



Государственное образовательное учреждение высшего
профессионального образования
«Московский Государственный Технический Университет
имени Н.Э. Баумана»

ОТЧЕТ

По лабораторной работе № 1
По дисциплине «Операционные системы»
на тему «Функции обработчика
прерывания системного таймера и
пересчет динамических приоритетов»

Исполнитель

Студент:

Неклепаева А.Н.

Группа:

ИУ7-53

Принял

Преподаватель:

Рязанова Н.Ю.

Москва

2019

Функции системных таймеров в защищённом режиме

Unix/Linux

по тикку:

- ведение статистики использования центрального процессора текущим процессом
- инкремент таймеров системы
- декремент кванта текущего потока
- проверка списка отложенных вызовов, отправление на выполнение отложенных вызовов на выполнение при достижении нулевого значения счётчика

по главному тикку:

- добавление в очередь на выполнение функций, относящихся к работе планировщика-диспетчера
- декремент времени, оставшегося до отправления одного из сигналов: SIGALARM (декремент будильников); SIGPROF (измерение времени работы процесса); SIGVTALARM (измерение времени работы процесса в режиме задачи)

по кванту:

- посылка сигнала SIGXCPU текущему процессу при истечении выделенного ему квант процессорного времени
- пробуждение системных процессов (swapper и pagedaemon), процедура wakeup перемещает дескрипторы процессов из очереди «спящих» в очередь «готовых к выполнению»

Windows

по тикку:

- инкремент счётчика системного времени
- декремент счётчиков отложенных задач
- декремент остатка кванта текущего потока
- добавление процесса в очередь DPC

по главному тикку:

- инициализация диспетчера настройки баланса (путём освобождения объекта «событие» каждую секунду)

по кванту:

- Инициализация диспетчеризации потоков (посредством добавления соответствующего объекта DPC в очередь)

Пересчет динамических приоритетов(только у пользовательских процессов)

Unix/Linux

Механизм планирования в традиционных системах базируется на приоритетах. Каждый процесс обладает приоритетом планирования, изменяющимся с течением времени. Планировщик всегда выбирает процессы, обладающие более высоким приоритетом. Для диспетчеризации процессов с равным приоритетом применяется вытесняющее квантование времени. Изменение приоритетов процессов происходит динамически на основе количества используемого ими процессорного времени. Если какой-либо из высокоприоритетных процессов будет готов к выполнению, планировщик вытеснит ради него текущий процесс даже в том случае, если тот не израсходовал свой квант времени.

Традиционное ядро Unix является строго невытесняющим. Если процесс выполняется в режиме ядра (например, в течение исполнения системного вызова или прерывания), то ядро не заставит такой процесс уступить процессор какому-либо высокоприоритетному процессу. Выполняющийся процесс может только добровольно освободить процессор в случае своего блокирования в ожидании ресурса, иначе он может быть вытеснен при переходе в режим задачи. Реализация ядра невытесняющим решает множество проблем синхронизации, связанных с доступом нескольких процессов к одним и тем же структурам данных ядра.

Приоритет процесса задается любым целым числом от 0 до 127 (приоритеты от 0 до 49 – зарезервировано для ядра, а прикладные процессы обладают диапазоном от 50 до 127). Чем меньше число, тем выше приоритет процесса. В структуре `proc` содержатся 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 отправить на выполнение. Когда процесс находится в режиме задачи, **p_pri** равен **p_usrpri**.

Когда процесс просыпается после блокирования в системном вызове, его приоритет будет временно повышен, чтобы дать ему предпочтение в режиме ядра. Поэтому планировщик использует **p_usrpri** для хранения приоритета, назначаемого процессу при возврате в режим задачи, а **p_pri** используется для хранения временного приоритета для выполнения в режиме ядра. Ядро системы связывает *приоритет сна* (величина, определяемая для ядра, поэтому лежит в пределах от 0 до 49) с событием или ожидаемым ресурсом, из-за которого процесс может быть заблокирован.

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 2х факторов: «любезности» и последней измеренной величины использования процессора. Степень любезности (*nice value*) – число от 0 до 39 (со значением 20 по умолчанию). Увеличение значения приводит к уменьшению приоритета.

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. Это может привести к понижению приоритета, что, в свою очередь, вызовет переключение контекста.

На каждом тике обработчик таймера увеличивает **p_cpu** на единицу для текущего процесса до максимального значения.

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 4.3 BSD для расчета фактора полураспада применяется следующая формула:

$$decay = (2 * load_average) / (2 * load_average + 1)$$

где *load_average* — это среднее количество процессов, находящихся в состоянии готовности к выполнению, за последнюю секунду. Фактор полураспада обеспечивает экспоненциально взвешенное среднее значение использования процессора в течение всего периода функционирования процесса.

