1. Как происходит загрузка операционной системы? Что такое первичный загрузчик? Вторичный? Как происходит загрузка бездисковых машин?

**Загрузка операционной системы** происходит в несколько этапов.

1. Первым этапом является загрузка первичного загрузчика, который находится на жестком диске или другом носителе информации. **Первичный загрузчик.** Первая стадия загрузки компьютера: программа, размер и возможности которой зависят от аппаратных требований и функций BIOS. Основная задача - загрузить вторичный загрузчик.

2. После загрузки первичного загрузчика начинается загрузка вторичного загрузчика, который находится в операционной системе. **Вторичный загрузчик.** Вторая стадия загрузки компьютера: программа, размер и возможности которой практически не зависят от аппаратных требований. Основная задача - полностью подготовить и запустить загрузку операционной системы.

3. Когда операционная система загружается, она начинает работу с пользовательским интерфейсом и другими приложениями.

**Загрузка бездисковых машин** происходит по-другому. Вместо жесткого диска используется сеть, и операционная система загружается из образа, который находится на сервере. При этом сначала загружается минимальный операционный системный образ, который затем запускает виртуальную машину и загружает полноценную операционную систему из сети.

2. Драйвер устройства. Функции драйвера в ОС семейства Unix.

**Драйвер устройства** - это программа, которая позволяет операционной системе взаимодействовать с конкретным устройством или аппаратным компонентом компьютера. Основной целью драйвера устройства является обеспечение абстрактного интерфейса между аппаратурой и операционной системой, позволяя операционной системе управлять и получать доступ к функциональности устройства.

В операционных системах семейства Unix, драйверы устройств являются частью ядра операционной системы и работают на низкоуровневом уровне для обеспечения доступа к аппаратным ресурсам.

Функции драйвера в UNIX-подобных операционных системах могут включать следующие:

1. **Инициализация и регистрация драйвера.** (Инициализация данных драйвера, регистрация обработчиков прерываний и создание специальных файлов устройства)

2. **Управление жизненным циклом устройства.** (Обнаружение, инициализация, освобождение ресурсов устройства, обработка сбоев или отказов)

3. **Определение интерфейса устройства.** (Функции для чтения/записи данных, управления состоянием устройства, настройки параметров и перехвата и обработки прерываний)

4. **Взаимодействие с ядром.** (Запросы к ядру для выделения памяти, управления потоками/процессами, графического вывода и других системных служб)

5. **Обеспечение безопасности и управления доступом.** (Авторизация и аутентификация пользователей, проверка полномочий и контроль доступа к ресурсам устройства)

6. **Обработка ошибок и управление исключительными ситуациями.** (Запись сообщений об ошибках, регенерация специальных команд или перезапуск устройства)

7**. Мониторинг и отладка.** (Журналирование событий, критических участков кода и запись дампов памяти для анализа ошибок)   
Дамп - содержимое рабочей памяти одного процесса, ядра или всей операционной системы.

3. Файловая система ISO 9660 (CDFS).

ISO 9660 (Compact Disk File System) - это стандарт файловой системы для оптических дисков, таких как CD и DVD. Файловая система ISO 9660 используется для создания образов дисков, которые могут быть загружены на компьютеры без установленной операционной системы или для установки операционной системы на новый компьютер. Эта файловая система поддерживает иерархическую структуру каталогов и файлов, а также ограничения на длину имени файла и размер файла. Она также поддерживает функции защиты данных, такие как контроль целостности данных и исправление ошибок.

4. Структура и принципы работы файловой системы NTFS.

NTFS (New Technology File System) - это проприетарная файловая система, разработанная компанией Microsoft, адаптированная для использования в операционных системах Windows NT и выше.

Структура и принципы работы файловой системы NTFS включают следующие особенности:

1. **Структурированное размещение данных.** Размещение файлов в блоках (кластерах) на диске, в отличие от более простых файловых систем, таких как FAT. Кластер - это минимальный размер данных, с которым NTFS работает. Затем файлы хранятся в виде контейнеров частей.

2. **Система атрибутов.** Система атрибутов, которая описывает файлы и директории. (Обязательно: имя, тип, дата создания и размер. Дополнительно: содержимое файла, разрешения доступа, временные метки и альтернативные потоки данных)

3. **Журналирование.** Обеспечение целостности файловой системы и быстрое восстановление после сбоев.

4. **Многоуровневая организация файловой системы.** Иерархическая структура каталогов для организации файлов и директорий.

5. **Сжатие данных.** Позволяет экономить место на диске без потери данных.

6. **Разреженные файлы.** Файлы могут содержать пропущенные области, не занимающие физическое пространство на диске для более эффективного использования.

7. **Безопасность и права доступа.**

Является предпочтительной файловой системой для большинства компьютеров под управлением Windows.

5. Файловая система FAT.

FAT (File Allocation Table) - это файловая система, которая используется для хранения файлов на различных устройствах, таких как жесткие диски, флеш-накопители и карты памяти. FAT была разработана в 1977 году компанией Microsoft и с тех пор стала одной из наиболее распространенных файловых систем в мире.

FAT поддерживает иерархическую структуру каталогов и файлов, а также ограничения на длину имени файла и размер файла. Она также поддерживает функции защиты данных, такие как контроль целостности данных и исправление ошибок.

Существует несколько версий FAT, для дискет, для жестких дисков и некоторых карт памяти, и для более крупных носителей, таких как флеш-накопители и жесткие диски.

6. Структура и особенности организации файловой системы UFS (FFS).

UFS (Unix File System), также известная как FFS (Fast File System), является исторической файловой системой, разработанной для операционных систем семейства Unix.

Структура и особенности:

1. **Суперблок** - это специальный блок в начале файловой системы UFS/FFS, который содержит информацию о файловой системе, такую как размер блока, размер файловой системы, количество инодов и другую системную информацию.

