



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»

## ОТЧЕТ

По лабораторной работе №1 часть 2

По курсу: «Операционные системы»

Тема: «Изучение функций системного таймера и особенностей пересчета  
динамических приоритетов для ОС Windows и Unix»

Студент:

Пронин А. С.

Группа:

ИУ7-52Б

Преподаватель:

Рязанова Н. Ю.

Оценка:

\_\_\_\_\_

Москва

2021

# Функции обработчика прерывания от системного таймера для Windows

## По тикку:

- инкремент счётчика системного времени;
- декремент остатка кванта текущего потока;
- декремент счетчиков отложенных задач;
- инициализация обработчика ловушки профилирования ядра.

## По главному тикку:

- инициализация объекта события для диспетчера настройки баланса (один раз в секунду)

## По кванту:

- инициализация диспетчеризации потоков:  
добавление соответствующего объекта в очередь DPC.

# Функции обработчика прерывания от системного таймера для Unix

## По тикку:

- инкремент счетчика тиков аппаратного таймера;
- инкремент счетчика использования процессора текущим процессом:  
инкремент поля `p_cpu` дескриптора текущего процесса на единицу, до максимального значения, равного 127;
- инкремент часов и других таймеров системы;
- декремент счетчика времени до отправления на выполнение отложенных вызовов;

- декремент кванта текущего потока.

### По главному тикку:

- инициализация отложенных вызовов (см. пояснения ниже) функций, относящиеся к работе планировщика;
- инициализация отложенного вызова (см. пояснения ниже) процедуры wakeup, которая перемещает дескрипторы процессов из очереди «спящих» в очередь «готовых к выполнению»;

Отложенные вызовы считаются обычными процедурами ядра и не должны выполняться с приоритетами прерываний. Поэтому обработчик прерываний таймера не выполняет эти вызовы на прямую. На каждом тике обработчик прерываний таймера проверяет, не нужно ли начать выполнение отложенного вызова. Если он находит ожидающий вызова, то выставляет флаг, указывающий на необходимость запуска *обработчика отложенного вызова*.

- декремент счетчика времени, оставшегося до отправления одного из сигналов:
  - SIGALARM - сигнал, посылаемый процессу по истечении действительного времени;
  - SIGPROF - сигнал, посылаемый процессу по истечении времени заданном в таймере профилирования;
  - SIGVTALARM - сигнал, посылаемый процессу по истечении времени, заданного в «виртуальном» таймере.

### По кванту:

- посылка текущему процессу сигнала SIGXCPU, если он израсходовал выделенный ему квант процессорного времени.

# Пересчет динамических приоритетов в Windows

В Windows при создании процесса, ему назначается базовый приоритет. Относительно базового приоритета процесса потоку назначается относительный приоритет. [1]

Планирование осуществляется на основании приоритетов потоков, готовых к выполнению. В Windows поток с более низким приоритетом вытесняется планировщиком, когда поток с более высоким приоритетом становится готовым к выполнению. Диспетчер настройки баланса сканирует очередь готовых потоков раз в секунду и, если обнаружены потоки, ожидающие выполнения более 4 секунд, диспетчер настройки баланса повышает их приоритет до 15. Как только квант истекает, приоритет потока снижается до базового приоритета. Если поток не был завершен за квант времени или был вытеснен потоком с более высоким приоритетом, то после снижения приоритета поток возвращается в очередь готовых потоков. Диспетчер настройки баланса сканирует лишь 16 готовых потоков и повышает приоритет не более чем у 10 потоков за один проход. При следующем проходе сканирование возобновляется с того места, где оно было прервано в прошлый раз. Наличие 10 потоков, приоритет которых следует повысить, свидетельствует о необычно высокой загрузке системы.

Windows использует 32 уровня приоритета (рисунок 1), от 0 до 31. Эти значения разбиваются на части следующим образом:

- шестнадцать уровней реального времени (от 16 до 31);
- шестнадцать изменяющихся уровней (от 0 до 15), из которых уровень 0 зарезервирован для потока обнуления страниц.

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).



Рис. 1: Уровни приоритета потоков

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).

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.

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

Рис. 2: Соответствие между приоритетами Windows API и ядра Windows

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

- повышение вследствие событий планировщика или диспетчера (сокращение задержек);
- повышение приоритета владельца блокировки;
- повышение приоритета после завершения ввода-вывода (сокращение задержек) (рисунок 3);
- повышение приоритета вследствие ввода из пользовательского интерфейса (сокращение задержек и времени отклика);
- повышение приоритета вследствие длительного ожидания ресурса исполняющей системы (предотвращение зависания);
- повышение вследствие ожидания объекта ядра;
- повышение приоритета в случае, когда готовый к выполнению поток не был запущен в течение длительного времени (предотвращение зависания и смены приоритетов);
- повышение приоритета проигрывания мультимедиа службой планировщика MMCSS.

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

Рис. 3: Рекомендуемые значения повышения приоритета

## Повышение приоритета проигрывания мультимедиа службой планировщика MMCSS

Повышение приоритета проигрывания мультимедиа обычно управляется службой пользовательского режима, которая называется службой планировщика класса мультимедиа — MultiMedia Class Scheduler Service (MMCSS). MMCSS работает с вполне определенными задачами, включая следующие:

- аудио;
- захват;
- распределение;
- игры;
- проигрывание;
- аудио профессионального качества;
- задачи администратора многооконного режима.

