

МГТУ им. БАУМАНА

ЛАБОРАТОРНАЯ РАБОТА №1 ч.2.

По курсу: "ОПЕРАЦИОННЫЕ СИСТЕМЫ"

**Функции обработчика системного
таймера. Пересчет динамических
приоритетов**

Работу выполнил: Мокеев Даниил, ИУ7-56

Преподаватель: Рязанова Н.Ю.

Москва, 2020

1 | Функции обработчика прерываний от системного таймера

1.1 Unix

1.1.1 По тикку

- Инкремент счетчика использования процессора текущим процессом;
- инкремент часов и других таймеров системы;
- декремент счетчика времени, оставшегося до отправления на выполнение отложенных вызовов и отправка отложенных вызовов на выполнение, при достижении нулевого значения счетчика;
- декремент кванта.

1.1.2 По главному тикку

- Добавление в очередь отложенных вызовов функций планировщик;
- пробуждение системных процессов swapper и pagedaemon;
- декремент счетчиков времени, оставшегося до отправления сигналов тревоги:
 - SIGALRM — сигнал будильника реального времени, который отправляется по истечении заданного промежутка реального времени;
 - SIGPROF — сигнал будильника профиля процесса, который измеряет время работы процесса;
 - SIGVTALRM — сигнал будильника виртуального времени, который измеряет время работы процесса в режиме задачи.

1.1.3 По кванту

- При превышении текущим процессом выделенного кванта, отправка сигнала SIGXCPU этому процессу.

1.2 Windows

1.2.1 По тикку

- Инкремент счетчика системного времени;
- декремент счетчиков отложенных задач;
- декремент остатка кванта текущего потока;
- активация обработчика ловушки профилирования ядра.

1.2.2 По главному тикку

- Инициализация диспетчера настройки баланса путем освобождения объекта «событие», на котором он ожидает.

1.2.3 По кванту

- Инициализация диспетчеризации потоков путем добавления соответствующего объекта DPC в очередь.

2 | Пересчет динамических приоритетов

2.1 Unix

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.

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.

2.1.1 Приоритеты процессов

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:

- `p_pri` — текущий приоритет планирования;
- `p_usrpri` — приоритет режима задачи;
- `p_ru` — результат последнего измерения использования процессора;
- `p_nice` — фактор «любезности», устанавливаемый пользователем.

Ядро системы связывает приоритет сна с событием или ожидаемым ресурсом, из-за которого процесс может заблокироваться. Приоритет сна является величиной,

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	43
Ожидание события	30	66

Таблица 2.1: Приоритеты сна

определяемой для ядра, и потому лежит в диапазоне 0-49 (Значение приоритета сна для терминального ввода - 28, для операций ввода-вывода - 20). Когда заблокированный процесс просыпается, ядро устанавливает `r_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 2.1 приведены значения приоритетов сна для систем 4.3BSD UNIX и SCO UNIX (OpenServer 5.0). 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.

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

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 `r_cpu` содержит величину последнего измерения использования процессора процессом. При создании процесса это поле инициализируется нулем. На каждом тике обработчик таймера увеличивает `r_cpu` на единицу для текущего процесса, вплоть до максимального значения — 127. Каждую секунду ядро вызывает процедуру `schedcpu`, которая уменьшает значение `r_cpu` каж-

дого процесса исходя из фактора «полураспада». В 4.3 BSD для расчета применяется формула:

$$decay = \frac{2 \cdot load_average}{2 \cdot load_average + 1},$$

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:

$$p_{usrpri} = PUSER + \frac{p_cpu}{4} + 2 \cdot p_nice,$$

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 приведет 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.

2.2 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.

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.

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.

2.2.1 Приоритеты потоков

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)

Класс приоритета	Realtime	High	Above	Norman	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

Таблица 2.2: Отображение приоритетов ядра Windows на WIndows API

Затем назначается относительный приоритет потоков в рамках процессов:

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

- наивысший — Highest (2);
- выше обычного — Above-normal (1);
- обычный — Normal (0);
- ниже обычного — Below-normal (-1);
- низший — Lowest (-2);
- простой — Idle (-15).

Относительный приоритет — это приращение к базовому приоритету процесса.

Соответствие между приоритетами Windows API и ядра системы приведено в таблице 2.2.

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:

- По завершении операции ввода-вывода;
- по окончании ожидания на событии или семафоре исполнительной системы;
- по окончании ожидания потоками активного процесса;
- при пробуждении GUI-потоков из-за операции с окнами;
- 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.

2.2.2 Повышение приоритета по завершении операции ввода-вывода

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 2.3 приведены приращения приоритетов.

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

Таблица 2.3: Рекомендованные значения повышения приоритета

Приоритет потока всегда повышается относительно базового приоритета. На рисунке 2.1 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.

