EFI) как замену системе БИОС. Данный интерфейс поддерживает диалоговую оболочку, при помощи которой пользователь может получить непосредственный доступ к устройствам компьютера.

3) Технология Plug-and-Play

Технология Plug-and-Play (PnP) позволяет операционной системе настраивать и использовать только что установленные аппаратные средства без участия пользователя.

Аппаратура, построенная по этой технологии, обладает следующими способностями:

- PnP-устройства могут однозначно определять себя в рамках операционной системы.
- PnP-устройства могут связываться с операционной системой для указания ресурсов и служб, необходимых для нормальной работы устройства.
- PnP-устройства могут определять необходимый драйвер и позволять операционной системе использовать его для настройки устройства.

ACPI — усовершенствованный интерфейс управления конфигурированием и энергопотреблением (Advanced Configuration and Power Inter-face) — определяет стандартный интерфейс операционной системы, используемый для настройки устройств и управления их энергопитанием.

14) Поддержка функций операционных систем аппаратными средствами. Кэширование и буферизация

1) Большинство систем выполняет кэширование посредством размещения в высокоскоростном запоминающем устройстве копии информации с медленных устройств, к которой впоследствии обращается процесс. Распоряжаться содержимым кэша, также называемого **элементами кэша (cache line)**, необходимо таким образом, чтобы свести к минимуму количество обращений к информации, не хранящейся в кэше. Такое обращение к кэшу называется неудачным обращением или **кэш-промахом (cache miss)**. Если информация, к которой об-ращаются, находится в кэше, то имеет место удачное обращение к кэшу, **кэш-попадание (cache hit)**. Управление кэшами осуществляется с исполь-зованием эвристики (heuristics), т.е. набора обобщающих практический опыт методов, которые приводят к хорошим результатам при относительно низких накладных затратах.

2) Буфер (buffer) — это область памяти для промежуточного хранения данных при выполнении операций передачи информации между устрой-ствами и процессами, которые работают на разных скоростях. Буферы повышают производительность системы, позволяя программному обеспечению и аппаратным устройствам обмениваться данными и запросами асинхронно (asynchronous transmission). Примерами буферов могут служить буферы накопителей на жестких магнитных дисках, буферы клавиатуры и принтеров.

Спулинг (spooling) — одновременное функционирование периферий-ных устройств в оперативном режиме, — это технология, при которой в роли посредника между процессом и низкоскоростным или ограниченным по раз-меру буфера устройством ввода/вывода выступает связующее устройство, например, дисковый накопитель. Спулинг позволяет про-цессам обращаться к устройству за выполнением операции даже в тот момент, когда устройство не готово к выполнению этого запроса.

15) Определение процесса. Состояние процессов

Во-первых, процесс — это логический объект. Каждый процесс обладает собственным адресным пространством, которое обычно состоит из области команд, области данных и области стека. В области команд находится программный код, выполняемый процессором. В области данных хранятся переменные, в нее входит память, динамически выделяемая процессу во время его выполнения. В области стека размещаются инструкции и локальные переменные, используемые при вызове процедур. Объем стека увеличивается при вызове вложенных процедур и уменьшается по их завершении.

Во-вторых, процессом называется «программа в стадии выполнения». Жизненный цикл процесса состоит из последовательности дискретных состояний.

Состояния: выполнение, если в данный момент ему выделен центральный процессор; готовность, если он мог бы сразу использовать процессор, предоставленный в его распоряжение; блокировка, если для того, чтобы продолжить работу, ему необходимо дождаться наступления определенного события. ОС создает список готовых (к выполнению) процессов, называемый также таблицей готовности, и список заблокированных процессов. Процессы размещены в списке готовых в порядке приоритета.

16) Управление процессом. Переход процессов из сост. в сост.

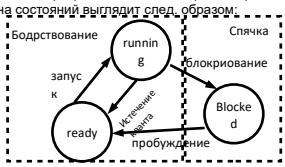
1. ОС попеременно выполняет несколько процессов, поэтому управление процессами должно быть организовано грамотно, чтобы при прерывании и возобновление работы того или иного процесса не возникали ошибки. Кроме того, процессы должны обладать возможностью взаимодействия с операционной системой для выполнения таких элементарных задач, как запуск нового процесса либо уведомления системы об окончании работы предыдущего.
2. Когда пользователь запускает определенную программу, создается процесс, помещаемый в таблицу готовности.

Смена состояний (state transition) — переход процесса из состояния готовности в со-стояние выполнения.

Предоставление центрального процессора первому процессу из списка готовых процессов называется диспетчеризацией (dispatching) или выбором процесса для выполнения. Диспетчеризация выполняется при помощи системной программы, называемой диспетчером

Чтобы предотвратить случайный либо умышленный монопольный захват ресурсов системы одним процессом, операционная си-стема задает в специальном аппаратном таймере прерываний (interrupting clock), называемом также интервальным таймером (interval timer), некоторое значение или квант времени (quantum), в течение которого данному процессу пользователя разрешено занимать процессор.

Смена состояний выглядит след. образом:



17) Блоки управления процессом и дескрипторы процесс.

Во время создания нового процесса ОС, как правило, выполняет целую последовательность операций.

Каждому процессу присваивается идентификационный номер процесса. Затем система создает блок управления процессом, называемый также дескриптором процесса, в который помещается информация, необходимая операционной системе для управления процессом.

В блоке управления процессом обычно содержится следующая информация:
1. PID; 2. текущее состояние процесса (выполняется, готов или блокирован); 3. Программный счетчик или счетчик команд— определяющий, какую по счету инструкцию программы процессор должен будет выполнить следующей; 4. приоритет процесса; 5. полномочия (данные, определяющие перечень ресурсов, к которым может иметь доступ данный процесс); 6. Указатель на родительский процесс, то есть процесс, создавший данный; указатели на дочерние процессы, то есть процессы, созданные данным процессом, если таковые имеются; 7. Указатели данных и инструкций процесса в памяти; 8. указатели выделенных процессу ресурсов (например, файлов).

Кроме того, в блоке управления процессом хранится содержимое регистров, называемое контекстом выполнения процессора, перед выходом из состояния выполнения.

Операционная система хранит указатели на блоки управления процессами в системной либо пользовательской таблице процессов (process table), чтобы ускорить доступ к нужной информации.

18) Операции над процессами

ОС должна иметь возможность выполнять определенные операции над процессами, в том числе: 1. создание процесса, 2. уничтожение процесса, 3. приостановка процесса, 4. возобновление процесса, 5. изменение приоритета процесса, 6. блокирование процесса, 7. пробуждение процесса, 8. запуск (выбор) процесса, 9. обеспечение взаимодействия процессов.

Процесс может порождать новые процессы, такой процесс называют родительским, а второй — дочерним. Каждый дочерний процесс имеет один родительский процесс, но каждый родительский процесс может иметь произвольное количество дочерних процессов В результате такого подхода формируется иерархическая структура процессов.

