UNIX

Функции системного таймера в защищенном режиме ОС UNIX

- **По тику**:
  - Инкремент часов и других таймеров системы
  - Инкремент поля p_cpu дескриптора текущего процесса на единицу, до максимального значения, равного 127
  - Инкремент счетчика тиков аппаратного таймера
  - Декремент счетчика времени, оставшегося до отправления на выполнение отложенных вызовов
  - Декремент кванта текущего потока

- **По главному тику**:
  - Инициализация отложенных вызовов функций, относящихся к работе планировщика, таких как пересчет приоритетов. Инициализация отложенного вызова заключается в посылке соответствующего сигнала или в изменении состояния процесса с S на R.
  - Декремент времени, оставшегося до отправления одного из сигналов:
    - SIGALARM(сигнал по истечении времени, предварительно заданного функцией alarm())
    - SIGPROF (сигнал, посылаемый процессу по истечении времени заданном в таймере профилирования)
    - SIGVTALARM (сигнал, посылаемый процессу по истечении времени, заданного в «виртуальном» таймере)
  - Пробуждение в нужный момент системных процессов, таких как swapper и pagedaemon путём инициализации отложенного вызова процедуры wakeup

- **По кванту**:
  - Посылка текущему процессу сигнала SIGXCPU, если тот превысил выделенный ему квант использования процессора. По получению сигнала обработчик сигнала прерывает выполнение процесса.

WINDOWS

Функции системного таймера в защищенном режиме ОС WINDOWS

Всего в ОС Windows 32 уровня запроса прерывания (от 0 до 31). Прерывания обслуживаются в порядке их приоритета. У интервального таймера системных часов высокое значение IRQL – CLOCK_LEVEL):

- **По тику**:
  - Инкремент счетчика системного времени
  - Декремент счетчиков времени отложенных задач
  - Декремент кванта текущего потока
  - Если активен механизм профилирования ядра, то инициализация отложенного вызова обработчика ловушки профилирования ядра путем постановки объекта в очередь DPC (обработчик ловушки профилирования регистрирует адрес команды, выполнявшейся на момент прерывания)
- **По главному тику**:
  - Освобождение объекта «событие», которое ожидает диспетчер настройки баланса. Диспетчер настройки баланса по событию от таймера сканирует очередь готовых процессов и повышает приоритет процессов, которые находились в состоянии ожидания дольше 4 секунд.
- **По кванту**:
  - Инициация диспетчеризации потоков (добавление соответствующего объекта в очередь DPC)

UNIX

Пересчет динамических приоритетов

Пересчитываться могут только приоритеты пользовательских процессов.

Современные алгоритмы планирования UNIX обеспечивают возможность одновременного выполнения интерактивных и фоновых приложений. Они хорошо подходят для систем общего назначения с несколькими подключенными пользователями. Эти алгоритмы обеспечивают малое время реакции для интерактивных приложений.

Планирование процессов в UNIX основано на приоритете процесса. Планировщик всегда выбирает процесс с наивысшим приоритетом. Приоритет процесса не является фиксированным и динамически изменяется системой в зависимости от использования вычислительных ресурсов, времени ожидания запуска и текущего состояния процесса. Если процесс готов к запуску и имеет наивысший приоритет, планировщик приостановит выполнение текущего процесса (с более низким приоритетом), даже если последний не «выработал» свой временной квант.

В традиционных системах Unix ядро является строго невытесняющим, однако в современных системах Unix ядро является вытесняющим – процесс в режиме ядра может быть вытеснен более приоритетным процессом в режиме ядра. Ядро было сделано вытесняющим для того, чтобы система могла обслуживать процессы реального времени, такие как аудио и видео.

Приоритет процесса задается любым целым числом, лежащим в диапазоне от 0 до 127. Чем меньше такое число, тем выше приоритет. Приоритеты от 0 до 49 зарезервированы для ядра, следовательно, прикладные процессы могут обладать приоритетом в диапазоне 50-127.

Структура proc содержит следующие поля, относящиеся к приоритетам:

- p_pri - текущий приоритет планирования
- p_usrpri - приоритет режима задачи
- p_cpu - результат последнего измерения использования процессора
- p_nice - фактор «любезности», устанавливаемый пользователем

Планировщик использует p_pri для принятия решения о том, какой процесс направить на выполнение. Когда процесс находится в режиме задачи, значение его p_pri идентично p_usrpri. Когда процесс просыпается после блокирования в системном вызове, его приоритет будет временно повышен для того, чтобы дать ему предпочтение для выполнения в режиме ядра. Следовательно, планировщик использует p_usrpri для хранения приоритета, который будет назначен процессу при возврате в режим задачи, а p_pri — для хранения временного приоритета для выполнения в режиме ядра.