В свою очередь, каждая из этих задач включает информацию о свойствах, отличающих их друг от друга. Одно из наиболее важных свойств для планирования потоков называется категорией планирования — Scheduling Category, которое является первичным фактором, определяющим приоритет потоков, зарегистрированных с MMCSS. На рисунке 4 показаны различные 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, не относящиеся к мультимедийным приложениям потоки, также могли получить ресурс.

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

Рис. 4: Категории планирования

## Пересчет динамических приоритетов в Unix

Вытесняемое ядро означает, что процесс в режиме ядра может быть вытеснен более приоритетным процессом в режиме ядра. [2] Это сделано для того, чтобы система могла обслуживать процессы реального времени, такие как:

1. аудио;
2. видео.

В современных системах UNIX/Linux ядро является вытесняемым.

Приоритет задается любым целым числом, лежащим в диапазоне от 0 до 127. Чем меньше такое число, тем выше приоритет. Приоритеты от 0 до 49 зарезервированы для ядра, они являются фиксированными величинами. Прикладные процессы могут обладать приоритетом в диапазоне 50-127. В первую очередь выполняются процессы с большим приоритетом, а процессы с одинаковыми приоритетами выполняются в течение кванта времени циклически друг за другом. На рисунке 5 приведены 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 этом случае `p_usrpri` будет использоваться для



p_pri	Текущий приоритет планирования
p_usrpri	Приоритет режима задачи
p_cpu	Результат последнего измерения использования процессора
p_nice	Фактор «любезности», устанавливаемый пользователем

Рис. 5: Поля структуры *proc*, относящиеся к приоритетам

хранения приоритета, который будет назначен процессу при возврате в режим задачи. Фактор любезности – целое число в диапазоне от 0 до 39 со значением 20 по умолчанию. Увеличение фактора любезности приводит к уменьшению приоритета процесса. Фактор любезности процесса может быть изменен суперпользователем с помощью системного вызова *nice*. 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.

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. На рисунке 6 показано событие и связанное с ним значение приоритета сна в системе 4.3BSD

Приоритет	Значение	Описание
PSWP	0	Свопинг
PSWP + 1	1	Страничный демон
PSWP + 1/2/4	1/2/4	Другие действия по обработке памяти
PINOD	10	Ожидание освобождения inode
PRIBIO	20	Ожидание дискового ввода-вывода
PRIBIO + 1	21	Ожидание освобождения буфера
PZERO	25	Базовый приоритет
TTIPRI	28	Ожидание ввода с терминала
TTOPRI	29	Ожидание вывода с терминала
PWAIT	30	Ожидание завершения процесса-потомка
PLOCK	35	Консультативное ожидание заблокированного ресурса
PSLEP	40	Ожидание сигнала

Рис. 6: Системные приоритеты сна

Каждую секунду, обработчик прерывания инициализирует отложенный вызов процедуры `schedcpu()`, которая уменьшает значение `p_cpu` каждого процесса исходя из фактора "полураспада", который рассчитывается по формуле 1, где *load\_average* - это среднее количество процессов, находящихся в состоянии готовности к выполнению, за последнюю секунду.

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

Процедура `schedcpu()` пересчитывает приоритеты для режима задачи всех процессов по формуле 2, где *PUSER* - базовый приоритет в режиме задачи, равный 50.

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

В результате, если процесс в последний раз, т.е. 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. Такая схема предотвращает бесконечное откладывание низкоприоритетных процессов по вине операционной системы. Ее применения предпочтительно процессам, осуществляющим много операций ввода-вывода, в противоположность процессам, производящим много вычислений.

## Вывод

Несмотря на то, что Windows и Unix разные операционные системы обработчик системного таймера выполняет схожие основные функции:

- инициализируют отложенные действия (такие как пересчет приоритетов);
- выполняют инкремент/декремент счетчиков времени:
  1. часов;
  2. таймеров;
  3. будильников реального времени;
  4. счетчиков времени отложенных действий.
- уменьшает квант процессорного времени, выделенного процессу.

Обе операционные системы являются системами разделения времени с вытеснением и динамическими приоритетами.

Приоритет пользовательского процесса в ОС Unix/Linux может динамически пересчитываться, в зависимости от фактора любезности, `r_sru` и базового приоритета. Приоритеты ядра являются фиксированными величинами.

При создании процесса в Windows, ему назначается базовый приоритет. Относительно базового приоритета процесса потоку назначается относительный приоритет, таким образом у потока нет своего приоритета. Приоритет потока пользовательского процесса может быть динамически пересчитан.

В любой системе у процесса базовый приоритет. Классическое ядро Unix не было многопоточным. Современные ядра и ядра Linux многопоточные.

## Список использованных источников

- [1] Ионеску А. Йосифович П. Руссинович М., Соломон Д. *Внутреннее устройство Windows*. 7-е изд. СПб.: Питер, 2018. 944 с.
- [2] Вахалия Ю. *UNIX изнутри*. СПб.: Питер, 2003. 844 с.