Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчет

к лабораторной работе №1 (2 часть)

по курсу «Операционные системы»

по теме «Функции системного таймера и пересчет динамических приоритетов»

студентки ИУ7-55Б

Оберган Татьяны

Преподаватель: Рязанова Н. Ю.

2019 г.

Функции обработчика прерывания системного таймера

Windows

По тику

- Инкремент счётчика системного времени
- Декремент счетчиков отложенных задач
- Декремент остатка кванта текущего потока.
- Активизация обработчика ловушки профилирования ядра

По главному тику

- Инициализация диспетчера настройки баланса (освобождение объекта «событие» каждую секунду)

По кванту

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

Unix/Linux

По тику

- Счет тиков аппаратного таймера
- Декремент кванта текущего потока.
- Инкремент времени использования процессора
- Наблюдение за списком отложенных вызовов.

По основному тику

- Добавление в очередь на выполнение функций, относящихся к работе планировщика-диспетчера (напр: пересчет приоритетов)
- Пробуждение системных процессов, таких, как swapper и pagedaemon (процедура wakeup перемещает дескрипторы процессов из очереди «спящих» в очередь «готовых к выполнению»)
- Обработка сигналов тревоги; декремент будильников и измерение времени работы процесса

По кванту

- Посылка текущему процессу сигнала SIGXCPU, если израсходован выделенный ему квант процессорного времени.

Пересчет динамических приоритетов

Windows

В Windows реализуется приоритетная, вытесняющая система планирования, при которой всегда выполняется хотя бы один работоспособный (готовый) поток с самым высоким приоритетом, с той оговоркой, что конкретные, имеющие высокий приоритет и готовые к запуску потоки могут быть ограничены процессами, на которых им разрешено или предпочтительнее всего работать.

Windows использует 32 уровня приоритета:

- 16 уровней реального времени (16-31)
- 16 изменяющихся уровней (0 – 15). Уровень 0 зарезервирован для потока обнуления страниц

Уровни приоритета потоков назначаются Windows API и ядром Windows. Сначала WinAPI систематизирует процессы по классу приоритета (присваивается при создании), затем назначается относительный приоритет отдельных потоков внутри этих процессов.

Поэтому в Windows API каждый поток имеет базовый приоритет, являющийся функцией класса приоритета процесса и его относительного приоритета процесса. В ядре класс приоритета процесса преобразуется в базовый приоритет путем использования процедуры PspPriorityTable и показанных ранее индексов PROCESS_ PRIORITY_CLASS, устанавливающих приоритеты 4, 8, 13, 14, 6 и 10 соответственно. Затем применяется относительный приоритет потока в качестве разницы для этого базового приоритета. Например, наивысший «Highest»-поток получит базовый приоритет потока на два уровня выше, чем базовый приоритет его процесса.

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

Насыщенные потоки пропускаются в коде обработки.

В то время как у процесса имеется только одно базовое значение приоритета, у каждого потока имеется два значения приоритета: текущее и базовое. Решения по планированию принимаются исходя из текущего приоритет. 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.

Как правило, пользовательские приложения и службы запускаются с обычным базовым приоритетом (normal), поэтому их исходный поток чаще всего выполняется с уровнем приоритета 8.

