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

ОТЧЕТ

По лабораторной работе №1 (часть 2)
По курсу «Операционные системы»
Тема: «Функции обработчика прерывания системного таймера и пересчет динамических приоритетов»

Студент: Щербатюк Д.С.
Группа: ИУ7-54
Преподаватель: Рязанова Н. Ю.

Москва, 2017

1. Функции системных таймеров в защищенном режиме

1.1 UNIX/LINUX

По тику

- Ведение учета использования центрального процессора
- Инкремент часов и других таймеров системы. Ведение показаний фактического времени.
- Добавление отложенных вызовов на выполнение при достижении нулевого значения счетчика. Проверка списка отложенных вызовов.
- Декремент кванта текущего потока.

По главному тику

- Добавление в очередь на выполнение функций, относящихся к работе планировщика-диспетчера
- Пробуждение системных процессов, таких, как swapper и pagedaemon (процедура wakeup перемещает дескрипторы процессов из очереди «спящих» в очередь «готовых к выполнению»)
- 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 текущему процессу сигнала SIGXCPU, если израсходован выделенный ему квант процессорного времени.

1.2 Windows

По тику

- Инкремент счётчика системного времени
- Декремент счетчиков отложенных задач
- Декремент остатка кванта текущего потока.

По главному тику

- Инициализация диспетчера настройки баланса (путём освобождения объекта «событие» каждую секунду)
- Пробуждение системных процессов, таких, как swapper и pagedaemon

По кванту

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

2. Пересчет динамических приоритетов(только у пользовательских процессов)

1.1 UNIX/LINUX

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.

В ядре системы SVR4 были определены несколько *точек вытеснения*. Эти точки являются определенными местами в коде ядра, в которых все структуры данных находятся в безопасном состоянии, а ядро системы готово начать выполнение большого объема операций. Когда достигается одна из точек вытеснения, ядро проверяет флаг **kprunrun**, который указывает на готовность к выполнению процесса реального времени. Если флаг установлен, ядро системы вытеснит текущий процесс.

Планирование процессов в UNIX основано на *приоритете* процесса. Планировщик всегда выбирает процесс с наивысшим приоритетом. Приоритет процесса не является фиксированным и динамически изменяется системой в зависимости от использования вычислительных ресурсов, времени ожидания запуска и текущего состояния процесса. Если процесс готов к запуску и имеет наивысший приоритет, планировщик приостановит выполнение текущего процесса (с более низким приоритетом), даже если последний не "выработал" свой временной квант.

Приоритет процесса/потока задается любым целым числом, лежащим в диапазоне от 0 до 139, то есть существует 140 уровней приоритета(для обычных потоков и потоков реального времени). Чем меньше такое число, тем выше приоритет. В Unix приоритеты от 0 до 49 зарезервированы для ядра, следовательно, прикладные процессы могут обладать приоритетом в диапазоне 50–139. В Linux потоки реального времени представлены приоритетами от 0 до 99, остальные отведены для обычных потоков разделения времени.

Каждый процесс имеет два атрибута приоритета: *текущий приоритет,* на основании которого происходит планирование, и *относительный приоритет,* называемый *nice number* (или просто nice), который задаётся при порождении процесса и влияет на текущий приоритет. По умолчанию он равен 0, но его можно

изменить при помощи системного вызова *nice(value)*, где *value* меняется от –20 до +19.

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

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

В UNIX структура proc содержит следующие поля, относящиеся к приоритетам:

p_pri	Текущий приоритет планирования	Используется для хранения временного приоритета для выполнения в режиме ядра.
p_usrpri	Приоритет режима задачи	Используется для хранения приоритета, который будет назначен процессу при возврате в режим задачи.
p_cpu	Результат последнего измерения использования процессора	Содержит величину результата последнего сделанного измерения использования процессора процессом. Инициализируется нулем.
p_nice	Фактор nice, устанавливаемый пользователем	(в диапазоне от 0 до 39 со значением 20 по умолчанию) Увеличение значения приводит к уменьшению приоритета.

Значения в данной структуре могут быть изменены в случае:

- Когда процесс находится в режиме задачи, значение его **p_pri** идентично **p_usrpri.**

- Когда процесс просыпается после блокирования в системном вызове, его приоритет будет временно повышен для того, чтобы дать ему предпочтение для выполнения в режиме ядра.

- Когда заблокированный процесс просыпается, ядро устанавливает значение его **p_pri,** равное приоритету сна события или ресурса (в диапазоне 0–49). Такой подход позволяет быстро завершить системный вызов, выполнение которого, в свою очередь, может блокировать некоторые системные ресурсы и приводить к бесконечному откладыванию.

- 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*).

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

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:

$$p\_usrpri = PUSER + \frac{p\_cpu}{4} + 2\ p\_nice$$

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

Если процесс до вытеснения другим процессом 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.

1.2 Windows

Ядро Windows не имеет центрального потока планирования. Вместо этого, когда поток не может больше выполняться, он сам вызывает планировщик, чтобы увидеть, не освободился ли в результате его действий поток с более высоким приоритетом планирования, который готов к выполнению. Если это так, то происходит переключение потоков.