2. **Иноды** - это структуры данных, которые содержат метаинформацию о файлах и директориях в файловой системе. Иноды располагаются в специально зарезервированных блоках внутри файловой системы.

3**. Аллокация блоков данных.** Использует блоки данных фиксированного размера для хранения содержимого файлов. Размер блока может варьироваться в зависимости от конфигурации файловой системы, но обычно используется размер 4 КБ.

4. **Структура директорий.** Директории организуются с помощью особых блоков, называемых "блоками директорий", которые содержат список записей, каждая из которых содержит имя файла и указатель на соответствующий инод. Они используются для производительного поиска и доступа к файлам в директориях.

5. **Журналирование**. Не включает встроенную поддержку журналирования. Это означает, что в случае сбоя системы или понижения надежности файловой системы, возможно восстановление файла за счет fsck (программа для проверки и восстановления файловых систем UFS/FFS).

6. **Поддержка символических и жестких ссылок** **на файлы**. Символьные ссылки - это файлы, содержащие путь к другому файлу. Жёсткие ссылки - это несколько записей в одной файловой системе, которые указывают на данные другого файла. Это позволяет создавать ссылки на файлы и общаться с ними на разных уровнях файловой системы.

7. **Ограничение размера файла.** В оригинальной системе ограничение составляло 2 ГБ. Однако, в поздних версиях этот размер был увеличен до 16 ТБ.

8. **Безопасность и доступ** поддерживает механизмы для управления правами доступа к файлам и директориям, включая режимы доступа UNIX.

7. Уровни RAID.

RAID (Redundant Array of Independent Disks) - это технология, которая позволяет объединять несколько физических дисков в одну логическую единицу хранения данных с целью повышения производительности и надежности.

Существует несколько уровней RAID:

1. **RAID 0** - используется для повышения производительности, объединяет несколько дисков в один логический диск без дублирования данных. Однако, если один из дисков выходит из строя, все данные на всех дисках теряются.

2. **RAID 1** - используется для повышения надежности, каждый диск в массиве дублируется на другой диск, что позволяет сохранить данные в случае выхода из строя одного из дисков. Однако, эта конфигурация не увеличивает производительность.

3. **RAID 5** - используется для повышения производительности и надежности, объединяет несколько дисков в один логический диск с распределением данных между всеми дисками и добавлением контрольной суммы для обнаружения ошибок. Если один из дисков выходит из строя, данные можно восстановить на основе контрольной суммы.

4. **RAID 6** - используется для повышения надежности, работает аналогично RAID 5, но использует две контрольные суммы для обнаружения и восстановления данных в случае выхода из строя двух дисков.

5. **RAID 10 (RAID 1+0)** - используется для повышения производительности и надежности, объединяет несколько дисков в две группы, каждая из которых работает в режиме RAID 1. Затем эти две группы объединяются в режиме RAID 0. Эта конфигурация обеспечивает высокую производительность и надежность, но требует большего количества дисков.

8. Распределение памяти алгоритмами близнецов и парных меток Ограничения этих алгоритмов.

Алгоритмы распределения памяти "близнецы" и "парные метки" являются методами управления памятью, используемыми в операционных системах для эффективного выделения и освобождения блоков памяти.

**Алгоритм "близнецы"**

- Разделяет доступную память на блоки размером, равным степени двойки.

- Каждый блок представляет собой структуру-близнеца, состоящую из двух половинок

- Если блок запрашивается для выделения, он будет разделен на две половинки.

- Если блок освобождается, две смежные половинки объединяются в больший блок.

Ограничения:

- Распределение памяти происходит только по размерам, являющимся степенями двойки.

- Для эффективного использования памяти необходимо предварительно заранее выделить большие блоки памяти.

**Алгоритм "парные метки"**

- Алгоритм использует связанный список свободных блоков памяти.

- Каждый блок имеет парную метку, которая указывает на следующий свободный блок памяти.

- Блоки памяти объединяются путем связывания парных меток для формирования более крупных блоков памяти.

Ограничения:

- Объединение и разделение блоков памяти требует обхода связанного списка, что может вызвать некоторые накладные расходы в производительности.

- Механизмы управления памятью для объединения и разделения блоков должны быть эффективно реализованы, чтобы избежать высокой фрагментации памяти.

9. Алгоритмы поиска жертвы при страничном обмене и кэшировании. Критерии выбора и влияние алгоритма на производительность. Что такое рабочее множество страниц?

Страничный обмен и кэширование - это важные концепции в управлении памятью компьютера, особенно в системах с виртуальной памятью. В процессе выполнения программы, данные и код могут быть разделены на страницы, которые хранятся в оперативной памяти или на диске.

Алгоритмы поиска жертвы при страничном обмене отвечают за определение, какие страницы следует удалить из оперативной памяти, чтобы освободить место для новых страниц. Основная идея состоит в том, что операционная система должна вытеснять наименее используемые страницы или те, которые весьма маловероятно понадобятся в ближайшем будущем.

Существует несколько алгоритмов выбора жертвы:

1. **FIFO (First-In-First-Out)** - этот алгоритм выбирает на удаление страницу, которая попала в память первой. Он прост в реализации, но не всегда эффективен, потому что не учитывает последовательность доступа к страницам.

2. **LRU (Least Recently Used)** - этот алгоритм выбирает на удаление страницу, которая была самой давно неиспользуемой. LRU является одним из популярных алгоритмов для выбора жертвы и обычно демонстрирует хорошую производительность.

3. **LFU (Least Frequently Used)** - этот алгоритм выбирает страницу на удаление, которая была использована реже всего.

4. **Optimal (OPT)** - этот алгоритм является оптимальным теоретическим решением, которое выбирает на удаление страницу, которая не будет использоваться дольше всех. Однако его реализация требует знания будущих обращений к памяти, что нереально. OPT используется в анализе алгоритмов для определения их производительности.

