



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 прерывания от системного таймера  
в системах разделения времени

**Дисциплина:** Операционные системы

Студент

ИУ7-54Б

(Группа)

Л.Е.Тартыков

(Подпись, дата)

(И.О. Фамилия)

Преподаватель

Н.Ю.Рязанова

(Подпись, дата)

(И.О. Фамилия)

*Москва, 2021*

# 1 Функции обработчика прерываний от системного таймера в системах разделения времени

## 1.1 ОС семейства Windows

Обработчик прерывания от системного таймера **по тик** выполняет следующие задачи:

- инкремент счетчика системного времени;
- декремент кванта текущего потока на величину, равную количеству тактов процессора, произошедших за тик. Если количество тактов процессора, затраченных потоком, достигает квантовой цели, то запускается обработка истечения кванта;
- декремент счетчиков времени отложенных задач;
- если активен механизм профилирования ядра, то инициализирует отложенный вызов обработчика ловушки профилирования ядра путем постановки в очередь DPC (Deferred Procedure Call - отложенный вызов процедуры): обработчик ловушки профилирования регистрирует адрес команды, выполнявшейся на момент прерывания.

Обработчик прерывания от системного таймера **по главному тик** выполняет следующие задачи:

- возвращает задействованный в системе объект “;событие“, который ожидает диспетчер настройки баланса;

Обработчик прерывания от системного таймера **по кванту** выполняет следующие задачи:

- инициализирует диспетчеризацию потоков путем постановки соответствующего объекта в очередь DPC;

## 1.2 ОС семейства Unix/Linux

Обработчик прерывания от системного таймера **по тик** выполняет следующие задачи:

- инкремент счетчика тиков аппаратного таймера;
- инкремент часов и других таймеров системы;
- декремент счетчика времени до отправления на выполнение отложенного вызова;
- инкремент счетчика использования процессора текущим процессом;
- декремент кванта текущего потока.

Обработчик прерывания от системного таймера **по главному тик** выполняет следующие задачи:

- регистрирует отложенные вызовы функций, относящихся к работе планировщика, такие как пересчет приоритетов;
- пробуждает из состояния прерываемого сна системных вызовов `swapper` и `pagedaemon`. Пробуждение - регистрация отложенного вызова процедуры `wakeup`, которая перемещает дескрипторы процессов из списка "спящие" в очередь готовых процессов;
- декремент счетчика времени, оставшегося до отправки одного из следующих сигналов:
  - `SIGALRM` - сигнал, посылаемый процессу по истечении времени, которое предварительно задано функцией `alarm()`;
  - `SIGPROF` - сигнал, посылаемый процессу по истечении времени, заданного в таймере профилирования;
  - `SIGVTALARM` - сигнал, посылаемый процессу по истечении времени, заданного в "виртуальном" таймере.

Обработчик прерывания от системного таймера **по кванту** выполняет следующие задачи:

- посылает текущему процессу сигнал SIGXCPU, если он израсходовал выделенный ему квант процессорного времени.

## 2 Пересчет динамических приоритетов

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

### 2.1 Пересчет динамических приоритетов в ОС семейства Windows

В ОС семейства Windows при создании процесса ему назначается базовый приоритет; относительно него потоку назначается относительный приоритет.

Планирование осуществляется на основании приоритетов потоков, готовых к выполнению. Планировщик вытесняет поток с более низким приоритетом, когда поток с более высоким приоритетом становится готовым к выполнению. По истечении кванта времени текущего потока ресурс передается первому потоку (с наибольшим приоритетом) в очереди готовых на выполнение.

Диспетчер настройки баланса выполняет сканирование очереди готовых потоков раз в секунду. Если обнаружены потоки, которые ожидают выполнение более четырех секунд, то диспетчер повышает их приоритет до пятнадцати. Как только квант процессорного времени истекает, приоритет потока снижается до базового. Если поток не был завершен за этот квант или был вытеснен потоком с более высоким приоритетом, то выполняется снижение приоритета и поток возвращается в очередь готовых потоков. Для минимизации расхода процессорного времени диспетчер настройки баланса сканирует лишь шестнадцать готовых потоков. Кроме того он повышает приоритет не более чем у десяти потоков за один проход. Если обнаружено десять потоков, приоритет которых необходимо повысить, то сканирование прекращается. При следующем проходе оно возобновляется с того места, где было прервано в прошлый раз. Наличие десяти потоков, приоритет которых следует повысить, го-

ворит о необычайной загруженности системы.