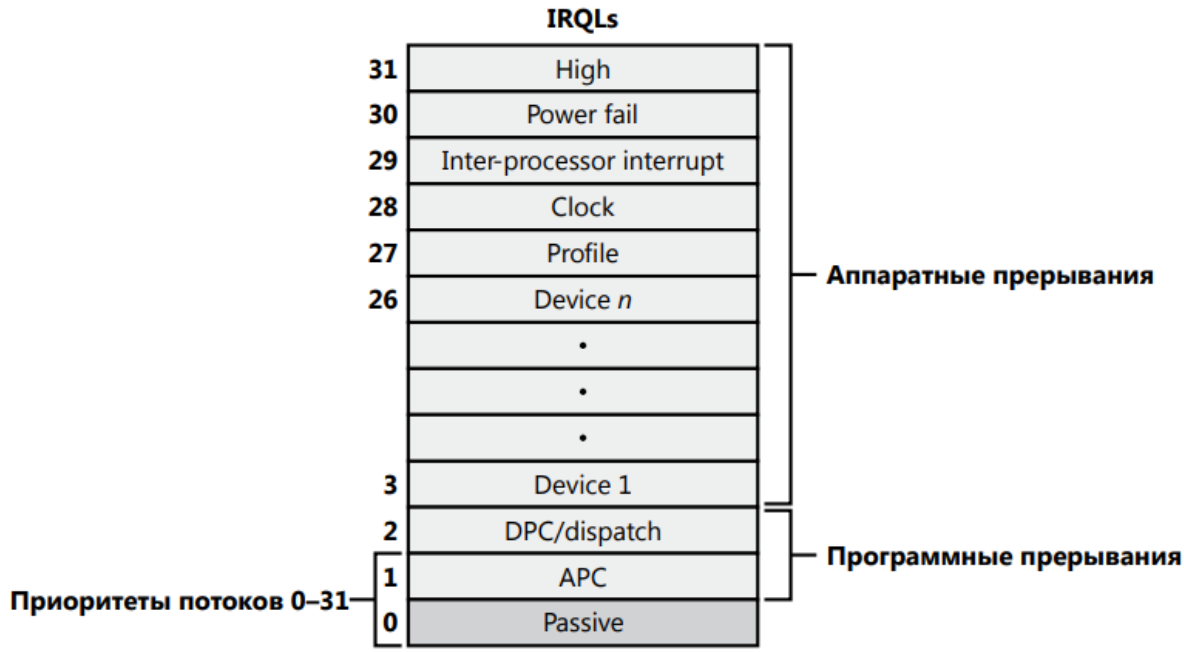

Рис. 5.15. Сопоставление приоритетов потоков с IRQL-уровнями на системе x86

На рис. 5.15 показаны уровни запроса прерываний (IRQL) для 32-разрядной системы. Потоки обычно запускаются на уровне IRQL 0 (который называется пассивным уровнем, потому что никакие прерывания не обрабатываются и никакие прерывания не заблокированы) или на уровне IRQL 1 (APC-уровень). Код пользовательского режима всегда запускается на пассивном уровне. Поэтому никакие потоки пользовательского уровня независимо от их приоритета не могут даже заблокировать аппаратные прерывания (хотя высокоприоритетные потоки реального времени могут заблокировать выполнение важных системных потоков).

Потоки, запущенные в режиме ядра, несмотря на изначальное планирование на пассивном уровне или уровне APC, могут поднять IRQL на более высокие уровни, например, при выполнении системного вызова, который может включать в себя диспетчеризацию потока, диспетчеризацию памяти или

ввод-вывод. Если поток поднимает IRQL на уровень dispatch или еще выше, на его процессоре не будет больше происходить ничего, относящегося к планированию потоков, пока уровень IRQL не будет опущен ниже уровня dispatch. Поток выполняется на dispatch-уровне и выше, блокирует активность планировщика потоков и мешает контекстному переключению на своем процессоре.

Планировщик Windows периодически настраивает текущий приоритет потоков, используя **внутренний механизм повышения приоритета:**

- Повышение приоритета вследствие событий планировщика или диспетчера
- Завершение ожидания
- Повышение приоритета владельца блокировки
- Повышение приоритета после завершения ввода-вывода

Таблица 5.6. Рекомендуемые значения повышения приоритета

Устройство	**Повышение приоритета**
Жесткий диск, привод компакт-дисков, параллельный порт, видеоустройство	1
Сеть, почтовый слот, именованный канал, последовательный порт	2
Клавиатура, мышь	6
Звуковое устройство	8

- Повышение при ожидании ресурсов исполняющей системы
- Повышение приоритета потоков первого плана после ожидания
- Повышение приоритета после пробуждения GUI-потока
- Повышения приоритета, связанные с перезагруженностью центрального процессора (CPU Starvation)

В Windows вытесняющая система планирования (переключение потоков может произойти в любой момент, а не только в конце кванта).

