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» (ИУ7)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.01 Информатика и вычислительная техника**

ОТЧЕТ

по лабораторной работе № 1 (часть 2)

Название: Функции системного таймера в защищённом режиме. Пересчёт динамических приоритетов.

Дисциплина: Операционные системы

Студент	ИУ7-52Б		Короткая В. М.
	(Группа)	(Подпись, дата)	(И.О. Фамилия)
Преподаватель			Рязанова Н. Ю.
		(Подпись, дата)	(И.О. Фамилия)

Москва, 2021

1. **Функции обработчика прерывания от системного таймера для двух классов ОС: Windows и Unix по литературе на примере защищенного режима:**

	Windows	Unix\Linux
- по тику;	– инкремент счётчика тактовых импульсов центрального процессора – декремент кванта текущего потока	– инкремент счётчика тиков, часов и других таймеров системы – декремент кванта текущего потока – декремент счётчика времени до выполнения самого раннего отложенного вызова, в случае, когда счётчик равен нулю, выставление флага
- по главному тику;	– активация диспетчера настройки баланса по сигналу от таймера, срабатывающего 1 раз в секунду	– добавление в очередь на выполнение функций, которые относятся к работе планировщика (например, пересчёт приоритетов) – пробуждение (вызов процедуры wakeup, производящей смену состояния sleep на running) системных процессов (таких, как swapper и pagedaemon) – декремент счётчиков времени до посылки сигналов тревоги (SIGALRM – сигнал истечения времени, заданного функцией alarm(), – SIGPROF – сигнал истечения времени выполнения процесса, а также ожидания завершения системных вызовов, SIGVTALRM – сигнал истечения таймера виртуального времени)
- по кванту	– инициализация диспетчеризации потоков (добавляется объект DPC в очередь)	– посылка текущему процессу сигнала SIGXCPU, если он израсходовал выделенный ему квант процессорного времени

2. Пересчет динамических приоритетов.

Динамическими приоритетами являются приоритеты пользовательских процессов.

В отчете пересчет динамических приоритетов рассматривается

отдельно для ОС семейства Windows и для ОС Unix.

Windows

При запуске каждому процессу назначается приоритет, который называется базовым, и приоритеты потоков определяются относительно приоритета процесса, в котором они создаются.

Реализуется приоритетная, вытесняющая система планирования. При такой системе обязательно выполняется хотя бы один готовый поток с самым высоким приоритетом, но оговаривается, что конкретные, имеющие высокий приоритет и готовые к запуску потоки могут быть ограничены процессами, на которых им разрешено или предпочтительнее работать.

В Windows уровни приоритетов потоков подразделяются на:

1) изменяющиеся (динамические) (0 – 15), из которых уровень 0 зарезервирован для потока обнуления страниц
2) реального времени (16 – 31)

Рис. 5.14. Уровни приоритета потоков

Как было отмечено выше, приоритеты только пользовательских процессов могут пересчитываться, поэтому Windows не регулирует приоритет потоков в диапазоне реального времени, по этой причине они всегда имеют один и тот же приоритет.

В отношении остальных потоков происходит пересчёт приоритетов, который осуществляется следующим образом: вычитается из текущего приоритета повышение первого плана, зачем вычитается повышение свыше обычного (эти два элемента хранятся в переменной PriorityDecrement) и вычитается 1.

Низшей границей полученного приоритета является базовый приоритет, а высшей - обычное значение.

Базовый же приоритет потоков является функцией класса приоритета процесса (Normal, Idle, Below Normal, Realtime или High) и его относительного приоритета 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

В Windows включён специальный механизм для изменения приоритетов готовых потоков, который называется диспетчер настройки баланса. Он ищет в очереди готовых потоков потоки, которые находятся в состоянии ожидания около 4 секунд. Если такой поток найден то, его приоритет повышается до 15 единиц. Как только выделенный квант истекает, приоритет этого потока снижается до обычного базового приоритета. Если он не успел завершиться, но есть готовый к запуску поток с более высоким приоритетом, то поток с пониженным приоритетом отправляется снова в очередь готовых потоков.

Важно, что диспетчер настройки баланса сканирует только 16 готовых потоков (чтобы минимизировать время, затрачиваемое на его работу, центральным процессором). Если на данном уровне приоритета имеется больше потоков, то диспетчер запоминает место, где он остановился, и начинает уже с него при следующем проходе по очереди.

Планировщик Windows также периодически настраивает текущий приоритет потоков, используя внутренний механизм повышения приоритета.

