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»

Лабораторная работа № 1 (вторая часть)

Тема

Функции системного таймера в защищенном режиме.
Пересчет динамических приоритетов

Студент Жигалкин Д.Р

Группа ИУ7-55Б

Оценка (баллы) _____

Преподаватель Рязанова Н.Ю

Москва.
2020 г.

Функции системного таймера в защищенном режиме

Тик - период времени между двумя последующими прерываниями таймера.

Главный тик - период времени равный n тикам таймера (число n зависит от конкретного варианта системы).

Квант представляет собой количество времени, предоставляемое потоку для выполнения

Функции системного таймера для ОС семейства Windows

По тику:

1) Декремент кванта. По истечении кванта основная задача системы – передать квант следующему процессу из очереди процессов, готовых к выполнению.
2) Инкремент счётчика системного времени
3) Декремент счетчиков отложенных задач

По главному тику:

1) Инициализация диспетчера настройки баланса

По кванту:

1) Инициализация диспетчеризации потоков, посредством добавления соответствующего объекта DPC (Deferred Procedure Call) в очередь

Функции системного таймера для ОС семейства Unix/Linux

По тику:

1) Инкремент часов, ведение показаний фактического времени
2) Инкремент счетчика использования процессора текущим процессом
3) Декремент кванта

4) Декремент счетчика времени, оставшегося до отправления на выполнение отложенных вызовов

По главному тику:

1) Пробуждение системных процессов swapper и pagedaemon

2) Добавление в очередь отложенных вызовов функций планировщика

По кванту:

1) При превышении текущим процессом выделенного кванта, отправка сигнала SIGXCPU это процессу.

Пересчет динамических приоритетов

Windows

В системе Windows реализовано вытесняющее планирование на основе уровней приоритета, при которой выполняется готовый поток с наивысшим приоритетом.

Процессорное время, выделенное на выполнение потока, называется квантом. Если поток с более высоким приоритетом готов к выполнению, текущий поток вытесняется планировщиком, даже если квант текущего потока не истек.

В Windows за планирование отвечает совокупность процедур ядра, называемая диспетчером ядра. Диспетчеризация может быть вызвана, если:

1) поток готов к выполнению;
2) истек квант текущего потока;
3) поток завершается или переходит в состояние ожидания;
4) изменился приоритет потока;
5) изменилась привязка потока к процессору.

В системе предусмотрено 32 уровня приоритетов: уровни реального времени (16-31), динамические уровни (1-15) и системный уровень (0).

Уровни приоритета потоков назначаются Windows API и ядром операционной системы.

Windows API сортирует процессы по классам приоритета, которые были назначены при их 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) простой --- Idle (1).

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

Затем назначается относительный приоритет потоков в рамках процессов:

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) низший --- Lowest (-2);

7) простой --- Idle (-15).

Относительный приоритет --- это приращение к базовому приоритету процесса.

Каким бы образом ни формировался приоритет потока, с точки зрения планировщика Windows важно только значение приоритета.

Процесс обладает только базовым приоритетом, тогда как поток имеет базовый, который наследуется от приоритета процесса, и текущий приоритет. Операционная система может на короткие интервалы времени повышать приоритеты потоков из динамического диапазона, но никогда не регулирует приоритеты потоков в диапазоне реального времени.

Приложения пользователя запускаются, как правило, с базовым приоритетом Normal. Некоторые системные процессы имеют приоритет выше 8, следовательно, это гарантирует, что потоки в этих процессах будут запускаться с более высоким приоритетом.

Система динамически повышает приоритет текущего потока в следующих случаях:

1) по завершении операции ввода-вывода;

2) по окончании ожидания на событии или семафоре исполнительной системы;

3) по окончании ожидания потоками активного процесса;

4) при пробуждении GUI-потоков из-за операции с окнами;

5) если поток, готовый к выполнению, задерживается из-за нехватки процессорного времени.