Поскольку 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.	Поток уже работает в режиме ядра для выполнения операции над диспетчером или объектом ввода-вывода.
2. Поток подает сигнал об объекте (например, выставляет *up* на семафоре).	Работающий поток также находится в ядре. Однако после сигнализации некоторого объекта он может продолжить выполнение, поскольку сигнализация объекта никогда не приводит к блокировке.
3. Истекает квант времени потока.	Происходит прерывание в режим ядра, в этот момент поток выполняет код планировщика.

В системе имеется 32 приоритета с номерами от 0 до 31. Сочетание класса приоритета и относительного приоритета отображается на 32 абсолютных значения

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

приоритета. Номер в таблице определяет *базовый приоритет (base priority)* потока. Кроме того, каждый поток имеет *текущий приоритет (current priority)*.

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: Реального времени — *Real-time (4)*, Высокий — *High (3)*, Выше обычного — *Above Normal (7)*, Обычный — *Normal (2)*, Ниже обычного — *Below Normal (5)* и Простоя — *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 Простоя — *Idle (–15)*.

Для использования этих приоритетов при планировании система поддерживает массив из 32 списков потоков, соответствующих всем 32 приоритетам (от 0 до 31). Каждый список содержит готовые потоки соответствующего приоритета. Базовый алгоритм планирования делает поиск по массиву от приоритета 31 до приоритета 0. Как только будет найден непустой список, поток выбирается сверху списка и выполняется в течение одного кванта. Если квант истекает, то поток переводится в конец очереди своего уровня приоритета и следующим выбирается верхний поток списка. Если готовых потоков нет, то процессор переходит в состояние ожидания, то есть переводится в состояние более низкого энергопотребления и ждет прерывания.

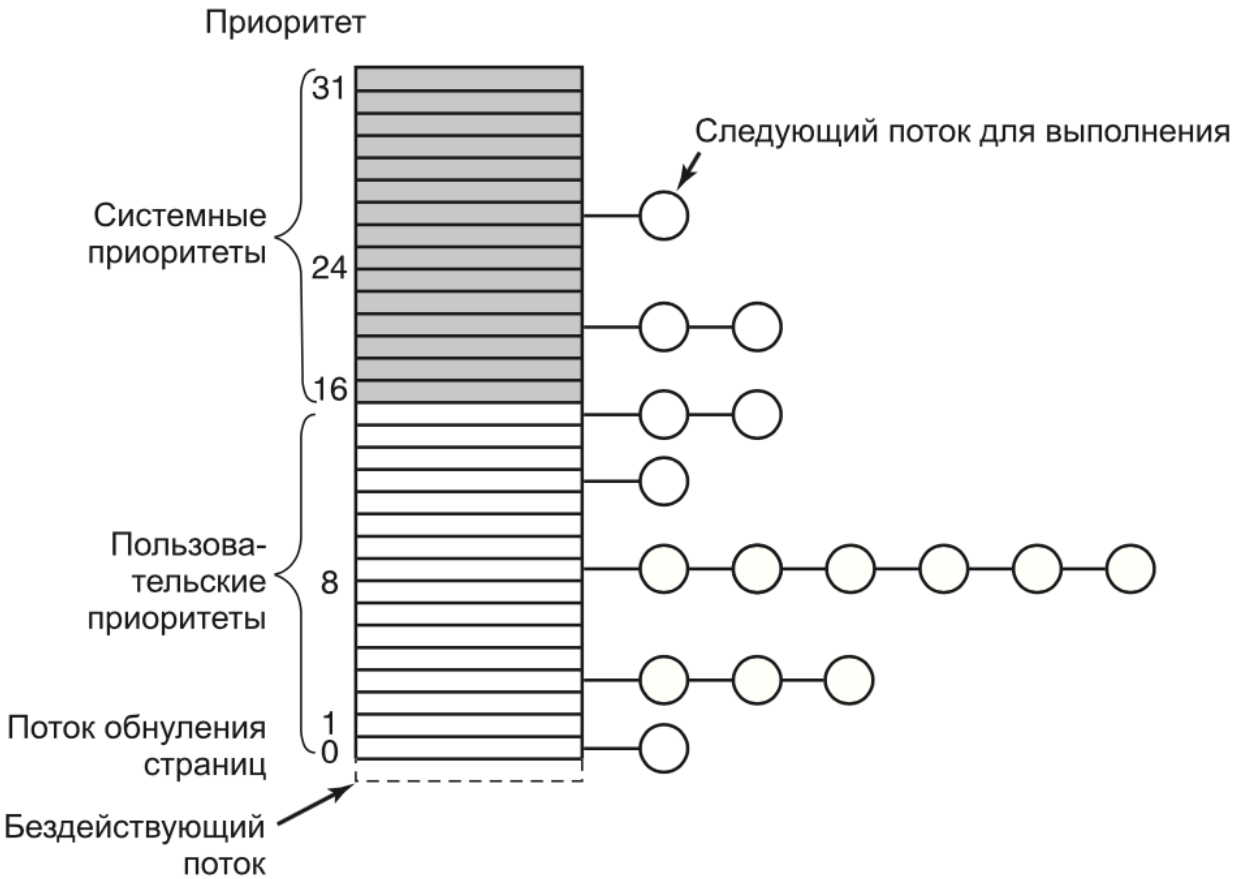

Рис. 11.13. Windows Vista поддерживает 32 приоритета для потоков

На схеме показано, что реально 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 у ровней, т о его приоритет станет равным только 15 (если приоритет потока равен 15, то повысить его нельзя).

