



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 (часть 2) по дисциплине «Операционные системы»**

Тема Функции обработчика прерывания системного таймера в системах разделения времени

Студент Шахнович Дмитрий Сергеевич

Группа ИУ7-52Б

Преподаватели Рязанова Н.Ю.

Москва, 2024

# **1 Функции обработчика прерывания системного таймера**

## **1.1 Unix/Linux**

### **По тикку**

- Инкремент счётчика реального времени и других счётчиков времени(например в SVR4, lbolt – счётчик тиков с момента запуска системы);
- Декремент кванта текущего потока;
- Инкремент счётчиков процессорного времени в режимах задачи и ядра;
- Декремент счётчика тиков отложенных вызовов. Если счётчик обнулился, то установка флага запуска ожидающего вызова.

### **По главному тикку**

- Пробуждение системных процессов (например pagedaemon, swapper для Unix, kswapd для Linux);
- Отложенный вызов функций планировщика (пересчёт приоритетов);
- Декремент счётчиков времени, до отправки сигналов будильников SIGVTALRM, SIGPROF, SIGALRM.

### **По кванту**

- Отправка сигнала SIGXCPU о том, что процесс израсходовал квант.

## **1.2 Windows**

### **По тикку**

- Инкремент счётчика реального времени;
- Декремент кванта текущего потока;
- Декремент счётчика тиков отложенных вызовов;
- Запись адреса выполняемого кода при активном профилировании ядра.

### **По главному тикку**

- Запуск диспетчеризации настройки баланса(каждую секунду);

### **По кванту**

- Отложенный вызов (DPC) диспетчеризации потоков.

## 2 Пересчёт динамических приоритетов

Операционные системы Unix и Windows являются системами разделения времени с динамическими приоритетами, соответственно процессорное время выделяется процессам в соответствии с их приоритетом, при этом их приоритеты меняются по ходу выполнения программы.

### 2.1 Unix/Linux

Планировщик Unix предоставляет процессор каждому процессу на небольшой промежуток времени, который называется **квантом времени**, а затем переключает на следующий процесс. При переключении процессов планировщик заставляет процессор выполнить **переключение контекста**. При этом действии ядро системы сохраняет аппаратный контекст, то есть текущие значения регистров общего назначения, управления памятью и других. После сохранения, ядро системы загружает аппаратный контекст процесса, который будет выполняться.

В системах Unix планировщик выбирает обладающие наиболее высоким приоритетом, среди всех. При этом если процессы обладают одинаковым приоритетом, то применяется вытесняющее квантование времени для этих процессов. При этом изменение приоритетов происходит динамически и если какой-то из высокоприоритетных процессов станет готов к выполнению, то он вытеснит текущий процесс, даже если тот не израсходовал свой квант времени.

Традиционное ядро Unix является строго невытесняющим (или по-другому «непрерываемым»), то есть процесс находящийся в режиме ядра не может быть вытеснен другим, даже более высокоприоритетным процессом. Выполняющийся процесс может добровольно освободить процессор, в случае блокирования на ресурсе, или вытеснен при переходе в режим задачи. Такой подход решает проблемы синхронизации, связанные с одновременным доступом нескольких процессов к одним и тем же объектам ядра.

Современные системы Unix/Linux являются вытесняющими, то есть процесс в режиме ядра может быть вытеснен более высокоприоритетным процессом.

Приоритет процесса может представляться любым целым числом в диапазоне от 0 до 127. Для системы 4.3BSD UNIX чем ниже это число, тем более высокий приоритет имеет процесс. Значения от 0 до 49 зарезервированы для режима ядра, остальные 50-127 отведены под прикладные процессы.

Структура *proc*, описывающая дескриптор процесса содержит следующие поля, характеризующие приоритет процесса:

- *p\_pri* – Текущий приоритет планирования;
- *p\_usrpri* – Приоритет процесса в режиме задачи;
- *p\_cpu* – Результат последнего измерения использования процессора;
- *p\_nice* – Фактор «Любезности» процесса, устанавливаемый пользователем.

Поле *p\_pri* используется планировщиком для принятия решения о том, какой процесс направить на выполнение. Значение *p\_pri* совпадает с *p\_usrpri*, когда процесс находится в ре-

жиме задачи. Когда процесс просыпается после блокирования в системном вызове, его значение  $p\_pri$  понижается, то есть приоритет повышается, для того, чтобы дать ему предпочтение среди остальных процессов.