Уничтожение процесса подразумевает его полное удаление из системы. Занимаемая им память и ресурсы возвращаются в распоряжение системы, информация о нем стирается из всех системных списков или таблиц, удаляется блок управления процессом

Изменение приоритета процесса, как правило, подразумевает изменение значения приоритета в блоке управления процессом. В зависимости от реализованного механизма планирования процессов в операционной систе-ме, может понадобиться перемещение указателя на PCB в другую очередь приоритетов.

19) Приостановка и возобновление работы процессам

Приостановленные процессы на некоторое время вычеркиваются из списка процессов, соревнующихся за процессорное время, но при этом не удаляются из системы. Приостановка выполнения процесса используется для выявления угроз безопасности и в целях отладки нового программного обеспечения.

Инициатором приостановки может выступать либо данный процесс, либо другой процесс.

Процесс может сам приостановить свое выполнение, только тогда, когда он выполняется. При этом происходит переход процесса из состояния выполнения в состояние приостановлен готов. Процесс в состоянии приостановлен готов может быть переведен в состояние готовности, то есть возобновлен (resume) другим процессом. Заблокированный процесс может быть переведен в состояние приостановлен блокирован другим процессом. Процесс, находящийся в состоянии приостановлен блокирован, может быть активизирован другим процессом, в результате чего произойдет переход из состояния приостановлен блокирован в состояние блокировки.

20) Переключение контекста

ОС выполняет переключение контекста, чтобы завершить работу выполняемого процесса и запустить процесс, который ранее находился в состоянии готовности. Для переключения контекста ядро системы вначале должно сохранить контекст выполнения процесса, пребывающего в состоянии выполнения, в блоке управления процессом PCB, а затем загрузить контекст выполнения намеченного к выполнению процесса, находящегося в состоянии готовности, из его PCB.

ОС очень часто обращается к блокам управления процессом PCB. Поэтому большинство процессоров имеет встроенные регистры, ссылающиеся на PCB выполняющегося процесса, ускоряя переключение контекста. Когда ОС инициирует переключение контекста, процессор сохраняет контекст выполнения текущего процесса в PCB. Благодаря этому предотвращается перезапись значений регистров процессора со стороны операционной системы либо других процессов. Кроме того, процессор упрощает и ускоряет переключение контекста, используя специальные инструкции сохранения и восстановления контекста выполнения процесса в или из блока управления процессом PCB, соответственно.

21) Прерывания. Обработка прерываний

Для обработки прерываний в составе операционной системы предусмотрены специальные программы, называемые **обработчиками прерыва-ний** (interrupt handlers). **Прерывания** — это наименее ресурсоемкий способ привлечь внимание процессора. Синхронные прерывания имеют место при попытках процесса выполнить запрещенные операции. Аппаратура формирует асинхронные прерывания, чтобы уведомить процессор об изменении своего состояния.

Обработка прерываний.

1) Линия, по которой передается прерывание, — электрическое соединение между материнской платой и процессором — становится активной. Разнообразные устройства посылают сигналы, которые делают активной линию прерываний.

2) Обнаружив активизацию линии прерываний процессор завершает выполнение текущих инструкций, после чего приостанавливает выполнение текущего процесса.

3) Далее процессор передает управление соответствующему обработчику прерываний. Каждому типу прерываний присваивается уникальное значение, используемое процессором в качестве индекса вектора прерываний (interrupt vector).

4) Обработчик прерываний выполняет соответствующие операции, основываясь на типе прерываний.

5) По завершении работы обработчика прерываний, состояние прерванного процесса (либо некоего «следующего» процесса, если ядро инициировало переключение контекста) восстанавливается.

6) Прерванный процесс (либо некий «следующий» процесс) продолжает выполнение. За определение того, какой процесс выполняется в данный момент (прерванный либо некий «следующий» процесс), ответственность возлагается на операционную систему.

22) Классы прерываний

Перечень прерываний, поддерживаемых системой, зависит от ее архитектуры. В спецификации IA-32 различаются два вида сигналов: прерывания и исключения. Прерывания уведомляют процессор о наступлении того или иного события, или об изменении статуса внешнего
устройства (завершении операции ввода/вывода). В архитектуре IA-32 предусмотрены программные прерывания, которые используются процессами для вызова системных функций.

Типы прерываний:

1) Ввод/вывод - Сигналы прерываний данного типа формируются оборудованием ввода/вывода. Они уведомляют процессор об изменении состояния канала либо устройства.

2) Таймер - в системе могут присутствовать устройства, генерирующие прерывания через определенные промежутки времени. С помощью таймера ОС определяет момент истечения выделенного процесс у кванта.

3) Межпроцессорное - Такие прерывания позволяют одному процессору отсылать сообщения другому процессору в многопроцессорной системе.

Классы исключений:

1) Промах - Деление на ноль, неверный формат данных, попытки выполнения запрещенных операций, попытки обратиться по адресу, выходящему за пределы физического адресного пространства, попытки пользовательского процесса выполнить привилегированные инструкции и попытки обращения к защищенным ресурсам.

2) Ловушка - Ошибки переполнения, либо достижением контрольной точки останова программы.

3) Аварийное завершение - Наступает в случае обнаружения процессором ошибки, после которой процесс восстановиться не в состоянии. Если процедура обработки исключений сама выдаст исключение, процессор не сможет справиться с двумя ошибками сразу. Это исключение двойного промаха.

Быстрое реагирование на прерывания и быстрый возврат управления прерванному процессу важен для повышения эффективности использования ресурсов и достижения высокого уровня интерактивности. Поэтому большинство процессоров позволяют ядру запрещать (disable) либо маскировать (mask) определенные типы прерываний. Процессор просто игнорирует прерывания данного типа либо сохраняет их в очереди отложенных прерываний,

23) Взаимодействие процессов: сигналы

Сигналами называют программные прерывания, уведомляющие процесс о наступлении определенного события.

Системные сигналы зависят от операционной системы и типов программных прерываний, поддерживаемых определенным процессором. При поступлении сигнала ОС сначала определяет, кому предназначен данный сигнал, а потом — как процесс должен на него отреагировать.

Процесс перехватывает сигнал и определяет процедуру, вызываемую операционной системой в случае поступления сигнала. Процесс может проигнорировать сигнал, то есть переложить ответственность за выполнение действия по умолчанию по обработке сигнала на операционную систему. Чаще всего по умолчанию задается аварийное завершение процесса. Другой операцией по умолчанию является дамп памяти, которая аналогична аварийному завершению процесса, только перед завершением генерируется файл ядра процесса, содержащий контекст выполнения процесса и данные из адресного пространства, что облегчает отладку программ. Третьим вариантом действия по умолчанию является игнорирование сигнала.

Процесс может заблокировать обработку сигнала путем его маскирования. Когда процесс маскирует сигнал определенного типа (сигнал приостановки), ОС блокирует сигналы этого типа до тех пор, пока маскирование сигнала не будет отключено.

