# Функции обработчика прерывания от системного таймера в системах разделения времени

## ОС семейства Unix

По тику:

* декремент кванта;
* инкремент счётчика реального времени;
* инкремент счетчика процессорного времени, полученного текущим процессором;
* инкремент счетчика времени с момента запуска системы;
* декремент счетчиков времени отложенных вызовов (при достижении каким-либо счетчиком нуля установка флага для запуска обработчика отложенного вызова).

По главному тику:

* инициализация отложенного вызова функций планировщика;
* вызов pagedaemon и swapper;
* декремент счетчиков времени до посылки сигналов: SIGALRM, SIGPROF, SIGVTALRM.

По кванту:

* посылка сигнала SIGXCPU текущему процессу, если он превысил выделенный для него квант процессорного времени.

## ОС семейства Windows

По тику:

* декремент кванта;
* инкремент счётчика реального времени;
* декремент счетчиков времени отложенных вызовов (при достижении каким-либо счетчиком нуля установка флага для запуска обработчика отложенного вызова).

По главному тику:

* инициализация диспетчера настройки баланса путем освобождения объекта «событие», на котором он ожидает;

По кванту:

* инициализация диспетчеризации потоков путем постановки соответствующего объекта в очередь DPC.

# Пересчет динамических приоритетов

ОС семейства Unix и Windows являются системами разделения времени с динамическими приоритетами и с вытеснением. При этом динамические приоритеты могут иметь только прикладные процессы.

## ОС семейства Unix

Планирование процессов в ОС семейства UNIX основано на приоритете процесса. Приоритет задается целым числом от 0 до 127. Чем меньше число, тем выше приоритет процесса. Приоритеты от 0 до 49 зарезервированы ядром операционной системы, пользовательские процессы могут обладать приоритетом от 50 до 127.

Структура proc содержит следующие поля, относящиеся к приоритетам:

1) p\_pri — текущий приоритет планирования;

2) p\_usrpri — приоритет режима задачи;

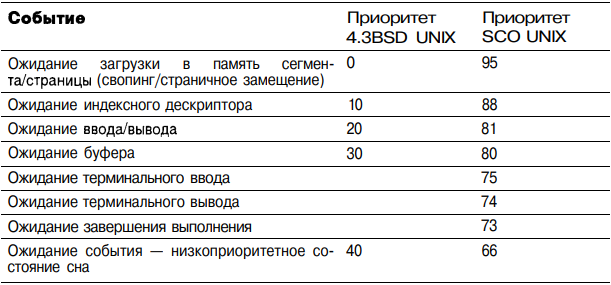
3) p\_cpu — результат последнего измерения использования процессора;

4) p\_nice — фактор «любезности», который устанавливается пользователем.

Планировщик использует значение p\_pri для принятия решения о том, какой процесс направить на выполнение. Значение p\_pri может быть повышено планировщиком для выполнения процесса в режиме ядра. При этом p\_usrpri будет использоваться для хранения приоритета, который будет назначен процессу при возврате в режим задачи.

Процессу, ожидающему недоступного в данный момент ресурса, система определяет значение приоритета сна выбираемое ядром из диапазона системных приоритетов и связанное с событием, вызвавшим это состояние.

Таблица системных приоритетов сна



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.

Приоритет в режиме задачи зависит от двух факторов: «любезности» и последней измеренной величины использования процессора. Увеличение степени любезности приводит к уменьшению приоритета.

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. Более того, каждую секунду ядро системы вызывает процедуру schedcpu() (запускаемую через отложенный вызов), которая уменьшает значение p\_cpu каждого процесса исходя из фактора «полураспада». В 4.3BSD для расчета фактора полураспада применяется следующая формула:

decay = (2 \* load\_avg) / (2 \* load\_ avg + 1)

где load\_avg — это среднее количество процессов, находящихся в состоянии готовности, за последнюю секунду. Процедура schedcpu() также пересчитывает приоритеты для режима задачи всех процессов по формуле

p\_usrpri = PUSER + (p\_cpu / 4) + (2 \* p\_nice)

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

В результате p\_cpu будет увеличен, если процесс до вытеснения другим процессом использовал большое количество процессорного времени. Это приведет к росту значения p\_usrpri, из чего следует понижение приоритета.