В Windows ответ на вопрос «Кто получит центральный процессор?» основывается на приоритете потока, а на практике:

- **Самостоятельное переключение** - поток может добровольно отказаться от использования процессора путем входа в состояние ожидания какого-нибудь объекта
- **Вытеснение** - В этом сценарии планировки поток с более низким приоритетом вытесняется, когда становится готовым к выполнению поток с более высоким приоритетом.

    - Поток с более высоким приоритетом завершил ожидание.
    - Произошло повышение или снижение приоритета потока.
- **Истечение кванта времени** - Когда у выполняемого потока истекает квант времени, Windows должна определить, нужно ли снижать приоритет потока, а затем определить, должно ли быть спланировано выполнение на процессоре другого потока.
- **Завершение выполнения потока** - Когда поток завершает свое выполнение, он переходит из состояния выполнения в состояние завершения

Unix/Linux

В Unix планировщик предоставляет процессор каждому процессу системы на небольшой период времени, после чего производит переключение на следующий процесс. Этот период называется *квантом времени*.

Переключение контекста - на самом низком уровне планировщик заставляет процессор производить переключения от одного процесса к другому.

Традиционное ядро UNIX является *строго невытесняющим*. Если процесс выполняется в режиме ядра, 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.

Приоритеты процессов:

Приоритет процесса задается любым целым числом от 0 до 127. Чем меньше число, тем выше приоритет.

- 0 – 49 – зарезервированы для ядра
- 50-127 – приоритеты прикладных процессов

Структура proc:

- p_pri – текущий приоритет планирования
- p_usrpri – приоритет режима задачи
- p_cpu – результат последнего измерения использования процессора. При создании процесса – 0.
- p_nice – фактор «любезности», устанавливаемый пользователем

Когда процесс находится в режиме задачи, значение p_pri идентично p_userpri.

Таблица 3.3. Системные приоритеты сна

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 ввода		75
Ожидание терминального вывода		74
Ожидание завершения выполнения		73
Ожидание события — низкоприоритетное состояние сна	40	66

Ядро системы связывает приоритет сна с событием или ожидаемым ресурсом, из-за которого процесс может заблокироваться. Приоритет сна является величиной, определяемой для ядра, и потому лежит в диапазоне 0-49. (Значение приоритета сна для терминального ввода – 28, для операций дискового ввода-вывода – 20).

Когда замороженный процесс просыпается, ядро устанавливает значение его p_pri, равное приоритету сна события или ресурса. Поскольку приоритеты ядра выше, чем приоритеты режима задачи, таки процессы будут назначены на выполнение раньше, чем другие, функционирующие в режиме задачи.

Приоритет в режиме задачи зависит от двух факторов: «любезности» (0 – 39) и последней измеренной величины использования процессора. Чем выше любезность, тем ниже приоритет.

На каждом тике инкрементируется p_cpu (max – 127). А каждую секунду ядро системы вызывает процедуру schedcpu(), которая

- уменьшает значение p_cpu каждого процесса исходя из фактора «полураспада»:
  decay = (2*load_average)/(2*load_average+1);
  где load_average – это среднее количество процессов, находящихся в состоянии готовности к выполнению, за последнюю секунду.
- Пересчитывает приоритеты для режима задачи всех процессов
  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 p_usrpri и, следовательно, к понижению приоритета.

Такая схема отдает предпочтение процессам, осуществляющим много операций ввода-вывода, в противоположность процессам, производящим много вычислений.

Вывод

Windows и Unix обработчик прерывания системного таймера выполняет очень похожие функции т.к. обе эти системы являются системами разделения времени. Общие функции обработчика: счет тиков системного времени, декремент кванта текущего потока, наблюдение за списком отложенных вызовов.

Системы планирования в этих ОС различаются: Windows – полностью вытесняющая, Unix – строго невытесняющая.

Литература

1) Соломон, Руссинович. Внутреннее устройство Windows. 466-532

2) Вахалия. UNIX изнутри 186-200