



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 дисциплине «Операционные системы»

Тема Прерывания таймера в Windows и UNIX

Студент Романов А. В.

Группа ИУ7-53Б

Преподаватель Рязанова Н. Ю.

Москва — 2020 г.

# 1 Функции обработчика прерывания от системного таймера в защищённом режиме

## 1.1 UNIX-системы

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

- инкремент счётчика тиков аппаратного таймера;
- декремент кванта текущего потока;
- обновление статистики использования процессора текущим процессом – инкремент поля `c_cpu` дескриптора текущего процесса до максимального значения 127.
- инкремент часов и других таймеров системы;
- декремент счетчика времени до отправления на выполнение отложенных вызовов, при достижении счетчиком нуля выставление флага для обработчика отложенных вызовов.

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

- регистрирует отложенные вызовы функций, относящиеся к работе планировщика, такие как пересчет приоритетов;
- пробуждение в нужные моменты системных процессов **swapper** и **pagedaemon**. Пробуждение обозначает регистрацию отложенного вызова процедуры **wakeup**, которая перемещает дескрипторы процессов из списка «спящих» в очередь готовых к выполнению.
- декремент счётчика времени, отвечающий за оставшееся время до отправки одного из сигналов:
  - **SIGALRM** – сигнал, посылаемый процессу по истечении времени, заданного функцией **alarm()**;
  - **SIGPROF** – сигнал, посылаемый процессу по истечении времени заданного в таймере профилирования;
  - **SIGVTALRM** – сигнал, посылаемый процессу по истечении времени, заданного в «виртуальном» таймере.

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

- послать сигнал текущему процессу сигнал **SIGXCPU**, если тот превысил выделенную для него квоту использования процессора.

## 1.2 Windows-системы

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

- инкремент счётчика системного времени;
- декремент кванта текущего потока на величину, равную количеству тактов процессора, произошедших за тик. В случае, если количество затраченных потомков тактов процессора достигает квантовой цели, запускается обработка истечения кванта;
- декремент счетчиков времени отложенных задач;
- в случае, если активен механизм профилирования ядра, инициализирует отложенный вызов обработчика ловушки профилирования ядра путём постановки объекта в очередь **DPC**: обработчик ловушки профилирования регистрирует адрес команды, выполнявшейся на момент прерывания.

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

- инициализация диспетчера настройки баланса путем сбрасывания объекта «событие», на котором он ожидает.

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

- инициализация диспетчеризации потоков – постановка соответствующего объекта в очередь **DPC**.

## 2 Пересчёт динамических приоритетов

В ОС семейства **UNIX** и **UNIX** динамически пересчитываться могут только приоритеты пользовательских процессов.

### 2.1 UNIX-системы

В современных системах **UNIX** ядро является вытесняющим. Это значит, что процесс в режиме ядра может быть вытеснен более приоритетным процессом, находящимся так же в режиме ядра. Это было сделано для того, чтобы система могла обслуживать процессы реального времени, например видео или аудио.

Согласно приоритетам процессов и принципу вытесняющего циклического планирования формируется очередь готовых к выполнению процессов. В первую очередь выполняются процессы с большим приоритетом. Процессы с одинаковыми приоритетами выполняются в течении кванта времени – циклически, друг за другом. В случае, если процесс, имеющий более высокий приоритет поступает в очередь готовых к выполнению процессов, планировщик вытесняет текущий процесс и предоставляет ресурс более приоритетному процессу.

Приоритет – это целое число, находящееся в диапазоне от 0 до 127. Чем меньше значение, тем выше приоритет процесса. Приоритеты ядра варьируются от 0 до 49, а приоритеты прикладных задач от 50 до 127. Приоритеты ядра являются фиксированными величинами, а приоритеты прикладных задач могут изменяться во времени в зависимости от следующих факторов:

- фактор любезности;
- последней измеренной величины использования процессора.

Фактор любезности – это целое число в диапазоне от 0 до 39 (по умолчанию 20). Чем меньше значение фактора любезности процесса, тем выше приоритет процесса. Фактор любезности процесса может быть изменен с помощью системного вызова **nice**, но только суперпользователем. Фоновым процессам задаются более высокие значения фактора любезности.