Динамическое повышение приоритета применяется только к потокам из динамического диапазона (1-15) и, независимо от приращения, приоритет потока не может оказаться выше 15.

По окончании определенных операций ввода-вывода Windows временно повышает приоритет потоков и потоки, ожидающие завершения

этих операций, имеют больше шансов немедленно возобновить выполнение и обработать полученные от устройств ввода-вывода данные.

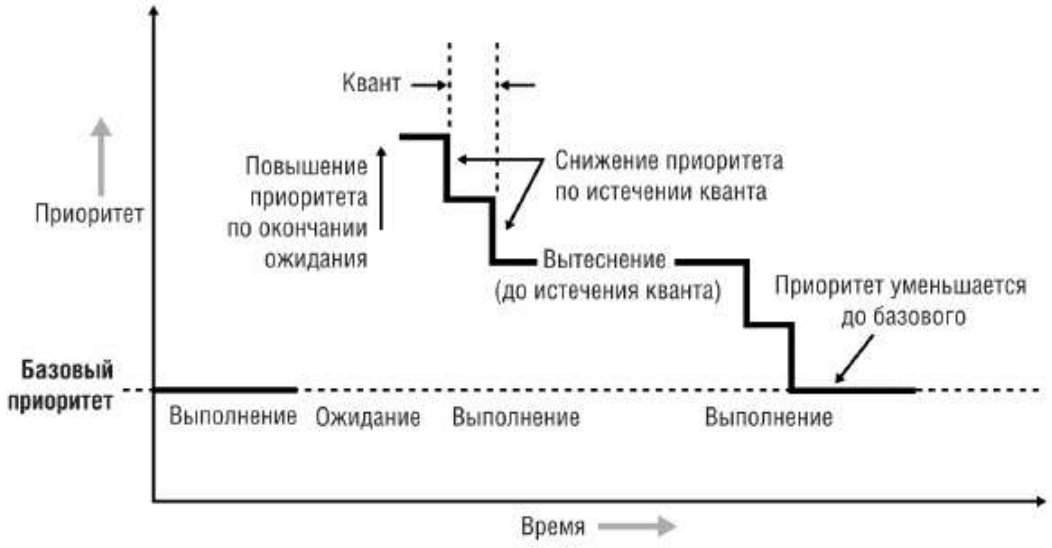
Драйвер устройства ввода-вывода через функцию IoCompleteRequest указывает на необходимость динамического повышения приоритета после выполнения соответствующего запроса.

В таблице ниже приведены приращения приоритетов.

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

Приоритет потока всегда повышается относительно базового приоритета. На рисунке ниже показано, что после повышения приоритета поток в течение одного кванта выполняется с повышенным приоритетом, а затем приоритет снижается на один уровень с каждым последующим квантом. Цикл продолжается до тех пор, пока приоритет не снизится до базового.



Если ожидание потока на событии системы или семафоре успешно завершается из-за вызова SetEvent, PulseEvent или ReleaseSemaphore, его приоритет повышается на 1.

Такая регулировка, как и в случае с окончанием операции ввода-вывода, позволяет равномернее распределить процессорное время --- потокам, блокируемым на событиях, процессорное время требуется реже, чем остальным. В данном случае действуют те же правила динамического повышения приоритета.

К потокам, пробуждающимся в результате установки события вызовом функций NtSetEventBoostPriority и KeSetEventBoostPriority, повышение приоритета применяется особым образом.

Если поток в активном процессе завершает ожидание на объекте ядра, функция ядра KiUnwaitThread повышает его текущий приоритет на величину значения PsPrioritySeparation. PsPriorituSeparation --- это индекс в таблице квантов, с помощью которой выбираются величины квантов для потоков активных процессов. Какой процесс является в данный момент активным, определяет подсистема управления окнами.

В данном случае приоритет повышается для создания преимуществ интерактивным приложениям по окончании ожидания, в результате чего повышаются шансы на немедленное возобновление потока приложения.