От выбранного алгоритма будет зависеть количество "промахов в кэше" (cache misses) - это ситуации, когда процессор не может найти нужную страницу в оперативной памяти и должен загрузить ее с диска. Чем больше промахов в кэше, тем ниже производительность программы.

**Рабочее множество страниц** - это набор страниц, который активно используется программой в течение определенного временного интервала (например, при выполнении какой-либо задачи). Рабочее множество страниц может изменяться со временем в зависимости от выполняемой работы и обращений к данным.

10. Инверсия приоритета. Способы ее предотвращения и способы обхода этой проблемы.

Инверсия приоритета - это ситуация, в которой низкоприоритетная задача занимает процессорное время, блокируя выполнение высокоприоритетной задачи. Это может привести к нарушению временных ограничений и плохой производительности системы реального времени.

Несколько способов предотвращения:

1. **Приоритет инвертирования.**

Высокоприоритетная задача временно повышает приоритет низкоприоритетной задачи, которая блокирует выполнение высокоприоритетной задачи. Таким образом, низкоприоритетная задача выполняется с приоритетом высокоприоритетной задачи до тех пор, пока не завершится или не освободит ресурсы, необходимые для выполнения высокоприоритетной задачи.

2. **Алгоритмы планирования с приоритетами.**

Высокоприоритетная задача всегда превосходит низкоприоритетную задачу. Такие алгоритмы обеспечивают более справедливое распределение процессорного времени между задачами с разными приоритетами.

3. **Мьютексы и семафоры.**

Позволяет синхронизировать доступ к общим ресурсам. При этом, если высокоприоритетная задача запрашивает доступ к ресурсу, который уже занят низкоприоритетной задачей, то высокоприоритетная задача будет блокироваться до тех пор, пока низкоприоритетная задача не освободит ресурс.

4. **Алгоритмы планирования с поддержкой реального времени.**

В системах реального времени применяются специализированные алгоритмы планирования, которые гарантируют выполнение задач в соответствии с их приоритетами и временными ограничениями. Эти алгоритмы обеспечивают более строгую гарантию соблюдения временных требований задач.

Семафоры используются для контроля доступа к ограниченному числу ресурсов. Они представляют собой объекты, имеющие счетчик, который уменьшается при каждом захвате ресурса и увеличивается при его освобождении.

Мьютекс — это специальный вид семафора, используемый для предотвращения одновременного доступа к критическому ресурсу. В отличие от семафора, мьютекс имеет только два состояния: заблокирован и разблокирован.

11. Сборщик мусора. Сборка мусора. Основные стратегии сборки мусора, их преимущества и недостатки.

**Сборщик мусора** - это компонент, ответственный за автоматическое управление памятью в программе. Его задача заключается в определении и освобождении неиспользуемых объектов в памяти.

Основные стратегии сборки мусора:

1. **Счетчик ссылок.**

- *Принцип работы:* Каждому объекту присваивается счётчик ссылок, который отслеживает количество ссылок на объект. Когда счетчик становится равным нулю, объект считается неиспользуемым и может быть удален из памяти.

- *Преимущества:* Прост в реализации, не требует остановки работы программы.

- *Недостатки:* Неэффективен в обработке циклических ссылок, так как они могут создать взаимную блокировку счетчиков ссылок и привести к утечкам памяти.

2. **Поколения.**

- *Принцип работы:* Объекты разделяются на группы или поколения в зависимости от их возраста. Объекты, которые дольше всего существуют, считаются более стабильными и перемещаются в старшие поколения. Сборка мусора сконцентрирована на молодых поколениях, так как большинство объектов быстро становятся неактивными.

- *Преимущества:* Эффективность для большинства типичных программ, поскольку большинство объектов быстро становятся мусором, и их можно удалять из памяти с меньшими затратами.

- *Недостатки:* Требует дополнительных накладных расходов для работы с поколениями объектов и перемещения объектов из одного поколения в другое.

3. **Стой-и-копируй.**

- *Принцип работы:* Память делится на две равные половины. При сборке мусора все активные объекты копируются из одной половины в другую, оставляя мусор в исходной половине. Затем весь мусор освобождается без необходимости поиска и обновления ссылок.

- *Преимущества:* Прост в реализации, избегает проблемы фрагментации памяти, так как объекты периодически перемещаются в другую половину памяти.

- *Недостатки:* Требует дополнительной памяти для копирования объектов, может быть неэффективен при больших объемах памяти.

4. **Маркировка и сжатие.**

- *Принцип работы:* Процесс сборки мусора состоит из двух этапов. Сначала выполняется маркировка всех активных объектов. Затем неактивные объекты удаляются, а оставшиеся активные объекты сжимаются, чтобы свободное пространство в памяти было сконцентрировано в одном месте.

- *Преимущества:* Избегает проблемы фрагментации памяти, свободное пространство сосредоточено, что улучшает производительность доступа к памяти.

- *Недостатки:* Требует дополнительного времени на сжатие активных объектов.

12. Что такое гармонически взаимодействующие последовательные процессы? Средства для реализации этой дисциплины в существующих системах.

**Гармонически взаимодействующие последовательные процессы** - это дисциплина планирования в системах реального времени, которая обеспечивает выполнение последовательных процессов с определенной гарантированной частотой и взаимодействием между ними, отсутствие блокировок и ожиданий.

*Средства и системы для реализации*

1**. Алгоритмы планирования.** Разработаны специальные алгоритмы планирования, которые позволяют управлять периодичностью и гарантированной частотой выполнения процессов.

2. **Ограничения времени выполнения.** процессов и контроль их соблюдения.

3. **Синхронизация и коммуникация.** Взаимодействие в заданный момент времени и в строго определенном порядке.

4. **Анализ и верификация.** Соответствие гармонически взаимодействующих последовательных процессов заданным требованиям и временным ограничениям.

13. Разделяемая память. Преимущества и недостатки по сравнению с другими методами межпроцессного взаимодействия.