24) Взаимодействие процессов: передача сообщений

Сообщения могут передаваться в одном направлении — тогда для любого сообщения один процесс является отправителем, а другой — получателем. Передача сообщений может также являться двунаправленной. Прием и отправка сообщений обычно реализуется в виде вызова системных функций.

В случае **блокирующей передачи** процесс вынужден ожидать, пока сообщение не будет доставлено получателю, требуя подтверждения приема.

При **неблокирующей передаче** процесс-отправитель может продолжить выполнение других операций, даже если сообщение еще не было доставлено получателю.

Блокирующая передача представляет собой пример синхронной связи, тогда как неблокирующая передача — асинхронной. Во время отправки сообщения можно указать процесс получатель либо опустить имя процесса, в таком случае будет произведена ***широковещательная*** **передача сообщения** всем процессам системы.

Реализацией механизма передачи сообщений является *канал* — защищенная операционной системой область памяти, которая выступает в качестве буфера, позволяющего нескольким процессам обмениваться между собой данными. ОС синхронизирует доступ к буферу — после того, как записывающий процесс закончит вести запись в буфер (вероятно, заполнив его), система приостановит работу записывающего процесса, позволив считывающему процессу начать чтение данных из буфера. По мере считывания данных из буфера, канал освобождается.
Отправители и получатели часто взаимодействуют между собой при помощи протокола квитирования используемого для подтверждения факта приема информации.

Механизм тайм-аута применяется для ограничения времени ожидания уведомления о доставке. Если уведомление о доставке сообщения не поступит по истечении заданного интервала, сообщение будет отправлено повторно.

25) Определение потока. Мотивы использования потоков

Поток, для обозначения которого иногда используется термин *легковесный* процесс, обладает доступом к большинству атрибутов базового процесса. Расписание выполнения потоков задается процессором, при этом каждый поток может выполнять свой набор инструкций независимо от других процессов и потоков. Как правило, потоки являются частью традиционных процессов, называемых тяжеловесными.

Термин «поток» обозначает определенный поток инструкций либо управляющий поток. Потоки одного процесса могут выполняться параллельно и взаимодействовать друг с другом для решения общих задач. В многопроцессорной системе одновременно может выполняться несколько потоков.

Потоки владеют подмножеством ресурсов процесса.

Управление потоками может осуществлять операционная система либо создавшее их пользовательское приложение.

Мотивы использования потоков:

1. Архитектура ПО. За счет модульности и структуры современных компиляторов, большинство современных приложений содержит фрагменты кода, которые могут выполняться независимо друг от друга.

2. Производительность. Проблема однопоточных приложений состоит в том, что независимые операции не могут выполняться на разных процессорах. В многопоточном приложении потоки могут обладать общим доступом к процессору (либо группе процессоров), в результате чего несколько задач могут выполняться параллельно.

3. Взаимодействие. Большинство приложений с целью синхронизации и обмена информацией друг с другом используют независимые компоненты.

26) Состояние потока: жизненный цикл потока

Каждый процесс содержит один управляющий поток и каждый такой поток проходит через серию дискретных состояний. Если процесс содержит несколько управляющих потоков, можно рассматривать каждый поток, как переходящий между дискретными **состояниями потока.**

Рассмотрим, в качестве примера следующую группу состояний, базируясь на реализации потоков в языке Java. В Java новый поток начинает свой жизненный цикл с состояния рождения (born).

В этом состоянии он пребывает до тех пор, пока программа не запустит поток, переведя его в состояние готовности (ready).

Поток в состоянии готовности с наивысшим приоритетом входит в состояние выполнения (running), как только процессор окажется в его распоряжении.

Выполняющийся поток входит в тупиковое состояние, называемое также состоянием смерти (dead) по окончанию выполнения всех инструкций либо в результате завершения потока по иным причинам.

Поток переходит в состояние блокировки (blocked) в тех случаях, когда он вынужден ожидать завершения операции ввода/вывода.

Когда процесс ожидает определенного события, говорят, что он находится в состоянии ожидания (waiting).

Поток возвращается в со-стояние готовности, когда о наступлении ожидаемого события его уведомляет (notifies) или будит (awakens) другой поток.

Поток в состоянии выполнения может войти в состояние сна (sleep) на определенный промежуток времени, называемый интервалом сна (sleep interval).

27) Операции над потоками

ОС может выполнять над потоками и процессами операции: 1. создание, 2. завершение, 3.приостановка, 4. возобновление, 5. засыпание, 6. пробуждение.

Механизм создания потока во многом напоминает создание процесса. Когда процесс порождает поток, библиотека по работе с потоками инициализирует специфические для потока структуры данных, где хранится содержимое регистров, программный счетчик и ID потока. Во время создания потока не требуется вмешательство операционной системы для инициализации ресурсов.

ОС может выполнять над потоками ряд операций, которые не имеют эквивалента среди операций над процессами:

1. отмена — поток или процесс могут досрочно завершить выполнение другого потока путем его отмены.

2. присоединение — в некоторых операционных системах при инициализации процесса создается так называемый первичный поток. Работа первичного потока ничем не отличается от любого другого потока, за тем лишь исключением, что при возврате из первичного потока, завершается выполнение всего процесса. Чтобы не допустить завершения процесса до окончания выполнения всех остальных потоков, первичный поток, как правило, переходит в состояние сна до тех пор, пока не закончат свою работу все созданные им потоки

28) Модели потока: потоки уровня пользователя

Способы реализации потоков в различных операционных системах могут различаться, но практически везде поддерживается одна из трех моделей потока: потоки уровня пользователя, потоки уровня ядра и комбинированная модель.

Каждый многопоточный процесс сам отвечал за хранение информации о состоянии процесса, задавая расписание выполнения потоков и обеспечивая примитивы для их синхронизации. Эти потоки, названные потоками уровня пользователя, подразумевают выполнение операций над потоками в пространстве пользователя, что означает создание потоков библиотеками времени выполнения, которые не могут выполнять привилегированные инструкции либо напрямую обращаться к примитивам ядра. Но при этом, ОС запускает многопоточный процесс целиком, не имея возможности выбирать для выполнения отдельные его потоки. Поэтому, говорят, что потоки уровня пользователя связаны **отношением многих-к-одному**.

Преимущества реализации потоков в пространстве пользователя:

1) Потоки пользовательского уровня не нуждаются в поддержке со стороны операционной системы.

2) Поскольку расписание выполнения потоков контролирует библиотека работы с потоками, а не ОС, разработчики приложений получают возможность более тонкой настройки алгоритма планирования в библиотеке работы с потоками под нужды конкретного приложения.

3) Кроме того, потоки уровня пользователя не требуют обращения к ядру для принятия решений по поводу планирования либо для выполнения процедуры синхронизации.

29) Модели потока: потоки уровня ядра.