Важной особенностью данного вида динамического повышения приоритета является то, что он поддерживается всеми системами Windows и не может быть отключен даже функцией SetThreadPriorityBoost.

Приоритет потоков окон пользовательского интерфейса повышается на 2 после их пробуждения из-за активности подсистемы управления окнами. Приоритет повышается по той же причине, что и в предыдущем случае, --- для увеличения отзывчивости интерактивных приложений.

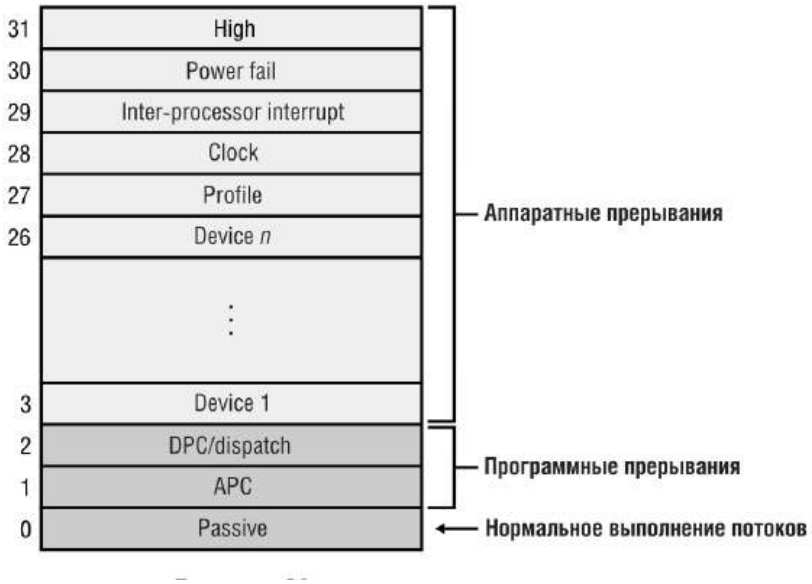
Раз в секунду диспетчер настройки баланса --- системный поток, предназначенный для выполнения функций управления памятью --- сканирует очереди готовых потоков и ищет потоки, которые находятся в

состоянии готовности в течение примерно 4 секунд. Диспетчер настройки баланса повышает приоритет таких потоков до 15. Причем в Windows 2000 и Windows XP квант потока удваивается относительно кванта процесса, а в Windows Server 2003 квант устанавливается равным 4 единицам. По истечении кванта приоритет потока снижается до исходного уровня. Если потоку все еще не хватило процессорного времени, то после снижения приоритета он возвращается в очередь готовых процессов. Через 4 секунды он может снова получить повышение приоритета.

Чтобы свести к минимуму расход процессорного времени, диспетчер настройки баланса сканирует только 16 готовых потоков за раз, а повышает приоритет не более чем у 10 потоков за раз.

Диспетчер настройки баланса не решает всех проблем с приоритетами потоков, однако позволяет потокам, которым не хватает процессорного времени, получить его.

Windows использует схему приоритетов прерываний, называемую уровни запросов прерываний (IRQL). Внутри ядра IRQL представляются в виде номеров до 0 до 31 для систем x86. Ядро определяет стандартный набор IRQL для программных прерываний, а HAL связывает IRQL с номерами аппаратных прерываний.



Прерывания обслуживаются в порядке их приоритета. Прерывания с большим приоритетом вытесняют прерывания с меньшим приоритетом.

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.

Unix/Linux

Классическое ядро UNIX является строго 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 освободить процессор в случае своего блокирования в ожидании ресурса, иначе он может быть вытеснен 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.

Однако современные ядра Linux, начиная с версии 2.5, являются полностью вытесняемыми, так как должны обеспечивать работу процессов реального времени.

Приоритет процесса в UNIX задается числом в диапазоне от 0 до 127, причем чем меньше значение, тем выше приоритет. Приоритеты 0--49

зарезервированы ядром операционной системы, прикладные процессы могут обладать приоритетом в диапазоне 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", устанавливаемый пользователем.