**Разделяемая память** - это метод межпроцессного взаимодействия, в котором несколько процессов имеют общий сегмент памяти, через который они могут обмениваться данными. Процессы могут читать и записывать данные в этот общий сегмент памяти, обеспечивая таким образом совместный доступ к информации.

Преимущества разделяемой памяти:

1. **Эффективность.** Разделяемая память обладает высокой производительностью, так как процессы могут напрямую обращаться к общей памяти без дополнительных накладных расходов, связанных с копированием данных.

2. **Быстрое взаимодействие.** Поскольку общая память находится в физической близости между процессами, обмен данными между ними происходит очень быстро и эффективно.

3. **Простота использования.** Использование разделяемой памяти в программе обычно проще, чем другие методы межпроцессного взаимодействия, такие как каналы или сокеты. Это требует меньше кода и меньше сложности в реализации.

Недостатки разделяемой памяти:

1. **Синхронизация.** Поскольку несколько процессов могут одновременно иметь доступ к общей памяти, необходимо правильно синхронизировать доступ и избегать гонок данных и других проблем, связанных с параллельным доступом.

2. **Общая память может быть сложна в отладке.** Трудно обнаружимые ошибки.

3. **Ограничения производительности.** Может возникнуть конфликт за ресурсы и ухудшение производительности при чтении или записи.

4. **Непереносимость.** Разделяемая память является особенностью операционной системы и может иметь ограничения или отличаться в разных средах выполнения, что затрудняет переносимость программы между различными платформами.

14. Реентерабельная программа.

**Реентерабельная программа** - это программа, которая может быть безопасно вызвана несколькими процессами или потоками одновременно без возникновения ошибок или непредсказуемого поведения.

Для того чтобы программа была реентерабельной, она должна быть написана таким образом, чтобы не использовать глобальные переменные или другие ресурсы, которые могут быть изменены другими процессами или потоками во время выполнения. Вместо этого программа должна использовать локальные переменные и передавать параметры через стек вызовов.

Реентерабельность особенно важна для многопоточных и многопроцессорных систем, где несколько процессов или потоков могут одновременно обращаться к одной и той же программе.

15. Приоритеты процессов и нитей. Управление приоритетами для нитей реального и разделенного времени. Где используется и для чего нужно динамическое изменение приоритета?

**Приоритеты процессов и нитей** являются механизмом управления планированием и приоритетом выполнения в многозадачных системах.

**Управление приоритетами для нитей реального времени.**

В системах реального времени приоритеты нитей используются для обеспечения выполнения задач в строго заданных временных ограничениях. Задачи с более высоким приоритетом имеют больше шансов быть выполненными вовремя. При этом степень динамического изменения приоритетов может быть ограничена, чтобы обеспечить предсказуемость выполнения и гарантировать временные требования системы реального времени.

**Управление приоритетами для нитей разделенного времени:**

В системах разделенного времени приоритеты нитей используются для определения частоты и порядка выполнения задач в условиях конкуренции за процессорное время. Чем выше приоритет нити, тем больше вероятность, что она будет выбрана для выполнения следующей.

**Динамическое изменение приоритета** может быть применено для адаптации системы в зависимости от текущих условий и требований. Оно может улучшить производительность, реактивность и эффективность выполнения.

Примеры использования.

1. **Адаптивное планирование.** Например, процессор может динамически повышать или понижать приоритет задачи в зависимости от ее использования процессорного времени.

2**. Управление ресурсами.** Например, задаче, которая активно использует процессор, может быть временно повышен приоритет, чтобы она была выполнена как можно быстрее, а затем приоритет восстановлен на более низкий уровень.

3. **Профилирование и адаптация.** Например, если система обнаруживает, что задача выполняется очень долго или часто блокируется на операциях ввода-вывода, ей может быть автоматически увеличен приоритет для ускорения ее выполнения.

16. Системы управления доступом. Полномочия и списки контроля доступа. Кольца доступа.

**Системы управления доступом -** это комплекс технических и программных средств, который позволяет контролировать доступ к объектам и помещениям, а также управлять этим доступом.

**Полномочия** в системе управления доступом определяют, какие действия и операции может выполнять пользователь или субъект системы. Это может включать такие полномочия, как чтение, запись, выполнение, удаление и другие операции в рамках определенных ресурсов или объектов системы. Могут быть установлены как для операционной системы, так и для отдельных приложений.

**Списки контроля доступа** представляют собой структуры данных, которые определяют разрешения или запреты для конкретных субъектов или группы субъектов на выполнение операций над определенными ресурсами или объектами.

**Кольца доступа**— это модель безопасности, используемая в операционных системах, которая разделяет привилегии и доступ к системным ресурсам на различные уровни (кольца). Она предполагает, что чем ниже уровень, тем больше привилегий имеет субъект системы. В классической модели существуют четыре кольца. Уровни кольца доступа позволяют ограничить доступ и предотвратить несанкционированное выполнение привилегированных операций. Это помогает обеспечить безопасность и целостность системы.

17. Планировщики разделенного времени. Динамическое управление приоритетами в системах разделенного времени.

**Планировщики разделенного времени** используются для управления выделением процессорного времени между различными задачами в системах, где несколько процессов конкурируют за ресурсы и выполняются параллельно.

**Динамическое управление приоритетами** в системах разделенного времени заключается в изменении приоритетов задач в режиме выполнения на основе их поведения, состояния и приоритета. Основной целью динамического управления приоритетами является обеспечение более эффективного распределения ресурсов и улучшения общей производительности системы.

Динамическое управление приоритетами в системах разделенного времени позволяет адаптировать планирование задач к текущим условиям и требованиям системы, повышая эффективность использования ресурсов и улучшая отзывчивость системы на различные типы задач.

18. Кооперативная и вытесняющая (preemptive) многозадачность. Преимущества и недостатки обоих архитектур.

