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»

Отчёт

по лабораторной работе № 1 (часть 2)

Название: Функции системного таймера и пересчет динамических приоритетов.

Дисциплина: Операционные системы

Студент ИУ7-55Б _____ Д.В. Сусликов

(Группа) (Подпись, дата) (И.О. Фамилия)

Преподаватель _____ Н.Ю. Рязанова

(Подпись, дата) (И.О. Фамилия)

Москва, 2020

Содержание

1 Функции системных таймеров в защищенном режиме

1.1 Windows

По тику:

1) Обновление системного времени (инкремент счетчика системного времени);

2) Декремент показания счетчика, отслеживающего продолжительность работы текущего потока (декремент кванта потока);

3) Декремент показаний счетчиков отложенных задач.

По главному тику:

1) Инициализация диспетчера настройки баланса (освобождение объекта "событие"каждую секунду).

По кванту:

1) Инициализация диспетчеризации потоков (добавление соответствующего объекта DPC в очередь).

1.2 Unix/Linux

По тику:

1) Обновление статистики использования процессора текущим процессом (инкремент счетчика времени использования процессора текущим процессом);

2) Обновление часов и других таймеров системы (инкремент часов и других таймеров системы);

3) Проверка необходимости выполнения отложенного вызова, при нахождении ожидающего вызова выставляется флаг, указывающий на необходимость запуска обработчика отложенного вызова.

По главному тику:

1) Добавление функций планировщика, например пересчет приоритетов, в очередь отложенных вызовов;

2) Пробуждение системных процессов, таких как swapper и pagedaemon (процедура wakeup перемещает дескрипторы процессов из очереди "спящих"в очередь "готовых к выполнению");

3) Декремент счетчиков времени, оставшегося до отправления сигналов тревоги:

    3.1) SIGALARM - сигнал будильника реального времени.

    3.2) SIGPROF - сигнал будильника профиля.

    3.3) SIGVTALARM - сигнал будильника виртуального времени.

По кванту:

1) Посылка текущему процессу сигнала SIGXCPU, если тот превысил выделенный ему квант использования процессора (процессорного времени).

2 Пересчет динамических приоритетов

В Windows и Unix/Linux могут пересчитываться только приоритеты пользовательских процессов.

2.1 В 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.

В то время как у процесса имеется только одно базовое значение приоритета, у каждого потока имеется два значения приоритета: текущее (динамическое) и базовое.

Исходный базовый приоритет потока наследуется от базового приоритета процесса. Процесс по умолчанию наследует свой базовый приоритет у того процесса, который его создал.

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

Повысить или понизить приоритет потока в динамическом диапазоне можно в любом приложении, но у вас должны быть привилегии, позволяющие повышать приоритет, использующийся при планировании для ввода значения в пределах динамического диапазона.

Windows использует 32 уровня приоритета, от 0 до 31 (31 — наивысший). Эти значения разбиваются на категории следующим образом:

1) шестнадцать уровней реального времени (от 16 до 31);

2) шестнадцать изменяющихся уровней (от 0 до 15), из которых уровень 0 зарезервирован для потока обнуления страниц.

Уровни приоритета потоков назначаются с двух позиций: от Windows API и от ядра Windows. Сначала Windows API распределяет процессы по классу приоритета, который им присваивается при создании.

1) реального времени — Real-time (4);

2) высокий — High (3);

3) выше обычного — Above Normal (6);

4) обычный — Normal (2);

5) ниже обычного — Below Normal (5);

6) уровень простоя — Idle (1).

А затем внутри этих процессов назначается приоритет отдельным потокам. Числа в скобках прибавляются к базовому приоритету процесса:

1) критичный по времени — Time-critical (15);

2) наивысший — Highest (2);

3) выше обычного — Above-normal (1);

4) обычный — Normal (0);

5) ниже обычного — Below-normal (–1);

6) самый низший — Lowest (–2);

7) уровень простоя — Idle (–15).

Таким образом, у каждого потока два приоритета: текущий и базовый. Решения, связанные с планированием, принимаются на основе текущего приоритета. Отображение Windows-приоритета на внутренние номерные приоритеты Windows показано в таблице.