Ядро системы связывает приоритет сна с событием или ожидаемым ресурсом. Когда заблокированный процесс просыпается ядро устанавливает его значение  $p\_pri$  равным приоритету сна или ресурса, при этом все эти приоритеты лежат в диапазоне 0-49, что повышает приоритет процесса по сравнению с процессами в режиме задачи. На рисунке 2.1 приведены значения приоритетов сна для систем 4.3BSD UNIX и SCO UNIX (OpenServer 5.0).

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

Рисунок 2.1 — Системные приоритеты сна

После завершения системного вызова перед возвращением в режим задачи ядро восстанавливает приоритет режима задачи, сохранённый перед выполнением системного вызова. Это может привести к понижению приоритета, что, в свою очередь, вызовет переключение контекста.

В режиме задачи приоритет зависит от двух факторов: «любезности» и последней измеренной величиной использования процессора.

Степень «любезности» процесса определяется числом от 0 до 39 со значением 20 по умолчанию. Увеличение этого числа приводит к уменьшению приоритета процесса, таким образом такой процесс будет уступать места более высокоприоритетным процессам. Уменьшить эту величину может только суперпользователь, так как это приведёт к увеличению приоритета процесса.

Поле  $p\_cpu$  содержит последний результат измерения использования процессора. Эта величина увеличивается на 1 на каждом тике, при выполнении процесса, вплоть до максимального значения в 127. Также каждую секунду ядро выпускает процедуру  $schedcpu()$  которая уменьшает значение  $p\_cpu$  по фактору «полураспада». В 4.3BSD Unix для этого используется формула 2.1

$$p\_cpu = p\_cpu \frac{2 * load\_average}{2 * load\_average + 1}, \quad (2.1)$$

где  $load\_average$  – среднее количество готовых к выполнению процессов за последнюю секунду. После расчёта  $p\_cpu$  рассчитывается приоритет процесса в режиме задачи 2.2.

$$p\_usrpri = PUSER + \frac{p\_cpu}{4} + 2 * p\_nice, \quad (2.2)$$

где  $PUSER = 50$  – базовый приоритет процессов в пользовательском режиме.

В результате долгое использование процессора повышает  $p\_pri$ , что понижает приоритет процесса, с другой стороны долгое нахождение в очереди процессов повышает его приоритет за счёт фактора «полураспада». Такая схема предотвращает бесконечное откладывание.

## 2.2 Windows

Планировщик Windows реализует приоритетная, вытесняющая система планирования. Единицей диспетчеризации, как и в случае `unix`, является процесс, которому при его создании задаётся базовый приоритет. Приоритеты потоков задаются относительно приоритета процесса.

Как и в случае планировщика `Unix`, планировщик Windows предоставляет потоку квант времени, при этом для выполнения всегда выбирается готовый к запуску поток с наивысшим приоритетом. Поток может перестать выполняться, если:

- появился более высокоприоритетный поток (например вышел из ожидания). Тогда текущий поток будет вытеснен более высокоприоритетным потоком;
- поток добровольно отказался от кванта, путём входа в ожидание какого-нибудь объекта (события, мьютекса, семафора, т.д.);
- квант, выделенный потоку закончился. Тогда квант может быть передан более высокоприоритетным потокам, если они есть, потокам с тем же приоритетом по очереди, если они есть, либо тому же самому потоку, если есть только потоки с более низким приоритетом.

Приоритет в Windows задаётся одним из 32 значений в диапазоне от 0 до 31, при этом 0 зарезервирован для потока обнуления страниц, 1-15 – изменяющиеся уровни, а 16-31 – уровни реального времени.

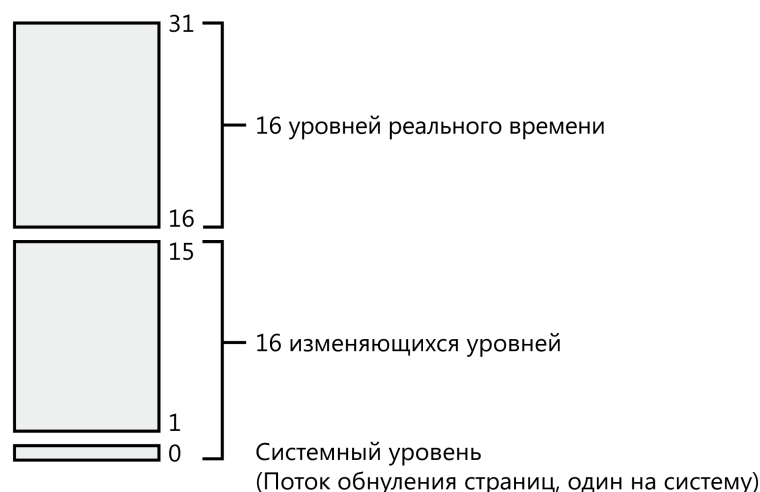


Рисунок 2.2 — Уровни приоритета потоков

Приоритет процесса присваивается Windows API при создании процесса и может быть одним из 6 значений:

- Реального времени – Real-time (4);
- Высокий – High (3);
- Выше обычного – Above Normal (7);
- Обычный – Normal (2);
- Ниже обычного – Below Normal (5);
- Простоя – Idle (1).