Дескриптор процесса **proc** содержит следующие поля, которые относятся к приоритету процесса:

- **p\_pri** – текущий приоритет планирования;

- **p\_usrpri** – приоритет процесса в режиме задачи;
- **p\_cpu** – результат последнего измерения степени загрузки процессора (процессом);
- **p\_nice** – фактор любезности, устанавливаемый пользователем.

Когда процесс находится в режиме задачи, значения **p\_pri** и **p\_usrpri** равны. Значение текущего приоритета **p\_pri** может быть повышено планировщиком для выполнения процесса в режиме ядра, а **p\_usrpri** будет использоваться для хранения приоритета который будет назначен когда процесс вернется в режим задачи.

Ядро системы связывает приоритет сна с событием или ожидаемым ресурсом, из-за которого процесс может блокироваться. В тот момент когда процесс просыпается, после того как был блокирован в системном вызове, ядро устанавливает приоритет сна в поле **p\_pri** – это значение приоритета в диапазоне от 0 до 49, зависящее от события или ресурса по которому произошла блокировка. В таблице 2.1 приведены значения приоритетов сна для систем **4.3BSD**.

Таблица 2.1: Таблица приоритетов 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                                  |

При создании процесса после **p\_cpu** инициализируется нулём. На каждом тике обработчик таймера увеличивает это поле для текущего процесса на единицу, до максимального значения, которое равно 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** фактор полураспада рассчитывается по формуле (2.1):

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

где **load\_average** – среднее количество процессов, находящихся в состоянии готовности к выполнению (за последнюю секунду).

Приоритеты для режима задачи всех процессов в процедуре **schedcpu()** пересчитываются 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, который равен 50.

Если процесс в последний раз использовал большое количество процессорного времени, его **p\_cpu** будет увеличен. Это приведёт к росту значения **p\_usrpri**, что приведет к понижению приоритету. Чем дольше процесс простаивает в очереди на исполнение, тем больше фактор полураспад уменьшает его **p\_cpu**, что приводит к повышению его приоритета. Данная схема предотвращает зависание низкоприоритетных по вине операционной системы. Применение такой схемы предпочтительнее процессам, которые осуществляют много операций ввода-вывода, в противоположность процессам, производящим много вычислений.

## 2.2 Windows-системы

В Windows при создании процесса, для него назначается базовый приоритет. Относительно базового приоритета процесса потоку назначается приоритет.

Планирование осуществляется на основе приоритетов потоков, готовых к выполнению. Поток с более низким приоритетом вытесняется потоком с более высоким приоритетом, в тот момент когда этот поток становится готовым к выполнению. По истечению кванта времени текущего потока, ресурс передается самому приоритетному потоку в очереди готовых на выполнение.

Раз в секунду диспетчер настройки баланса сканирует очередь готовых потоков, и, в случае, 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. В случае, 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.

Для того чтобы минимизировать расход процессорного времени, диспетчер настройки баланса сканирует только 16 готовых потоков. Диспетчер повышает приоритет не более чем у 10 потоков за один проход. Если он обнаруживает 10 потоков, приоритет которых

следует повысить, он прекращает сканирование. При следующем проходе сканирование возобновляется с того места, где оно было прервано. Наличие 10 потоков, приоритет которых нужно повысить, говорит о высокой загрузке системы.

Windows использует 32 уровня приоритета, которые описываются целыми числами от 0 до 31, а 31 – наивысший приоритет. Приоритеты от 16 до 31 – уровни реального времени, от 0 до 15 – динамические уровни. Уровень 0 зарезервирован для потока обнуления страниц.

Уровни приоритета потоков назначаются с двух позиций: **Windows API** и ядра операционной системы. **Windows API** сортирует процессы по классам приоритета, которые были назначены при их создании:

- реального времени (real-time, 4);
- высокий (high, 3);
- выше обычного (above normal, 6);
- обычный (normal, 2);
- ниже обычного (below normal, 5).
- простой (idle, 1).