Когда процесс завершил выполнение системного вызова и находится в состоянии возврата в режим задачи, его приоритет сбрасывается обратно в значение текущего приоритета в режиме задачи. Измененный таким образом приоритет может оказаться ниже, чем приоритет какого-либо иного запущенного процесса; в этом случае ядро системы произведет переключение контекста.

Процессу, ожидающему недоступный в данный момент ресурс, система определяет значение, выбираемое ядром из диапазона системных приоритетов и связанное с событием, вызвавшее это состояние.

В таблице приведены значения приоритетов сна для систем UNIX и SCO UNIX 5.0. Направление роста значений приоритета для этих систем различно — в BSD UNIX большему значению соответствует более низкий приоритет.

Событие	Приоритет 4.3BSD UNIX	Приоритет SCO UNIX
Ожидание загрузки в память сегмента/страницы (свопинг/страничное замещение)	0	95
Ожидание индексного дескриптора	10	88
Ожидание ввода/вывода	20	81
Ожидание буфера	30	80
Ожидание терминального		75

ввода		
Ожидание терминального вывода		74
Ожидание завершения выполнения		73
Ожидание события — низкоприоритетное состояние сна	40	66

Когда процесс пробуждается, ядро устанавливает значение текущего приоритета процесса равный приоритету сна. Поскольку приоритет такого процесса находится в системном диапазоне и выше, чем приоритет режима задачи, вероятность предоставления процессу вычислительных ресурсов весьма велика. Такой подход позволяет, в частности, быстро завершить системный вызов, выполнение которого, в свою очередь, может блокировать некоторые системные ресурсы.

Приоритет в режиме задачи зависит от двух факторов:

- Фактора «любезности» (*nice*).Степень любезности (*nice value*) является числом в диапазоне от 0 до 39 со значением 20 по умолчанию. Увеличение значения приводит к уменьшению приоритета. Фоновым процессам автоматически задаются более высокие значения. Уменьшить эту величину для какого-либо процесса может только суперпользователь, поскольку при этом поднимется его приоритет.
- Степени загруженности процессора в момент последнего обслуживания им процесса.

Системы разделения времени пытаются выделить процессорное время таким образом, чтобы конкурирующие процессы получили его примерно в равных количествах. Такой подход требует слежения за использованием процессора каждым из процессов. Поле ***p_cpu*** структуры ***proc*** содержит величину результата последнего сделанного измерения использования процессора процессом. При создании процесса значение этого поля инициализируется нулем. На каждом тике обработчик таймера увеличивает

p_cpu на единицу для текущего процесса до максимального значения, равного 127.

Каждую секунду ядро системы вызывает процедуру *schedcpu()* (запускаемую через отложенный вызов), которая уменьшает значение ***p_cpu*** каждого процесса исходя из фактора «полураспада» (*decay factor*).

В системе SVR3 фактор полураспада равен 0.5.

В 4.3BSD для расчета фактора полураспада применяется следующая формула:

$$decay = \frac{2 * load\_average}{2 * load\_average + 1}$$

где *load_average* — это среднее количество процессов, находящихся в состоянии готовности к выполнению, за последнюю секунду.

Процедура *schedcpu()* также пересчитывает приоритеты для режима задачи всех процессов по формуле

$$p\_usrpri = PUSER + \frac{p\_cpu}{4} + 2 * p\_nice$$

где *PUSER* — базовый приоритет в режиме задачи, равный 50.

В результате, если процесс в последний раз использовал большое количество процессорного времени, его ***p_cpu*** будет увеличен. Это приведет к росту значения *p_usrpri* и, следовательно, к понижению приоритета. Чем дольше процесс простаивает в очереди на выполнение, тем больше фактор полураспада уменьшает его ***p_cpu***, что приводит к повышению его приоритета. Такая схема предотвращает бесконечное откладывание низкоприоритетных процессов.

Таким образом, приоритет процесса в режиме задачи может быть динамически пересчитан по следующим причинам:

- Вследствие изменения фактора любезности процесса системным вызовом nice

- В зависимости от степени загруженности процессора процессом p_cpu
- Вследствие ожидания процесса в очереди готовых к выполнению процессов
- Приоритет может быть повышен до соответствующего приоритета сна вследствие ожидания ресурса или события
- Процесс может быть переведен из режима задачи в режим ядра, в этом случае его текущий приоритет назначается из диапазона режима ядра

Динамический пересчет приоритетов процессов в режиме задачи позволяет избежать бесконечного откладывания.