	Real-time	High	Above-Normal	Normal	Below-Normal	Idle
Time Critical	31	15	15	15	15	15
Highest	26	15	12	10	8	6
Above Normal	25	14	11	9	7	5
Normal	24	13	10	8	6	4
Below Normal	23	12	9	7	5	3
Lowest	22	11	8	6	4	2
Idle	16	1	1	1	1	1

Соответствие между приоритетами ядра Windows и Windows API

Система динамически повышает приоритет текущего потока в следующих случаях:

1) вследствие событий планировщика или диспетчера;

2) вследствие завершения ввода/вывода;

3) вследствие ввода из пользовательского интерфейса;

4) вследствие слишком продолжительного ожидания ресурса исполняющей системы;

5) в случае, когда готовый к запуску поток не был запущен в течение определенного времени.

Решения по планированию принимаются исходя из текущего приоритета. Как поясняется в следующем разделе, посвященном повышениям приоритета, система при определенных обстоятельствах на короткие периоды времени повышает приоритет потоков в динамическом диапазоне (от 1 до 15).

Сценарии повышения приоритета:

I) Повышение вследствие событий планировщика или диспетчера (сокращение задержек).

Следующие сценарии имеют дело с объектом диспетчера, входящим в сигнальное состояние:

1) В очередь потока поставлен APC-вызов.
2) Событие установлено или отправило сигнал.
3) Таймер установлен, или системное время изменилось (таймеры должны быть перезапущены).
4) Мьютекс был освобожден или ликвидирован.
5) Был освобожден семафор.
6) Произошел выход из процесса.
7) Событие установлено или отправило сигнал.
8) Изменено состояние потока.

II) 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.

Устройство	**Повышение**
Жесткий диск, CD-ROM, параллельный порт, видео	1
Сеть, почтовый слот, именованный канал, последовательный порт	2
Клавиатура, мышь	6
Звуковая карта	8

III) Повышение вследствие ввода из пользовательского интерфейса (сокращение задержек и времени отклика)

Потоки — владельцы окон получают при пробуждении дополнительное повышение приоритета на 2 из-за активности при работе с окнами, например, при поступлении сообщений. Смысл этого повышения — содействие интерактивным приложениям.

IV) Повышение вследствие слишком продолжительного ожидания ресурса исполняющей системы (предотвращение зависания)

Когда поток пытается получить ресурс исполняющей системы (ERESOURCE; подробнее об объектах синхронизации см. в главе 8 части 2), который уже находится в исключительном владении другого потока, он должен быть в состоянии 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 500 мс. Если по окончании этих 500 мс ресурс все еще находится во владении, исполняющая система пытается предотвратить зависание центрального процессора путем получения блокировки диспетчера, повышения приоритета потока или потоков, владеющих ресурсом, до 15, перезапуска их квантов и выполнения еще одного ожидания.

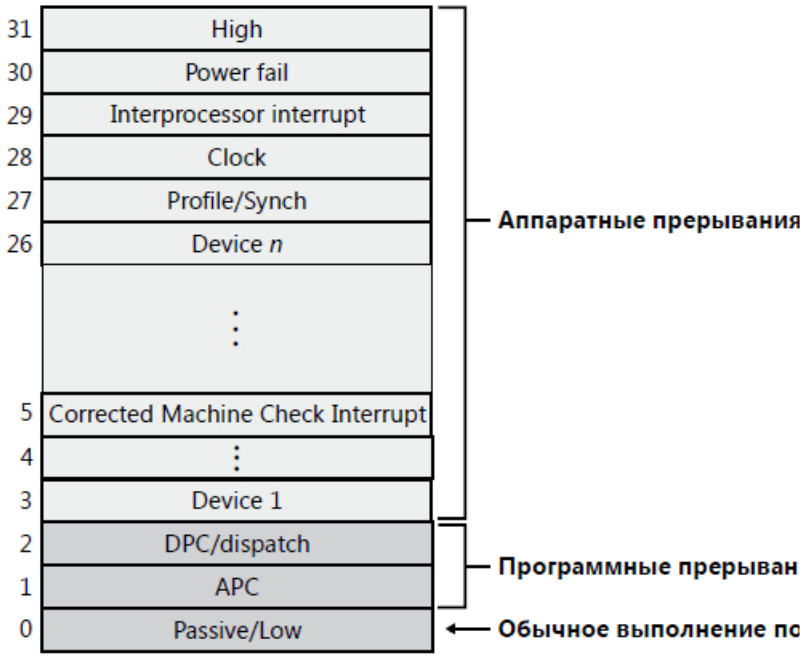
V) Повышения приоритета, связанные с перезагруженностью центрального процессора

в Windows включен общий механизм ослабления загруженности централь-

ного процессора, который называется диспетчером настройки баланса и является частью потока. Он один раз в секунду 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. 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.