## ОС семейства Windows

В ОС семейства Windows код, отвечающий за планирование рассредоточен по ядру, единого модуля или процедуры с названием «планировщик» нет. Совокупность процедур, выполняющих эти обязанности, называется диспетчером ядра.

При создании процесса ему назначается приоритет, который является базовым для приоритетов потока. Планирование в Windows осуществляется на уровне потоков.

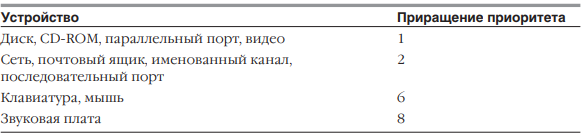
Уровни приоритета потока назначаются с учетом двух разных точек зрения — Windows API и ядра Windows. Windows API сначала упорядочивает процессы по классам приоритета, назначенным при их создании, а затем — по относительному приоритету индивидуальных потоков в рамках этих процессов.

У каждого потока есть два значения приоритета: текущее и базовое. Начальный базовый приоритет потока наследуется от базового приоритета процесса. Но решения, связанные с планированием, принимаются на основе текущего приоритета, так как в определенных обстоятельствах система может на короткое время повышать приоритеты потоков в динамическом диапазоне (1–15). Приоритет потока всегда повышается относительно базового, а не текущего уровня и независимо от приращения приоритет потока никогда не будет больше 15.

Windows никогда не изменяет приоритеты потоков в диапазоне реального времени (16–31), поэтому у таких потоков базовый приоритет идентичен текущему.

Windows может динамически повышать значение текущего приоритета потока в одном из пяти случаев:

* после завершения операций ввода-вывода;

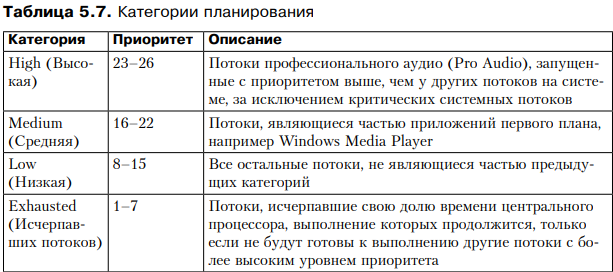
Таблица рекомендуемых значений повышения приоритета

* по окончании ожидания на событии или семафоре исполнительной системы (приоритет повышается на 1);
* по окончании операции ожидания потоками активного процесса (приоритет повышается на величину текущего значения PsPrioritySeparation);
* при пробуждении GUI-потоков из-за операций с окнами (приоритет повышается на 2);
* если поток, готовый к выполнению, задерживается из-за нехватки процессорного времени (приоритет повышается до 15).

Для динамического повышения приоритета при нехватке процессорного времени раз в секунду диспетчер настройки баланса сканирует 16 потоков их очереди готовых потоков и ищет те, которые находятся в состоянии готовности в течение примерно 4 секунд. Обнаружив такой поток, диспетчер настройки баланса повышает его приоритет до 15. В Windows 2000 и Windows XP квант потока удваивается относительно кванта процесса. В Windows Server 2003 квант устанавливается равным 4 единицам. Как только квант истекает, приоритет потока немедленно снижается до исходного уровня.

**Повышения приоритетов для мультимедийных приложений и игр.**

Для повышения приоритета потоков, на которых выполняются мультимедийные приложения, в клиентские версии Windows была внедрена служба MMCSS (MultiMedia Class Scheduler Service), которая работает с такими задачами как аудио, игры, проигрывание и т. д.



