15. Мультизадачная ОС с вытеснением

Мультизадачная операционная система с вытеснением (preemptive multitasking) — это система, которая позволяет нескольким задачам (процессам или потокам) выполняться одновременно. В отличие от кооперативной мультизадачности, где задача должна добровольно уступить управление, в вытесняющей системе ядро ОС может прервать текущую задачу, чтобы предоставить процессор другим задачам. Это обеспечивает более равномерное распределение ресурсов и повышает отзывчивость системы, особенно в многопользовательских средах.

16. Циклическое планирование

Циклическое планирование (Round Robin Scheduling) — это алгоритм управления процессами, который назначает каждому процессу фиксированный временной интервал (квант времени) для выполнения. Если процесс не завершает свою работу в течение этого времени, он приостанавливается, и управление передается следующему процессу в очереди. Этот метод обеспечивает справедливое распределение процессорного времени между всеми активными процессами и часто используется в многозадачных системах.

17. Приоритетное планирование

Приоритетное планирование — это алгоритм управления процессами, который назначает каждому процессу приоритет. Процессы с более высоким приоритетом получают доступ к процессору раньше, чем процессы с более низким приоритетом. Этот метод позволяет более критически важным задачам выполняться быстрее, но может привести к проблеме "голодания", когда процессы с низким приоритетом никогда не получают возможность выполниться.

18. Кооперативное планирование

Кооперативное планирование (cooperative multitasking) — это метод, при котором процессы управляют своим временем выполнения самостоятельно. Каждая задача должна самостоятельно уступить управление другим задачам. Это может привести к проблемам, если одна из задач не завершает выполнение или зависает, так как другие задачи не смогут получить доступ к процессору.

19. ОС реального времени

Операционная система реального времени (real-time operating system, RTOS) — это система, которая обеспечивает выполнение задач в строго заданные временные рамки. В таких системах важна не только правильность результатов выполнения, но и соблюдение временных ограничений. RTOS используются в критически важных приложениях, таких как системы управления, автоматизация, авиация и медицина.

20. Приоритет процесса

Приоритет процесса — это значение, которое указывает, насколько важно выполнение данного процесса по сравнению с другими процессами. В большинстве ОС более высокие приоритеты означают, что процесс получит доступ к процессору раньше, чем процессы с более низкими приоритетами. Приоритеты могут назначаться вручную или автоматически, в зависимости от нагрузки на систему.

21. Поток уступает процессор другому потоку

Выражение "поток уступает процессор другому потоку" означает, что текущий поток (выполняемая единица работы в рамках процесса) временно прекращает выполнение, позволяя другому потоку (или процессу) получить доступ к процессору. Это может происходить в результате различных механизмов планирования, таких как использование временных квантов в циклическом планировании или в ответ на события, такие как блокировка.

22. Windows: как поток может уступить процессор?

В Windows поток может уступить процессор с помощью системных вызовов, таких как `Sleep`, который приостанавливает выполнение потока на заданный период, или `WaitForSingleObject`, который ожидает завершения определенного события. Также поток может уступить процессор, если он ожидает ввода-вывода или блокирует ресурсы.

23. Windows: базовый приоритет потока, как он вычисляется и диапазон его изменения

Базовый приоритет потока — это начальное значение приоритета, назначаемое потоку при его создании. Он варьируется от 0 до 31, где 0 — это самый низкий приоритет (для системных потоков), а 31 — самый высокий. Приоритет может изменяться в зависимости от других факторов, таких как приоритет родительского процесса или использование системных вызовов для изменения приоритета потока.

24. Windows: назначение и принцип применения системного вызова ResumeThread

Системный вызов `ResumeThread` используется для возобновления выполнения приостановленного потока. Когда поток приостанавливается с помощью `SuspendThread`, его выполнение останавливается, но ресурсы остаются выделенными. Вызов `ResumeThread` увеличивает счетчик, который отслеживает состояние приостановления, позволяя потоку снова начать выполнение.

25. Windows: назначение и принцип применения системного вызова WaitForSingleObject

Системный вызов `WaitForSingleObject` используется для ожидания завершения или сигнализации о событии, связанном с объектом, например, потоком или дескриптором события. Когда поток вызывает этот вызов, он блокируется до тех пор, пока указанный объект не станет доступным или не будет завершен. Это полезно для синхронизации потоков и управления ресурсами.

26. Windows: назначение и принцип применения системных вызовов GetProcessPriorityBoost, GetThreadPriorityBoost, SetProcessPriorityBoost, SetThreadPriorityBoost

- `GetProcessPriorityBoost`: возвращает текущее состояние увеличения приоритета процесса (включено или выключено).

- `GetThreadPriorityBoost`: возвращает текущее состояние увеличения приоритета потока.

- `SetProcessPriorityBoost`: устанавливает состояние увеличения приоритета для процесса, что позволяет процессу временно повышать свой приоритет.

- `SetThreadPriorityBoost`: устанавливает состояние увеличения приоритета для потока. Эти вызовы помогают управлять приоритетами и обеспечивать более справедливое распределение ресурсов.

27. Linux: принцип идентификации процессов и потоков

В Linux процессы и потоки идентифицируются с помощью уникальных идентификаторов процессов (PID). Потоки реализуются как легковесные процессы (LWP) с общими ресурсами, такими как память и файловые дескрипторы. Каждому потоку назначается свой уникальный PID, а также он наследует PID родительского процесса. Это позволяет эффективно управлять и синхронизировать выполнение процессов и потоков.

28. Linux: планировщик потоков

Планировщик потоков в Linux — это компонент ядра, который управляет распределением процессорного времени между потоками и процессами. Он использует различные алгоритмы планирования (например, CFS — Completely Fair Scheduler), чтобы обеспечить эффективное выполнение задач с учетом их приоритетов и ресурсов. Планировщик также обрабатывает переключение контекста между потоками, минимизируя накладные расходы и задержки.

29. Linux: принцип использования значения nice

Значение `nice` — это параметр, который влияет на приоритет процесса. Оно варьируется от -20 (максимальный приоритет) до 19 (минимальный приоритет). Значение `nice` применяется в планировщике процессов для определения, как много процессорного времени получает данный процесс по сравнению с другими процессами. Чем ниже значение `nice`, тем выше приоритет процесса.

30. Linux: политики планирования, политика по умолчанию

В Linux существует несколько политик планирования, включая:

- \*\*CFS (Completely Fair Scheduler)\*\* — по умолчанию для обычных процессов.

- \*\*FIFO (First In, First Out)\*\* — для процессов реального времени.

- \*\*RR (Round Robin)\*\* — для потоков реального времени.

По умолчанию используется CFS для всех процессов, что обеспечивает справедливое распределение ресурсов.

31. Linux: системный вызов, с помощью которого поток может уступить процессор

В Linux поток может уступить процессор с помощью системного вызова `sched\_yield`. Этот вызов сообщает планировщику, что текущий поток готов уступить процессор другим потокам с тем же приоритетом. Это позволяет улучшить производительность в многопоточных приложениях, давая другим потокам возможность выполниться.