API-функция SetPriorityClass позволяет изменять класс приоритета процесса до одного из этих уровней.

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).

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

Таким образом, в **Windows API** каждый поток имеет базовый приоритет, являющийся функцией класса приоритета процесса и его относительного приоритета процесса. В ядре класс приоритета процесса преобразуется в базовый приоритет. В таблице 2.2 приведено соответствие между приоритетами **Windows API** и ядра системы приоритета.

Таблица 2.2: Соответствие между приоритетами **Windows API** и ядра Windows

|                      | <b>real-time</b> | <b>high</b> | <b>above normal</b> | <b>normal</b> | <b>below normal</b> | <b>idle</b> |
|----------------------|------------------|-------------|---------------------|---------------|---------------------|-------------|
| <b>time critical</b> | 31               | 15          | 15                  | 15            | 15                  | 15          |
| <b>highest</b>       | 26               | 15          | 12                  | 10            | 8                   | 6           |
| <b>above normal</b>  | 25               | 14          | 11                  | 9             | 7                   | 5           |
| <b>normal</b>        | 24               | 13          | 10                  | 8             | 6                   | 4           |
| <b>below normal</b>  | 23               | 12          | 9                   | 7             | 5                   | 3           |
| <b>lowest</b>        | 22               | 11          | 8                   | 6             | 4                   | 2           |
| <b>idle</b>          | 16               | 1           | 1                   | 1             | 1                   | 1           |

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

- повышение приоритета после завершения операций ввода-вывода;
- повышение приоритета владельца блокировки;
- повышение приоритета вследствие ввода из пользовательского интерфейса;
- повышение приоритета вследствие длительного ожидания ресурса исполняющей системы;
- повышение приоритета вследствие ожидания объекта ядра;
- повышение приоритета в случае, когда готовый к выполнению поток не был запущен в течение длительного времени;
- повышение приоритета проигрывания мультимедиа службой планировщика **MMCSS**.

Текущий приоритет потока в динамическом диапазоне может быть понижен до базового путем вычитания всех его повышений. В таблице 2.3 приведены рекомендуемые значения повышения приоритета для устройств ввода-вывода.



Таблица 2.3: 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          |

### 2.2.1 MMCSS

Потоки, на которых выполняются различные мультимедийные приложения, должны выполняться с минимальными задержками. В Windows эта задача решается путем повышения приоритетов таких потоков драйвером **MMCSS** – MultiMedia Class Scheduler Service. Приложения, которые реализуют воспроизведение мультимедиа, указывают драйверу **MMCSS** задачу из списка:

- аудио;
- игры;
- распределение;
- захват;
- воспроизведение;
- задачи администратора многоэкранного режима.

Одно из наиболее важных свойств для планирования потоков – категория планирования – первичный фактор определяющий приоритет потоков, зарегистрированных в **MMCSS**. Различные категории планирования представлены в таблице 2.4.

Функции **MMCSS** временно повышают приоритет потоков, зарегистрированных с **MMCS** до уровня, соответствующего их категориям планирования. Далее, их приоритет снижается до уровня, соответствующего категории **Exhausted**, для того чтобы другие потоки могли получить ресурс.

## Вывод

Функции обработчика прерывания от системного таймера в защищенном режиме в системах Unix и Windows выполняют одинаковые действия:

Таблица 2.4: 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 продолжиться, только если не будут готовы к выполнению другие потоки с более высоким уровнем приоритета |

- выполняют декремент счетчиков времени: часов, таймеров, счетчиков времени отложенных действий, будильников реального времени;
- выполняют декремент кванта текущего процесса в Linux и декремент текущего потока в Windows;
- инициализируют отложенные действия, относящиеся к работе планировщика, такие как пересчёт приоритетов.

Обе системы являются системами разделения времени с динамическими приоритетами и вытеснением. Это объясняется тем, что такой подход обеспечивает более высокой эффективность использования всех вычислительных ресурсов. Пересчет динамических приоритетов пользовательских процессов выполняется для того чтобы исключить их бесконечное откладывание.