**affinity mask** - это [битовая маска](https://en.wikipedia.org/wiki/Mask_(computing)), указывающая, на каком [процессоре(процессорах)](https://en.wikipedia.org/wiki/Central_processing_unit) [должен выполняться поток или](https://en.wikipedia.org/wiki/Process_(computing))процесс [планировщиком](https://en.wikipedia.org/wiki/Scheduling_(computing)) [операционной системы](https://en.wikipedia.org/wiki/Operating_system). Установка маски сходства для определенных процессов, работающих под управлением Windows, может быть полезна, поскольку существует несколько системных процессов (особенно на контроллерах домена), которые ограничены первым процессором / ядром. Таким образом, исключение первого процессора может привести к повышению производительности приложения.

1. **Поясните понятие «мультизадачная OS с вытеснением».**

ОС, которая выполняет несколько задач одновременно, выделяет квант времени, определяет, когда это время истекает и переключает задачу.

1. **Поясните понятие «циклическое планирование».**

Одним из самых старых, простых, справедливых и наиболее часто используемых считается алгоритм **циклического планирования**. Каждому процессу назначается определенный интервал времени, называемый его **квантом**, в течение которого ему предоставляется возможность выполнения. Если процесс к завершению кванта времени все еще выполняется, то ресурс центрального процессора у него отбирается и передается другому процессу. Разумеется, если процесс переходит в заблокированное состояние или завершает свою работу до истечения кванта времени, то переключение центрального процессора на другой процесс происходит именно в этот момент.

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

1. **Поясните понятие «приоритетное планирование».**

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

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

1. **Поясните понятие «кооперативное планирование».**

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

1. **Поясните понятие «OS реального времени».**

Система, которая гарантирует фиксированное время для выполнения задачи.

Если операция *должна* быть проведена точно в срок (или в определенный период времени), то мы имеем дело с **системой жесткого реального времени**. Другой разновидностью подобных систем является **система мягкого реального времени**, в которой хотя и нежелательно, но вполне допустимо несоблюдение срока какого-нибудь действия, что не наносит непоправимого вреда.

1. **Поясните понятие «приоритет процесса».**

**приоритет**[**процесса**](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) — число, ориентируясь на значение которого [планировщик процессов](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D1%81%D0%BF%D0%B5%D1%82%D1%87%D0%B5%D1%80_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%BE%D0%B9_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D1%8B) может выдавать [процессу](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) больше или меньше [процессорного](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%BE%D1%80) [времени](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%BE%D1%80%D0%BD%D0%BE%D0%B5_%D0%B2%D1%80%D0%B5%D0%BC%D1%8F).

Это значение, которое влияет на выбор процесса, который будет выполняться следующим.

1. **Поясните выражение «поток уступает процессор другому потоку».**

Это значит, что текущий поток прерывает свое выполнение. При этом управление процессором передается следующему из очереди потоку.

1. **Windows: как поток может уступить процессор?**

Sleep(0) – процесс становится в конец очереди.

1. **Windows: что такое базовый приоритет потока, как он вычисляется и диапазон его изменения?**

Базовый приоритет потока – сочетание класса приоритета процесса и приоритета потока, изменяется в пределах [1,31], по умолчанию – 8, приоритеты возрастающие;

так вычисляется:



1. **Windows: поясните назначение и принцип применения системного вызова SetThreadIdealProcessor.**

SetThreadIdealProcessor устанавливает предпочтительный процессор для потока, т.е. процессор на котором он будет работать.

DWORD SetThreadIdealProcessor(

HANDLE hThread, // дескриптор потока

DWORD dwIdealProcessor // номер идеального процессора

);

dwIdealProcessor - Номер привилегированного процессора для потока. Значение MAXIMUM\_PROCESSORS сообщает системе, что поток не имеет привилегированного процессора.

Если функция завершается успешно, величина возвращаемого значения - предшествующий привилегированный процессор или MAXIMUM\_PROCESSORS, если поток не имеет такового.

Если функция завершается с ошибкой, величина возвращаемого значения равна - (минус) 1

1. **Windows: поясните назначение и принцип применения системного вызова ResumeThread.**

Запускает поток, после его системного вызова SuspendThread.

DWORD ResumeThread (

HANDLE hThread // дескриптор потока

);

Уменьшает счетчик приостановленных потоков. Когда счетчик приостановки уменьшается до нуля, выполнение потока возобновляется.

1. **Windows: поясните назначение и принцип применения системного вызова WaitForSingleObject.**

DWORD WaitForSingleObject(

HANDLE **hHandle**, //Дескриптор потока.

DWORD **dwMilliseconds** //Интервал ожидания в миллисекундах.(если INFINITE - то функция вернет только тогда , когда объект сигнализируется.)

);

Ожидает, пока указанный объект не перейдет в сигнальное состояние или пока не истечет интервал времени ожидания.