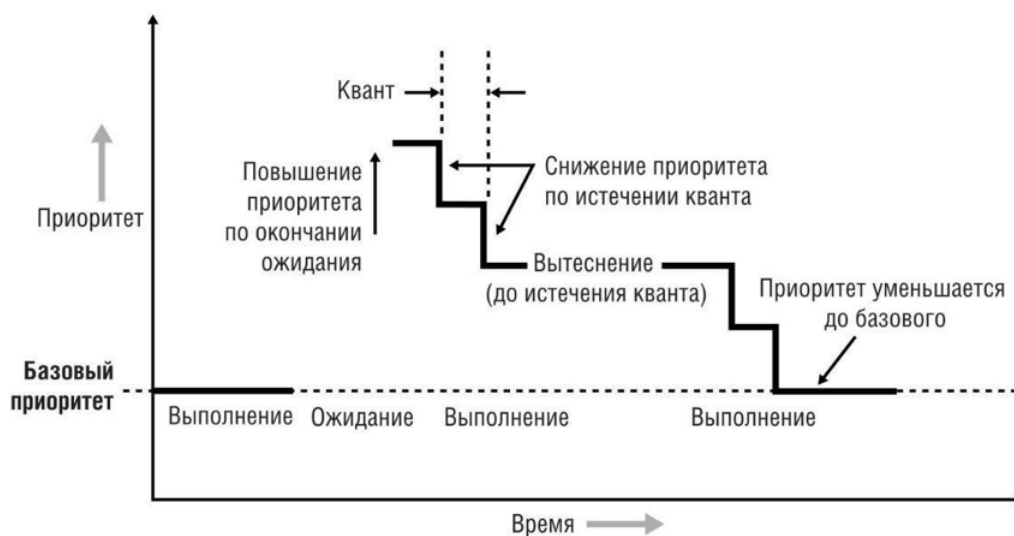


Рис. 2.1: Динамическое изменение приоритета

2.2.3 Повышение приоритета по окончании ожидания на событии или семафоре

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.

2.2.4 Повышение приоритета по окончании ожидания потоками активного процесса

Если поток в активном процессе завершает ожидание на объекте ядра, функция ядра `KiUnwaitThread` повышает его текущий приоритет на величину значения `PspPrioritySeparation`. `PspPrioritySeparation` — 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`.

2.2.5 Повышение приоритета при пробуждении GUI потоков

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.

2.2.6 Повышение приоритета при нехватке процессорного времени

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 пример-

но 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 секундам. 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.

2.2.7 Уровни запросов прерываний

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 (см. рис. 2.2).

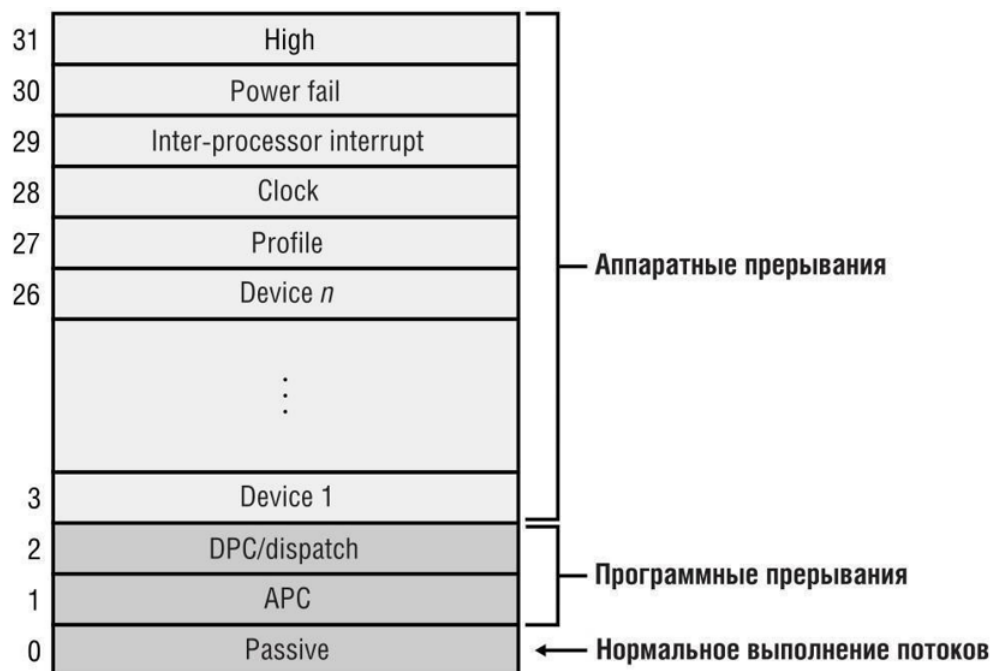


Рис. 2.2: Уровни запросов прерываний

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 ло-

вушки. 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.

3 | Заключение

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:

- Инкремент счетчика системного времени;
- декремент кванта;
- добавление функций планировщика в очередь отложенных вызовов;
- декремент счетчиков времени, оставшегося до выполнения отложенных вызовов;
- отправка отложенных действий на выполнение.

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.