Потоки уровня ядра позволяют обойти ограничения потоков уровня пользователя, связывая каждый поток с отдельным контекстом выполнения. Потоки уровня ядра связаны **отношением один-к-одному**, т.е. каждому пользовательскому потоку ставится в соответствие поток ядра, запуск которого может выполнять операционная система. Потоки ядра разделяют общее адресное пространство процесса, каждый поток ядра хранит специфические данные потока, такие как содержимое регистров и идентификатор потока.

Преимущества потоков, связанных отношением 1-к-1:

1) Ядро может одновременно запускать на разных процессорах потоки одного процесса, что позволяет повысить производительность приложений, предназначенных для параллельного выполнения.

2) Ядро в состоянии управлять каждым потоком по отдельности, то есть система может запустить поток, находящийся в состоянии готовности, даже если другие потоки данного процесса заблокированы. Это позволяет повысить интерактивность приложений, которые должны реагировать на вводимые пользователем данные, а также общее быстродействие в тех случаях, когда приложение может выиграть от параллельного выполнения фрагментов кода.

Процесс, использующий потоки уровня ядра, может менять присвоенные операционной системой приоритеты потоков, задавая приоритет планирования для каждого своего потока.

Программное обеспечение, использующее потоки уровня ядра является менее переносимым. Разработчики приложений, использующие потоки уровня ядра, вынуждены переделывать свои приложения для работы с API потоков на каждой операционной системе.

Еще одним недостатком потоков уровня ядра является потребление большего количества ресурсов по сравнению с потоками уровня пользователя.

30) Модели потока: совместное использование потоков разных уровней.

В гибридной модели потоки связываются отношением многих-ко-многим. Как предполагает данное название, в этой модели нескольким потокам уровня пользователя ставится в соответствие группа потоков ядра. Количество потоков уровня пользователя и ядра может не совпадать.

Накладные расходы для потоков, связанных отношением многих-ко-многим, уменьшаются (по сравнению с один-к-одному) благодаря использованию **пула потоков**. Данная технология позволяет приложению задавать количество требуемых потоков уровня ядра.

Применение пула потоков позволяет существенно уменьшить количество ресурсоёмких операций по созданию и уничтожению потоков.

Потоки, которые существуют постоянно, называются **рабочими потоками**, поскольку они могут выполнять самые разнообразные функции, в зависимости от того, какие пользовательские потоки связаны с ними.

Преимущество потоков, связанных отношением многих-к-одному заключается в том, что приложения, в состав которых они входят, позволяют повышать производительность библиотеки работы с потоками за счет настройки алгоритма планирования.

Активацию планировщика выполняет поток ядра, уведомляющий библиотеку работы с потоками уровня пользователя о наступлении определенных событий (например, о блокировании потока либо освобождении процессора). Данный тип потоков ядра носит название «потоков активации планировщика», поскольку библиотека работы с потоками уровня пользователя может выполнять операции планирования выполнения потоков только после «активации» (прихода уведомления о появлении события), — иногда данную методику называют upcall-механизмом.

Главным ограничением модели потоков многих-ко-многим является усложнение архитектуры операционной системы, а также отсутствие стандартного способа реализации данной модели потоков.

31) Доставка сигналов потока.

Сигналы прерывают выполнение процессов, и при этом вырабатываются программным обеспечением - операционной системой либо пользовательским процессом. Сигналы превратились в стандартный механизм взаимодействия процессов.

Когда операционная система передает сигнал процессу, тот, реагируя на него, приостанавливает свое выполнение и вызывает обработчик сигналов. Когда обработчик сигналов закончит работу, процесс возобновляет свое выполнение.

Существует два типа сигналов: синхронные и асинхронные. **Синхронный сигнал** возникает непосредственно в результате выполнения команд процессом или потоком. **Асинхронный сигнал** формируется в результате появления событий, не связанных с выполнением текущих инструкций, поэтому в нем обязательно должен задаваться ID процесса, чтобы система имела возможность определить получателя сигнала. Асинхронные сигналы используются для уведомления процесса о завершении операции ввода/вывода, приостановке выполнения процесса, продолжении выполнения процесса, либо уведомления о том, что процесс должен завершить свое выполнение.

Если каждый процесс системы состоит из одного управляющего потока, доставка сигналов осуществляется след. образом. Если сигнал синхронный, он доставляется тому процессу, который в данный момент времени выполняется на процессоре, инициировавшем этот сигнал. Если же сигнал является асинхронным, операционная система сможет доставить сигнал целевому процессу, только если в данный момент времени он выполняется, в противном случае операционная система добавит сигнал в очередь **отложенных сигналов**, чтобы осуществить его доставку после того, как получатель перейдет в состояние выполнения.

Вариант с многопоточным процессом. Если сигнал синхронный, он доставляется тому потоку, который в текущий момент времени выполняется процессором, выступившим в роли инициатора сигнала. Если же сигнал является асинхронным, операционная система должна каким-то образом идентифицировать получателя. Один из вариантов решения предполагает задание ID потока отправителем.

Маскирование сигнала позволяет потоку запретить прием сигналов определенного типа. Поток может запретить прием всех сигналов кроме тех, которые его интересуют. Маскирование сигналов позволяет процессу распределить обработку сигналов между потоками.

32) Завершение работы потоков.

Когда процесс завершает работу обычным образом (например, путем вызова системной функции exit), операционная система немедленно удаляет поток из системы. Выполнение потока может завершиться досрочно в результате исключения (например, при обращении к запрещенной области памяти) либо по приходу сигнала отмены от процесса или потока. Поскольку потоки взаимодействуют между собой, внося изменения в разделяемые данные, приложение может выдать незаметные на первый взгляд ошибочные результаты в случае непредвиденного завершения одного из потоков. Поэтому библиотека работы с потоками должна очень внимательно следить за тем, когда и каким образом можно будет удалить поток из системы. Поток может запретить прием сигналов отмены с помощью маскирования. Обычно, он делает это только при выполнении фрагментов программного кода, которые нельзя прерывать до полного завершения.

33) Файловые системы: иерархия данных.

Информация хранится в компьютерах в соответствии с определенной иерархией данных. Самый нижний уровень этой иерархии состоит из битов. Сочетания битов, составляющие двоичные коды, используются для представления всех данных в компьютерных системах. В последовательности из n битов можно хранить 2^n различных сочетаний битов.

Следующий уровень иерархии — это последовательности битов фиксированной длины, например, байты , символы и слова. Байт в применении к устройствам хранения обычно состоит из восьми битов.

Слово — это цепочка битов, которую процессор может обработать одновременно.

Символы — это значения байтов, используемые для представления цифр, букв, знаков пунктуации и специальных знаков. Во многих системах символ состоит из 8 битов.

Поле — это группа символов. Запись — это набор полей. Файл — это группа взаимосвязанных записей.

Самый высокий уровень иерархии —это файловая система или база данных. Файловые системы — это наборы файлов, а базы данных — это наборы данных.