1. **Windows: поясните назначение и принцип применения системных вызовов GetProcessPriorityBoost, GetThreadPriorityBoost, SetProcessPriorityBoost, SetThreadPriorityBoost.**

GetProcessPriorityBoost, GetThreadPriorityBoost возвращает true/false в зависимости от того, разрешено ли динамическое изменение приоритета процесса/потока.

SetProcessPriorityBoost, SetThreadPriorityBoost задают это разрешение.

1. **Linux: поясните принцип идентификации процессов и потоков и поясните почему он такой.**

Номер процесса всегда равен номеру главного потока, потому что ядро Линукса не различает потоки и процессы.

1. **Linux: Поясните понятие «планировщик потоков».**

Это алгоритм, который определяет порядок выполнения потоков.

Планировщик — это часть ядра, которая решает какая запущенная нить будет выполняться процессором следующей.

Планировщик потоков ядра отвечает за то, какие потоки выполняются на процессорах системы. Каждый поток выполняется до тех пор, пока прерывание таймера не сигнализирует о том, что пора переключаться на другой поток (квант закончился), или до тех пор, когда потоку нужно ждать какого-то события (завершения ввода-вывода или снятия блокировки), либо до тех пор, пока работоспособным не станет поток с более высоким приоритетом (которому требуется процессор). При переключении с одного потока на другой планировщик обеспечивает сохранение регистров и прочего состояния оборудования. Затем планировщик выбирает для выполнения на процессоре другой поток и восстанавливает ранее сохраненное состояние (для выбранного потока).

1. **Linux: поясните принцип использования значения nice –процесса, диапазон его изменения, для какого режима работы планировщика это значение применяется?**

nice – значение любезности, чем меньше, тем выше приоритет

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

- nice – диапазон приоритетов [-20, 19], default = 0;(это у Смелова, но вроде есть инфа что 10)

Чтобы установить значение nice ниже нуля, требуются права суперпользователя

РЕЖИМ РАБОТЫ: вытесняющий(other) (Планировщик может принудительно забирать управление у потока (например по таймеру или при появлении потока с большим приоритетом))

1. **Linux: перечислите политики планирования, какая действует по умолчанию?**

Linux, политики планирования процесса: 1) стандартная (OTHER, разделения времени); - по умолчанию 2) FIFO-политика (FIFO, реального времени); 3) карусельная (round-robin) политика (RR); 4) пакетная политика(BATCH).

1. **Linux: как выяснить действующую политику планирования для процесса с помощью файловой системы proc?**

/proc/PID/sched (в строчке police будет)

0- OTHER

1-FIFO

2-RR

1. **Linux: с помощью какого системного вызова поток может уступить процессор.**

уступить процессор shed\_yield()

1. **Linux: чем отличается системный вызов nice от вызова setpriority.**

nice прибавляет к текущему значению любезности nice аргумент, setpriority устанавливает значение nice

1. **Linux: поясните понятие «планировщик ввода вывода», каким образом можно выяснить какие планировщики ввода/ вывода доступны?**

планировщики ввода/вывода (I/O) – программная прослойка между блочными устройствами (дисковые устройства) и низкоуровневыми драйверами ввода/вывода.

Задача планировщика – оптимизация доступа процесса к дисковому устройству (уменьшение времени поиска данных, обеспечение приоритетности, гарантировать данные за заданное время), лифтовые алгоритмы (elevator) – операции в порядке чтения (записи) ближайших секторов.

dmesg | grep scheduler

1. **Linux: перечислите известные вам планировщики ввода/ вывода, кратко охарактеризуйте их.**

NOOP – простой I/O-планировщик, общая FIFO- очередь read/write-запросов, объединяет однотипные запросы для сокращения операций.

CFQ (Completely Fair Queueing, наиболее справедливая очередь) – у каждого процесса своя очередь, у каждой очереди свой квант времени, планировщик циклически обходит очереди, обслуживает очередь в течении кванта, чтение в очереди имеет приоритет. Поддержка i/o-приоритетов.

BFQ (Budget Fair Queueing) – базируется на CFQ, каждой CFQ-очереди выделяется бюджет, который растет для процессов с интенсивным вводом/выводом.

Deadline - I/O-планировщик пытается выполнить запрос в указанное время, две очереди read и write, read-очередь приоритетнее, запросы объединяются в пакеты по принципу «алгоритма лифта».

MQ-Deadline – модификация Deadline для новых устройств.

Kyber - для работы с быстрыми устройствами, две очереди read и write, read-очередь приоритетнее измеряетcя время завершения каждого запроса и корректирует фактический размер очереди для достижения установленных в настройках задержек.

1. **Linux: каким образом можно выяснить тип планировщика действующего для блокового устройства?**

cat /sys/block/hda/queue/scheduler