**Кооперативная и вытесняющая многозадачность** представляют собой различные подходы к организации и управлению выполнением нескольких задач в операционной системе.

**Кооперативная многозадачность** предполагает, что задачи явно сотрудничают друг с другом и определяют момент, когда они добровольно освобождают центральный процессор для выполнения других задач. В этом случае задачи осуществляют контроль передачи управления друг другу, обычно посредством явных вызовов или сигналов.

*Преимущества кооперативной многозадачности*

- Простота реализации. Нет необходимости в сложных средствах синхронизации или контроля за ресурсами.

- Эффективное использование ресурсов. (Процессорное время и память.)

*Недостатки кооперативной многозадачности*

- Несправедливое распределение ресурсов. Если одной задаче необходимо выполняться в течение длительного времени, это может привести к деградации производительности.

- Уязвимость системы. Если задача повреждена или зациклена, она может забрать все доступные ресурсы, и другие задачи будут заблокированы.

**Вытесняющая многозадачность** предполагает, что операционная система имеет возможность вытеснить текущую выполняющуюся задачу и передать управление другой задаче. В этом случае контроль передачи управления между задачами полностью находится в руках операционной системы, и задачи не зависят от сотрудничества друг с другом.

*Преимущества вытесняющей многозадачности*

- Справедливое распределение ресурсов.

- Безопасность системы. Предотвращение длительного занятия ресурсов одной задачей и обеспечение защиты от ошибок и атак.

- Отзывчивость и отказоустойчивость. Обеспечивает устойчивость к сбоям.

*Недостатки вытесняющей многозадачности*

- Сложность реализации. Требуется сложная инфраструктура планировщика задач и механизмов переключения контекста.

- Потребление ресурсов. Переключение контекста между задачами может снизить производительность системы.

19. Ввод-вывод в режиме опроса и по прерываниям. Преимущества и недостатки.

**Ввод-вывод в режиме опроса и по прерываниям** - это два основных метода управления операциями ввода-вывода в компьютерных системах.

**Ввод-вывод в режиме опроса**

*Преимущества*

- Простота и удобство реализации. Программист полностью контролирует операции ввода-вывода.

- Предсказуемость и точная временная координация операций ввода-вывода. (с остальными задачами)

- Эффективность в системах с малым количеством устройств ввода-вывода и низкой нагрузкой.

*Недостатки*

- Централизация управления. Постоянно использует процессорное время и создает накладные расходы на опрос устройств, даже когда они не готовы к передаче данных.

- Значительная загрузка процессора.

- Трудность масштабирования. В больших системах может возникнуть проблема с многопоточностью и потоковой безопасностью при выполнении опроса устройств, особенно если устройства работают с различной скоростью.

**Ввод-вывод по прерываниям**

*Преимущества*

- Эффективность использования процессорного времени. Прерывания сигнализируют процессору о готовности устройств и активизируют обработчики прерываний только при необходимости.

- Параллельное выполнение операций ввода-вывода

- Снижение нагрузки на процессор. Процессор передает управление обработчику прерывания только в момент готовности устройства к передаче данных.

*Недостатки*

- Сложность реализации.

- Увеличение накладных расходов. Время на переключение контекста и вызов обработчика прерывания.

- Трудности синхронизации и многопоточности.

20. Мертвая и живая блокировки.

**Мертвая блокировка** и **живая блокировка** - это две разновидности проблем, возникающих при синхронизации параллельных процессов или потоков.

**Мертвая блокировка** происходит, когда каждый из заблокированных процессов не может продолжить свою работу, так как ожидает освобождения ресурса, но сам не может освободить этот ресурс, так как это зависит от другого заблокированного процесса.

Простейшим примером мертвой блокировки является "взаимная блокировка", когда процесс А удерживает ресурс X и ожидает ресурс Y, которым владеет процесс B, а процесс B в свою очередь удерживает ресурс Y и ожидает ресурс X.

*Характеристики*

- Зависимость друг от друга.

- Потеря производительности.

- Бесконечное ожидание.

**Живая блокировка** происходит, когда процессы не ожидают ресурсов и выполняют операции, но не могут продвинуться дальше в своей работе, так как они оказываются в "зацикленном" состоянии и постоянно отвечают на действия друг друга, не решая основную задачу.

Примером живой блокировки может быть ситуация, когда два водителя на перекрестке пытаются пропустить друг друга, но постоянно дергаются и никто не движется вперед.

*Характеристики*

- Зацикленное взаимодействие. Процессы находятся в постоянном цикле изменения состояний.

- Потеря производительности.

21. Спинлоки и их применение.

**Спинлоки** - это механизм синхронизации, который используется в многопоточных приложениях для предотвращения одновременного доступа к общим ресурсам несколькими потоками. Они позволяют потоку заблокировать доступ к общему ресурсу до тех пор, пока другой поток не освободит его.

**Применение**

1. *Защита критических секций кода*. Критическая секция - это участок кода, который может быть выполнен только одним потоком за раз. Спинлоки позволяют заблокировать доступ к этой секции до ее завершения.

2. *Использование общих ресурсов*. Если несколько потоков используют общий, то спинлоки могут предотвратить конфликты при доступе к этому ресурсу.

3*. Ожидание завершения операций.*

4. *Предотвращение блокировки потоков*.

5. *Реализация алгоритмов параллельной обработки данных*.

22. Семафоры Дийкстры. Мутексы, двоичные семафоры и семафоры общего вида. Семафоры Unix System V IPC. Наборы семафоров

**Семафоры Дийкстры** - это средство синхронизации процессов, представляют собой целочисленные переменные, которые могут принимать только неотрицательные значения и поддерживают две операции - wait (ожидание) и signal (сигнал).

Операция wait уменьшает значение семафора на 1, если оно больше нуля, иначе процесс блокируется до тех пор, пока значение семафора не станет положительным. Операция signal увеличивает значение семафора на 1 и разблокирует один из процессов, ожидающих на этом семафоре.