Термин **том(volume)** обозначает часть накопителя, в которой может размещаться множество файлов. Физический том ограничивается в размере емкостью одного накопителя, а логический — например, используемый виртуальной машине — может быть распределен по множеству физических накопителей.

34) Файлы. Определение. Операции над файлами.

Файл представляет собой именованный набор данных, над которым можно выполнять следующие операции:

- Открывать — подготавливать файл к обращениям.
- Закрывать — блокировать дальнейшие обращения к файлу до нового открытия.
- Создавать.
- Уничтожать.
- Копировать.
- Переименовывать
- Отображать — выводить содержимое файла на экран или распечатывать.

Отдельные элементы данных, хранящиеся в файлах, могут подвергаться следующим операциям:

- Читать— копировать данные из файла в память процесса.
- Записывать — копировать данные из памяти процесса в файл.
- Обновлять — изменять содержимое существующего элемента данных в файле.
- Вставлять — добавлять в файл новый элемент данных.
- Удалять — удалять элемент данных из файла.

Файлы могут характеризоваться следующими свойствами:

- Размером
- Расположением — местом, где хранится файл
- Режимом доступа — ограничениями на доступ к файлу.
- Типом — назначением файла. Например, исполняемый файл содержит исполняемые инструкции для процесса.
- Изменчивостью — частотой внесения изменений в данные, хранящиеся в файле.
- Активностью — процентом записей в файле, к которым выполняются обращения в течение заданного периода времени.

Файлы могут состоять из одной или более записей. Физическая запись или физический блок — это единица информации, считываемая с накопителя или записываемая на него. Логическая запись , или логический блок — это набор данных, воспринимаемый как единица программами. Если физическая запись содержит ровно одну логическую, то говорят, что файл состоит из неблокированных записей. Если физическая запись содержит несколько логических, то говорят, что файл состоит из сблокированных(блок) записей. В файле с фиксированной длиной записи все записи имеют одинаковую длину; в файле с записями произвольной длины записи могут иметь любую длину вплоть до размера блока.

35) Файловые системы. Основные понятия.

Файловая система организует файлы и управляет доступом к хранящимся в них данным.

Файловая система отвечает за:

- Управление файлами — реализует механизмы хранения файлов, обращения к ним, их разделения и обеспечения их безопасности.
- Управление вспомогательными устройствами хранения — выделяет пространство под файлы на вторичных и третичных устройствах хранения.
- Целостность файлов — гарантирует, что хранимая в файлах информация не будет повреждена.
- Методы доступа — методы, позволяющие получить доступ к хранимым данным.

Файловая система в основном занимается управлением вторичными устройствами хранения, особенно дисковыми накопителями, но она может обращаться и к данным в файлах, хранящихся на других носителях. Файловые системы должны предоставлять возможность структурирования файлов способами, удобными для приложений и переноса данных между файлами. Механизм разделения файлов должен реализовать различные типы контролируемого доступа — например, доступ для чтения , доступ для записи, доступ для выполнения и их сочетания.

Физические имена указывают, где расположен файл. Символьные имена дают системе возможность реализовать для пользователей логическое представление их данных, присваивая осмысленные имена файлам и операциям над файлами.

Чтобы предотвратить как случайную потерю данных, так и их намеренное уничтожение, система должна располагать возможностями резервного копирования, и возможностями восстановления.

Файловые системы могут также предоставлять возможности шифрования (encryption) и дешифрования (decryption) данных.

36) Файловые системы. Директории. Примеры.

Чтобы организованно размещать и быстро находить файлы, файловые системы используют директории . Это файлы, содержащие списки имен файлов и информацию об их размещении в системе. В отличие от других файлов, директории не хранят пользовательские данные.

Ниже перечислены типичные поля файла директории.

1) Имя — символьная строка, содержащая имя файла
2) Местоположение — физический блок или логический адрес файла в файловой системе.
3) Размер- количество байтов, занимаемые файлов.
4) Тип — описание назначения файла (файл данных или файл директории)
5) Время обращения — время посл. Обращениям
6) Время изменения - время последнего изменения файла.
7) Время создания - время создания файла.

37) Файловые системы. Одноуровневые и иерархически структурированные файловые системы.

1) Простейшая организация файловой системы — это **одноуровневая** или плоская файловая система. В таких системах все файлы хранятся в одной директории. В одноуровневых системах имя каждого файла должно быть уникальным. Одноуровневые системы применяются редко.

2) Для большинства сред лучше подходит иерархическая файловая система, организованная следующим образом.

Корень отмечает, в какой позиции устройства хранения начинается корневая директория. Директории — это файлы, которые могут указывать на другие файлы и директории. Корневая директория указывает на различные пользовательские директории.

Пользовательская директория содержит запись о каждом находящемся в ней пользовательском файле; каждая такая запись указывает позицию, в которой соответствующий файл находится на устройстве хранения. Имена файлов должны быть уникальными только в пределах одной и той же пользовательской директории. В таких иерархически структурированных файловых системах каждая директория может иметь несколько поддиректорий, но только одну родительскую директорию. Имя файла обычно формируется как путь к этому файлу из корневой директории.

Например, в двухуровневой файловой системе с пользователями Алешка, Ибрагим и Василий, в которой у пользователя Ибрагим есть файлы Rock и Lab, путь к файлу Rock можно записать в виде ROOT:Ibragim:Rock. В этом примере ROOT обозначает корневую директорию, а символ двоеточия (:) разделяет части пути. В windows представление было бы такое C:\Ibragim\Rock.

38) Файловые системы: относительные и абсолютные пути.

Многие системы поддерживают концепцию **рабочей директории**, чтобы упростить навигацию с применением путей. Рабочая директория (обозначаемая символом точки «.» в Windows и UNIX-системах) дает возможность пользователям задавать пути, не начинающиеся из корневой директории. Предположим, что текущей рабочей директорией выбрана home/user/ в файловой системе. Относительный путь к директории /home/user/any/music будет выглядеть как any/music. Такой прием позволяет сокращать пути при обращениях к файлам. Если файловая система встречает относительный путь, она формирует абсолютный путь , то есть путь, начинающийся из корня, выполняя рабочей директории и относительного пути. Затем система обращается к требуемому файлу по абсолютному пути. Файловая система обычно хранит ссылку на родительскую директорию рабочей директории, то есть директорию уровнем выше рабочей в системной иерархии.

39) Файловые системы: ссылки. Метаданные.

1) Ссылка — это запись в директории, ссылающаяся на файл данных или директорию, обычно размещенные в другой директории.

Мягкая ссылка, также называемая символьной ссылкой в UNIX-системах, ярлыком в системах Windows и псевдонимом в системах MacOS — это запись в директории, содержащая путь к файлу. Файловая система находит файл, на который указывает ссылка, перемещаясь в системе директорий по указанному в ссылке пути. Поскольку в мягких ссылках хранятся логические позиции файлов в файловых системах, то если пользователь перемещает файл в другую директорию или переименовывает его, мягкие ссылки станут некорректными.