Планировщик использует поле **p_pri** для принятия решения о том, какой процесс отправить на выполнение. Значения **p_pri** и **p_usrpri** идентичны, когда процесс находится в режиме задачи. Когда процесс просыпается после блокировки в системном вызове, его приоритет временно повышается. Планировщик использует **p_usrpri** для хранения приоритета, который будет назначен процессу при переходе из режима ядра в режим задачи, а **p_pri** --- для хранения временного приоритета для выполнения в режиме ядра.

Ядро связывает приоритет сна (0-49) с событие или ожидаемым ресурсом, из-за которого процесс может быть заблокирован. Когда блокированный процесс просыпается, ядро устанавливает **p_pri**, равное приоритету сна события или ресурса, на котором он был заблокирован, следовательно, такой процесс будет назначен на выполнение раньше, чем другие процессы в режиме задачи. В таблице ниже приведены значения приоритетов сна для систем 4.3BSD UNIX и SCO UNIX. Такой подход позволяет системным вызовам быстрее завершать свою работу. По завершении процессом системного вызова его приоритет сбрасывается в значение текущего приоритета в режиме задачи. Если при этом приоритет

окажется ниже, чем приоритет другого запущенного процесса, ядро произведет переключение контекста.

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	30	75
Ожидание терминального вывода	30	74
Ожидание завершения выполнения	30	73
Ожидание события	40	66

Приоритет в режиме задачи зависит от "любезности" и последней измеренной величины использования процессора. Степень любезности --- это число в диапазоне от 0 до 39 со значение 20 по умолчанию. Степень любезности называется так потому, что одни пользователи могут быть поставлены в более выгодные условия другими пользователями посредством увеличения кем-либо из последних значения уровня любезности для своих менее важных процессов.

Системы разделения времени стараются выделить процессорное время таким образом, чтобы все процессы системы получили его в примерно равных количествах, что требует слежения за использованием процессора. Поле **p_cpu** содержит величину последнего измерения использования процессора процессом. При создании процесса это поле инициализируется нулем. На каждом тике обработчик таймера увеличивает **p_cpu** на единицу для текущего процесса, вплоть до максимального значения --- 127. Каждую секунду ядро вызывает процедуру **schedcpu**, которая уменьшает значение **p_cpu** каждого процесса исходя из фактора "полураспада". В 4.3 BSD для расчета применяется формула

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

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

Кроме того, процедура **schedcpu** также пересчитывает приоритеты режима задачи всех процессов по формуле

$$p\_usrpri = PUSER + \frac{p\_cpu}{2} + 2 * p\_nice$$

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

Таким образом, если процесс до вытеснения другим процессом использовал большое количество процессорного времени, его **p_cpu** будет увеличен, что приведен к увеличению значения **p_usrpri**, и, следовательно, к понижению приоритета.

Чем дольше процесс простаивает в очереди на выполнение, тем меньше его **p_cpu**. Это позволяет предотвратить зависания низкоприоритетных процессов. Если процесс большую часть времени выполнения тратит на ожидание ввода-вывода, то он остается с высоким приоритетом.

В системах разделения времени фактор использования процессора обеспечивает справедливость при планировании процессов. Фактор полураспада обеспечивает экспоненциально взвешанное среднее значение использования процессора в течение функционирования процесса. Формула, применяемая в SVR3 имеет недостаток: вычисляя простое экспоненциальное среднее, она способствует росту приоритетов при увеличении загрузки системы.

Вывод

Операционные системы UNIX и Windows являются системами разделения времени с вытеснением. В связи с этим обработчики прерываний от системных таймеров в них выполняют схожие 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.

Декремент кванта является основной функцией обработчика прерывания от системного таймера.

Классическое ядро UNIX **является строго невытесняющим**. Ядра Linux, начиная с версии 2.5, и ядра операционных систем Windows являются полностью вытесняющими для обеспечения работы процессов реального времени.