Затем для отдельных потоков назначается относительный приоритет потока, который имеет либо положительное значение либо отрицательное:

- 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);
- Простоя – Idle (-15).

Таким образом в Windows API каждый поток имеет базовый приоритет, являющийся функцией класса приоритета процесса и его относительного приоритета процесса. На рисунке 2.3 представлена таблица, отображающая относительный приоритет и приоритет процесса на базовый приоритет потока.

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

Рисунок 2.3 — Отображение приоритетов ядра Windows на Windows API

У каждого потока есть 2 вида приоритетов: текущий и базовый. Текущий используется планировщиком при выборе следующего потока. В общем случае текущий приоритет равен базовому, однако он может быть повышен планировщиком. Такие повышения могут иметь следующую природу:

- **Повышение вследствие событий планировщика или диспетчера:** событие установлено или отправило сигнал, мьютекс был освобожден или ликвидирован, был освобожден семафор, в очередь была вставлена запись или очередь была очищена и др. – повышение приоритета на 2;
- **Повышения приоритета, связанные с завершением ожидания:** увеличение при-

оритета потока, вышедшего из ожидания в состояние ready – повышение приоритета на 1;

— **Повышение при ожидании ресурсов исполняющей системы:** если некоторый поток блокируется при получения ресурса системы, который уже находится в исключительном владении другого потока, то для избежания зависания приоритет владеющего потока повышается через некоторое время, после начала ожидания – приоритет повышается до 14 (если был < 14);

— **Повышение приоритета потоков первого плана после ожидания:** большее повышение приоритета интерактивных процессов для улучшения их отзывчивости – приоритет повышается на текущее значение переменной *PsPrioritySeparation*;

— **Повышения приоритета, связанные с перезагруженностью центрального процессора:** раз в секунду диспетчер настройки баланса, который является частью системного потока, канирует очередь готовых потоков в поиске тех из них, которые находятся в состоянии ready около 4 секунд и повышает их приоритет – приоритет повышается до 15 (если был < 15);

— **Повышение приоритета после завершения ввода-вывода:** 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. Рекомендованные значения для данных повышений представлены на рисунке 2.4, хотя действительные значения повышения определяются драйвером соответствующего устройства (как видно из рисунка аудио устройство имеет более высокий приоритет из-за того, что слух человека чувствительней к задержкам, по сравнению со зрением).

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

Рисунок 2.4 — Рекомендуемые значения повышения приоритета

Большинство описанных выше повышений приоритетов перестают действовать после завершения кванта или нескольких, выделенных процессу. На рисунке 2.5 представлен пример повышения и понижения приоритетов.

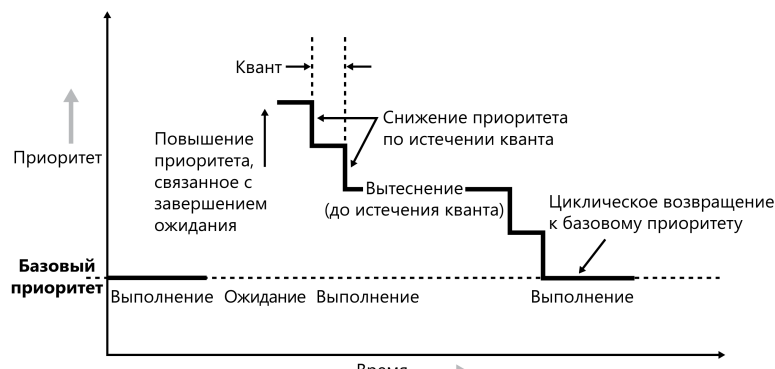


Рисунок 2.5 — Повышение и снижение приоритета

Для различных мультимедийных приложений, вроде воспроизведение видео/аудио, трёхмерных компьютерных игр требуются минимальные задержки, которые стандартные средства планировщика реализовать не могут, для клиентских версий была разработана служба MMCSS, чьей целью было гарантировать проигрывание мультимедийного контента приложений, зарегистрированных с этой службой, без каких-либо сбоев.