Жесткая ссылка — это запись в директории, указывающая позицию файла в устройстве хранения. Файловая система находит данные из указываемого ссылкой файла, прямо обращаясь к заданному физическому блоку. Во время операций (напр. дефрагментация) размещение файлов может изменяться, и файловая система должна будет обновлять записи о файлах в директориях. Система может хранить в файле указатель на каждую жесткую ссылку на этот файл. Если размещение файла изменяется, то по указателям можно найти и обновить все жесткие ссылки.

Если пользователь уничтожает ссылку на файл, файловая система должна определить, уничтожать ли сам файл. Для этого системы обычно ведут учет жестких ссылок на файлы. Если на какой-то файл не указывает ни одна жесткая ссылка, и этот файл можно удалять. Мягкие ссылки не учитываются при таких проверках.

2) В файловых систем хранятся не только данные и директории, но и специальные данные, например, о свободных блоках накопителей , о времени последнего изменения файлов. Эта информация, называемая **метаданными**, обеспечивает целостность файловой системы и не может быть непосредственно изменена пользователем. Многие файловые системы также создают суперблок, в котором хранится информация, обеспечивающая целостность файловой системы.

40) Файловые системы: дескрипторы файлов. Монтирование

1) Чтобы избежать многократных просмотров директорий, система хранит в оперативной памяти таблицу, ведущую учет открытых файлов. Во многих системах операция открытия файла возвращает **дескриптор файла**, неотрицательное целое число, являющееся индексом для таблицы открытых файлов. Таблица открытых файлов часто содержит блоки управления файлами. В этих блоках содержится информация, нужная системе для управления файлом, иногда называемая атрибутами файла. Файловые атрибуты сильно отличаются в разных системах.

В блоке управления файлом может содержаться:

- Символьное имя файла;
- Данные о его расположении на накопителе;
- Организационная структура (например, файл последовательного доступа или произвольного доступа);
- Данные управления доступом (например, о том, какие пользователи могут обращаться к файлу и какие они могут выполнять операции);
- Данные о типе файла.
- Характер файла.
- Дата и время создания файла;

2) Монтирование. В файловых системах предусматривается возможность монтирования множества файловых систем. Монтирование объединяет множество систем в общее пространство имен — набор файлов, к которым может обратиться одна файловая система. Объединенное пространство имен предоставляет пользователям доступ к данным из разных источников, как если бы все файлы находились в родной файловой системе. Команда монтирования назначает в родной файловой системе директорию, называемую точкой монтирования.

Файловые системы управляют директориями монтирования с помощью **таблиц монтирования**. В этих таблицах содержится информация о путях к директориям, используемым как точки монтирования, и устройствах, на которых хранятся монтируемые файловые системы.

41) Размещение файлов: непрерывное размещение файлов

Файловые системы, реализующие непрерывное размещение файлов, выделяют под файлы непрерывные участки адресов в пространствах устройств хранения. Пользователь заранее указывает размер области, которую нужно выделить под файл. Если непрерывную область заданного размера выделить невозможно, файл создан не будет.

Преимущество непрерывного размещения в том, что следующие друг за другом логические записи оказываются размещенными рядом и физически. Скорость доступа к ним больше, и не требует дополнительных операций позиционирования.

Недостаток систем с непрерывным размещением файлов в том, что у них проблемы с внешней фрагментацией. Системы с непрерывным размещением файлов могут обладать низкой производительностью, если размер файлов будет изменяться с течением времени. Если размер файла превышает изначально заданный, то файл должен быть целиком перенесен на новый участок подходящего размера. На это требуются дополнительные операции ввода/вывода.

42) Размещение файлов: размещение файлов в виде связных списков.

Использование связного списка секторов — это один из способов реализации фрагментированного размещения Каждая запись в директории указывает на первый сектор, занятый файлом на накопителе с подвижными головками. Раздел данных сектора содержит данные файла; раздел указателя содержит указатель на следующий сектор, занятый файлом. Поскольку файлы могут занимать множество секторов, головка чтения/записи должна последовательно обращаться к каждому сектору файла, пока не найдет запрошенную запись.

Минусы: Поскольку записи из файла могут быть рассеяны по диску,прямой и последовательный доступ к логическим записям может потребовать множества операций позиционирования помимо первой, необходимой для чтения начального сектора файла. Необходимость хранения указателей в списке уменьшает объем, доступный для хранения собственно данных в каждом секторе.

Схема с блочным размещением - вместо выделения отдельных секторов выделяются непрерывные последовательности секторов — блоки. Система пытается выделять под файлы новые блоки, выбирая блоки максимально близкие к существующим, предпочтительно на том же цилиндре. Каждое обращение к файлу требует определения нужного блока и сектора в блоке.

При сцеплении блоков каждая запись в директории указывает на первый блок в файле. Блоки, принадлежащие к одному файлу, состоят из двух разделов: раздела данных и раздела указателя на следующий блок файла. Поиск нужной записи в файле потребует просмотра цепочки блоков вплоть до блока, содержащего нужную запись. Цепочку блоков нужно просматривать с начала, процесс поиска может быть медленным из-за перемещений от блока к блоку. Операции вставки и удаления сводятся к изменению указателей в блоках. В некоторых системах для ускорения поиска используются двухсвязные списки.

43) Размещение файлов: табличное фрагментированное размещение.

Файловая система **хранит указатели на блоки файлов**, чтобы уменьшить потребность в операциях позиционирования для доступа к нужным данным. **Записи в директориях указывают на первые блоки файлов**. **Номер блока** в таблице выделения блоков используется **в качестве индекса**, по которому **определяется следующий блок файла**. **Поскольку указатели хранятся централизованно**, **их таблицу можно загрузить в кэш и быстро определять цепочки блоков**, в которых хранятся файлы, соответственно уменьшая время доступа.

Чтобы уменьшить время доступа, эта **таблица должна храниться на диске в непрерывном блоке и кэшироваться в оперативной памяти.** Если блоки файла рассеяны по диску, то и записи в таблице об этих блоках тоже рассеяны по таблице. Поэтому системе может понадобиться загрузить в память несколько блоков таблицы, что может сильно ухудшить время реагирования и кэширование такой большой таблицы может занять существенную часть оперативной памяти.

44) Размещение файлов: индексированное фрагментированное размещение.

У каждого **файла есть один или более индексных блоков**. **Индексный блок содержит список указателей**, **указывающих на блоки с данными из файла.** Запись о файле в его директории **содержит указатель на индексный блок** этого файла. **Чтобы найти требуемую запись**, файл. *сис.* просматривает директории и находит на диске индексный блок файла. Затем она загружает индексный блок в память и использует содержащиеся в нем указатели, чтобы определить физический адрес нужного блока с данными. В большинстве реализаций индексных блоков **последние несколько записей в них резервируются для хранения указателей на другие индексные блоки**. Этот прием называется **сцеплением. Основное преимущество сцепления** - поиск выполняется собственно в индексных блоках.