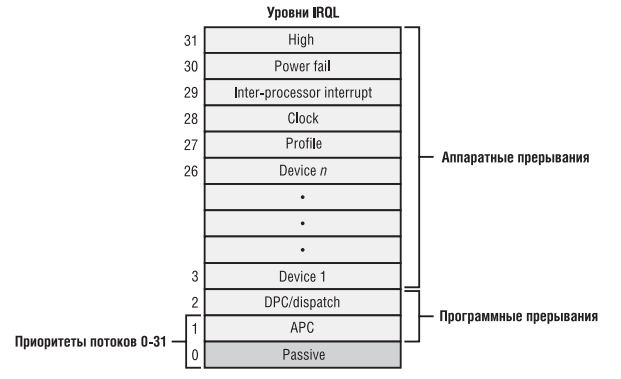
Функции MMCSS временно повышают приоритет потоков, зарегистрированных с MMCSS до уровня, который соответствует категории планирования. Потом их приоритет снижается до уровня, соответствующего категории планирования Exhausted, для того, чтобы другие потоки тоже могли

Сама служба MMCSS выполняется с приоритетом 27, поскольку ей нужно вытеснять любые Pro Audio-потоки с целью снижения их приоритета до категории Exhausted.

**Уровни запросов программных прерываний (IRQL)**

Windows устанавливает свою собственную схему приоритетности прерываний, известную как уровни запросов прерываний (IRQL). В ядре IRQL— уровни представлены в виде номеров от 0 до 31 на системах x86, где более высоким номерам соответствуют прерывания с более высоким приоритетом. Прерывания обслуживаются в порядке их приоритета.

Уровни запросов прерываний (IRQL) в x86-системах



Прерывания обслуживаются в порядке их приоритета, и прерывания с более высоким уровнем приоритета получают преимущество в обслуживании. При возникновении прерывания с высоким уровнем приоритета процессор сохраняет состояние прерванного потока и запускает связанный с прерыванием диспетчер системных прерываний. Тот, в свою очередь, поднимает IRQL и вызывает процедуру обработки прерывания. После выполнения этой процедуры диспетчер прерываний понижает IRQL-уровень процессора до значения, на котором он был до возникновения прерывания, а затем загружает сохраненное состояние машины. Прерванный поток продолжает выполнение с того места, в котором оно было прервано.

**Потоки в ОС семейства UNIX**

В системах разделения времени процесс переходит в состояние блокировки по разным причинам: истек квант или запрошен дополнительный ресурс, который не может быть получен сразу. Процесс переходит в состояние ожидания освобождения этого ресурса. С точки зрения переключения контекста это затратное действие.

Чтобы сократить эти затраты Unix предложил потоки. В их основе лежит идея сократить затраты на переключение полного контекста. Но в очереди к процессору стоит смесь потоков из разных процессов, поэтому, если процессор переключается на выполнение потока, того же процесса, то будет переключаться только аппаратный контекст, а если переключение произойдет на выполнение потока уровня другого процесса, то будет переключен полный контекст. Прогнозировать какой поток будет следующим в очереди невозможно, но среднестатистически при таком подходе экономия времени есть.

Потоки делятся на 2 типа: уровня пользователя и уровня ядра.

О потоках уровня пользователя системе ничего не известно. Для работы с такими потоками нужна библиотека потоков уровня пользователя.

Потоки уровня ядра создаются соответствующими системными вызовами. Ядро имеет о них полную информацию

.

# Выводы

ОС семейства Unix и Windows являются системами разделения времени с динамическими приоритетами и с вытеснением. Поэтому функции обработчика прерывания от системного таймера в этих системах выполняют схожие задачи:

* декремент кванта;
* инкремент счётчика реального времени;
* декремент счетчиков времени отложенных вызовов (при достижении каким-либо счетчиком нуля установка флага для запуска обработчика отложенного вызова).

При этом динамические приоритеты могут иметь только прикладные процессы.