Таблица 5.6. Рекомендуемые значения повышения приоритета

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

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 у стройством с меньшим г арантированным 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 у правления 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 1 5. 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, т о после снижения приоритета он возвращается в очередь готовых потоков. В итоге через 4 секунды его приоритет может быть с нова 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 у 10 потоков за один проход. Обнаружив 10 потоков, приоритет которых следует повысить (что говорит о высокой загруженности 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 исполняется код режима ядра, Windows использует *приоритеты прерываний IRQL*.

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.

На рис. 5.15 показаны уровни запроса прерываний (IRQL) для 32-разрядной системы. 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.

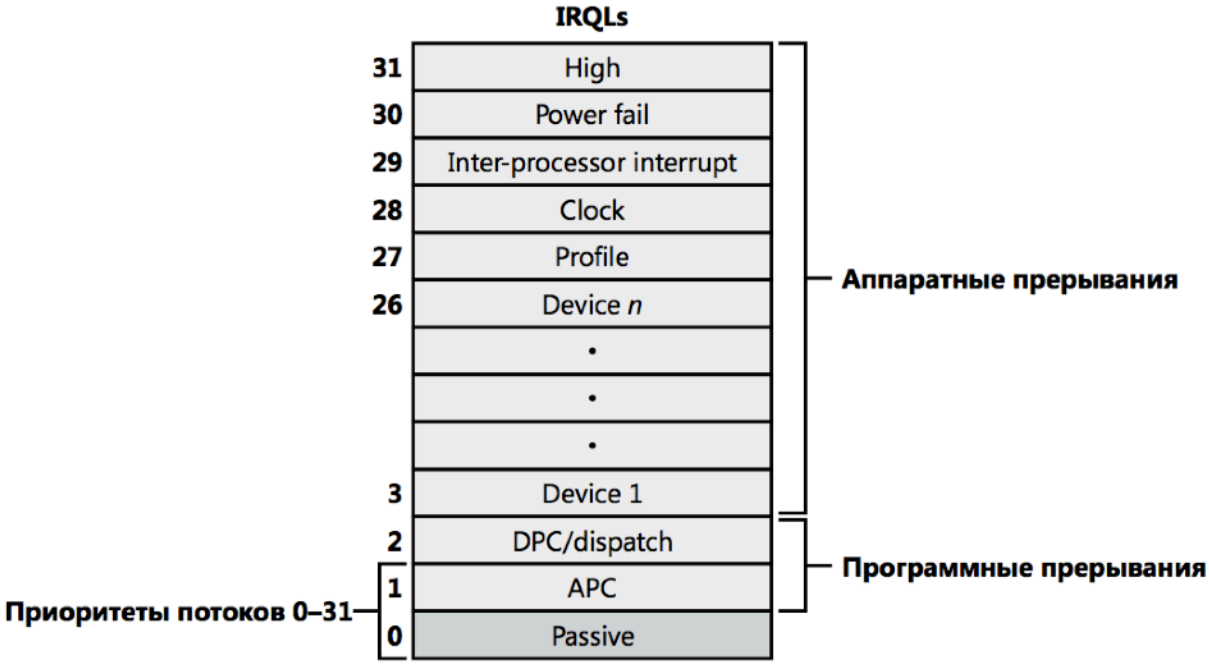

Рис. 5.15. Сопоставление приоритетов потоков с IRQL-уровнями на системе x86

Потоки, запущенные в режиме ядра, несмотря на изначальное планирование на пассивном уровне или уровне APC, могут поднять IRQL на более высокие уровни.

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.

Поток, запущенный в режиме ядра, может быть запущен на APC-уровне, если он запускает специальный APC-вызов ядра, или он может временно поднять IRQL до APC-уровня, чтобы заблокировать доставку специальных APC-вызовов ядра. Поток, выполняемый в режиме ядра на APC-уровне, может быть прерван в пользу потока с более высоким приоритетом, запущенным в пользовательском режиме на уровне *passive*.

Вывод

И в ОС Windows, и UNIX обработчик системного таймера выполняет схожие основные функции:

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

Это обусловлено тем, что обе операционные системы являются системами разделения времени с вытеснением и динамическими приоритетами.

Однако в планировании семейства этих систем сильно различаются. Классический Unix имеет невытесняющее ядро, а Windows является полностью вытесняющей. Алгоритмы планирования имеют схожие черты и основаны на очередях, но взаимодействия планировщика и потоков в данных ОС имеют явные различия, к примеру, в Windows потоки сами вызывают планировщик для пересчета их приоритетов.