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, 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.

Windows

Windows является полностью вытесняющей, то есть переключение потоков может произойти в любой момент, а не только в конце кванта текущего потока.

Планирование вызывается при следующих условиях:

- 1) Выполняющийся поток блокируется на семафоре, мьютексе, событии, вводе/выводе и т.д;
- 2) Поток подает сигнал об объекте;
- 3) Истекает квант времени потока.

Windows использует 32 уровня приоритета, от 0 до 31. Эти значения разбиваются на части следующим образом:

- шестнадцать уровней реального времени (от 16 до 31);
- шестнадцать изменяющихся уровней (от 0 до 15), из которых 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 (номера представляют внутренний индекс `PROCESS_PRIORITY_CLASS`, распознаваемый ядром): Реального времени — Real-time (4), Высокий — High (3), Выше обычного — Above Normal (7), Обычный — Normal (2), Ниже обычного — Below Normal (5) и Простая — 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) и Простая — Idle (–15).

Таблица 5.3. Отображение приоритетов ядра Windows на Windows API

Класс приоритета/ Относительный приоритет	Realtime	High	Above	Normal	Below Normal	Idle
Time Critical (+ насыщение)	31	15	15	15	15	15
Highest (+2)	26	15	12	10	8	6
Above Normal (+1)	25	14	11	9	7	5
Normal (0)	24	13	10	8	6	4
Below Normal (–1)	23	12	9	7	5	3
Lowest (–2)	22	11	8	6	4	2
Idle (– насыщение)	16	1	1	1	1	1

Решения по планированию принимаются исходя из текущего приоритета. Система при определенных обстоятельствах на короткие периоды времени повышает приоритет потоков в динамическом диапазоне (от 0 до 15). Windows никогда не регулирует приоритет потоков в диапазоне реального времени (от 16 до 31), поэтому они всегда имеют один и тот же базовый и текущий приоритет.

Имеется четыре категории приоритетов: real-time, user, zero и idle (значение которого фактически равно –1). Приоритеты 16–31 называются системными и предназначены для создания систем, удовлетворяющих ограничениям реального времени, таким как предельные сроки, необходимые для мультимедийных презентаций. Потоки с приоритетами реального времени выполняются до потоков с динамическими приоритетами (но не раньше DPC и ISR).

Потоки приложений обычно выполняются с приоритетами 1–15. Как правило, пользовательские приложения и службы запускаются с обычным базовым приоритетом (normal), поэтому их исходный поток чаще всего выполняется с уровнем приоритета 8.

Повышение приоритета вступает в действие немедленно и может вызвать изменения в планировании процессора. Однако если поток использует весь свой следующий квант, то он теряет один уровень приоритета и перемещается вниз на одну очередь в массиве приоритетов. Если же он использует второй полный квант, то он перемещается вниз еще на один уровень, и так до тех пор, пока не дойдет до своего базового уровня (где и останется до следующего повышения). Повышение приоритета потока в Windows применяется только для потоков с приоритетом динамического диапазона (0-15). Но каким бы ни было приращение, приоритет потока никогда не будет больше 15. Таким образом, если к потоку с приоритетом 14 применить динамическое повышение на 5 уровней, то его приоритет станет равным только 15 (если приоритет потока равен 15, то повысить его нельзя).

Приоритет потока повышается:

- Когда операция ввода-вывода завершается и освобождает находящийся в состоянии ожидания поток, то его приоритет повышается (чтобы он мог опять быстро запуститься и начать новую операцию ввода-вывода). Важно, что для запросов на ввод/вывод, адресованных устройством с меньшим гарантированным временем отклика, предусматриваются большие приращения приоритета.
- Если поток ждал на семафоре, мьютексе или другом событии, то при его освобождении он получает повышение приоритета на два уровня, если находится в фоновом процессе, и на один уровень во всех остальных случаях. Это целесообразно, так как потокам, блокируемым на событиях, процессорное время требуется реже, чем остальным (это позволяет равномернее распределять процессорное время).
- Если поток графического интерфейса пользователя просыпается по причине наличия ввода от пользователя, то он также получает повышение.
- Если поток, готовый к выполнению, задерживается из-за нехватки процессорного времени. Раз в секунду диспетчер настройки баланса (системный поток, предназначенный главным образом для выполнения функций управления памятью), проверяет очереди готовых потоков и ищет потоки, которые находятся в состоянии готовности (Ready) в течение 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. Если этот поток не успел закончить свою работу и если другой поток с более высоким приоритетом готов к выполнению, то после снижения приоритета он возвращается в очередь готовых потоков. В итоге через 4 секунды его приоритет может быть снова повышен. Чтобы свести к минимуму расход процессорного времени, диспетчер настройки баланса сканирует лишь 16 готовых потоков. Если таких потоков с данным уровнем приоритета более 16, он запоминает тот поток, перед которым он остановился, и в следующий раз продолжает сканирование именно с него. Кроме того, он повышает приоритет не более чем говорит о высокой загрузке системы), он прекращает сканирование. При следующем проходе сканирование возобновляется с того места, где оно было прервано в прошлый раз.