**Мутексы**- это особый тип семафоров, который может принимать только два значения - 0 и 1. Они используются для реализации взаимного исключения, что позволяет только одному процессу одновременно получить доступ к общему ресурсу.

**Двоичные семафоры** - это семафоры, которые могут принимать только два значения - 0 и 1. Они подобны мутексам, но могут быть использованы для других целей, кроме взаимного исключения.

**Семафоры общего вида** - это семафоры, которые могут принимать любые неотрицательные значения. Они могут использоваться для решения различных задач синхронизации, например, для ограничения доступа к общему ресурсу нескольким процессам.

**Семафоры Unix System V IPC** - это семафоры, реализованные в операционной системе Unix по стандарту System V. Они поддерживают те же операции, но имеют более сложный интерфейс и требуют использования специальных системных вызовов для работы с ними.

**Наборы семафоров** - могут быть созданы и использованы одновременно несколькими процессами. Это позволяет решать более сложные задачи синхронизации, например, синхронизацию нескольких ресурсов или ограничение доступа к общему ресурсу разным группам процессов.

23. Линки в транспьютере. Диспетчер задач в транспьютере

**Линки в** **транспьютере** - это специальные команды, которые позволяют процессам обмениваться данными и синхронизироваться друг с другом. Линки могут быть однонаправленными или двунаправленными и могут передавать как простые данные, так и сложные структуры данных.

**Диспетчер задач в транспьютере** - это компонент операционной системы, который отвечает за распределение вычислительных ресурсов между процессами и управление их выполнением в соответствии с приоритетами и другими параметрами. Диспетчер задач также отвечает за управление памятью и вводом-выводом.

24. Определение задачи реального времени. Чем системы РВ отличаются от систем разделенного времени? Пример архитектуры ОС реального времени.

**Задача реального времени** - это задача, которая должна быть выполнена в строго заданные сроки. Такие задачи часто встречаются в системах управления и контроля, например, в автоматизированных производственных линиях или системах безопасности.

Системы **реального времени** отличаются от систем **разделенного времени** тем, что они должны гарантировать выполнение задач в строго заданные сроки. Для этого они используют специальные алгоритмы планирования и выделяют вычислительные ресурсы только для задач реального времени, игнорируя все остальные задачи.

**Примером архитектуры** ОС реального времени может служить микроядерная архитектура, где ядро операционной системы содержит только базовые функции, а все остальные компоненты, включая диспетчер задач, работают как отдельные процессы. Это позволяет более гибко управлять выделением ресурсов и обеспечивать более высокую степень надежности и безопасности системы.

25. Журнальные файловые системы. Принципы работы. Для чего это нужно?

**Журнальные файловые системы** - это тип файловой системы, который использует журналирование для обеспечения целостности данных и повышения скорости работы. Они являются одним из наиболее распространенных типов файловых систем в современных операционных системах.

**Принцип работы** журнальных файловых систем заключается в том, что они сохраняют записи о всех изменениях, происходящих с файлами и каталогами, в специальном журнале. Этот журнал является своего рода "черным ящиком", который позволяет восстановить данные в случае сбоя или ошибки.

**Преимущества**

*-Сокращают время восстановления после сбоя*. В обычных файловых системах при сбое может потребоваться длительное время для проверки целостности данных и восстановления поврежденных файлов. В журнальных файловых системах этот процесс происходит намного быстрее, так как данные уже сохранены в журнале.

-*Повышение производительности.* Уменьшают количество операций записи на диск. Вместо того, чтобы каждый раз записывать изменения в файловую систему, они записываются в журнал, а затем применяются к файловой системе в более эффективном порядке.

26. Как реализуется многопоточность на однопроцессорной машине. Что такое контекст процесса? Какие особенности процессора влияют на скорость переключения процессов?

**Многопоточность** на однопроцессорной машине **реализуется** путем переключения контекста процесса. Операционная система распределяет свободное процессорное время между различными процессами, делая видимым для пользователя и программного обеспечения возможность параллельного выполнения различных задач.

**Контекст процесса** —информация о текущем состоянии исполняемого процесса. Переключение между процессами включает сохранение текущего контекста процесса и загрузку контекста нового процесса, которое позволяет операционной системе возвращаться к выполнению каждого процесса в нужное время.

**Особенности процессора**, которые влияют на скорость переключения процессов

1. *Частота процессора*.

2. *Количество ядер*.

3. *Кэш-память*.

4. *Алгоритм планирования*.

5. *Размер страницы памяти* - если размер страницы памяти слишком маленький, то будет больше операций ввода-вывода при переключении процессов, что замедлит работу системы.

27. Методы реализации виртуальной памяти. Базовая адресация, сегментная и страничная виртуальная память.

**Виртуальная память** - это механизм, который позволяет программам использовать больше памяти, чем физически доступно на компьютере. Виртуальная память реализуется с помощью разричных методов.

**Базовая адресация** - это метод виртуальной памяти, который используется в простых системах. В этом методе каждая программа имеет свой базовый адрес, который используется для доступа к памяти. Когда программа загружается в память, она занимает непрерывный блок памяти, начиная с базового адреса.

**Сегментная виртуальная память** - это метод, который разделяет программу на несколько сегментов, каждый из которых может быть загружен в отдельный блок памяти. Каждый сегмент имеет свой базовый адрес, который может быть размещен в любом месте виртуальной памяти.

**Страничная виртуальная память** - это метод, который разделяет программу на блоки фиксированного размера, называемые страницами. Каждая страница имеет свой базовый адрес, который может быть размещен в любом месте виртуальной памяти. Когда программа обращается к адресу памяти, который не загружен в физическую память, операционная система загружает соответствующую страницу в память. Требует меньшей логики управления памятью, чем сегментная виртуальная память.

