27. Мультизадачная операционная система с вытеснением (Preemptive Multitasking OS) позволяет выполнять несколько процессов или задач одновременно. Этот тип операционной системы имеет механизм, позволяющий системе автоматически выталкивать (вытеснять) текущий выполняющийся процесс с процессора, чтобы предоставить процессор другому процессу с более высоким приоритетом. Это обеспечивает более справедливое распределение ресурсов процессора между процессами и позволяет обеспечивать реактивность системы.

28. Циклическое планирование (Round Robin Scheduling) - это метод планирования задач в мультизадачной системе, при котором каждой задаче предоставляется определенное квант времени для выполнения на процессоре. После истечения кванта времени задача переходит в конец очереди готовых задач, и процессор переходит к выполнению следующей задачи в очереди. Этот процесс повторяется в цикле. Циклическое планирование обеспечивает справедливое распределение времени процессора между задачами, но может привести к нежелательным задержкам, особенно если некоторые задачи требуют больше времени для выполнения, чем другие.

29. Приоритетное планирование (Priority Scheduling) - это метод планирования задач, при котором каждой задаче присваивается приоритет, и задачи выполняются в порядке приоритета. Задачи с более высоким приоритетом имеют приоритет при использовании процессора, что позволяет им завершать выполнение быстрее. Приоритеты могут быть статическими или динамически изменяемыми.

30. Кооперативное планирование (Cooperative Scheduling) - это метод планирования задач, при котором задачи сами управляют переключением контекста и освобождают процессор для других задач. Это означает, что задачи должны явно уступать процессор другим задачам, и в противном случае они могут заблокировать выполнение других задач. Кооперативное планирование может быть менее надежным, чем вытесняющее планирование, так как оно зависит от сотрудничества задач.

31. Операционная система реального времени (Real-time OS) - это операционная система, предназначенная для выполнения задач в строго определенные временные сроки. В системах реального времени задачи должны быть выполнены точно и вовремя, и задержки могут иметь критические последствия. ОС реального времени предназначена для обработки таких задач и обеспечения предсказуемости выполнения.

32. Приоритет процесса - это числовое значение, которое определяет, насколько важен процесс в контексте выполнения задач в операционной системе. Процесс с более высоким приоритетом получает больше времени на процессоре и завершает выполнение быстрее, чем процессы с более низким приоритетом.

33. Выражение "поток уступает процессор другому потоку" означает, что текущий исполняемый поток (или процесс) завершает свою работу и освобождает процессор для выполнения другого потока (или процесса). Это может произойти по истечении кванта времени (в случае циклического планирования) или при явной передаче управления другому потоку.

34. В Windows поток может уступить процессор, вызвав функцию `Sleep` или `SwitchToThread`. `Sleep` приостанавливает выполнение текущего потока на определенное количество миллисекунд, а `SwitchToThread` пытается передать управление другим готовым потокам.

35. Базовый приоритет потока в Windows - это числовое значение, которое определяет приоритет выполнения потока. Он вычисляется как сумма приоритета процесса и приоритета потока. Диапазон изменения базового приоритета находится между 0 и 31.

36. `SetThreadIdealProcessor` - это системный вызов Windows, который позволяет установить предпочтительный процессор (ядерный номер) для выполнения указанного потока. Это может быть полезно для оптимизации производительности и управления аффинностью потока к конкретным процессорам.

37. `ResumeThread` - это системный вызов Windows, который возобновляет выполнение приостановленного потока.

38. `WaitForSingleObject` - это системный вызов Windows, который приостанавливает выполнение потока до тех пор, пока указанный объект (например, мьютекс или семафор) не станет сигнализированным или не истечет заданный таймаут.

39. Системные вызовы `GetProcessPriorityBoost`, `GetThreadPriorityBoost`, `SetProcessPriorityBoost` и `SetThreadPriorityBoost` используются для управления приоритетом процессов и потоков в Windows. Они позволяют устанавливать и получать информацию о повышении приоритета для процессов и потоков.

40. В Linux процессы и потоки идентифицируются с помощью их уникальных идентификаторов PID (Process ID) и TID (Thread ID). Процессы в Linux имеют свои собственные адресные пространства, в то время как потоки могут совместно использовать адресное пространство внутри процесса.

41. Планировщик потоков (Thread scheduler) в Linux управляет выполнением потоков внутри процессов. Он решает, какой поток должен получить доступ к процессору в каждый момент времени.

42. Значение `nice` в Linux используется для управления приоритетом процесса. Чем выше значение `nice`, тем ниже приоритет у процесса. Диапазон изменения `nice` обычно составляет от -20 до 19. Низкое значение `nice` предоставляет процессу более высокий приоритет, а высокое значение `nice` уменьшает приоритет процесса.

43. В Linux существует несколько политик планирования, включая SCHED\_OTHER (по умолчанию), SCHED\_FIFO и SCHED\_RR.

44. Для определения действующей политики планирования для процесса в Linux, можно использовать файл `/proc/<PID>/sched`. Здесь `<PID>` - идентификатор процесса. В файле можно найти информацию о текущей политике планирования и параметрах.

45. В Linux поток может уступить процессор, вызвав системный вызов `sched\_yield`.

46. Системный вызов `nice` используется для изменения приоритета процесса, в то время как системный вызов `setpriority` позволяет установить приоритет процесса или потока. `setpriority` более общий, так как может использоваться для установки приоритетов для разных классов ресурсов.

Планировщик ввода/вывода (I/O Scheduler) в операционной системе управляет доступом к устройствам ввода/вывода, таким как жесткие диски, SSD-накопители, USB-устройства и другие. Он определяет порядок выполнения операций ввода/вывода для эффективного использования ресурсов устройств и улучшения производительности системы.

### Как узнать доступные планировщики ввода/вывода в Linux:

На системах Linux можно узнать доступные планировщики ввода/вывода с помощью командной строки и файла `/sys`.

### Для перечисления доступных планировщиков ввода/вывода в Linux:

```bash

cat /sys/block/<device>/queue/scheduler

```

Где `<device>` - это имя блочного устройства (например, sda, sdb и т.д.).

### Некоторые известные планировщики ввода/вывода в Linux:

1. \*\*noop\*\*: Простейший планировщик, который не выполняет никаких операций планирования. Используется для устройств, которые уже имеют собственные механизмы планирования, например, некоторые SSD.

2. \*\*cfq (Completely Fair Queuing)\*\*: Ранее был основным планировщиком ввода/вывода для Linux. Имеет функции планирования с учетом времени ожидания запросов ввода/вывода.

3. \*\*deadline\*\*: Основная идея этого планировщика - гарантировать, что запросы ввода/вывода будут выполнены в определенном временном окне. Это может быть полезно для систем, где важна низкая задержка.

4. \*\*bfq (Budget Fair Queuing)\*\*: Планировщик, который стремится обеспечить справедливое распределение времени доступа к устройствам, а также пытается уменьшить задержку для интерактивных приложений.

### Определение действующего планировщика для блочного устройства в Linux:

```bash

cat /sys/block/<device>/queue/scheduler

```

Это покажет текущий используемый планировщик ввода/вывода для указанного блочного устройства.

Например:

```bash

cat /sys/block/sda/queue/scheduler

```

Эта команда выведет текущий планировщик ввода/вывода для устройства sda.