В действительности эта служба не повышает приоритет, как это делает планировщик, а меняет базовый приоритет процессов и потоков на время её работы. Важным свойством службы являются категории планирования, которые делят зарегистрированные процессы по приоритетам (представлены на рисунке 2.6).

Категория	Приоритет	Описание
High (Высокая)	23–26	Потоки профессионального аудио (Pro Audio), запущенные с приоритетом выше, чем у других потоков на системе, за исключением критических системных потоков
Medium (Средняя)	16–22	Потоки, являющиеся частью приложений первого плана, например Windows Media Player
Low (Низкая)	8–15	Все остальные потоки, не являющиеся частью предыдущих категорий
Exhausted (Исчерпавших потоков)	1–7	Потоки, исчерпавшие свою долю времени центрального процессора, выполнение которых продолжится, только если не будут готовы к выполнению другие потоки с более высоким уровнем приоритета

Рисунок 2.6 — Категории планирования

Механизм, положенный в основу MMCSS, повышает приоритет потоков внутри зарегистрированного процесса до уровня, соответствующего их категории планирования и относительного приоритета внутри этой категории на гарантированный срок. Затем он снижает категорию этих потоков до Exhausted, чтобы другие, не относящиеся к мультимедийным приложениям потоки, также получили шанс на выполнение. По умолчанию мультимедийные потоки получают 80% процессорного времени. При этом сама служба MMCSS выполняется с приоритетом 27, так как ей нужно вытеснить любые управляемые ей потоки.

## IRQL



Контроллеры прерываний сами устанавливают приоритетность своих прерываний, однако Windows определяет свою схему приоритетности, называемую IRQL, которая представлена на рисунке 2.7.

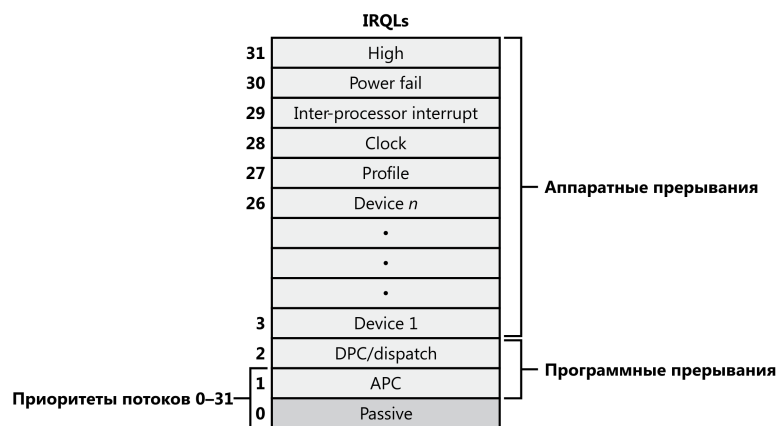


Рисунок 2.7 — Сопоставление приоритетов потоков с IRQL-уровнями на системе x86

Потоки обычно запускаются на уровне IRQL 0 или на уровне IRQL 1 (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.

Потоки, запущенные в режиме ядра, несмотря на изначальное планирование на пассивном уровне или уровне APC, могут поднять IRQL на более высокие уровни, например, при выполнении системного вызова, который может включать в себя диспетчеризацию потока, диспетчеризацию памяти или ввод-вывод. 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, так как прерывания обрабатываются в порядке из приоритетности. Поток выполняется на dispatch-уровне и выше, блокирует активность планировщика потоков и мешает контекстному переключению на своем процессоре.

# ЗАКЛЮЧЕНИЕ

Так как семейства операционных систем Windows и Unix являются системами разделения времени, то функции их обработчиков системного таймера схожи. Основные из них:

- Декремент кванта, по окончании которого инициализируется диспетчеризация, которая передаёт квант другому процессу;
- Инкремент счётчиков времени;
- Инициализация работы планировщика;
- Декремент счётчиков отложенных вызовов.

В обоих семействах также используются динамически пересчитываемые приоритеты, которые решают позволяют работу нескольких процессов и решают проблемы бесконечного откладывания.

## СПИСОК ИСПОЛЬЗОВАННЫХ МАТЕРИАЛОВ

- 1) Вахалия Ю. UNIX изнутри. - СПб.: Питер, 2003. - 844 с.
- 2) Робачевский А. Операционная система UNIX. - СПб.: БХВ-Петербург, 2002. - 528 с.
- 3) Русинович М., Соломон Д. Внутреннее устройство Microsoft Windows. - 6-е изд. - СПб.: Питер, 2013. - 800 с.