28. Что такое абсолютный и относительный загрузчики? Структура абсолютного и перемещаемого загрузочных модулей. Что такое позиционно-независимый код?

**Абсолютный загрузчик** загружает программу в память по абсолютным адресам исходного кода. Это означает, что программе требуются фиксированные адреса в памяти для выполнения. Изменение этих адресов после компиляции может потребовать полной перекомпиляции.

**Относительный загрузчик** загружает программу в память по относительным адресам. Это позволяет загружать программы в любые свободные области памяти. Он более гибок и позволяет избежать конфликтов адресации, связанных с абсолютным адресным пространством.

**Абсолютный загрузочный модуль** содержит программный код и данные, которые должны быть загружены в определенные абсолютные адреса памяти. Такая структура является жестко завязанной на адреса, в которые загружается модуль.

**Перемещаемый загрузочный модуль** содержит код и данные, которые могут быть загружены в любые доступные области памяти. Эта структура более гибкая и позволяет осуществлять загрузку кода и данных в различные области памяти, не зависящие от абсолютных адресов.

**Позиционно-независимый** **код** - это тип исполняемого кода, который может быть загружен и выполнен независимо от конкретных адресов памяти. Такой код содержит относительные адреса вместо абсолютных.

29. Устойчивые к сбоям файловые системы. Методы реализации устойчивых ФС.

**Устойчивая файловая система** — это ФС, которая способна сохранять целостность и доступность данных даже в случае сбоев или отказов в работе компьютера или хранилища данных.

**Методы реализации**

1. *Журналирование* – все операции записи в ФС сохраняются в журнале. Если происходит сбой или отказ, то при перезапуске ФС использует журнал для восстановления целостности данных.

2. *Копирование* – данные дублируются на несколько независимых дисков.

3. *Резервное копирование* – данные регулярно копируются на внешнее хранилище.

4. *Проверка целостности данных* – каждый блок данных имеет контрольную сумму. При сбое ФС может использовать контрольные суммы для проверки целостности данных и их восстановления.

5. *Распределенные файловые системы* – данные хранятся на нескольких компьютерах или серверах.

30. Приоритеты процессов и нитей. Управление приоритетами для нитей реального и разделенного времени. Где используется и для чего нужно динамическое изменение приоритета?

**Приоритеты процессов и нитей** определяют порядок их выполнения в системе. Чем выше приоритет, тем больше ресурсов (времени процессора, памяти и т.д.) будет выделено для выполнения данного процесса или нити.

Операционные системы, поддерживающие **нити реального времени**, позволяют разработчикам задавать более точные требования к приоритетам, что позволяет выполнить задачи в строго определенные временные рамки.

**Нити разделенного времени** работают в обычном режиме операционной системы без строгих временных ограничений. Приоритеты нитей разделенного времени также важны, но они менее жестко регламентированы.

**Динамическое изменение приоритетов** имеет целью адаптировать работу процессов или нитей в зависимости от текущей нагрузки на систему или других условий.

**Использование**

1. *Планирование* (смена приоритетов в зависимости от текущей загрузки процессора или других факторов)

2. *Реакция на события* (могут быть временно изменены, чтобы обеспечить более быстрый отклик системы)

3. *Управление ресурсами* (повышение приоритета критически важных задач, понижение приоритета задач, которые потребляют много ресурсов)

31. Понятия инода и связи в файловых системах ОС семейства Unix.

**Инод** - это структура данных в файловой системе Unix, которая содержит метаданные о файле или каталоге, такие как разрешения доступа, владелец, группа, время создания и изменения, размер файла и указатели на блоки данных. Каждый файл или каталог в Unix-системе имеет свой уникальный инод.

**Связь** - это отношение между именем файла и его инодом. В Unix-системах файл может иметь несколько имен (жесткие ссылки), которые указывают на один и тот же инод. Это позволяет создавать различные пути к одному и тому же файлу в файловой системе.

**Символическая связь** - это особый тип связи, который указывает на другой файл или каталог по его имени, вместо использования инода, что позволяет создавать ссылки на файлы и каталоги в разных частях файловой системы или даже на других устройствах.

32. Сигналы в системах семейства Unix.

**Сигналы в системах семейства Unix** - это механизм передачи информации или уведомления процессам и нитям о различных событиях, таких как завершение процесса, ошибка ввода-вывода, прерывание пользователем и т.д.

Каждый сигнал имеет уникальный номер и может быть отправлен процессу или нити, который может зарегистрировать обработчик сигнала, вызванного при его получении.

Сигналы могут быть разделены на две категории: стандартные и пользовательские. Стандартные сигналы имеют заранее определенное значение и используются для определенных целей. Пользовательские могут быть использованы процессом или нитью для передачи информации другому процессу или нити.

33. Почтовые ящики (mailbox) в VAX/VMS.

**В VAX/VMS почтовые ящики (mailbox)** представляют собой специальные файлы, которые используются для хранения и обмена электронными сообщениями. Каждый пользователь имеет свой собственный почтовый ящик, в котором хранятся полученные и отправленные сообщения.

Почтовые ящики в VAX/VMS могут быть доступны как через интерфейс командной строки, так и через специальные почтовые программы. Пользователи могут читать, отправлять и удалять сообщения из своих почтовых ящиков, а также настраивать различные параметры своей электронной почты.

34. Программные каналы (трубы) в системах семейства Unix.

**Программные каналы (трубы) в системах семейства Unix** - это механизм передачи данных между процессами. Они создаются с помощью системного вызова pipe() и представляют собой однонаправленный канал, который может использоваться для передачи данных от одного процесса к другому.

Каналы могут быть созданы как в родительском, так и в дочернем процессе. Данные, записанные в канал одним процессом, могут быть прочитаны другим процессом. Каналы могут использоваться для передачи данных между процессами, запущенными на одной машине, или между процессами, запущенными на разных машинах через сеть.

35. Динамическое выделение памяти в ОС семейства Unix и стандарте POSIX.