В Windows присутствует схема приоритетов прерываний IRQL. Правило IRQL гласит, что код с более низким IRQL не может вмешиваться в работу кода с более высоким IRQL, и наоборот — код с более высоким IRQL не может вытеснять код, работающий с более низким IRQL. 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 прерванный поток возобновляется с той точки, где он был прерван.

Потоки обычно запускаются на уровне IRQL0 (который называется пассивным уровнем, потому что никакие прерывания не обрабатываются и никакие прерывания не заблокированы) или на уровне IRQL1 (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).



Уровни IRQL

2.2 UNIX/Linux

Классическое ядро Linux было невытесняемым. Это означает, что если процесс выполняется в режиме ядра, то ядро не заставит этот процесс уступить процессорное время какому-либо более приоритетному процессу. Выполняющийся процесс может только добровольно освободить процессор в случае своего блокирования в ожидании ресурса, иначе он может быть ветеснен при переходе в режим задачи. Такая реализация ядра позволяет решить множество проблем синхронизации, связанных с доступом нескольких процессов к одним и тем же структурам данных ядра. Современные ядра Linux являются полностью вытесняемыми. Это обусловлено поддержкой процессов реального времени, например видео или аудио.

Приоритеты процессов:

Приоритет процесса в Unix задается любым целым числом в диапозоне от 0 до 127 и это число меньше, тем выше приоритет. Приоритеты от 0 до 49 зарезервированы для ядра, а прикладные процессы могут обладать приоритетом в диапазоне от 50 до 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 отправить на выполнение.

Когда процесс находится в режиме задачи p_pri и p_usrpri идентичны. 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.

Процессу, ожидающему недоступного в данный момент ресурса, система определяет значение приоритет сна выбираемое ядром из диапазона 0-49 и связанное с событием, вызвавшее это состояние.

Когда замороженный процесс просыпается, значение p_pri устанавливается равным приоритету сна события или ресурса, на котором он был заблокирован. Системные приоритеты сна представлены в таблице ниже.

Событие	Приоритет 4.3BSD UNIX	Приоритет SCO UNIX
Ожидание загрузки в память страницы	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	40	66

Системные приоритеты сна

Поскольку приоритеты ядра выше, такие процессы будут назначены на выполнение раньше, чем другие, функционирующие в режиме задачи. Такой подход позволяет системным вызовам быстро завершать работу, что является желательным, так как процессы во время выполнения вызова могут занимать некоторые ключевые ресурсы системы, не позволяя другим ими пользоваться. Когда процесс завершил выполнение системного вызова, его приоритет сбрасывается обратно в значение текущего приоритета в режиме задачи, произойдет переключение контекста.

Приоритет в режиме задачи зависит от "любезности"(nice) и последней измеренной величины использования процессора. Степень любезности - это число в диапозоне от 0 до 39 со значением 20 по умолчанию. Увеличение nice приводит к уменьшению приоритета. При этом суперпользователь может поднять приоритет.

Системы разделения времени пытаются выделить процессорное время таким образом, чтобы конкурирующие процессы системы получили его в пример-

но равных количествах. Поле p_cpu при создании процесса инициализируется нулем. На кажом тике обработчик таймера увеличивает p_cpu на удиницу для текущего процесса до максимального значения 127.

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

По формуле:

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

где load_average - среднее количество процессов в состоянии готовности за последнюю секунду.

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

$$\text{p\_usrpri} = \text{PUSER} + (\text{p\_cpu}/4) + (2 * \text{p\_nice}),$$

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

Если процесс в последний раз использовал большое количество процессорного времени, его p_cpu будет увеличен, что приведет к увеличению значения p_usrpri, и 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.

В системах разделения времени фактор использования процессора обеспечивает равные и честные условия при планировании процессов. Фактор полураспада обеспечивает экспотенциально взвешенное среднее значение использования процессора в течение всего периода функционирования процесса.

3 Заключение

Операционные системы Windows и Unix/Linux - системы с разделением времени с динамическими приоритетами и вытеснением. Поэтому обработчик системного таймера выполняет похожие функции:

1) инкремент счетчика системного времени;
2) декремент кванта;
3) добавление функций планировщика в очередь отложенных вызовов;
4) декремент счетчиков времени, оставшегося до выполнения отложенных вызовов;
5) отправка отложенных вызовов на выполнение.

Алгоритмы планирования основаны на очередях, но работа планировщиков различна: планировщик в Windows вызывается самими процессами, тогда как в Unix/Linux за это отвечает ядро.