Для обеспечения поддержки мультизадачности системы, когда выполняется код режима ядра, Windows использует приоритеты прерываний IRQL.

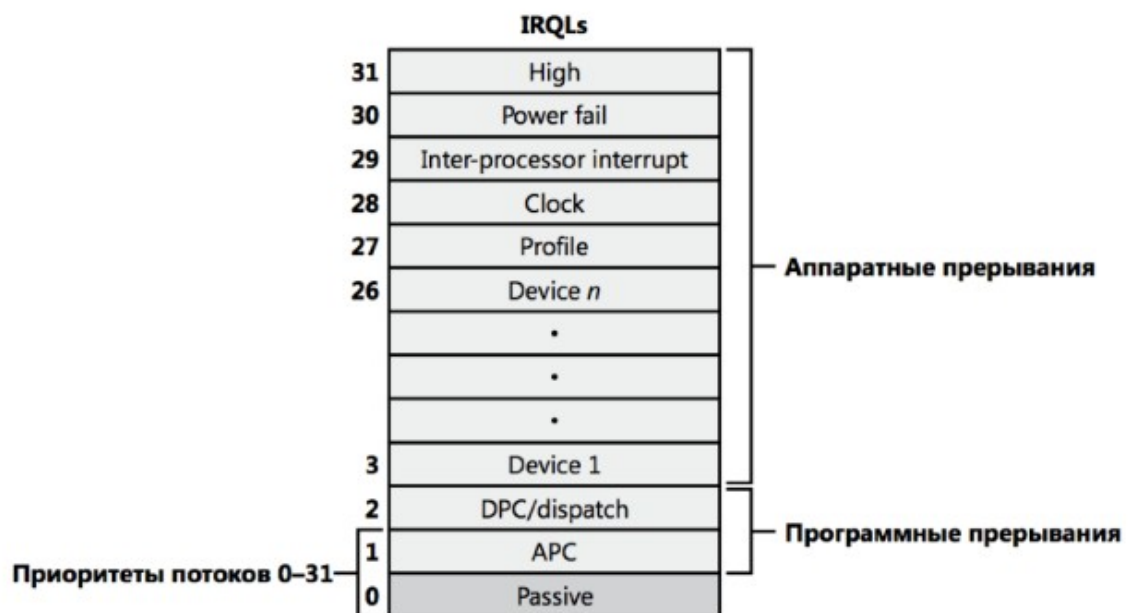


Рис. 5.15. Сопоставление приоритетов потоков с IRQL-уровнями на системе x86

Прерывания обслуживаются в порядке их приоритета. При возникновении прерывания с высоким приоритетом процессор сохраняет информацию о состоянии прерванного потока и активизирует сопоставленный с данным прерыванием диспетчер ловушки. Последний повышает IRQL и вызывает процедуру обслуживания прерывания (ISR). После выполнения ISR диспетчер прерывания понижает IRQL процессора до исходного уровня и загружает сохраненные ранее данные о состоянии машины. Прерванный поток возобновляется с той точки, где он был прерван. Когда ядро понижает IRQL, могут начать обрабатываться ранее замаскированные прерывания с более низким приоритетом. Тогда

вышеописанный процесс повторяется ядром для обработки и этих прерываний.

Выводы

Обработчик системного таймера выполняет схожие функции в обоих рассматриваемых семействах ОС Windows и Unix. А именно:

- обновление системного времени
- уменьшение кванта процессорного времени, выделенного процессу
- запуск планировщика задач
- отправление отложенных вызовов на выполнение

Это объясняется тем, что и Windows, и Unix — это системы разделения времени с вытеснением и динамическими приоритетами. В обоих семействах ОС учитывается ожидание процесса в очереди готовых процессов, чтобы исключить бесконечное откладывание процессов. Но системы планирования данных семейств ОС имеют ряд различий. Классический Unix имеет невытесняющее ядро, Windows является полностью вытесняющей ОС. Алгоритмы планирования в обеих ОС основаны на очередях, но взаимодействия планировщика и потоков в данных ОС различны. В Windows потоки сами вызывают планировщик для пересчёта их приоритетов.