Некоторые сценарии повышения приоритета:

- повышение вследствие событий планировщика или диспетчера;
  Например, мьютекс был освобождён или ликвидирован, или был освобождён семафор. Причём, если поток находится в фоновом процессе, то он получает повышение приоритета на два уровня, и на один в остальных случаях.
- повышение вследствие продолжительного ожидания ресурса;
- повышение вследствие завершения ввода-вывода;

Когда операция ввода-вывода завершается, то приоритет потока, который при этом был освобождён повышается, чтобы приступить к новой операции 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 первого плана после ожидания;
- повышение приоритета после пробуждения GUI-потока;
  Потоки-владельцы окон получают повышение приоритета на 2.
- повышение приоритета, связанного с перезагруженностью ЦП (CPU Starvation)

Unix

Традиционное ядро UNIX является строго невытесняемым. То есть, если процесс выполняется в режиме ядра, то ядро не может заставить такой процесс уступить процессор какому-либо другому процессу с высоким приоритетом. Современные же ядра UNIX являются полностью вытесняемыми, так как это требование обусловлено поддержкой процессов реального времени, таких как, аудио, видео.

Часть ядра, которая распределяет процессорное время между процессорами, называется планировщиком. Он отдаёт предпочтение тем процессам, у которых более высокий приоритет. Значения приоритетов изменяются динамически на основе количества используемого процессорного времени.

Приоритет задаётся целым числом от 0 до 127, причём, чем меньше число, тем выше приоритет. От 0 до 49 зарезервированы для ядра, остальное выделено под прикладные процессы (то есть от 50 до 127).

Приоритеты ядра – фиксированные величины, а приоритеты прикладных задач могут изменяться.

Приоритет процесса периодически повышается ядром системы, пока тот ещё не выполняется, но он будет понижен, как только получит какое-то количество процессорного времени.

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. И может получиться так, что приоритет оказывается ниже приоритета другого запущенного процесса. Тогда ядро производит переключение контекста.

В традиционных системах UNIX приоритет процесса определяется двумя факторами:

1. любезность
   - с помощью системного вызова nice можно изменить его значение, тем самым, оказывается влияние на приоритет процесса (но возможность увеличивать приоритет доступна только суперпользователям). Увеличение этого фактора, уменьшает приоритетность
2. утилизация
   - фактор, который определяется степенью последней загруженности CPU процессом, он позволяет системе динамически изменять приоритет процесса

Структура proc содержит следующие поля:

p_pri

- текущий приоритет планирования;

p_usrpri

- используется для хранения приоритета, который будет назначен процессу при возврате в режим задачи; пересчитывается процедурой schedcpu() по формуле: $p\_usrpri = PUSER + \frac{p\_cpu}{4} + 2 * p\_nice$, где PUSER – базовый приоритет в режиме задачи, равный 50;

p_cpu

- результат последнего измерения использования процессора; при создании процесса значение равно нулю, максимальное значение – 127. На каждом тике увеличивается на единицу обработчиком таймера, и каждую секунду процедурой schedcpu() уменьшается в зависимости от фактора «полураспада», который рассчитывается по формуле:

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

где load_average – среднее количество процессов за последнюю секунду, которые находятся в состоянии готовности к выполнению;

p_nice

- фактор любезности, устанавливаемый пользователем

Получается, что, если процесс использовал большое количество процессорного времени, то значение p_cpu будет увеличено. Значит, p_usrpri также возрастёт, следовательно, приоритет понизится. И наоборот, если процесс долгое время простаивал в очереди, тем больше фактор «полураспада» уменьшает p_cpu, поэтому приоритет повышается.

Планировщик устроен следующим образом:

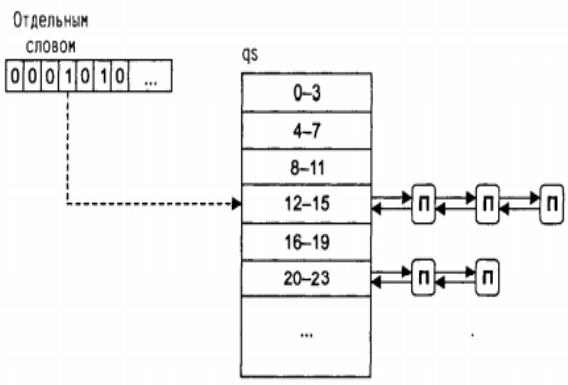

Рис. 5.2. Структуры данных планировщика в системе BSD

- Массив qs состоит из 32 очередей готовых к выполнению процессов, каждая из которых отведена под четыре соседствующих приоритета, например, 0 очередь соответствует 0-3 и т.д. В каждой очереди содержится начало двунаправленного связанного списка структур proc.
- whichqs – глобальная переменная, хранящая битовую маску, в которой для каждого элемента массива qs зарезервирован один бит. Если в очереди находится хотя бы один процесс, то этот бит устанавливается.

Вывод

В силу того, что Windows и UNIX – это системы разделения времени с динамическими приоритетами и вытеснением, их обработчики прерывания от системного таймера выполняют схожие функции, такие как: обновление системного времени, декремент кванта текущего потока, обновление счётчиков времени, оставшегося до выполнения отложенных вызовов.

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

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