Динамическое выделение памяти в операционных системах из семейства Unix, особенно в соответствии со стандартом POSIX, предполагает использование различных механизмов для управления памятью в процессах.

Функция malloc используется для выделения блока памяти определенного размера, а функция calloc - для выделения блока памяти и его инициализации нулевыми значениями. Функция realloc позволяет изменить размер ранее выделенного блока памяти, а функция free освобождает выделенную ранее память.

Динамическое выделение памяти позволяет программам эффективно управлять доступом к памяти и использовать ее по мере необходимости. Однако неправильное использование этих функций может привести к утечкам памяти или ошибкам в работе программы.

36. Флаги событий в RSX и VMS. Что такое AST?

**Флаги событий** в операционных системах RSX и VMS используются для управления асинхронными событиями, такими как завершение операции ввода-вывода или появление новых данных в канале связи. Флаги событий могут быть установлены для определенного устройства или процесса, и когда событие происходит, операционная система уведомляет об этом процесс, выполняющий ожидание на флаге.

**AST** - это механизм, используемый в операционных системах RSX и VMS для обработки асинхронных событий. AST позволяет программистам определить обработчик событий, который будет вызван автоматически при возникновении определенного события, такого как завершение операции ввода-вывода или появление новых данных в канале связи.

37. Разделяемые библиотеки формата ELF.

Формат исполняемых и разделяемых объектных файлов **(ELF)** является стандартным форматом файлов для исполняемых программ, разделяемых библиотек и объектных файлов в операционных системах семейства Unix.

Разделяемые библиотеки в формате ELF представляют собой файлы, содержащие исполняемый код и данные, которые могут быть использованы несколькими программами одновременно. Они загружаются в память при выполнении программы и могут быть общими для нескольких процессов, что позволяет экономить память и упрощает обновление программ. ELF-файлы содержат метаинформацию, необходимую для загрузки и выполнения программы.

38. Механизм setuid в ОС семейства Unix.

Механизм **setuid** в операционной системе семейства Unix позволяет запускать программы от имени другого пользователя, обладающего большими привилегиями, например, администратора системы. Это позволяет пользователю выполнить задачи, которые требуют повышенных прав доступа, необходимых для доступа к определенным файлам или ресурсам.

Однако, использование механизма setuid может быть опасно, если программу запускают от имени пользователя с повышенными привилегиями без должной проверки и контроля. Неправильное использование setuid может привести к возможности выполнения злонамеренного кода или созданию уязвимостей в системе.

39. Загружаемые модули и разделяемые библиотеки Win32/Win64 (PE).

Загружаемые модули и разделяемые библиотеки (DLL) в операционных системах используют формат PE для представления исполняемых файлов, библиотек и других модулей. PE формат предоставляет метаданные, необходимые для загрузки и выполнения программ.

**Разделяемые библиотеки** в формате PE загружаются в память при выполнении программы и могут быть общими для нескольких процессов. Они также содержат информацию о зависимостях от других библиотек и другую метаинформацию.

**Загружаемые модули** являются специальными разделяемыми библиотеками, предназначенными для динамической загрузки приложений во время выполнения. Они позволяют приложениям обмениваться данными и функциями, что способствует эффективному использованию памяти и обеспечивает возможность обновления кода без перекомпиляции основной программы.

40. Права доступа к файлам в ОС семейства Unix.

В операционной системе семейства Unix каждый **файл имеет три типа прав доступа**: чтение, запись и выполнение. Каждый тип прав доступа может быть установлен для трех типов пользователей: владельца файла, группы пользователей и всех остальных пользователей.

Также в Unix существует концепция пользователей и групп пользователей. Каждый пользователь имеет свой уникальный идентификатор пользователя (UID), а каждая группа пользователей имеет свой уникальный идентификатор группы (GID). Пользователь может принадлежать нескольким группам.

41. Запуск задач в ОС семейства Unix.

В операционных системах семейства Unix, используется формат исполняемых файлов ELF. ELF-файлы содержат исполняемый код, данные, таблицы символов, информацию о зависимостях от других библиотек и другую метаинформацию, необходимую для загрузки и выполнения программы. Он обеспечивает гибкость и расширяемость для разработчиков программного обеспечения и позволяет эффективно управлять зависимостями и ресурсами при выполнении программ.

При запуске задач в ОС Unix, ядро операционной системы загружает исполняемый файл в память и начинает его выполнение. Если исполняемый файл зависит от разделяемых библиотек, они также загружаются в память. Разделяемые библиотеки могут быть общими для нескольких процессов, что позволяет экономить память и упрощает обновление кода.

42. Асинхронный ввод-вывод в стандарте POSIX

**Асинхронный ввод-вывод** - это механизм ввода-вывода, который позволяет выполнять операции ввода-вывода без блокировки процесса. В стандарте POSIX асинхронный ввод-вывод реализуется с помощью функций aio\_read, aio\_write и aio\_error.

В POSIX асинхронный ввод-вывод реализуется с помощью следующих функций и структур:

1. *aio\_context\_t* (представляет контекст асинхронного ввода-вывода)

2. *aiocb* (структура используется для инициирования асинхронного запроса на ввод-вывод)

3. *aio\_read, aio\_write* (отправляют запрос на чтение или запись и возвращают управление сразу после того, как запрос на ввод-вывод был отправлен, и не блокируют основной поток)

4. *aio\_return* (позволяет получить результат завершенной операции)

5. *aio\_error* (завершилась ли операция с ошибкой)

6. *Функции обратного вызова* (позволяет обработать результат операции ввода-вывода, когда он станет доступен)

Асинхронный ввод-вывод может быть полезен в приложениях, которые работают с большими объемами данных или с сетевыми соединениями, где операции ввода-вывода могут занимать значительное время. Использование асинхронного ввода-вывода позволяет увеличить производительность приложения и уменьшить задержки.