WINDOWS

В Windows при создании процесса ему назначается приоритет и он обычно называется базовым приоритетом процесса. Приоритет потока вычисляется относительно базового приоритета процесса.

Пересчет динамических приоритетов

Пересчитываться могут только приоритеты пользовательских процессов.

В Windows реализуется приоритетная, вытесняющая система планирования, при которой всегда выполняется хотя бы один работоспособный (готовый) поток с самым высоким приоритетом. Windows осуществляет планировку потоков по вытесняющему принципу. Такой подход имеет смысл, если учесть, что процессы не запускаются, а только предоставляют ресурсы и контекст, в котором запускаются потоки.

Код Windows, отвечающий за планирование, реализован в ядре. Поскольку этот код рассредоточен по ядру, единого модуля или процедуры с названием «планировщик» нет. Совокупность процедур, выполняющих эти обязанности, называется *диспетчерам ядра*. Диспетчеризация потоков может быть вызвана любым из следующих событий:

- Поток готов к выполнению – например, он только что создан или вышел из состояния ожидания.
- Поток выходит из состояния Running (выполняется), так как его квант истек или поток завершается либо переходит в состояние ожидания.
- Приоритет потока изменяется в результате вызова системного сервиса или самой Windows.
- Изменяется привязка к процессорам, из-за чего поток больше не может работать на процессоре, на котором он выполнялся.

В Windows используется 32 уровня приоритета: целое число от 0 до 31, где 31 – наивысший приоритет, из них:

- от 16 до 31 – уровни реального времени
- от 0 до 15 – динамические уровни, уровень 0 зарезервирован для потока обнуления страниц

Уровни приоритета потоков назначаются исходя из двух разных позиций: одной от Windows API и другой от ядра Windows. Сначала Windows API систематизирует процессы по классу приоритета, который им присваивается при создании:

- Реального времени — Real-time (4)
- Высокий — High (3)
- Выше обычного — Above Normal (7)
- Обычный — Normal (2)
- Ниже обычного — Below Normal (5)
- Простоя — Idle (1)

Затем назначается относительный приоритет отдельных потоков внутри этих процессов. Здесь номера представляют изменение приоритета, применяющееся к базовому приоритету процесса:

- Критичный по времени — Time-critical (15)
- Наивысший — Highest (2)
- Выше обычного — Above-normal (1)
- Обычный — Normal (0)
- Ниже обычного — Below-normal (–1)

- Самый низший — Lowest (–2)
- Простоя — Idle (–15)

При назначении потоку уровня «критичный по времени» поток получает наивысший возможный для класса его процесса приоритет:

- 31 - для потока процесса уровня реального времени
- 15 - для потока процесса динамического уровня

При назначении потоку уровня простоя поток получает самый низкий возможный для класса его процесса приоритет:

- 16 - для потока процесса уровня реального времени
- 1 - для потока процесса динамического уровня

Поток характеризуют текущее(динамическое) и базовое значения приоритета.

Текущий приоритет потока в динамическом диапазоне – от 1 до 15 – может быть повышен планировщиком вследствие следующих причин.

- **Повышение вследствие событий планировщика или диспетчера (сокращение задержек)**
  При наступлении события диспетчера вызываются процедуры с целью проверить не должны ли на локальном процессоре быть намечены какиелибо потоки, которые не должны быть спланированы.

- **Повышения приоритета, связанные с завершением ожидания**
  Повышения приоритета, связанные с завершением ожидания (unwait boosts), пытаются уменьшить время задержки между потоком, пробуждающимся по сигналу объекта (переходя тем самым в состояние Ready), и потоком, фактически приступившим к своему выполнению в процессе, который не находился в состоянии ожидания (переходя тем самым в состояние Running).

- **Повышение приоритета владельца блокировки**
  Поскольку блокировки ресурсов исполняющей системы (ERESOURCE) и

блокировки критических разделов используют основные объекты диспетчеризации, в результате освобождения этих блокировок осуществляются повышения приоритета, связанные с завершением ожидания. С другой стороны, поскольку высокоуровневые реализации этих объектов отслеживают владельца блокировки, ядро может принять более взвешенное решение о том, какого вида повышение должно быть применено с помощью AdjustBoost.

- **Повышение вследствие завершения ввода-вывода (сокращение задержек)**
  Windows дает временное повышение приоритета при завершении определенных операций ввода-вывода, при этом потоки, ожидавшие ввода-вывода, имеют больше шансов сразу же запуститься и обработать то, чего они ожидали.