Чтобы **ускорить просмотр файлов**, индексные блоки обычно кэшируются в оперативной памяти. **Файловые системы обычно размещают индексные блоки рядом с блоками данных**, на которые они ссылаются, и **к блокам данных можно быстро обращаться** после считывания индексного блока.

В UNIX индексные блоки называются **индексными узлами**, хранящие **атрибуты файлов** (владелец, размер, время создания и последнего изменения) и **адреса** некоторых **блоков данных файла** и **указатели на индексные блоки с продолжениями**, называемые **косвенными блоками. Структуры индексных узлов поддерживают до трех уровней косвенных блоков**.

В косвенных блоках первого уровня хранятся указатели на блоки данных. **В косвенных блоках второго уровня** хранятся только указатели на косвенные блоки первого уровня. **В косвенных блоках третьего уровня** хранятся только указатели на косвенные блоки второго уровня. **Сильная сторона: для поиска любого блока с данными нужно выполнить максимум четыре перехода по указателям** (указатель из индексного узла и максимум три указателя из косвенных блоков).

45) Файловые системы: Управление свободным пространством.

Список свободных блоков — это связный список блоков, в котором указано размещение свободных блоков. Последняя запись в блоке списка свободных блоков хранит указатель на следующий блок этого списка; последняя запись в последнем блоке списка хранит null-указатель. Когда **системе нужно выделить для файла новый блок**, она **находит адрес свободного блока** в списке свободных блоков, **записывает данные в этот блок и удаляет запись об этом блоке из списка. Файл. сис. выделяет блоки** из начала списка и добавляет освободившиеся блоки в конец списка. Указатели на начало и конец списка могут храниться в суперблоке файловой системы. **Свободный блок можно найти**, выполнив один переход по указателю; **добавление нового блока в список** тоже требует добавления одного указателя.

По мере создания и удаления файлов свободное пространство на накопителе становится все более фрагментированным, и соседние записи в списке свободных блоков будут указывать на несмежные блоки накопителя. **В результате следующие друг за другом данные могут быть записаны в далеко отстоящие друг от друга блоки, и время доступа к этим данным может возрасти**.

Битовый массив. В одном из вариантов реализации бит содержит значение 1, если соответствующий ему блок занят, и 0, если свободен. **Одно из основных преимуществ битовых массивов**, что файловая система может быстро определить, доступны ли в некоторой области накопителя непрерывные последовательности свободных блоков.

46) Доступ к файлам: Матрица контроля доступа

Матрицы контроля доступа, в которой перечислены все пользователи и все файлы в системе. В **ячейке aij мат- рицы хранится 1, если пользователь i может обращаться к файлу j; в против- ном случае aij = 0.** В системе с большим количеством пользователей и множеством файлов размер матрицы будет весьма большим. Чтобы **сделать** концепцию матрицы **пригодной для практического применения, нужно использовать различные коды** для обозначения разных **видов доступа — только для чтения, только для записи, только для исполнения, для чтения и записи и так далее.**

47) Доступ к файлам: Контроль доступа по классам пользователя.

Классификация пользователей:

1) Владелец - пользователь, создавший файл и обладающий неогран. доступом к файлу, может менять права других пользователей на доступ к этому файлу.

2) Указзаный пользователь. - поллзователь, которому дали права на доступ к файлу.

3) Группа - члены группы могут обладать доступом к связанным с проектом файлам других членов группы.

4) Общедоступный — опер. система позволяет создавать общедоступные файлы, обращаться к которым могут все пользователи, но не изменять.

48) Защита целостности данных: резервное копирование и восстановление.

Физическое резервное копирование создает копию содержимого устройства хранения на битовом уровне. Оно **не сохраняет** информацию о **логической структуре** файловой системы. Поэтому **физическая резервная копия не может разделить содержащиеся в ней файлы,** поэтому физические эти копии копии должны содержать всю файловую систему целиком.

Логическое резервное копирование сохраняет данные файловой системы и ее логическую структуру. Поскольку логические резервные копии хранят **данные в стандартном формате в виде структуры директорий**, они п**озволяют операционным системам** с различными собственными форматами файлов **считывать и восстанавливать данные из резервных копий**, и данные могут восстанавливаться на множестве разнородных систем и позволяют восстанавливать из них только отдельные файлы. **Однако** поскольку **логическое резервное копирование считывает только данные, предоставляемые файловой системой,** оно **может пропустить**, например, **скрытые файлы или метаданные**, которые были бы скопированы при побитовом физическом резервном копировании накопителя системы. **Сохранение файлов в стандартном формате может быть неэффективным из-за** накладных расходов, связанных с **преобразованием между форматом архива и исходным форматом файла.**

Инкрементное резервное копирование это логическое резервное копирование, при котором копируются только данные в файловой системе, изменившиеся со времени предыдущего копирования.

49) Защита целостности данных: журнальные файловые системы

Журнальная файловая система (LFS — Logged File System), также называемая протоколирующей файловой системой выполняет все операции с файлами как протоколируемые транзакции. В LFS **запрос на создание файлов** в новой директории выполняется так - файловая система сначала записывает данные файла foo в журнал, затем записывает ме-таданные файла foo (т.е. индексный узел), позволяющие системе найти данные из этого файла. После этого в журнал добавляется запись о новой директории.

Поскольку измененные директории и метаданные всегда записываются в конец журнала, LFS может понадобиться просмотреть весь журнал, чтобы найти конкретный файл. **Поэтому LFS кэширует информацию о размещении файлов** в файловой системе и время от времени записывает **карты индексных узлов** (inode maps) или суперблоки, указывающие размещение других метаданных. Это позволяет операционной системе находить и кэшировать метаданные файлов достаточно быстро при загрузке системы. Некоторые файловые системы **хранят в журналах только метаданные**. В этом случае система **сначала изменяет метаданные**, внося в журнал новую запись, **а затем обновляет данные** в файловой системе.

Как освободить место для поступающих данных. LFS может периодически просматривать журнал и определять, какие блоки можно освободить, поскольку другие блоки содержат измененные копии данных из этих блоков. Затем можно размещать новые данные в освободившихся блоках, но при этом новые данные окажутся сильно фрагментированными. **LFS может создавать непрерывные свободные участки в журнале**, копируя данные в конец журнала.

50) Оперативная память: стратегии управления памятью.

Стратегии управления памятью определяют поведение выбранной организации памяти при различных уровнях нагрузки. Управление памятью реализуется с помощью комплекса программных и специализированных аппаратных средств.

Стратегии загрузки определяют, когда нужно перемещать новые данные или инструкции в оперативную память со вторичных устройств хранения. Они делятся на две категории: стратегии **загрузки по требованию** и стратегии **предварительной загрузки.**

Стратегии размещения определяют, где в оперативной памяти система должна размещать загружаемые программы и данные. Делятся на **первых подходящих участках**, в **наиболее подходящих участках**, и в **наименее подходящих участках.**

Если оперативная память слишком заполнена, чтобы вместить новую программу, система должна удалить некоторые программы и данные, находящиеся в памяти. **Стратегии замены** определяют, какие элементы нужно удалять из памяти.