В операционных системах семейства Windows используется 32 уровня приоритета, от 0 до 31, как показано на рисунке 2.1:



Рисунок 2.1 – Уровни приоритета потоков.

Уровни приоритета потоков назначаются Windows API и ядром операционной системы. Сначала Windows API систематизирует процессы по классу приоритета, который присваивается им при их создании:

- реального времени (real-time) - 4;
- высокий (high) - 3;
- выше обычного (above normal) - 7;
- обычный (normal) - 2;
- ниже обычного (below normal) - 5;
- простой (idle) - 1.

Затем назначается относительный приоритет потоков в рамках процессов:

- критичный по времени (real-time) - 15;

- наивысший (high) - 2;
- выше обычного (above normal) - 1;
- обычный (normal) - 0;
- ниже обычного (below normal) - -1;
- низший (lowest) - -2;
- простой (idle) - -15.

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

Соответствие между приоритетами Windows API и ядра системы приведено на рисунке 2.2:

<b>Класс приоритета/ Относительный приоритет</b>	<b>Realtime</b>	<b>High</b>	<b>Above</b>	<b>Normal</b>	<b>Below Normal</b>	<b>Idle</b>
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 2.2 – Уровни приоритета потоков.

Текущий приоритет потока в динамическом диапазоне (от 1 до 15) может быть повышен планировщиком вследствие следующих причин:

- вследствие события планировщика или диспетчера;
- повышение приоритета владельца блокировки;
- после завершения ввода/вывода (см. таблицу 2.1);
- вследствие ввода из пользовательского интерфейса;
- длительного ожидания ресурса исполняющей системы;

- вследствие ожидания объекта ядра;
- в случае, когда готовый к выполнению поток не был запущен в течение длительного времени;
- проигрывание мультимедиа службой планировщика MMCSS.

Текущий приоритет потока в динамическом диапазоне может быть понижен до базового путем вычитания всех повышений.

Таблица 2.1 – Рекомендуемые значения повышения приоритета.

Устройство	Приращение
Диск, CD-ROM, параллельный порт, видео	1
Сеть, почтовый ящик, именованный канал, последовательный порт	2
Клавиатура, мышь	6
Звуковая плата	8

### 2.1.1 MMCSS

Мультимедийные потоки должны выполняться с минимальными задержками. Повышение приоритета проигрывания обычно выполняется службой пользовательского режима - служба планировщика класса мультимедиа (Multimedia Class Scheduler Service). MMCSS работает со следующими задачами:

- аудио;
- захват;
- распределение;
- игры;
- проигрывание;
- аудио профессионального качества;



- задачи администратора многооконного режима.

Каждая из этих задач включает информацию о свойствах, которые отличаются друг от друга. Одним из наиболее важным свойством для планирования потоков является категория планирования (scheduling category). Она является первичным фактором, который определяет приоритет потоков, зарегистрированных с MMCSS. На рисунке 2.3 показаны различные категории планирования:

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 2.3 – Категории планирования.

Механизм, положенный в основу MMCSS, повышает приоритет потоков внутри зарегистрированного процесса до уровня, соответствующего их категории планирования. Затем он снижает категорию этих потоков до exhausted, чтобы другие, которые не относятся к мультимедийным приложениям, могли получить ресурс.

## 2.2 Пересчет динамических приоритетов в ОС семейства Unix/Linux

В современных системах Unix/Linux в режиме ядра процесс может быть вытеснен более приоритетным процессом. Ядро было сделано вытесняющим для того, чтобы система могла обслуживать процессы реального времени, такие как аудио, видео.