  Рекомендуемые приращения приоритета:

Устройство	Приращение
Диск, CD-ROM, параллельный порт, видео	1
Сеть, почтовый ящик, именованный канал, последовательный порт	2
Клавиатура, мышь	6
Звуковая плата	8

Приоритет потока всегда повышается относительно базового, а не текущего уровня.

- **Повышение при ожидании ресурсов исполняющей системы**
  Когда поток пытается получить ресурс исполняющей системы (ERESOURCE), который уже находится в исключительном владении другого потока, он должен войти в состояние ожидания до тех пор, пока другой поток не освободит ресурс. Для ограничения риска взаимных исключений исполняющая система выполняет это ожидание, не входя в бесконечное ожидание ресурса, а интервалами по пять секунд. Если по окончании этих пяти секунд ресурс все еще находится

во владении, исполняющая система пытается предотвратить зависание центрального процессора путем завладения блокировкой диспетчера, повышения приоритета потока или потоков, владеющих ресурсом, до значения 14 (если исходный приоритет владельца был меньше, чем у ожидающего, и не был равен 14), перезапуска их квантов и выполнения еще одного ожидания.

- **Повышение приоритета потоков первого плана после ожидания** Смысл этого повышения заключается в улучшении отзывчивости интерактивных приложений: если дать приложениям первого плана небольшое повышение приоритета при завершении ожидания, у них повышаются шансы сразу же приступить к работе, особенно когда другие процессы с таким же базовым приоритетом могут быть запущены в фоновом режиме.

- **Повышение приоритета после пробуждения GUI-потока** Потоки-владельцы окон получают при пробуждении дополнительное повышение приоритета на 2 из-за активности системы работы с окнами, например поступление сообщений от окна. Система работы с окнами (Win32k.sys) применяет это повышение приоритета, когда вызывает функцию KeSetEvent для установки события, используемого для пробуждения GUI-потока. Смысл этого повышения похож на смысл предыдущего повышения — содействие интерактивным приложениям.

- **Повышения приоритета, связанные с перезагруженностью центрального процессора.**
  В Windows также включен общий механизм ослабления загруженности центрального процессора, который называется диспетчером настройки баланса и является частью потока (речь идет о системном потоке, который существует главным образом для выполнения функций управления памятью). Один раз в секунду этот поток сканирует очередь готовых потоков в поиске тех из них, которые находятся в состоянии ожидания (то есть не были запущены) около 4 секунд. Если такой поток будет найден, диспетчер настройки баланса повышает его приоритет до 15 единиц и устанавливает квантовую цель

эквивалентной тактовой частоте процессора при подсчете 3 квантовых единиц. Как только квант истекает, приоритет потока тут же снижается до обычного базового приоритета. Если поток не был завершен и есть готовый к запуску поток с более высоким уровнем приоритета, поток с пониженным приоритетом возвращается в очередь готовых потоков, где он опять становится подходящим для еще одного повышения приоритета, если будет оставаться в очереди следующие 4 секунды.

- **Повышение приоритетов для мультимедийных приложений и игр**

  В клиентские версии Windows была внедрена служба MMCSS, чьей целью было гарантировать проигрывание мультимедийного контента приложений, зарегистрированных с этой службой, без каких-либо сбоев.

MMCSS работает с вполне определенными задачами, включая следующие:

- аудио
- игры
- проигрывание
- аудио профессионального качества
- задачи администратора многооконного режима

Функции MMCSS временно повышают приоритет потоков, зарегистрированных с MMCSS до уровня, соответствующего их категориям планирования. Затем, их приоритет снижается до уровня, соответствующего категории Exhausted, чтобы другие потоки также могли получить ресурс.

Windows никогда не повышает приоритет потоков в диапазоне реального времени (16-31).

Текущий приоритет потока в динамическом диапазоне может быть понижен до базового приоритета путем вычитания всех повышений.

Вывод

Для семейства ОС Windows и для семейства ОС Unix/Linux функции обработчика прерывания от системного таймера в защищенном режиме схожи так как эти ОС являются системами разделения времени. Общие функции обработчика:

1. Инициализируют отложенные действия, относящиеся к работе планировщика, такие как пересчет приоритетов

2. Выполняют декремент счетчиков времени: часов, таймеров, будильников реального времени, счетчиков времени отложенных действий

3. Выполняют декремент кванта.

Что казается планирования в операционных системах, то классическое ядро UNIX является строго невытесняемым. Windows является полностью вытесняемой. Динамический пересчет приоритетов пользовательских процессов используется прежде всего для избежания проблемы бесконечного откладывания.

Также есть схожесть в планировщиках задач: в основе работы лежит взаимодействие с очередями, подчиняясь определенным правилам, могут изменяться.