51) Оперативная память: выделение непрерывных блоков памяти в однопольз. сис.

В ранних компьютерных системах в любой момент времени мог работать на машине только один пользователь. Ему были доступны все ресурсы машины.

В таблице показана организация памяти в однопользовательских системах с выделением непрерывных блоков. (Табл: сверху вниз — ОС, Пользователь, не используется).

Разработчики систем создали библиотеки, реализовавшие стандартные функции — системы управления вводом/выводом (Input/Output Control System — IOCS).

52) Оперативная память: оверлеи.

Программист разделяет программу на логические блоки. Когда программе становится ненужным какой-то блок, программа может удалить его из памяти и разместить на его месте какой-то другой блок или его часть.

Оверлеи позволяли программистам «расширять» оперативную память. Однако ручное создание оверлеев требовало тщательного и долгого планирования, и программистам требовались глубокие знания об организации памяти в системе. Программа со сложной структурой оверлеев часто с трудом поддавалась изменению. Стало ясно, что операционная система должна изолировать программистов от сложных задач управления памятью, таких, как оверлеи. **Структура:**

53) Оперативная память: мультипрограммные системы с фиксированным распределением памяти.

В первых мультипрограммных системах использовался мультипрограммный режим с фиксированным распределением памяти. В **этом режиме система делит оперативную память на несколько разделов** (partitions) фиксированного размера. В каждом разделе размещается одна задача, и система быстро переключает процессор между задачами, чтобы создать впечатление одновременного выполнения этих задач. **Чтобы устранить проблему нерационального использования оперативной памяти**, разработчики создали перемещаемые трансляторы, компиляторы и загрузчики.

Система может ограничить каждый раздел двумя регистрами — верхней и нижней границами, которые также называются **базовым регистром** и **регистром предела.** Когда процесс генерирует обращение к оперативной памяти, система проверяет, не меньше ли запрашиваемый адрес, чем значение в базовом регистре, и не больше ли он, чем значение в регистре предела.

Системы с фиксированным распределением памяти страдают от **внутренней фрагментации** (internal fragmentation), которая проявляется, если объем программной части процесса и его данных меньше, чем объем раздела, в котором этот процесс выполняется.

54) Оперативная память: мультипрограммные системы с изменяемым распределением памяти.

Оптимальное решение по выделению памяти - каждому процессу ровно столько памяти, сколько ему требуется (в пределах объема, доступного в оперативной памяти). Такой режим называется **мультипрограммным с изменяемым распределением памяти (variable-partition multiprogramming).**

Система просматривает очередь и размещает каждую задачу в доступной области оперативной памяти, после чего задача становится процессом.

Системы с изменяемым распределением памяти не страдают от внутренней фрагментации. Но при любой организации памяти часть памяти тратится бесполезно. Потери неочевидны, пока процессы не заканчивают выполняться, после чего в памяти появляются дыры, в которых система может размещать новые процессы. По мере того, как процессы продолжают завершаться, дыры становятся все меньше. Это явление называется **внешней фрагментацией** (external fragmentation), здесь общий объем дыр достаточен, чтобы вместить новый процесс, но каждая дыра в отдельности слишком мала для размещения в ней процесса.

Методы уменьшения внешней фрагментации. Когда процесс в системе с изменяемым распределением памяти завершается, система может определить, является ли освободившаяся область смежной с уже свободными областями. Затем система запишет в **список свободных областей**, что или появилась новая свободная область, или уже существующая свободная область увеличилась. Процесс объединения смежных свободных областей в одну большего размера называется **сращением.**

Уплотнением памяти (сборка мусора) - процессы в памяти перемещаются к краю адресного пространства. При этом образуется единая свободная область вместо множества раздельных свободных областей памяти. В результате вся свободная память составляет единую непрерывную область, и ждущий процесс можно запускать.

55) Оперативная память: мультипрограммные системы с подкачкой.

В некоторых системах с подкачкой **в оперативной памяти одновременно находится только один процесс. Этот процесс выполняется** до тех пор, **пока он может продолжать использование процессора. Если использование процессора приостанавливается, процесс передает память и процессор другому процессу.** В результате система поочередно передает память каждому процессу. **Когда процесс освобождает ресурс,** система выгружает предыдущий процесс на вторичное устройство хранения и загружает в оперативную память следующий процесс. **Когда система закачивает процесс обратно в оперативную память,** содержимое памяти процесса и другие значения извлекаются обратно со вторичного устройства хранения.

56) Стратегии размещения в памяти.

Стратегия размещения в памяти определяет, в каких областях в оперативной памяти размещаются загружаемые программы и данные.

1) Стратегия размещения в первых подходящих участках размещает задачу в первом же найденном свободном участке достаточного размера.

2) Размещение в следующем подходящем участке, когда поиск подходящей дыры начинается в том месте, где закончился предыдущий поиск.

3) Стратегия размещения в наиболее подходящих участках размещает задачу в свободном участке, наиболее близком к задаче по размеру, чтобы оставить минимальную по размеру дыру.

4) Стратегия размещения в наименее подходящих участках размещает задачу в дыру, в которую она подходит хуже всего.

57) Виртуальная память.

Одно из решений ограниченности памяти - создать иллюзию наличия большого объема памяти. Это фундаментальная идея, реализованная в виде **виртуальной памяти.**

Адреса, по которым обращаются процессы, называются **виртуальными адресами**.

Адреса, доступные в оперативной памяти, называются **реальными адресами.**

При каждом обращении процесса по виртуальному адресу система должна преобразовать этот адрес в реальный. **С**истемы виртуальной памяти используют специальные аппаратные устройства, называемые **устройствами управления памятью** , способные быстро преобразовывать виртуальные адреса в реальные.

Виртуальное адресное пространство процесса - это множество виртуальных адресов, по которым может обращаться процесс.

Множество реальных адресов, доступных в конкретной системе, называется **реальным адресным пространством.**

Механизмы динамической трансляции адресов (Dynamic Address Translation — DAT) преобразуют виртуальные адреса в физические во время выполнения программ.

Искусственная непрерывность — отображение непрерывных областей виртуальных адресных пространств процессов во фрагментированные области физической памяти.

Суперблок может содержать:

- Идентификатор файловой системы, определяющий тип используемой файловой системы.
- Информацию о количестве блоков в файловой системе.
- Информацию о расположении свободных блоков.
- Информацию о расположении корневой директории.
- Дату и время последнего изменения файловой системы.
- Информацию, показывающую, нуждается ли файловая система в проверке

Если суперблок будет поврежден или уничтожен, файловая система может быть не в состоянии обратиться к данным в файлах. Незначительные ошибки в данных суперблока могут привести к повреждению данных пользователей. Большинство файловых систем хранят избыточные копии суперблока, распределенные по накопителю.