Очередь готовых к выполнению процессов формируется согласно приоритетам процессов и принципу вытесняющего циклического планирования: в первую очередь выполняются процессы с большим приоритетом. Если процессы имеют одинаковый приоритет, то они выполняются в течение кванта времени циклически друг за другом. Если в очередь готовых к выполнению поступает процесс, который имеет более высокий приоритет, то планировщик вытесняет текущий процесс и предоставляет ресурс более приоритетному.

Приоритет представляет собой целое число из диапазона от 0 до 127 (чем меньше число, тем выше приоритет):

- в диапазоне от 0 до 49 находятся приоритеты ядра;
- в диапазоне от 50 до 127 - приоритеты прикладных задач.

Приоритеты прикладных задач могут изменяться во времени в зависимости от следующих двух факторов:

- фактор любезности;
- последней измеренной величиной использования процесса.

Фактор любезности - целое число в диапазоне от 0 до 39 со значением 20 по умолчанию. С увеличением фактора любезности происходит уменьшение приоритета процесса. Фоновым процессам автоматически задаются более высокие значения этого фактора. Фактор любезности может быть изменен суперпользователем при помощи системного вызова `nice()`.

На рисунке 2.4 приведены поля структуры `proc`, относящиеся к приоритетам:

<code>p_rti</code>	Текущий приоритет планирования
<code>p_usrpri</code>	Приоритет режима задачи
<code>p_cpu</code>	Результат последнего измерения использования процессора
<code>p_nice</code>	Фактор «любезности», устанавливаемый пользователем

Рисунок 2.4 – Поля структуры `proc`, относящиеся к приоритетам.

Планировщик использует `p_pri` для принятия решения о том, какой процесс направить на выполнение. У процесса, находящегося в режиме задачи, значения `p_pri` и `p_usrpri` идентичны. Значение текущего приоритета `p_pri` может быть повышено планировщиком для выполнения процесса в режиме ядра. В таком случае `p_usrpri` будет использоваться для хранения приоритета, который будет назначен процессу при возврате в режим задачи.

Ядро системы связывает приоритет сна с событием или ожидаемым ресурсом, из-за которого процесс может блокироваться. Когда процесс просыпается после блокирования в системном вызове, ядро устанавливает в поле `p_pri` приоритет сна - значение приоритета из диапазона от 0 до 49, зависящее от события или ресурса, по которому произошла блокировка.

При создании процесса поле `p_cpu` инициализируется нулем. На каждом тике обработчик таймера увеличивает поле `p_cpu` текущего процесса на единицу до максимального значения, равного 127. Каждую секунду обработчик прерывания инициализирует отложенный вызов процедуры `schedcpu()`, которая уменьшает значение `p_cpu` каждого процесса исходя из фактора “полураспада”. В системе 4.3BSD для расчета фактора полураспада используется формула (2.1):

$$decay = \frac{2 \cdot load\_average}{2 \cdot load\_average + 1} \quad (2.1)$$

где `load_average` - среднее количество процессов, которые находятся в состоянии готовности к выполнению, за последнюю секунду.

Процедура `schedcpu()` пересчитывает приоритеты для режима задачи всех процессов по формуле (2.2):

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

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

В результате, если процесс в последний раз процесс использовал большое количество процессорного времени, то его `p_cpu` будет увеличен. Это приведет к росту значения поля `p_usrpri` и, следовательно, к понижению приоритета. Чем дольше процесс простаивает в очереди на испол-

нение, тем больше фактор полураспада уменьшает его  $p_{ср}$ ; это приводит к повышению его приоритета. Такая схема предотвращает зависание низкоприоритетных процессов по вине операционной системы. Её применение предпочтительнее процессам, которые осуществляют множество операций ввода-вывода, в противоположность тем, которые производят много вычислений.

# Вывод

Функции обработчика прерывания от системного таймера в защищенном режиме для ОС семейств Windows и Unix/Linux выполняют схожие задачи:

- инициализируют отложенные действия, относящиеся к работе планировщика (пересчет приоритетов);
- выполняют декремент счетчиков времени: часов, таймеров, счетчиков времени отложенных действий, будильников реального времени;
- выполняют декремент кванта (текущего потока в Windows, текущего процесса в Unix/Linux).

Такая схожесть объясняется тем, что обе системы являются системами разделения времени с динамическими приоритетами и вытеснением.